Circuit Techniques for High-Speed Serial and Backplane Signaling

Marcus Henricus van Ierssel

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy, Department of Electrical and Computer Engineering University of Toronto

© Copyright by Marcus Henricus van Ierssel 2007
Circuit Techniques for High-Speed Serial and Backplane Signaling

Marcus Henricus van Ierssel
Doctor of Philosophy
Department of Computer and Electrical Engineering
University of Toronto
2007

ABSTRACT

This thesis describes three contributions towards the design and modeling of high-speed serial and backplane signaling systems. Specifically, these contributions address the issues of jitter and intersymbol interference in clock and data recovery circuits.

First, this thesis proposes the modeling of jitter in clock and data recovery (CDR) systems using an event-driven model that accurately includes the effects of power-supply noise, the finite bandwidth in the phase-detector’s front-end sampler, and ISI in the system’s channel. These continuous-time jitter sources are captured in the model through their discrete-time influence on sample based phase-detectors. Modeling parameters for these disturbances are directly extracted from the circuit implementation. The jitter predicted by the event-driven model, implemented in Simulink, is within 12% of that predicted by an Hspice simulation, but with a simulation speed that is 1800 times higher.

Second, this thesis proposes a new adaptation technique for a 4-PAM decision-feedback equalizer (DFE). The DFE adapts to the channel impulse response by observing an intermittent calibration sequence sent across the channel. Uninterrupted signaling is maintained across a parallel bus by providing an additional channel and using multiplexers to reroute the signals of the channel being calibrated. Using the intermittent calibration sequence, instead of the conventional
LMS adaptation technique, removes the need to generate an error signal, eliminating the associated analog blocks. Also presented is a novel method of using the DFE adaptation circuits to extract the system’s pulse response. The complete transceiver is implemented in a 0.18 μm CMOS process.

Finally, a hybrid CDR is proposed that increases the jitter tolerance of a phase-tracking CDR by a factor of 32 at frequencies below its loop filter’s bandwidth, while maintaining the high-frequency jitter tolerance of a 5x blind-oversampling CDR. Measured data from a 0.11um CMOS testchip at 2.4Gb/s confirm a 200UI peak-to-peak jitter tolerance for a 200kHz jitter. The testchip is operational from 1.9Gb/s to 3.5Gb/s and consumes 115mW at 2.4Gb/s.
Acknowledgements

A PhD is a long journey, and like most journeys it is made richer by the people one meets along the way. Primary thanks goes to my supervisor, Prof. Ali Sheikholeslami, for convincing me to start this journey, and for then dragging me kicking and screaming to its finish.

I thank the defense committee for their time and valuable suggestions: Prof. Dave Johns, Prof. Tony Carusone and Prof. Paul Chow. Thank you also to the external examiner Yuriy Greshishchev of Nortel Networks for taking the time to provide his evaluation, as well as travelling to Toronto to participate in the defense.

There are a large number of students who deserve thanks for help, friendship, and tolerance. While any attempt at an exhaustive list would be futile, it would be wrong to not explicitly thank my fellow members of the Ali-group. Thanks go to Yadi Eslami for being an ideal cube-mate and tolerating all my irritating habits. Thanks go to Kostas Pagiamtzis for sharing many hours discussing circuits, politics, and listening to my rants. Thank you to Joyce Wong for her part in taping out a chip under a punishing timetable. Thank you to Oleksiy Tyshchenko for his help even while he laboured under his own punishing schedule. Although there are too many to list, my thanks go out to all those other students I have encountered over the years.

Special thanks also go to those at Fujitsu with whom I have worked over the course of this degree. In particular, I would not have been able to get to this point without the generous assistance of both Hirotaka Tamura and Bill Walker.

It has also been my honour to have known and worked with Laura Fujino and Prof. K.C. Smith. over the course of my studies. Through their assistance, I was able to attend ISSCC six times as a student volunteer. Their support and assistance through the course of my degree greatly enriched my journey.

I also need to thank my friends and family for their support and understanding over the years. That they remain after years of virtual abandonment is a testament to their commitment, and I am indebted to them. Finally I must thank my wife, Shannon, for her support and understanding throughout this endeavour, especially during the 120 hour work weeks prior to tapeouts.

Marcus van Ierssel

March 30, 2007
# Table of Contents

## Chapter 1  Introduction

1.1 Motivation ........................................................................................................ 1
1.2 Design Challenges and Approaches ............................................................... 2
1.3 Thesis Objectives and Contributions .............................................................. 3
1.4 Thesis Outline .................................................................................................. 4

## Chapter 2  Fundamental Problems in High-Speed Signaling

2.1 Channel impairments and their implications .................................................. 6
  2.1.1 Inter-symbol interference ....................................................................... 7
  2.1.2 Timing uncertainty ............................................................................. 11
2.2 Decision Feedback Equalization .................................................................... 12
  2.2.1 Adaptive Equalization ....................................................................... 14
2.3 Clock and Data Recovery ............................................................................... 17
  2.3.1 Phase Tracking .................................................................................. 18
  2.3.2 Blind-oversampling Clock and Data Recovery .................................. 24

## Chapter 3  Event-Driven CDR Modeling

3.1 Event-Driven CDR Simulation ....................................................................... 31
3.2 Modeling Power Supply Noise ....................................................................... 35
  3.2.1 Model Verification ............................................................................ 39
3.3 Sampler Impulse Response ............................................................................ 40
  3.3.1 Model Verification ............................................................................ 43
3.4 Event-Driven Implementation of the Data Filter .......................................... 44
  3.4.1 ISI Model Verification ....................................................................... 46
3.5 Putting it all Together - System Level Simulation Results .......................... 46
  3.5.1 Clock jitter due to supply noise ......................................................... 47
  3.5.2 Clock jitter due to limited bandwidth ............................................... 48
  3.5.3 Simulation Time ............................................................................... 51
3.6 Summary ....................................................................................................... 51

## Chapter 4  An Adaptation Technique for 4-PAM Decision-Feedback Equalization

4.1 Basic DFE Architecture ............................................................................... 54
4.2 Traditional adaptive DFE architecture ......................................................... 56
4.3 Proposed adaptive DFE using intermittent calibration sequence ............... 58
Chapter 1

Introduction

High-speed signaling, in this thesis, refers to the transfer of digital data at bit-rates in excess of approximately 1Gb/s. High-speed signaling can be distinguished from conventional signaling not only by the high bit-rates, but also by the complexity of the required circuits. Conventional signaling systems transmit and receive full-swing single-ended signals using simple circuits such as CMOS inverters. In contrast, high-speed signaling systems typically use low-swing differential signaling and contain complex circuits having transistor counts that can exceed several thousand. This complexity has arisen primarily due to the non-ideal channel between the transmitter and the receiver. While significant progress has been made in channel materials, cost pressures have resulted in the continued use of older materials. As a result, increasing bit-rates require increasingly more complex solutions to the resulting problems, particularly those of timing uncertainty and intersymbol interference. This thesis presents two implemented design solutions, one addressing each of these problems. In addition, this thesis presents an modeling framework for the simulation of timing uncertainty in clock and data receiver designs that extends the functionality of existing discrete-time models.

1.1 Motivation

High speed signaling is used in a wide variety of digital systems, such as the computer, digital television, and network router. Field programmable gate arrays (FPGAs) are commonly used to implement portions of many of these systems. As with most integrated circuit technology, the internal speed and density of FPGAs has seen order of magnitude increases in the past few decades. To fully utilize this increased performance at a system level requires a similar increase in the bandwidth of the FPGA interconnect. To satisfy this requirement, FPGA vendors have introduced gigabit rate high-speed serial I/O blocks that implement functions such as equalization and clock recovery into many of their high-performance devices [1][2].
The International Technology Roadmap for Semiconductors (ITRS) roadmap predicts that signaling bandwidth will continue to rise exponentially for at least the next decade, as illustrated in Fig. 1.1 [3], reaching 50Gb/s by 2017. As illustrated in the following section, the design solutions that allow current designs to operate at 5Gb/s will not suffice as we progress towards 50Gb/s, requiring new solutions.

1.2 Design Challenges and Approaches

While the bit-rate in high-speed signaling has risen exponentially over time, cost pressures have resulting in the continued use of older channel materials. This channel is often a trace on a printed circuit board (PCB) that is a few inches long. These PCBs are usually made from a material known as FR4, which has remained unchanged for many decades [4]. When signaling rates were at 25Mb/s, the corresponding bit-period (40ns) was substantially larger than the propagation delay of a 6” PCB trace (around 1ns). This allowed the trace to be modeled as a single electrical node connecting the transmitter to the receiver. However, at a signaling rate of 5Gb/s, the corresponding bit-period is 200ps, which is smaller than the 1ns propagation delay through the same trace. As a result, the trace must now be modeled as a transmission-line, with the associated properties of finite signal-propagation speed, trace impedance, high-frequency attenuation, and crosstalk.

Numerous problems result when attempting to perform high-speed signaling over PCB traces due to the above mentioned properties. Among the most difficult problems to solve are intersymbol-interference (ISI), and timing uncertainty in the receiver.
ISI is a distortion of the received signal due to limited channel bandwidth—in this case from the high-frequency attenuation in the PCB trace. This distortion takes the form of a time-domain smearing of the signal such that pulses spread out and interfere with adjacent pulses. Beyond a certain threshold, ISI can cause sufficient distortion that a sampling receiver can no longer accurately resolve the level of the transmitted pulse.

To maintain signal integrity in the presence of ISI, current high-speed signaling implementations use equalization. Equalizers are filters that compensate for the high-frequency attenuation caused by the channel. These filters have a transfer function which is ideally the inverse of the channel transfer function. Historically, when equalizers were first used they were simple discrete-time feed-forward equalizers in the transmitter [5]. As channel attenuation increased, transmitter equalization alone often proved inadequate, and designs introduced additional linear equalizers in the receiver [6]. Both of these solutions may result in noise enhancement and increased crosstalk. To combat these issues, some designs have employed decision-feedback equalization (DFE) [7]. Chapter 4 presents the design of a DFE using adaptive equalization.

In addition to compensating for ISI, the high-speed signaling system also needs to provide timing information to the receiver so the boundaries between the data symbols can be determined. Uncertainty in the timing of these boundaries is referred to as timing uncertainty, and like ISI can be attributed to properties of the channel. As described earlier, when data rates were sufficiently low the propagation time through a PCB trace was much smaller than the symbol period. In this case, a global clock could be distributed to all the components in a system. As signaling rates increased, the propagation times through the PCB traces became comparable to the clock periods, and global clocking gave way to source-synchronous clocking [8]. As signaling rates rose still further, the shrinking clock periods made it increasingly difficult to match the propagation times on separate clock and data traces. The clock signal is now often left out entirely, instead relying on clock and data recovery (CDR) in the receiver to extract the clock information embedded within the transmitted data [9]. Chapter 5 presents the design of a semi-blind oversampling CDR designed for increased jitter tolerance.

1.3 Thesis Objectives and Contributions

This thesis explores solutions to the problems of channel equalization and timing recovery. An equalization solution is presented in the form of an adaptive 4-PAM DFE. Timing recovery is
addressed in an event-driven CDR model with enhanced simulation accuracy, and a hybrid CDR design which significantly increases the jitter tolerance of a conventional CDR.

The first contribution of this thesis is an enhanced event-driven model for simulation of CDR jitter. This model accurately includes the effects of power-supply noise, the finite bandwidth of the front-end sampler, and ISI in the system’s channel. The model exploits the nature of CDRs with sample-based phase detectors to allow these jitter sources to be described only in terms of their impact on the phase detector’s samples. Modeling parameters for these disturbances are directly extracted from the circuit implementation. The model has a simulation accuracy close to that of an Hspice simulation, but with a simulation speed that is 1800 times faster.

The second contribution of this thesis is a 4-PAM adaptive DFE. The DFE adapts to the channel impulse response by observing an intermittent calibration sequence sent across the channel. Using an intermittent calibration sequence instead of the conventional LMS adaptation technique eliminates the hardware typically used to generate the adaptation error signal. This results in increased speed and reduced hardware complexity. Also presented is a novel method of using the DFE adaptation circuits to extract the system’s pulse response.

The third contribution of this thesis is a semi-blind oversampling CDR designed for increased jitter tolerance. This hybrid CDR increases the jitter tolerance of the conventional phase-tracking CDR by a factor of 32 at low frequencies, while maintaining the high-frequency jitter tolerance of a 5x blind-oversampling CDR. A test chip was fabricated to confirm the functionality of the CDR.

1.4 Thesis Outline

This thesis is organized as follows. Chapter 2 provides an overview of the problems, and some of their solutions, encountered in the design of a high-speed signaling transceiver, as well as establishing the terminology used in the remainder of this dissertation. The key contributions of this thesis are presented in Chapters 3, 4, and 5. Chapter 3 presents enhanced event-driven modeling techniques for the simulation of CDR jitter. Chapter 4 presents an adaptive 4-PAM DFE. Chapter 5 presents a semi-blind oversampling CDR designed for increased jitter tolerance. The results of this thesis are summarized in Chapter 6, along with a discussion on potential future areas of investigation.
Chapter 2

Fundamental Problems in High-Speed Signaling

This chapter presents some of the fundamental problems in transceiver design for high-speed signaling, and describes their corresponding design solutions. Our main focus in this chapter is on equalization and clock and data recovery (CDR) as they provide the necessary context for the contributions described in this dissertation.

High-speed signaling, in general, refers to Gigabit rate communication between a transmitter and a receiver through a communication channel. This channel can take many forms, such as a network interface, (e.g. ethernet cable), a peripheral interface (e.g. USB cable), or a PCB trace between two chips, in the case of high-speed chip-to-chip signaling. Despite the varied nature of these different channels, all the applications that use them must solve similar design problems. However, the different applications require different solutions owing to their differing cost functions. For example, a typical desktop computer will have only one network interface and a small number of peripheral interfaces, while having hundreds of high-speed chip-to-chip connections. On the other hand, network interfaces often communicate over hundreds of feet of cable, peripheral interfaces over a few feet of cable, and chip-to-chip signaling interfaces over less than one foot of PCB trace. As a result, network applications often require complex solutions that may need an entire chip to implement, peripheral applications require a moderately complex solution that can still be integrated with other devices, and the chip-to-chip signaling applications tend to emphasize low power and occupy only a small fraction of the chip area.

The block diagram of a generic high-speed signaling transceiver is shown in Fig. 2.1, where the transmitter is integrated on one chip along with the data source, and the receiver is integrated on

![Fig. 2.1 High-speed signaling transceiver](image-url)
another chip along with the data consumer. The purpose of the transceiver is to move data from the data source to the data consumer through a non-ideal channel. This channel, typically, has a limited bandwidth and a finite signal propagation speed. The role of the transceiver, therefore, is to compensate for the channel non-idealities and to provide for an acceptably low bit error rate (BER). While a contemporary transceiver consists of a wide variety of circuits blocks, such as line drivers, serializers, deserializers and samplers [10], the bulk of its circuit design complexity exists to compensate for the channel impairments.

In the remainder of this chapter we discuss these channel impairments and their corresponding design solutions. Limited channel bandwidth is shown to cause inter-symbol interference, and a technique known as equalization is usually used to overcome it. An increasingly popular form of equalization, known as decision feedback equalization, is discussed in detail. Finite signal propagation speed is shown to cause timing uncertainty in the receiver, requiring increasingly complex clock distribution schemes. Of these schemes, the CDR is currently found in the majority of transceiver designs, and is also examined in detail.

2.1 Channel impairments and their implications

The channel in serial and backplane signaling usually comprises multiple elements, such as cables, connectors, and traces on a printed circuit board (PCB). Although all of these elements have different properties, the impairments found in the PCB trace are still representative of the impairments found in other elements. The cross-sectional structure of a typical 50 Ohm PCB trace is shown in Fig. 2.2(a), where a 0.024” wide copper trace runs over a 0.013” thick FR4 substrate with an underlying ground plane. It is the combination of the trace, FR4 material, and ground plane which is considered the communications channel. This channel is often treated as a linear time-invariant system with a typical frequency response shown in Fig. 2.2(b) for a 30cm-long trace. At high frequencies (typically 1 GHz and above), skin effect in the copper conductor and dielectric losses in the substrate cause this channel to increasingly attenuate the propagating signal [11]. This high-frequency channel attenuation results in inter-symbol interference that impairs the ability of the receiver to resolve the transmitted symbol from a sample of the received signal. We will discuss this topic in Section 2.1.1.

In addition to high-frequency attenuation, the channel also suffers from finite signal propagation speed. For the structure shown in Fig. 2.2(a), the propagation speed is $1.6 \times 10^8$ m/s (approximately
half the speed of light). At this speed, it takes almost 19 bit periods at 10Gb/s for data to propagate through the above 30 cm trace. Uncertainty in the propagation speed makes it difficult to synchronize circuits across multiple chips. This contributes to timing uncertainty in the receiver as it impairs its ability to determine when to sample the received signal. Timing uncertainty is discussed in Section 2.1.2.

2.1.1 Inter-symbol interference

In the time-domain, the result of high-frequency attenuation is the temporal spreading of a symbol transmitted during one symbol-period into adjacent symbol-periods [12]. This process, known as intersymbol-interference (ISI), is illustrated in Fig. 2.3 for the channel described above. This figure shows a transmitted 100ps square pulse and its corresponding signal observed at the receiver. It can be seen that the received waveform fails to reach full amplitude during the symbol period, and fails to reach zero before the following symbol period. As the symbol rate increases, a larger fraction of a pulse’s energy is spread across neighbouring symbols, to the point that it is
difficult to identify the transmitted symbol, eventually resulting in an unacceptable BER at the receiver.

The solution commonly used to compensate for ISI is equalization. An equalizer is a filter with a transfer function that is ideally the inverse of the channel’s transfer function (over the frequency range of interest). The cascade of the channel and the equalization filter has a transfer function that is approximately flat over the bandwidth of the signal, eliminating ISI. These equalizers can be implemented in the transmitter [5][13][14], receiver [15][16][6], or both [17][18][20].

Equalization performed in the transmitter is known as pre-equalization [5][13][14]. This type of equalization boosts the high-frequency content of the transmitted signal before the attenuation effects of channel take place. In a time-domain view of pre-equalization, the transitions of the transmitted signal are emphasized, in order to counter the attenuation effect of the ISI on the received pulse. An example of this is shown in Figure 2.4. After passing through the channel, the combination of pre-equalization and channel attenuation cancel out, leaving an ISI-free signal at input to the receiver.

Simple filter implementations are often the preferred choice for equalization when dealing with Gb/s signaling rates. One example of a simple pre-equalization filter is the two-tap FIR filter design

Fig. 2.3 Impact of ISI on 100ps pulse
shown in Figure 2.5. This circuit produces a load current \( I_{\text{load}} \) that is proportional to the difference between the current symbol, \( x_n \), and a fraction of the previous symbol, \( x_{n-1} \):

\[
I_{\text{load}} = I_0 x_n - I_1 x_{n-1}
\]

\[
= I_0 \left( x_n - \frac{I_1}{I_0} x_{n-1} \right)
\]  

(2.1)

By controlling this fraction through the ratio \( I_1/I_0 \), the designer can control the degree of pre-emphasis. This scheme is used in [5] in the design of a 1.0625Gb/s transceiver with digitally-controlled current-sources allowing a controllable amount of equalization. The popularity of this technique lies in its simple implementation, requiring only an extra differential pair and a delay element to create \( x_{n-1} \) from \( x_n \).

Equalization performed in the receiver is known as post-equalization [6][15][16], as it is applied to the distorted received signal. As in pre-equalization, the goal of post-equalization is to
compensate for the effects of ISI due to previous symbols. However, unlike the transmitter, the receiver does not have easy access to the previously transmitted symbols (from which the ISI is derived) and must operate using the previously received symbols—which may not be necessarily identical. The previously received symbol can be acquired using one of two approaches. In the first approach, the analog level of the received signal is sampled during every symbol period and a fraction of this symbol is subtracted during the subsequent symbol period. This approach is known as linear equalization. In the second approach, the digital output of the receiver is used to provide the previous symbol. This approach is known as decision feedback equalization (DFE). This is a non-linear approach due to the non-linear nature of the decision circuit within the feedback loop.

In [6], a linear post-equalizer is implemented using a switched-capacitor design that approximates the channel response as an RC filter, allowing the state of the ISI to be modelled as only dependent on the previously sampled receiver input. With this approximation, equalization can be performed by subtracting a fraction of the previously sampled receiver input from the current input. This concept is shown in Figure 2.6(a), with the equivalent signal flow graph in Figure 2.6(b). The switched-capacitor implementation of this design is shown in Figure 2.6(c), where $V$ is the receiver input, $V_{tt}$ is a reference voltage, and $\phi_1$ and $\phi_2$ are two phases of a clock. Each clock phase is synchronized with one bit of transmission. Note that because the system requires one clock phase per bit period, this design requires two of these circuits, interleaved on...
opposite clock phases. The degree of equalization performed, $\alpha$, is determined by the ratio of capacitors $C_1$ and $C_2$.

Both the pre-equalizer and linear post-equalizer described above suffer from noise enhancement, which degrades the signal to noise ratio (SNR) at the receiver [12] and can result in data errors. In any real system, there will be noise introduced into the signal between the transmitter and receiver. In pre-equalization schemes, high-frequency signal energy is increased at the cost of low-frequency signal energy. The high-frequency energy is then attenuated in the channel, decreasing the total received signal energy and resulting in a reduced SNR. Similarly, in linear post-equalization schemes, the equalizer amplifies the high-frequency signal energy, but also amplifies the high-frequency noise energy. While the high-frequency signal energy is usually only a modest fraction of the total signal energy, the amplified (enhanced) high-frequency channel noise can increase the total noise energy by as much as an order of magnitude or more. Thus, linear post-equalizers also result in a decreased SNR. This process of noise-enhancement limits the usefulness of linear equalization in high-attenuation environments.

Noise enhancement can be avoided through the use of non-linear equalization techniques, such as the DFE, which uses the digital decisions of the receiver to drive the equalization filter. Because the decision process eliminates noise in the received signal, the DFE amplifies the high-frequency signal energy without boosting the high-frequency noise energy. This improves the SNR at the receiver, providing increased link reliability. The DFE is discussed in more detail in Section 2.2.

2.1.2 Timing uncertainty

To sample the received signal at the correct time, the receiver requires timing information, usually in the form of a reference clock. Once again, it is the channel impairments that interfere with the acquisition of this reference, requiring the use of complicated circuit solutions such as clock and data recovery (CDR).

When signaling speeds are low enough (less than 25MHz) to consider short PCB traces as single electrical nodes, a global clock can be broadcast to all circuits in a system to provide synchronization, as shown in Fig. 2.7(a). Above this speed, the signal propagation times through PCB traces become comparable to the bit-periods, and source-synchronous clock systems become necessary to maintain synchronization [8]. In these schemes, shown in Fig. 2.7(b), the clock and data are transmitted together such that they share a common propagation delay. However, as
signaling rates approaching 1GHz, it becomes increasingly difficult to match the propagation times of the data and clock signals, and CDRs become necessary. In systems using CDRs, only the data signal is transmitted, and the CDR uses transitions within the data signal to extract timing information and reconstruct the clock signal, as shown in Fig. 2.7(c). The recovered clock is then used to sample and recover the data. At data rates in the 10Gb/s range, high-speed signaling systems rely almost entirely on CDRs, although this is also due to the economic advantage of eliminating the package pins and signal traces previously used for clock transmission. We discuss CDRs in detail in Section 2.3.

2.2 Decision Feedback Equalization

A DFE is an equalizer that uses the past decisions of the receiver and an estimate of the channel impulse response to create and subtract a replica of the ISI from the current symbol [7][19][20][22]. The block diagram of a simple DFE is shown in Fig. 2.8, where \( x[n] \) is the received signal, \( y[n] \) is the equalized signal, and \( \hat{y}[n] \) is the receiver’s decision on the transmitted symbol, based on the slicing of \( y[n] \). The decision feedback filter, \( W[z] \), is an N-tap discrete time FIR filter of the form:

Fig. 2.7 Clocking schemes: (a) global, (b) source-synchronous, (c) clock and data recovery
Thus the equalized signal, $y(n)$, can be expressed as:

$$y[n] = x[n] - \sum_{i=1}^{N} w_i \hat{y}[n-i]$$  \hspace{1cm} (2.3)

The summation in this equation represents the predicted ISI in the current symbol. Because $\hat{y}[n]$ represents the reconstruction of the transmitted symbol sequence, it is only possible for the summation to represent the ISI if $W(z)$ is equal to the discrete-time impulse response of the channel. As we will see later, adaptive equalization is often used to ensure this equality.

The primary advantage of the DFE architecture is that it does not cause noise enhancement. This is because the source of the feedback path is the noiseless decision, $\hat{y}[n]$, not the noisy $y[n]$. Assuming that the decisions are correct (or that they have a very low BER), then the slicing process eliminates noise on $y[n]$ to create the discrete levels of $\hat{y}[n]$. This non-linear feedback system boosts the high-frequency content of the signal without boosting the noise power, as linear filters do.

Another advantage of the non-linear DFE feedback is the simplicity of its implementation. The feedback consists of digital samples which can be delayed using flip-flops, instead of the analog delay elements needed for linear discrete-time filters.

While the DFE benefits from the lack of noise enhancement, the decision-based feedback can result in error propagation. If the DFE makes a wrong decision, the non-linear feedback will distort future symbols, potentially resulting in further errors. As a result of this error propagation, it is common for DFE errors to occur in bursts [12].
Another disadvantage of the DFE is its inability to compensate for pre-cursor ISI. Because the DFE uses past decisions to calculate the ISI present in the current symbol, it is impossible for the DFE to calculate the pre-cursor ISI, as this would require knowledge of future decisions.

The primary challenge of the DFE, however, is the critical path formed by its feedback loop, which limits its symbol rate. The critical path of the DFE shown in Fig. 2.8 starts at the output, \( \hat{y}[n] \), going through the loop filter, \( W[z] \), and then through the signal adder to \( y[n] \). This signal is then sliced, producing the next instance of \( \hat{y}[n] \). For the DFE to function correctly, the delay through this path must be less than the symbol period of \( x[n] \). Typically, the dominant contributors towards this delay are the settling time of the \( y[n] \) node due to its large capacitive load, and the propagation delay through the slicer, usually implemented as a sense-amp latch.

### 2.2.1 Adaptive Equalization

In addition to requiring sufficient speed in the feedback path, the DFE (or any equalizer) requires its filter parameters to match the channel characteristics. Since knowledge of the channel ISI is often approximate at design time, and may even vary with process and temperature, the only viable technique to guarantee correct ISI compensation is adaptive equalization. Adaptive equalization refers to dynamically adjusting the filter parameters of an equalizer in order to minimize ISI in its received signal [23]. This sub-section describes the operation of a generic adaptive equalizer, and then tailors the concept to DFEs.

Fig. 2.9 shows the conceptual block diagram of an adaptive equalizer where the received signal, \( x[n] \), is applied to the equalization filter \( W[z] \), producing the equalized signal \( y[n] \). Adaptation of the equalization filter is driven by the equalization error signal, \( err[n] \). This signal is defined as

\[
err[n] = y[n] - \hat{y}[n]
\]

![Fig. 2.9 Conceptual adaptive equalization block diagram](image-url)
the difference between the equalized signal, \( y[n] \), and a reference signal, \( \delta[n] \), representing the transmitted data sequence:

\[
err[n] = \delta[n] - y[n]
\] (2.4)

In the well known least-mean-square (LMS) adaptation scheme [12], the equalizer adapts the filter coefficients, \( w_i[n] \), to minimize the average power of the error signal, \( E[err^2[n]] \). This average error power varies over the multi-dimensional space defined by all possible \( w_i \). The steepest-descent algorithm [24] assumes that this space is well-behaved, such that there is a point in this space with minimum error power, and that following the error gradient from any point in the space will lead to this point of minimum error power. The steepest-descent algorithm exploits this by iteratively traversing the error gradient towards the minimum. This iterative process is expressed mathematically as:

\[
w_i[n + 1] = w_i[n] - \mu \left( \frac{\partial E[err^2[n]]}{\partial w_i} \right)
\] (2.5)

where \( \mu \) is the adaptation gain that determines the size of the coefficient steps, and is chosen for a balance between adaptation speed and convergence stability. If the gain is set too high, the coefficients can overshoot and potentially diverge from the minimum.

Since we do not know the value of \( E[err^2[n]] \) a priori, the LMS algorithm replaces the expected error squared with the instantaneous error squared:

\[
w_i[n + 1] = w_i[n] - \mu \left( \frac{\partial err^2[n]}{\partial w_i} \right) = w_i[n] - 2\mu err[n] \left( \frac{\partial err[n]}{\partial w_i} \right)
\] (2.6)

Although this simplification results in many individual adaptation iterations in the wrong direction, on average the adaptation iterations will still proceed toward the minimum of \( E[err^2[n]] \) [12]. We can further simplify (2.6) with a substitution of \( err[n] = \delta[n] - y[n] \) from (2.4):

\[
w_i[n + 1] = w_i[n] + 2\mu err[n] \left( \frac{\partial y[n]}{\partial w_i} \right)
\] (2.7)
This equation, which describes the functionality of the *adaptive algorithm* block in Fig. 2.9, is valid for all LMS adaptive equalizers. The difference between various implementations stem from the derivations of the error signal and its partial derivative.

We now tailor the adaptive algorithm to the specific context of an adaptive DFE [25][26]. Fig. 2.10(a) shows the block diagram of an adaptive DFE, derived from the combination of Fig. 2.8 and Fig. 2.9. When (2.7) is applied to the adaptive DFE, we can simplify $\frac{\partial y[n]}{\partial w_i}$ by evaluating the derivative of (2.3), which defines the equalized DFE signal $y[n]$:

$$\frac{\partial y[n]}{\partial w_i} = \hat{y}[n-i] \tag{2.8}$$

Fig. 2.10 DFE error signal generation based on (a) reference, (b) DFE decision
Thus the partial derivative can be acquired directly from the delayed decisions used in the feedback filter. When this is substituted into (2.7), the DFE adaptation algorithm simplifies to:

\[ w[n + 1] = w_i[n] + 2\mu err[n] \hat{y}[n - i] \]  

(2.9)

To implement this algorithm we still need to determine the error signal, \( err[n] \). In Fig. 2.10(a), \( err[n] \) is derived using the reference signal, \( \delta[n] \). Usually, this reference is not available in the receiver. If it were, then there would be no need to equalize \( x[n] \), as we could just use \( \delta[n] \). Conveniently, for reliable systems with low bit-error rates, it is reasonable to assume that the DFE output, \( \hat{y}[n] \), is correct and can be substituted for \( \delta[n] \). This substitution is reflected in the block diagram shown Fig. 2.10(b).

This approach (and its related variations) to error signal generation has the effect of introducing additional capacitive loading placed on the \( y[n] \) node, slowing the critical path of the DFE. In order to avoid this problem, we propose in Chapter 4 an adaptive DFE architecture that does not require any additional error signal generation hardware connected to this critical node.

### 2.3 Clock and Data Recovery

A CDR uses the transitions between different data bits to align the phase of the recovered clock edge to the phase of the embedded clock, as shown in Fig. 2.11. The recovered clock is then used to sample the received signal and recover the data.

The traditional approach to CDR design uses a local oscillator placed within a feedback loop that causes the phase of the local oscillator to track the phase of the embedded clock, as derived from the data transitions [9][21][28][29]. This approach is known as phase-tracking, and is described in the diagram shown in Fig. 2.11.

**Fig. 2.11 Clock and data timing relationship**
Section 2.3.1. Another approach to clock and data recovery uses blind-oversampling of the received sequence, followed by digital logic that determines the clock phase, and recovers the data [30][31]. This approach, described in Section 2.3.2, has the advantage of a fully digital implementation.

The accuracy with which a CDR recovers and tracks the embedded clock directly impacts the reliability of the system [32]. If the recovered clock is unable to track timing changes in the received signal, then the signal will be sampled at the wrong time, resulting in errors. Predicting the accuracy of clock recovery at design time is critical to guarantee the correct operation of the design.

While frequency-domain analysis on linearized CDR models provides good initial performance estimates, time-domain simulation is often required to provide more accurate predictions. This is particularly true when strong non-linearities are present in the CDR. These simulations can be very time consuming, often taking weeks or even months for high-accuracy circuit simulations. In chapter 3 of this thesis we present an event-driven simulation technique for time-domain analysis of CDR behaviour that provides fast and accurate results.

2.3.1 Phase Tracking

The phase-tracking clock recovery loop in a CDR, shown in Fig. 2.12, is identical to that of a PLL [33], except that the phase-detector reference is the received data signal, not a reference clock. The phase-detector compares the phase of the recovered clock to the phase of the received signal, and generates an output that is, ideally, proportional to the phase difference of its inputs. This phase-difference output is then low-pass filtered, and used to adjust the phase of the recovered clock.
This clock is typically derived from a VCO, as shown in Fig. 2.12, but can also be derived from a phase interpolator that shifts the phase of a plesiochronous reference clock [34][35].

To analyse the behaviour of a CDR with respect to input and output clock phase, it is common to draw a corresponding signal flow graph, as shown in Fig. 2.13, where the $\Phi_{in}(s)$ is the phase of the clock embedded in the received signal, and $\Phi_{osc}(s)$ is the phase of the recovered clock. In this graph, the phase detector has been modeled by an adder and a gain stage of $K_{pd}$, the low-pass loop filter is represented by its transfer function, $H_{lp}(s)$, and the VCO is represented as an integrator with a frequency gain of $K_{vco}$.

Although phase detection in the CDR occurs as a series of discrete events, the signal flow graph represents it as a continuous-time process. This provides a useful way of analysing the CDR at frequencies significantly lower than the clock rate. Various types of CDR behaviour, such as jitter transfer, jitter tolerance, and loop stability, can be analysed in the frequency domain using this approach.

The jitter transfer function is defined as the ratio of the recovered clock phase to the phase of the received signal. Using the signal flow graph in Fig. 2.13, this ratio can be written as:

$$
\frac{\Phi_{osc}(s)}{\Phi_{in}(s)} = \frac{K_{pd}K_{vco}H_{lp}(s)}{s + K_{pd}K_{vco}H_{lp}(s)} \quad (2.10)
$$

This function typically takes the form of a low-pass filter, and shows how well a CDR attenuates the jitter in its received signal. In systems with a cascade of CDRs, it is also important to ensure that the jitter transfer function has little or no peaking. Peaking will be amplified by the cascade, resulting in large amplitude jitter that may result in data recovery errors. Many high-speed interconnect protocols specify jitter transfer function targets that must be met by the CDR [36].
Jitter tolerance [37] is defined as the maximum amplitude of sinusoidal jitter at a given frequency that can be applied to a CDR before it begins to generate errors. These errors can be predicted by combining the jitter transfer function with the non-idealities in the phase detector. Most phase detectors have outputs that are periodic over a $2\pi$ phase-difference window, as shown in Fig. 2.14. Hence, a clock that lags the data by a phase of $1.5\pi$ is indistinguishable from a clock that leads by $0.5\pi$. When a sinusoidal phase jitter, $\Phi_{in}(s)$, is applied to the CDR input, the CDR attempts to track that jitter with the recovered clock phase, $\Phi_{osc}(s)$, where the accuracy of this tracking is defined by the jitter transfer function (2.10). The jitter tolerance is determined by the maximum phase input, $\Phi_{in}(s)$, that keeps the phase difference between input and recovered clocks less than $\pi$ so as to prevent aliasing in the phase detector output. That is,

$$\left| \Phi_{osc}(s) - \Phi_{in}(s) \right| < \pi \quad (2.11)$$

By combining (2.10) and (2.11), we can determine the maximum tolerable jitter amplitude:

$$|\Phi_{in}(s)| < \pi \left[ 1 + \frac{K_{pd}K_{vco}H_{lp}(s)}{s} \right], \text{ or}$$

$$J_{tol}(\omega) = \left| 1 + \frac{K_{pd}K_{vco}H_{lp}(j\omega)}{j\omega} \right| \quad (2.13)$$

To get meaningful results from the jitter transfer function and jitter tolerance, we need to know the values of $K_{pd}$, $K_{vco}$, and $H_{lp}(s)$. These values depend on the implementation and architecture.
of the CDR. The following section examines a bang-bang CDR implementation using 2x oversampling phase detection.

2x Oversampling CDR Implementation

The 2x-oversampling phase-detector [5], also known as Alexander or bang-bang phase detector, is a popular choice for CDR implementations in high-speed signaling systems. This phase detector samples the received signal at twice the symbol rate, once near the center of the data symbol, and once near the boundary between two adjacent data symbols, as shown in Fig. 2.15(a). These samples are then used to produce an early/late judgement about the local clock. If the value of the boundary sample matches that of the preceding data sample, then the clock is judged to be early; conversely, if the value of the boundary sample matches that of the following data sample, then the clock is judged to be late. The decision logic required for this comparison is reflected by the truth table shown in Fig. 2.15(b).

The early/late signals from the phase detector are then used to control a charge-pump, producing a current which is low-pass filtered to generate the VCO control voltage, as shown in Fig. 2.15(c). The VCO creates the recovered clock, which is used to clock the phase-detector on both edges of the clock; the data sample is clocked by the rising edge, and the boundary sample is clocked by the falling edge.

To determine the performance of this CDR, we must extract the quantities reflected in the signal flow graph shown in Fig. 2.13. However, the $k_{pd}$ of the oversampling phase detector is undefined. An ideal phase detector outputs a signal that is linearly proportional to the phase difference of the signals at its input. In contrast, the oversampling phase detector outputs a binary early or late signal that is the same regardless of how early or late the clock is.

In practice, this phase detector non-linearity is mitigated by the presence of random jitter and ISI. Without these, if the clock was slightly early, the phase detector would always report it as early. However, for a slightly early clock, the jitter and ISI will sometimes be sufficient to make the clock late. The earlier the clock is, the less frequently the jitter will be sufficient to make the clock late. Similarly, if the clock is less early, jitter will make it late more frequently. Averaged over many samples, this will give the average phase detector output a finite $K_{pd}$ that depends on the jitter.
statistics [38]. In some cases, the phase of the boundary sample clock is intentionally dithered to artificially introduce jitter and establish a predetermined $K_{pd}$ [35].

The transfer function of the RC loop filter in this example is defined as:

$$H_{lp}(s) = \frac{V_{cntl}(s)}{i_{cp}(s)} = \frac{1 + sRC}{sC}$$  \hspace{1cm} (2.14)

By substituting $K_{pd}$ and $H_{lp}(s)$ into both (2.10) and (2.13), we can calculate the jitter transfer function and jitter tolerance, respectively, of the 2x oversampling CDR:
where \( Q = \frac{1}{\omega_0 R C} \) and \( \omega_0^2 = \frac{K_{pd} K_{vco}}{C} \). If we now assume \( Q = 0.5 \), we can plot the jitter transfer and jitter tolerance as shown in Fig. 2.16.

Given (2.16), it is clear that the only way to increase the jitter tolerance of this CDR is to increase \( \omega_0 \). However, \( \omega_0 \) must usually be at least a factor of 100 lower than the clock frequency, otherwise the loop may become unstable. This limits the maximum jitter tolerance of the CDR. Further increases in jitter tolerance require architectural changes.

While the CDR presented in this thesis focuses on enhancing jitter tolerance, the choice of \( \omega_0 \) is a compromise between competing goals for jitter tolerance, jitter transfer, and jitter generation (the jitter generated by the CDR itself). A high \( \omega_0 \) is required to increase jitter tolerance and to decrease jitter generation, while a low \( \omega_0 \) is required to decrease jitter transfer.

---

**Fig. 2.16** 2x Oversampling CDR (a) jitter transfer, (b) jitter tolerance
2.3.2 Blind-oversampling Clock and Data Recovery

The traditional phase-tracking CDR uses feedback to keep the local clock aligned with the clock embedded in the received signal. In contrast, in a blind oversampling CDR [39][40][41] there is no correlation between the phase of the sampling clock and the phase of the embedded clock. As shown in Fig. 2.17, a multi-phase clock is used to take multiple samples of the received signal (3 or more) during each nominal symbol period. Phase detection is then performed by identifying transitions in this set of samples. The clock phase is then used to downsample the oversampled data by picking one sample per data symbol.

Without a feedback mechanism, like that of the phase-tracking CDR, the local and embedded clocks may drift out of phase by many UI. This phase difference is typically absorbed in one of two ways: padding bits or a FIFO. To compensate for slight frequency mismatches (a few ppm) between the local and embedded clocks, additional padding bits can be added to the transmitted data sequence. These bits do not contain valid data, and can be dropped to eliminate accumulated phase error. When the local clock is frequency (but not phase) locked, a FIFO can be used to absorb any transitory phase difference between local and embedded clocks, where the size of this FIFO determines the maximum phase difference.

The use of blind-oversampling clock and data recovery has several advantages. The implementation of the design is entirely digital, simplifying the design process, and making it more amenable to process scaling. In addition, this architecture can respond much faster to changes in clock phase, compared to the phase-tracking approach. The tracking speed of a phase-tracking CDR is limited by stability concerns in its feedback path. In contrast, the blind-oversampling CDR

![Fig. 2.17 Block diagram of a blind oversampling CDR](image-url)
can behave non-causally by performing phase detection on a set of samples, and then applying the recovered phase to these same samples, which have been delayed to compensate for the phase-detection delay.

There are also some disadvantages to the use of blind-oversampling. Taking multiple samples per symbol period increases the hardware requirements, which scale almost linearly with the oversampling ratio. It also requires multi-phase clocking, which further complicates the design of the clock-source and clock distribution network.

The concept of blind oversampling is illustrated through an example in Fig. 2.18. This figure shows a received signal oversampled by a factor of 5, producing a total of 40 samples over 8 nominal bit periods. Corresponding to each data sample, there is a sample phase, measured in units of 1/5 UI. The 40 samples are divided into 2 windows of 20 samples each. Windowing is used such that all the samples in the window are processed in parallel, reducing the operating speed of the circuit and averaging phase variations over multiple transitions. In each window of data samples, the data transitions are used to identify the phase of the received signal. In the first window of our example, a transition occurs between sample phases 1 and 2. If we define the transition phase as the sample phase following the transition, then the transition phase in this window is 2. Similarly, in the second window we identify the transition phase as 0. This process is depicted in the figure by an arrow pointing from the transition in the data samples to the boxed transition phase. Once recovered, the transition phase is then used to downsample the data by picking the samples $\frac{1}{2}$ UI away from the transitions, equivalent to an offset of $2\frac{1}{2}$ sample periods. In the first window, since the transition occurs somewhere between sample phase 1 and 2 (on average at a phase of $1\frac{1}{2}$), we downsample the data samples at a sample phase of $2\frac{1}{2} + 1\frac{1}{2} = 4$ (the data samples in the shaded circles). Similarly, in the second window we downsample the data samples at a sample phase of 2.
While a blind-oversampling CDR can track the change in phase from one window of samples to the next, there is a limit to the degree to which the phase can change between windows. This limit can be illustrated using the example in Fig. 2.18. In the first window, the transition occurs at a sample phase of 2, while in the second window it occurs at 0. Because phase is circular in nature, it is impossible to determine without further information if the sample phase changed by +3/5 UI or -2/5 UI between the two transitions. The received signal in our example shows a change of -2/5 UI, and has 5 bits between the transitions in the data—the same number of bits recovered by our example. However, if the phase change in the received signal was +3/5 UI, then there would be 6 bits between the transitions, but the data samples would be the same as in the above example, resulting (erroneously) in the same 5 recovered bits. If the wrong phase change is identified, the number of downsampled bits between transitions will be incorrect.

To avoid this ambiguity, blind-oversampling is limited to tracking a transition-to-transition phase change of less than $\frac{1}{2}$ UI. Because the phase is quantized by the process of oversampling, this results in a phase-change limit that varies with the oversampling ratio, $OSR$. For example, for $OSR = 3$, a phase change of $\frac{1}{3}$ UI is the largest discrete phase-change less than $\frac{1}{2}$ UI, while for $OSR = 4$, a phase change of $\frac{1}{4}$ UI is the largest discrete phase-change less than $\frac{1}{2}$ UI. This can be expressed mathematically as:

$$\Delta \phi_{in}\bigg|_{max} = \frac{\text{Floor}[(OSR - 1)/2]}{OSR}$$

which is illustrated in Fig. 2.19. Notice that the points lay on two curves that asymptotically approach $\frac{1}{2}$ UI: one curve for even ratios, and one for odd ratios. In all cases, an even-numbered oversampling ratio is an inferior choice, as the next-lowest odd oversampling ratio provides a higher phase-change limit. In addition, while $OSR = 3$ is the minimum required for blind-oversampling ($\Delta \phi_{in}\big|_{max} = 0$ for $OSR = 2$), there is little incentive in going above $OSR = 5$, as this provides diminishing returns for the required increase in hardware complexity. In this thesis, all further discussions of blind oversampling use $OSR = 5$, corresponding to a phase-change limit of $\frac{2}{5}$ UI.

While $\frac{2}{5}$ UI is the transition-to-transition phase-change limit, windowing of the samples as described above further constrains the phase-change limit. With a windowed phase-detection scheme, where the average phase is detected and then applied over the whole window, the phase-
change limit applies from one window with transitions to the next window with transitions. The maximum *rate* of allowable phase-change is thus limited by the maximum interval between transitions. This interval is a function of the run-length in the received sequence. For a $2^{31}-1$ PRBS sequence applied to a CDR using a 4 UI window, this restricts the rate of phase change to no more than $2/5$ UI over the 8 windows (32 UI) required for the runlength of 31. More precisely, the maximum rate of phase-change limit can be expressed as:

$$
\left. \frac{d\phi_{in}}{dt} \right|_{max} = \frac{2/5 \text{UI}}{32 \text{UI}} = \frac{1 \text{UI}}{80 \text{UI}}
$$

(2.18)

From this maximum rate of phase change we can determine the jitter tolerance of a blind-oversampling CDR as a function of frequency. Jitter tolerance is defined as the largest amplitude of sinusoidal jitter that can be applied to the CDR input without causing errors. Thus we define this input jitter as:

$$
\phi_{in} = 2\pi A_j \sin(2\pi f_j t) \text{ rads}
$$

(2.19)

where $A_j$ is the jitter amplitude in UI, and $f_j$ is the jitter frequency. The maximum rate of phase-change in this signal is:
If we now equate (2.20) to (2.18), then we can solve for the maximum jitter amplitude, $A_j$, that can be tracked by the above example of an oversampling CDR.

$$J_{tol} = 2A_j = \frac{1}{80\pi t_b f_j} \text{UI}$$

Hence the jitter tolerance is inversely proportional to the jitter frequency. This jitter tolerance is limited, however, at both high and low frequencies. At low frequencies, when a FIFO is used to retime the data to the reference clock, the jitter amplitude, measured in unit-intervals or bit-periods, cannot exceed the depth of the FIFO, otherwise it will overflow. At high jitter frequencies, the jitter period is much smaller than the runlength, hence a transition might not occur during one jitter period. In this case, the maximum phase-change between transitions is $2A_j$, which cannot exceed $\frac{2}{5} \text{UI}$, the transition-to-transition phase-change limit. Hence at high-frequencies the jitter tolerance is:

$$J_{tol} = 2A_j = \frac{2}{5} \text{UI}$$

The combined low-, mid-, and high-frequency jitter tolerance of the blind-oversampling CDR, using an oversampling ratio of 5, and a $2^{31}-1$ PRBS data sequence is shown in Fig. 2.20.
If we compare the jitter tolerance of the phase-tracking CDR, shown in Fig. 2.16, to that of the blind-oversampling CDR, shown in Fig. 2.20, we see that the phase-tracking CDR does well at low frequencies, while the blind-oversampling CDR does well at the moderately high-frequencies where (2.21) holds. In Chapter 5, we describe a CDR architecture that combines the properties of the phase-tracking and blind-oversampling CDR to achieve a jitter tolerance that is the product of the individual jitter tolerances.
Chapter 3

Event-Driven CDR Modeling

Many modern high-speed signaling systems do not transmit a clock signal, relying instead on the data transitions (zero crossings) to recover the clock. These zero crossings however are displaced from their original positions when subjected to the limited bandwidth and group delay variation of the transmission channel [10] and decision circuits [42][43]. In addition, the power supply noise and other circuit noise [44][45] also interfere with the zero crossings. As a result, the recovered clock at the receiver includes a combination of deterministic jitter and random jitter [46] that affects the overall performance of the system.

Previous works [47] derive the relationship between the zero crossing and the channel ISI in the form of equations using a linearized CDR model and frequency-domain analysis. This approach, however, does not apply to the non-linear behaviour of bang-bang [48] CDR architectures which require the use of time-domain simulations. These simulations are often of long duration due to the CDR loop filter having a bandwidth that is orders of magnitude lower than the operating frequency. To reduce the duration of these time-domain simulations, recent works have been presented that use discrete-time simulation techniques [49][50][51]. However, these simulations focus on architectural level behaviour and, aside from supply noise coupling to the VCO [52], they do not include the above mentioned jitter sources.

In this work, we strive to extend these discrete-time simulation techniques to incorporate the effect of jitter due to supply-noise, channel ISI, and the limited bandwidth (finite aperture window) of the front-end samplers. These techniques are implemented in Matlab using an event-driven model, a class of discrete-time simulation model. Using this model, we demonstrate a simulation accuracy close to that of continuous-time simulations and a simulation speed close to that of traditional discrete-time simulations.

The rest of this chapter is organized as follows. Section 3.1 provides an introduction to event-driven CDR modeling. This includes a simplified event-driven CDR model that serves as a baseline
model for the subsequent sections. Section 3.2 describes the modeling of jitter due to power supply noise. Section 3.3 describes the impulse response characterization of the CDR’s front-end samplers. Section 3.4 describes a discrete-time filter implementation that models the effects of limited bandwidth of the channel and front-end samplers. Section 3.5 compares the CDR jitter predicted by the proposed event-driven model with the CDR jitter predicted by Hspice [53]. This section also compares the two simulation approaches in terms of their speed. Finally, Section 3.6 summarizes the chapter.

3.1 Event-Driven CDR Simulation

An event-driven simulation refers to a class of discrete-time simulations where each simulation time-step corresponds to the occurrence of an event [54]. As a result, the simulation time-step is determined by the interval between events. This is in contrast with another class of discrete-time simulations, called unit-time simulations, where the simulation time-step is fixed and the occurrence of an event is determined by the simulator. The difference between these two classes of discrete-time simulations is illustrated in Fig. 3.1 through the relationship between simulation-time and simulation-state (e.g. voltage, current, etc.) in two examples. Both examples model a simple system with only one state variable, say \( v[t_n] \), where \( t_n \) denotes the n-th simulation time-step. In the unit-time simulation, \( t_n \) is advanced by a constant increment, \( \Delta t \), and the value of \( v[t_n] \) is calculated, as shown in Fig. 3.1(a). In the event-driven simulation, on the other hand, the value of \( v[t_n] \) is used to calculate the next event-time: \( t_{n+1} = t_n + f(v[t_n]) \), as shown in Fig. 3.1(b). That is, the time-step in this case is a function of the simulation-state.

![Fig. 3.1 Simulation time/state flowchart of: (a) unit-time model, (b) event-driven model](image-url)
While the unit-time simulation is often simple to implement, it requires a $\Delta t$ small enough to guarantee that events are not missed between timesteps. The event-driven simulation, on the other hand, uses a coarse but variable timestep that captures all events of interest, resulting in faster simulations.

Event-driven simulation can be used to model systems where the system behaviour can be described by discrete-time state-variables defined at times coinciding with the major system events (such as a clock edge). This is self evident with digital logic. With continuous-time analog circuits, this involves summarizing the circuit behaviour in the span between major events with state-variables at those events. For example, a charge-pump may be described by the total charge sunk between events.

This approach has sufficient flexibility to model a wide range of systems, so long as they experience major synchronizing events. Since most communication circuits (i.e., transceivers) transfer digital data from one chip to another, they must have a clocked interface somewhere in the receiver. Thus, most transceivers fall into the category of systems that can be modeled using event-driven techniques.

Fig. 3.2(a) shows a simplified transceiver where a transmitter sends internally generated data over an ideal channel to a CDR. The event-driven model of this system is shown in Fig. 3.2(b), consisting of three blocks: an event-scheduler, a transmitter event-routine, and a CDR event-routine. The event-scheduler maintains the times at which it must next trigger the TX, $t_{tx,n}$, and the CDR, $t_{rx,n}$, and generates these trigger signals accordingly. For example, if the next TX and CDR trigger times are 250ps and 300ps, respectively, the event-scheduler will set the current simulation time to 250ps, and generate a TX trigger. This trigger will cause the TX to generate new data, $v_{tx}[t_{tx,n}]$, and to calculate the next time it needs to be triggered (say $t_{tx,n+1} = 310\text{ ps}$). This completes one event. Next, the event-scheduler sets the current simulation time to 300ps and accordingly generates a trigger for the CDR. The CDR recovers a new data bit, $v_{rx}[t_{rx,n}]$, and calculates the next time it should be triggered (say $t_{rx,n+1} = 350\text{ ps}$). The latter information is sent back to the event-scheduler, and the process continues.

This technique can be used to capture the functionality of the bang-bang CDR [35] shown in Fig. 3.3(a), using the event-routine described below. We will use this event-routine throughout this chapter to simulate CDR clock jitter. The CDR consists of a bang-bang phase detector, a discrete-
time loop filter, and a phase interpolator that adjusts the phase of an independent reference clock. To create a corresponding event-routine, we first identify the triggering events. For this CDR, edges of the recovered clock are the natural choice for these events, as all changes of digital state are synchronized with them. In addition, the purpose of our simulation is to analyze jitter in the recovered clock, and this can be accomplished directly by analyzing the sequence of event-times if they coincide with the clock edges.

The resulting event-routine, representing only the basic CDR functionality, is represented by the unshaded blocks in Fig. 3.3(b). The logical functionality of the digital portions of the CDR, the phase detector and discrete-time loop filter, are inherently well suited to reproduce in the discrete-time event-routine, as shown in the top half of the figure. When the event-routine is triggered (at the time of the next recovered clock edge), the model of these blocks update the state of the recovered data, \( v_{rx}[t_{rx, n}] \), and that of the phase-code, which controls the phase-interpolator. The block modeling the phase-interpolator takes the updated state (the phase-code) and uses it to generate the next CDR event time, \( t_{rx, n+1} \). As shown in the bottom of the figure, this is
accomplished by adding the period of the reference clock, $t_{\text{clk}}$, to the current simulation-time, $t_{rx,n}$, and then introducing a phase-offset determined by changes in the phase-code.

The accuracy of this model in jitter prediction relies heavily on the accuracy of the event-times, defined as the times of the recovered clock edges. This event-time, as discussed earlier, is not uniform, and is influenced by the supply noise, by channel losses (through ISI), and by the limited bandwidth in the front-end samplers (also through ISI). These influences are not captured by the typical event-routine described above. In this work we use the shaded blocks in Fig. 3.3(b) to
capture these influences as a perturbation of the sampling process occurring in the CDR front-end. More specifically, we will introduce the data filter block to capture perturbations of the sample *value*, and the sampling-time offset block to capture perturbations of the sample *timing*.

In the CDR shown in Fig. 3.3(a), supply noise can potentially cause jitter when applied to any of the analog blocks. However, if we assume differential signaling in the data-path and good common-mode rejection in the samplers, then effects of supply noise will be limited to the perturbation of the recovered clock, and hence the sample *timing*. This is modeled by the *sampling-time offset* block in Fig. 3.3(b). The characterization and implementation of the sampling-time offset block is described in Section 3.2.

Limited bandwidth in the data path, encompassing the channel and receiver’s front-end samplers, also introduces jitter into the CDR. It does this by perturbing the sample *value* of the front-end phase-detector samplers. While models of the channel preceding the CDR are often available, models describing the bandwidth of the phase-detector samplers are not well known. Section 3.3 describes a technique to characterize the impulse response of the phase-detector samplers. For modeling purposes, the impulse responses of the CDR’s channel and phase-detector samplers can be combined into a single filter, the *data filter* block in Fig. 3.3(b), preceding the ideal samplers in our CDR event-routine. We provide an event-driven implementation of this filter in Section 3.4.

### 3.2 Modeling Power Supply Noise

This section describes the characterization of CDR jitter due to power supply noise. While oscillator jitter due to supply noise has been well modeled [55], the jitter generated in the remainder of the CDR is frequently neglected. Fig. 3.4 shows the clock path of the CDR described in Section 3.1, consisting of a phase-interpolator, clock-buffer, and a front-end sampler in the CDR’s phase-detector. When noise is applied to the power supply of any of these blocks, it will perturb the clock

---

**Fig. 3.4 CDR clock path and sampler**

---
signal. This will in turn affect the sample timing of the phase-detector’s sampler. The difference between the resulting sample times of a noisy and a noiseless power supply is the sampling-time offset.

We calculate the sampling-time offset as a function of the supply noise using a time-varying sampling time sensitivity function (STSF) for the circuits to be modeled. The STSF, represented by $\Gamma(\tau)$, expresses the change in the phase detector sampling time as a function of the time that an impulse of noise is applied to the power supply of the circuit being modeled. The time of the applied noise impulse is measured relative to the clock edge applied to the sampler. The sampling time offset, $\Delta\phi_n$, at the $n^{th}$ clock edge occurring at $t_n$ for an arbitrary noise source, $V_{\text{noise}}(t)$, is then expressed by:

$$
\Delta\phi_n = \int_{-\infty}^{\infty} V_{\text{noise}}(\tau - t_n)\Gamma(\tau)d\tau
$$

(3.1)

This technique is similar to that used by Hajimiri [55] to model the effects of noise in oscillators using an impulse sensitivity function (ISF), but differs in two ways: First, in oscillators the ISF is periodic, due to internal feedback of phase jitter, while with this technique the STSF is of finite duration. Second, in oscillators, the supply noise is applied to an oscillator stage and the resulting jitter measured by the shift in the zero-crossing at the output of that stage, while our technique applies the supply noise to the component being modeled, and measures the resulting sampling-time offset by the change in the phase-detector’s sample timing. In the following part of this section, we apply the above technique to determine the sampling-time offset for the clock buffer shown in Fig. 3.4.

First, we apply an impulse of noise to the buffer’s power supply (approximately 10% of the supply voltage in amplitude, as described later) and measure the resulting sampling-time offset. This process is performed repeatedly, each time applying the noise impulse at a different time. By definition, plotting the sampling-time offset against the impulse time produces the STSF. Note that while the noise impulse is applied to the buffer (the circuit being characterized), the quantity being measured is the sampling-time offset of the sampler.

We use circuit simulation to determine the sampling-time offset needed for this characterization process. Fig. 3.5 shows the schematic of the sampler used in our CDR, as well as the simulated waveforms of the important nodes in this circuit. The CDR’s received signal ($v_{\text{in}_p}$, $v_{\text{in}_n}$) is
applied to the clocked differential pair and feed the back-to-back inverters on the top to create rail-to-rail values. When the sampler is in reset mode (clock low), the outputs are pre-charged high. When the sampler is enabled (clock high), the outputs start to drop. If a constant non-zero differential voltage is applied to the data inputs, the outputs will split and the positive feedback of the back-to-back inverters will eventually cause the sampler to latch with rail-to-rail values. If the differential input voltage is zero, however, the outputs will not split, and instead converge to an intermediate value (i.e. a zero differential output). The circuit can also become metastable when it is clocked during a data transition between two non-zero, and opposite differential voltages. We define the sampling-time by the timing of a data transition, with respect to the clock edge, which results in a metastable output.

In simulation, a noise impulse is applied to the power supply, and the clock timing is adjusted until the circuit is judged to be metastable. The adjustment of the clock timing is accomplished using the parameter optimization feature in Hspice; the clock timing is optimized to produce a minimum differential output voltage at a point in time sufficiently past the clock edge to demonstrate metastability. The change in clock timing caused by a noise impulse at time \( \tau \) past the clock edge provides a point solution to the STSF, \( \Gamma(\tau) \). Full STSF characterization is then achieved by sweeping the impulse time using multiple simulations.
Fig. 3.6 shows the STSFs of the three individual components in Fig. 3.4: The phase interpolator, the clock buffer, and the sampler. The STSF of the phase interpolator is much longer in duration than those of the clock buffer and the sampler. This indicates that the buffer and sampler are only sensitive to supply noise during their transient states (at the clock edges), while the phase interpolator has internal analog nodes that remain sensitive to noise throughout the clock cycle. The resulting greater area under the phase interpolator’s STSF implies that it is far more sensitive to supply noise.

Having determined the STSF of these blocks, we now use (3.1) to determine the sampling-time offset due to a sinusoidal noise defined by \( \sin(2\pi f_{\text{noise}}t + \theta) \):\[
\Delta \phi_n = \int_{-\infty}^{\infty} \sin(2\pi f_{\text{noise}}(\tau - t_n) + \theta) \Gamma(\tau) d\tau \tag{1}
\]
which for \( t_n = 0 \) simplifies to:
\[
\begin{align*}
\Delta \phi_n &= \int_{-\infty}^{\infty} \left( e^{j\theta}/2j \delta(f - f_{\text{noise}}) - e^{-j\theta}/2j \delta(f + f_{\text{noise}}) \right)^* \Gamma(f) df \\
&= e^{-j\theta}/2j \Gamma(f_{\text{noise}}) - e^{j\theta}/2j \Gamma^*(f_{\text{noise}}) \\
&= \sin(\theta) \text{real}[\Gamma(f_{\text{noise}})] - \cos(\theta) \text{imag}[\Gamma(f_{\text{noise}})] 
\end{align*}
\]
where $\Gamma(f)$ is the Fourier transform of $\Gamma(\tau)$. To determine $\Delta\phi_n$ when $t_n \neq 0$, we can still use (2), but capture the noise sinusoid’s phase with an equivalent phase, $\theta$, at $t_n = 0$. During an event-driven CDR simulation $\Delta\phi_n$ is then determined using (2) at each sample point. Because the STSF only needs to be extracted once for each design, the overhead of modeling the noise effects is limited to the evaluation of (2) once per sample.

This approach can be extended to include the effect of periodic supply noise other than sinusoidal. In general, the convolution of the STSF and the noise waveform produces a solution for (3.1) over the entire noise period. This result is stored in a look-up table in the model, and the sampling-time offset block need only keep track of the current position within the noise waveform during each sample, and use this position as an index into the lookup-table.

In our event-driven model $\Delta\phi_n$ is introduced into the model using the sampling-time offset block shown in Fig. 3.3(b). When a data transition occurs near the sample time, this allows the CDR model to determine if the supply noise changes the sampling time sufficiently to alter the sampler’s binary output. This will in turn change the early/late decision of the phase detector.

### 3.2.1 Model Verification

To verify the accuracy of the model described above, we compare the sampling-time offset predicted by our model to the results of Hspice simulations using a sinusoidal noise source of $100\text{mV} \sin(2\pi f_{\text{noise}}t + \theta)$, with various $f_{\text{noise}}$ and $\theta$. The circuit we use for this verification is the phase-interpolator within the clock path shown in Fig. 3.4, designed in a 0.11\(\mu\)m CMOS process.

The results of this comparison are shown in Fig. 3.7, confirming the accuracy of our modeling approach. The results show a very good agreement between the Hspice and Matlab simulations. The worse case mismatch occurred for a 1GHz sinusoidal supply noise where our model predicts a peak sampling-time offset of 25.6ps, compared to 22.8ps predicted by Hspice—a 12% error. Note however, that the MatLab simulations complete in less than a second, while the Hspice simulations required many hours to complete.

To verify the linear approximation assumed by this model, we ran simulations with supply noise amplitudes ranging from 25mV to 200mV on a 1.2V nominal supply. Within this range, our simulations produced consistent results with no sign of non-linearity.
3.3 Sampler Impulse Response

An ideal sampler samples its input voltage instantaneously. In reality, the sampling is performed over a window of finite duration, known as the aperture window. We can describe this process by a sampling function, $s(t)$, applied to the input voltage $v(t)$ at time $t_n$:

$$\hat{v}(t_n) = \int v(\tau)s(\tau - t_n)d\tau$$  \hspace{1cm} (3.2)

In the ideal case, where $s(t) = \delta(t)$, this produces the instantaneous sample $v(t_n)$.

This sampling function becomes easier to incorporate into a system level model if we apply a variable substitution, $g(t) = s(-t)$ to (3.2):

$$\hat{v}(t_n) = \int v(\tau)g(t_n - \tau)d\tau = v(t) \otimes g(t)|_{t = t_n}$$  \hspace{1cm} (3.3)

where $\otimes$ denotes the convolution operation. This shows that the sampling process is equivalent to applying $v(t)$ to a filter with impulse response $g(t)$, followed by an ideal sampler triggered at time $t_n$. This is depicted in Fig. 3.8, along with a quantization block that slices the sampled value into one of two logical levels (in the case of binary signaling). Our goal here is to find a $g(t)$ such that

![Fig. 3.7 MatLab vs. HSpice simulated sample-time offset for various $f_{noise}$](image-url)
the system shown in Fig. 3.8 will have the same input-output characteristics as that of the circuit shown in Fig. 3.5.

Given an arbitrary input \( v(t) \), \( g(t) \) can be determined if we can extract \( x(t) \), the output of the \( g(t) \) filter block, from the circuit of Fig. 3.5. However, \( x(t) \) does not exist in the physical circuit implementation, neither is it observable from the output of the quantizer, except for when \( x(t) \) becomes zero. At \( x(t) = 0 \), the output of the quantizer is undefined and the output of the circuit becomes metastable. By finding a set of input signals that produce this output condition, we can reconstruct the sampler’s impulse response.

A set of input signals that satisfy this condition are of the form \( A_1 \delta(t-t_1) + A_2 \), where \( A_1 \) and \( A_2 \) can be determined for each \( t \) so as to produce the metastable output. At the output of the filter \( g(t) \), this input signal results in:

\[
x(t) = [A_1 \delta(t-t_1) + A_2] \ast g(t) = A_1 g(t-t_1) + A_2 G(0),
\]

where \( G(0) \) is the DC component of \( g(t) \).

Since the input to the quantizer is the solution to (3.4) at \( t = 0 \), the metastable output condition can only occur when:

\[
x(t) = A_1 g(-t_1) + A_2 G(0) = 0
\]

(3.5)

This can be rearranged as:

\[
g(-t_1) = -A_2 \frac{G(0)}{A_1}
\]

(3.6)

This implies that if we keep \( A_1 \) constant, then \( g(-t_1) \) becomes directly proportional to the DC component, \( A_2 \), of the input signal that causes the sampler to become metastable. Simulating to find this value provides the required indirect measurement of the filter \( g(-t) \).
To determine $g(t)$ for a given value of $t_1$, we first perform a simulation to find the amplitude of the DC level, $A_2$, required to cause metastability. As in the previous section, our simulation uses Hspice optimization to find this metastable point. For a given $t_1$, we optimize $A_2$ with the goal of producing a differential output voltage of zero. By sweeping the timing of the delta function, $t_1$, across the sampler’s aperture window, and at each point determining $A_2$, we determine the scaled and time reversed impulse response of the sampler. The exact value of the proportionality constant, $G(0)/A_1$, is irrelevant, as the quantizer output is a function of only the sample’s polarity. Fig. 3.9(a) shows the simulated voltages for the input, output and clock nodes of the sampler in Fig. 3.5, showing both the case of fully switched outputs, as well as metastable outputs.

Fig. 3.9(b) shows the normalized value of $A_2$ (and hence $g(-t)$) found through simulation. The results show that the impulse response begins when the clock input reaches NMOS threshold voltage, turning on the clocked NMOS device, and ends when the output nodes drop by a PMOS threshold voltage, turning on the PMOS devices of the back-to-back inverters.
Additional simulation show that \( g(-t) \) is a function of the clock waveform, particularly its rise time. For the sampler shown in Fig. 3.5, clocks with rise times less than 50ps produced results similar to that shown in Fig. 3.9(b), where the trailing edge of \( g(-t) \) is a decaying exponential with a time constant equal to that of the sampler’s output nodes. As the rise time increases above 50ps, the width of \( g(-t) \) also increases, implying lower latch bandwidth.

To implement the above modeling technique in an event-driven CDR simulation, the filter modeling the finite bandwidth of the sampler is combined with the filter modeling the lossy data channel as the *data filter* shown in Fig. 3.3(b). The implementation of this combined filter is discussed in Section 3.4.

### 3.3.1 Model Verification

To verify our model, we test it with waveforms representative of what would be experienced when the sampler is integrated into a CDR. The only sampler outputs that are of interest in this case are those that could produce either a high or a low output given a small perturbation, that is, when the sampler is clocked near transitions in its input data signal. Our verification simulations examine the output of our sampler when clocked near data transitions, using varying transition times, \( t_{\text{trans}} \), to emulate the data dependant slew rate resulting from channel ISI.

The simulation testbench is illustrated in Fig. 3.10(a). A data signal that is transitioning between 0 and 1 is applied to the simulated sampler, along with a rising clock edge. This setup is then

![Fig. 3.10 Model verification simulation results](image-url)
simulated, using Hspice, to determine the sampling-time offset, $t_{STO}$, required to generate a metastable output. We then determine the sampling-time offset again, this time by convolving the impulse response we previously extracted with the data signal, and locating the zero-crossing. This comparison is performed for varying data transition times, and the results are illustrated in Fig. 3.10(b). The upper-left and lower-right portions of the figure show the timing regions where both methods indicate that the sampler will output 1 and 0, respectively. The shaded region in the center shows where the methods disagree, with its upper and lower boundary curves denoting the metastability points for the Hspice and impulse response models, respectively. The gap between the two curves corresponds to less than 0.3ps of timing error. This is expected to result in a jitter prediction error of similar magnitude in the CDR simulation.

3.4 Event-Driven Implementation of the Data Filter

This section describes the event-driven implementation of the data filter in Fig. 3.3(b). As shown in Fig. 3.11(a), this filter models the ISI from the lossy data channel and the limited bandwidth of the phase-detector samplers. Formally, its effect on the received data sequence can be represented by:

$$y(t) = x(t) \otimes h(t), \quad (3.7)$$

where the transmitted signal, $x(t)$, is convolved with the combined impulse response of the data channel and phase detector, $h(t)$, to generate $y(t)$, the input to the idealized CDR sampler.

Unlike discrete time filter implementations, the filter implementation required in a CDR operates in the presence of jitter and changing cycle times. Because of the resulting irregular timesteps, a filter implementation using z-domain techniques is not possible. Instead, we calculate the filter output using the step-response, $s(t)$, of the system and exploit the fact that the transmitted signal is not an arbitrary waveform, but rather a series of pulses of non-uniform duration. In other words, we can express $x(t)$ as the following:

$$x(t) = \sum_{n=0}^{\infty} x[n](u(t-t_n) - u(t-t_{n+1})), \quad (3.8)$$

where $x[n]$ is the $n^{th}$ transmitted data symbol, $u(t)$ is the step function, and $t_n$ and $t_{n+1}$ are the start and end times of the symbol, respectively.
Substituting this into (3.7) allows us to express the filter output as:

\[ y(t) = \sum_{n=0}^{\infty} x[n] (s(t - t_n) - s(t - t_{n+1})) \]  

(3.9)

Since the step response is usually limited to only a few bit periods in duration, we only need to evaluate the first few elements in this summation. In addition, the step response can be predetermined and stored in a lookup table for use during simulation. Since we use binary signaling and normalize the transmitted data symbols \( x[n] \) to \{1, 0\}, the filter implementation is reduced to a small number of table lookups and additions (or subtractions). A block-level implementation of this concept is shown in Fig. 3.11(b), where the summation in (3.9) is simplified to only include the effect of the previous \( N \) data symbols. The computation can be further simplified by realizing that the table lookups required for the trailing edge calculation are a subset of those required for the
leading edge contribution and can be shared. Note that in the trailing edge calculation we discard the current data bit because, by definition, its trailing edge has not yet occurred. The summation of the leading and trailing edge contributions produces the signal level, $y(t)$, at the input to the receiver’s ideal sampler.

### 3.4.1 ISI Model Verification

To verify our model, we use it to predict the output of a data channel with a random data sequence applied to it. These results are then compared to the output of an Hspice simulation of the same system. The result of this simulation is shown in Fig. 3.12 where the continuous waveform is the Hspice output, and the crosses are the sample values determined using the step response model. The RMS error for the data samples is 1.1%. Most of the error is due to the truncated summation used in Fig. 3.11 (N=3 in this example), and could be reduced at the expense of increased computation.

### 3.5 Putting it all Together - System Level Simulation Results

In previous sections, we examined CDR modelling at a component level. In this section, we incorporate these component models in a complete event-driven CDR model. We then compare the simulated jitter in our event-driven CDR model to the jitter in Hspice simulations of the same CDR. This model comparison is performed in two parts: First we evaluate the use of the sampling-time offset to capture jitter due to supply noise. Second, we evaluate the use of the event-driven data filter to capture jitter due to limited bandwidth in the channel and in the phase-detector samplers.
The CDR being modelled uses the architecture described in Section 3.1, operating at 3.2Gbps. It uses 2x oversampling phase detection, built around the sampler shown in Fig. 3.5. The loop filter is a first-order digital low-pass filter, with a bandwidth of 4MHz. The phase interpolator is built using a current-starved inverter architecture [35], and interpolates an ideal 3.2GHz clock to generate the local clock. The local clock is then supplied to the phase detector through an inverter-based clock buffer. The channel model used in this system has a first-order response, with a 3dB bandwidth of 1.6GHz.

The event-driven model of the above CDR is implemented in Matlab’s Simulink [56] using the structure shown in Fig. 3.3(b). The data filter block is structured in the manner described in Section 3.4. The impulse response of this filter is the convolution of the channel and sampler impulse responses, where the sampler impulse response is determined using the process described in Section 3.3. The sampling-time offset block is implemented using the technique described at the end of Section 3.2 for periodic supply noise waveforms.

3.5.1 Clock jitter due to supply noise

In this section, we compare our event-driven model in Matlab against Hspice simulations in predicting jitter in the recovered clock due to supply noise.

For these simulations we choose an FM modulated supply noise:

\[
\phi(t) = 0.1 \sin\{2\pi[3.2e9 + 0.5e9 \sin(2\pi 5e6 t)]t\}
\]

That is, an FM noise having a peak amplitude of 0.1V with the noise frequency centered around 3.2GHz (the CDR’s bit-rate) and FM modulation with a peak frequency-deviation of 500MHz, and with a modulation frequency of 5MHz. This FM noise source allows the demonstration of two properties: First, while the sampling-time offset can be determined as a continuous time function by convolving the FM noise source with the circuit’s STSF, it is only sampled during the clock edge events. This results in the aliasing of the noise down to a frequency band centered around DC. Second, in the frequency domain, the amplitude of the STSF shown in Fig. 3.6 drops to less than half of its DC value from 2GHz-4GHz, overlapping the range of instantaneous frequencies of the FM noise source. The implication of this is that while noise at 2.7GHz and at 3.7GHz will both be aliased to 500MHz, the sampling-time offset due to the 3.7GHz noise will be attenuated compared to that of the 2.7GHz noise.
The effects of these properties can be seen in Fig. 3.13, where the top curve shows the instantaneous frequency of the FM noise source, the middle and bottom curves show the instantaneous jitter (with reference to an ideal clock) in the recovered clock determined using Hspice and event-driven simulations, respectively. The effect of the aliasing can be seen in both simulations where the jitter frequency in the CDR output goes through zero when the instantaneous noise frequency equals 3.2GHz. It can also been seen that the jitter amplitude is lower in both simulations at the higher instantaneous noise frequencies.

**3.5.2 Clock jitter due to limited bandwidth**

This section compares the simulated jitter due to limited bandwidth as modeled by the data filter block in our event-driven model to the jitter predicted by an equivalent Hspice simulation. We perform this comparison in two steps: First, we look at the jitter due to limited bandwidth in the channel in order to verify the performance of the data filter block. Second, we examine jitter due to
limited bandwidth in the phase detector’s samplers to verify the characterization of their impulse response.

To demonstrate jitter due to limited bandwidth, we choose a data sequence that begins with a repeating sequence of ‘11110000’ switching to ‘10101010’ after 100ns. During transmission of the first sequence, ISI in the received signal will cause the signal transitions to be delayed compared to the signal transitions occurring during transmission of the second sequence. The CDR should track this change of transition location, producing jitter in the recovered clock.

The simulated jitter due to limited bandwidth in the channel is shown in Fig. 3.14. The top waveform shows the transmitted data pattern, while the middle and bottom waveforms show the jitter in the recovered clock for the Hspice and event-driven simulations, respectively. Both simulations show the roughly the same 60ps change in recovered clock phase.

To demonstrate the jitter due to limited bandwidth in the phase detector samplers, characterized in Section 3.3, we artificially introduced large parasitic capacitors into the samplers to exaggerate
their impulse response. Simulation results using the data filter to model the limited bandwidth of the modified samplers are shown in Fig. 3.15. The top waveform shows the same transmitted data pattern as before, while the middle and bottom waveforms show the jitter of the recovered clock for the Hspice and event-driven simulations, respectively. Once again, there is a good correspondence between the jitter predictions, roughly 8ps, in both simulations.

The need to exaggerate the impulse response of this sampler to produce measurable results makes it appear that jitter in a realistic implementation is negligible. While this may be true in a simple full-rate implementation, a fractional-rate system employing interleaving to increase the bit-rate can run fast enough to require such modeling.
3.5.3 Simulation Time

While the previous section demonstrated the accuracy of our event-driven CDR simulation techniques, the real advantage of these techniques becomes apparent when comparing the simulation time required by the two techniques. As shown in Table 3.1, simulating for about 600 cycles in Hspice takes 2 hours. A conventional fixed-step discrete time simulation with a step size 1/32th of the nominal bit rate required 30 seconds to perform the same simulation -- a 240x speed-up. In comparison, the event-driven model took only 4 seconds, 1800x faster than hspice and 7.5x faster than the fixed-step simulation.

3.6 Summary

This chapter presented event-driven CDR simulation techniques that allow the quick and accurate prediction of jitter in the recovered clock. These techniques introduce the modeling of supply noise, the characterization of bandwidth limitations in the phase detector samplers, and a discrete-time filter implementation for non-uniform time intervals.

The supply noise was modeled as a sampling-time offset in the phase detector samplers. This sampling-time offset was determined by taking the dot product of the supply noise waveform and the sampling time sensitivity function (STSF) of the circuit. We also described the process for characterizing the STSF of the circuit.

The limited bandwidth of the phase detector samplers was modeled as a filter preceding an ideal sampler. The process of characterizing the impulse response of this filter was also presented.

A discrete-time filter for non-uniform time-steps was implemented by describing the transmitted data sequence as a sequence of rising and falling steps, instead of pulses, and applying the system’s step-response to each step. This process simplifies the computation of received sample values to only a few operations.
The event-driven modeling techniques presented in this chapter offer a simulation accuracy close to that of Hspice, but with a simulation speed-up of 1800 times. This was confirmed through a CDR simulation in both.
Chapter 4

An Adaptation Technique for 4-PAM Decision-Feedback Equalization

Current high-speed serial and backplane signaling systems operate at Gigabit rates, causing significant high-frequency channel attenuation within the bandwidth of the signal. Two approaches to counter this problem are 4-PAM signaling and decision-feedback equalization. 4-PAM signaling halves the symbol-rate to reduce the impact of high-frequency attenuation at the cost of increased design complexity. Decision-feedback equalization uses past decisions of the receiver to predict and to eliminate the ISI resulting from high-frequency attenuation. When the channel properties are not known in advance, this equalization must be made adaptive. However, the circuits required to provide adaptive functionality typically reduce the maximum bit-rate at which the system can operate. This chapter proposes an adaptation technique for a 4-PAM decision-feedback equalization (DFE) which mitigates this bit-rate penalty. In addition, the proposed design adaptively determines the 4-PAM reference levels, provides phase-alignment to a source-synchronous clock, and provides channel pulse-response monitoring capability.

The DFE proposed in this chapter is implemented as part of a complete transceiver, shown in Fig. 4.1. The transceiver uses 4-PAM signaling and 4-phase clocking, resulting in the transmission of 8 bits over each channel during each system clock period. The transmitter contains a pseudo-random bit sequence (PRBS) data generator producing an 8-bit wide output. These 8 bits are then applied to an 8-to-2 serializer and then driven off-chip with a 4-level PAM driver. The receiver portion of the chip comprises the adaptive DFE, a phase recovery block, and a data retiming block. A built-in error detector compares the transmitter’s PRBS data sequence with the received data sequence and counts the resulting errors.

The remainder of this chapter is organized as follows: Section 4.1 reviews the basic DFE architecture. Section 4.2 describes the speed penalty in traditional adaptive DFE implementations due to the additional circuits introduced into the DFE critical path. Section 4.3 introduces the
proposed adaptive DFE architecture which eliminates many of these additional circuits. Section 4.4 describes the circuit-level implementation of the DFE. The measured and simulated results of the fabricated design are presented in section 4.5. Section 4.6 summarizes the contributions of this work.

4.1 Basic DFE Architecture

A DFE is an equalizer that uses the past decisions of the receiver and an estimate of the channel impulse response to create and subtract a replica of the ISI from the current symbol [7][20][22]. The block diagram of a simple 4-PAM DFE is shown in Fig. 4.2, where \( x[n] \) is the received signal, \( y[n] \) is the equalized signal, and \( \hat{y}[n] \) is the receiver’s decision on the transmitted symbol, based on the slicing of \( y[n] \). 4-PAM signaling uses 4 symbols to encode data, normalized here to \{3, 1, -1, -3\}. To decode these 4 symbols the receiver requires three slicers, shown symbolically together as one block, with thresholds midway between the adjacent symbol levels, \{2, 0, -2\}. The output of these slicers form a 3-bit thermometer coded representation of the 4 symbol levels. The decision feedback filter, \( W[z] \), is an N-tap discrete time FIR filter that replicates the channel’s discrete-time impulse response, \( H[z] \):

* The design of the transceiver was a 2 person effort, this chapter discusses only this author’s contributions—the design and implementation of the DFE receiver. A detailed description of the transmitter and other components can be found in the M.A.Sc thesis of Joyce Wong [37].
Thus the equalized signal, \( y(n) \), can be expressed as:

\[
y[n] = x[n] - \sum_{i=1}^{N} w_i \hat{y}[n-i]
\]  

(4.2)

Since \( \hat{y}[n] \) is the reconstruction of the transmitted data, the last term in (4.2) represents the DFE’s estimate of the ISI. This estimate is then subtracted from the received signal to produce an ISI-free signal, \( y[n] \).

Since the \( W(z) \) coefficients are not always known in advance or, if known, might change over time, adaptive equalization is often used to determine and to track them. However, as described in the following section, traditional approaches to adaptive equalization have a detrimental effect on the maximum bit-rate of a DFE.
4.2 Traditional adaptive DFE architecture

As described in Section 2.2.1, the commonly used LMS adaptation algorithm requires an error signal to guide the adaptation. This error signal, as shown in Fig. 4.3, is defined as the difference between the equalized signal, \( y[n] \), and a reference signal that is approximated by the DFE output, \( \hat{y}[n] \). In high-speed DFE applications where the process technology is being pushed to its limit, the slicer delay consumes a sufficient fraction of the symbol period, precluding a direct implementation of this difference operation. This is because the slicer input, \( y[n] \), has changed by the time the DFE output, \( \hat{y}[n] \), is ready to be subtracted from it.

The conventional solution to this problem is shown conceptually in Fig. 4.4(a). The solution is similar to DFE loop unrolling [58] in that the error signal for all 4 possible received symbols are determined simultaneously [27]. Then, once the received symbol is determined, a multiplexer is used to select the appropriate error signal. The actual implementation of this scheme is shown in Fig. 4.4(b), where the 4 subtractors are replaced with 4 additional slicers (again shown symbolically together as one block). These 4 slicers have thresholds equal to levels of the 4 possible received symbols, \{3, 1, -1, -3\}. For each recovered symbol, \( \hat{y}[n] \), the output of error slicers with \( \hat{y}[n] \) as its threshold indicates the sign of the error. For example, if \( y[n] = 2.9 \) and \( \hat{y}[n] = 3 \),
then the error slicer with a threshold of 3 would produce a low output, indicating a negative error. The resulting system produces $\text{sgn}(err[n])$ instead of $err[n]$, a well known derivative of the LMS adaptation technique.

While solving the problem of error generation, this technique requires 4 additional slicers (for 4-PAM signaling) to generate the error signal. These additional slicers add capacitance to the critical path node, $y[n]$. Present designs attempt to eliminate these additional slicers, but have only succeeded in reducing their number [59]. The resulting increased loading on the DFE’s critical path reduces its speed, and hence its maximum data-rate. In addition, the error slicers add design complexity and area overhead.

While the technique presented in [59] eliminates 3 of the 4 error slicers shown in Fig. 4.4(b), the technique presented in the following section eliminates the final error slicer to achieve an additional 20% reduction in critical path loading. This comes, however, at the cost of requiring a special calibration sequence.

Fig. 4.4 Lookahead error signal generation (a) conceptional (b) implementation
4.3 Proposed adaptive DFE using intermittent calibration sequence

To reduce the penalties associated with the use of adaptive equalization, we propose a technique that eliminates the error slicers of Fig. 4.4(b) and, instead, uses the existing symbol slicers to produce the required error signal, as illustrated in Fig. 4.5. To permit this change, we introduce an intermittent calibration sequence. This calibration sequence is designed such that the symbol slicers generate \( \text{err}[n] \) instead of \( \hat{y}[n] \). This output is then used to guide the adaptation control of \( W(z) \), while the filter’s data input is driven by the known calibration sequence. As described later, this same technique is used to guide the adaptation of the 4-PAM reference levels and the phase locking of the local clock to the received data.

The intermittent calibration sequence is designed around the use of a special 0-symbol which is used only during calibration. While 4-PAM signaling uses the symbols \{3, 1, -1, -3\} to encode data, our calibration sequence consists of the repeating sequence ‘3 0 0 0’, with adaptation occurring only during reception of the 0-symbol. The use of a 0-symbol has two advantages. First, it allows the existing ‘0’ slicer (normally used to distinguish 1 from -1) to generate its error signal. Second, its slicing level does not need to be calibrated against a reference voltage—‘0’ is inherent to a differential comparator design. If we were to use a calibration sequence such as ‘3 -3 -3 -3’, then we would require a ‘-3’ slicer to generate the error signal. While this could be achieved by scaling the threshold of the ‘-2’ slicer, this slicing level is initially unknown and also requires adaptation.
The 0-symbol, equivalent to 0 V differential, is implemented by splitting the transmit driver into two equal halves; during data transmission the two halves support each other, while during ‘0’ transmission they oppose each other. Because the driver is only split in half, its total area is not changed. As a result, the only overhead to produce the 0-symbol is the inversion in the data path of one of the two driver halves.

The use of an intermittent calibration sequence has the undesirable side-effect of interrupting the continuous flow of data over a channel. To provide uninterrupted signaling, our system uses N+1 channels to transmit N signals. For a 32-bit bus this requires an area and channel overhead of only 3%. For test purposes, we have chosen N=4 to demonstrate a small, yet non-trivial implementation. Such a system is shown in Fig. 4.6, where the receiver in each channel has an instantiation of the DFE in Fig. 4.5. In this system, multiplexers take one channel out of service at a time, seamlessly transferring its signaling duties to the previously out-of-service channel. The newly out-of-service channel then uses the calibration sequence to adjust its DFE filter coefficients and 4-PAM reference levels, and to adjust the phase of the source-synchronous clock to match the phase of the clock embedded in the data. However, in systems where the channel characteristics are
unlikely to change, adaptation can be performed only once during startup, eliminating the need for this channel-multiplexing scheme.

The architecture of the DFE used in each channel, excluding the adaptation functionality, is shown in Fig. 4.7. The DFE uses a 4-phase quarter-rate clock, such that 4 consecutive 4-PAM symbols are decoded by 4 interleaved DFE slices. This 3-tap DFE subtracts ISI due to the past three symbols, and is characterized by:

\[
y[n] = x[n] - w_1 \hat{y}[n-1] - w_2 \hat{y}[n-2] - w_3 \hat{y}[n-3]
\]  

(4.3)

where \(x[n]\) is the received sequence, and \(\hat{y}[n]\) is a DFE decision. \(w_1, w_2,\) and \(w_3\) are the DFE filter coefficients, which equal the channel’s discrete-time impulse-response of \(h_1, h_2,\) and \(h_3,\) respectively, once the DFE has adapted.

Because our system is 4-way interleaved, the delayed \(\hat{y}[n]\) are obtained directly from the outputs of the other interleaved DFE slices, without the need for delay elements. For example, as shown in Fig. 4.7 for the 0° slice, the outputs of the 90°, 180°, and 270° slices provide \(\hat{y}[n-1],\) \(\hat{y}[n-2],\) and \(\hat{y}[n-3]\), respectively.
The following sub-sections describe how the DFE uses the intermittent calibration sequence to provide three different adaptive functions. First, we describe how the filter coefficients, \( w_x \), are adapted to \( h_x \). Second, we describe how the 4-PAM reference levels ‘2’ and ‘-2’ are adapted to match the received signal amplitude. Finally, we describe how our source-synchronous system adjusts the phase of our 4-phase clock to match the phase of the received data.

### 4.3.1 DFE filter coefficient adaptation

This sub-section describes how the intermittent calibration sequence is used to adapt the DFE filter coefficients. We first show the simplifications resulting from the application of our calibration sequence to an LMS adaptive DFE. We then show that these simplifications combined with the DFE architecture shown in Fig. 4.7 allow an adaptation implementation using minimal additional hardware.

Fig. 4.8(a) shows the implementation of a 3-tap adaptive DFE filter \( W[z] \). The top half of the figure shows the implementation of the feedback portion of Eq. (4.3). During equalization past decisions, \( \hat{y}[n-i] \), are multiplied by the filter coefficients, \( w_i \), and then added together. When in adaptation mode, the filter uses the \( w_i \) update blocks to adapt the filter coefficients according to the LMS algorithm described in Section 2.2.1, modified to use the error sign:

\[
 w_i[n + 1] = w_i[n] + 2\mu \text{sgn}(err[n])\hat{y}[n-i] \tag{4.4}
\]

While this update algorithm is identical to the conventional adaptation algorithm, our implementation differs in the way we generate the \( 2\mu \text{sgn}(err[n])\hat{y}[n-i] \) term. These changes are illustrated in Fig. 4.8(a). Whereas \( \text{sgn}(err[n]) \) was previously generated by dedicated slicers as shown in Fig. 4.4(b), we now derive the \( \text{sgn}(err[n]) \) term from the 0-level DFE slicer output, denoted by \( \hat{y}_0[n] \), as described earlier. Furthermore, we now replace the delayed DFE decisions, \( \hat{y}[n-i] \), with the known calibration sequence. In addition, because the calibration sequence length matches the interleaving depth of four, each of the interleaved DFE slices will see a constant 3-symbol subset of the 4-symbol calibration sequence. These subsets are shown at the bottom of Fig. 4.8(a), and assume that the phase recovery, described later, has aligned the ‘3’ symbol to occur during the 0° clock phase.

This constant sub-set of the calibration sequence results in a fixed 0 or 3 at the inputs to the \( w_i \) update blocks, allowing significant simplifications. These simplifications are shown in Fig. 4.8(b).
Fig. 4.8 LMS adaptation using calibration sequence (a) generic implementation (b) simplified implementation of 90° slice
for the $90^\circ$ slice, where the update blocks with a 0-input are eliminated and the block with a 3-input absorbs this constant into the $2\mu$ gain factor. The result is an adaptive filter that provides adaptation only for the first filter-tap coefficient, $w_1$, and does so with a reduced hardware complexity. Similarly, the $180^\circ$ slice generates $w_2$, and the $270^\circ$ slice generates $w_3$. These coefficients are then shared between all four interleaved DFE slices. The $0^\circ$ slice has 0 at the inputs to all its multipliers, and hence does not contribute towards filter adaptation, although we will use it later to adapt the 4-PAM reference level.

The block diagram for the circuit implementation of our adaptive filter is shown in Fig. 4.9(a), again only for the $90^\circ$ slice. Because the error signal, $\hat{y}_0[n]$, is a binary signal, the discrete-time
The integrator $\frac{1}{z-1}$ can be implemented using an up/down counter. The output of this counter drives a DAC which generates $w_1$. The DAC determines the adaptation gain, $6\mu$, and is set using a reference current, described later in Section 4.4. During calibration $w_1$ is multiplied by a constant ‘3’ to reproduce the ISI resulting from the ‘3’ in the calibration sequence. This is then subtracted from the received signal. The up/down counter is the only component in this system, aside from a few switches, that does not appear in a non-adaptive DFE, and it is driven by the digital node $\hat{y}[n]$, instead of the speed-critical analog node $y[n]$.

Intuitively, the operation of the circuit can be explained by examining the received waveform of the calibration sequence, $x(t)$, shown in Fig. 4.9(b). The phase-recovery block, described later in Section 4.3.3, aligns the ‘3’ symbol of the calibration sequence to the $0^\circ$ clock. Because a ‘0’ is transmitted during the $90^\circ$ clock phase, the input to the DFE during this period will consist only of the first ISI component, $h_1$, of the ‘3’ from the previous symbol period (assuming the ISI in the system is limited to four symbol periods in duration). When this input is sliced, the counter increments if the input is greater than 0. This in turn causes a larger signal to be subtracted from the received signal, shifting the equalized signal, $y(t)$, at the slicer input down as indicated by the dashed line. As shown in Fig. 4.9(c), this continues until the slicer input reaches 0. At this point the filter coefficient $w_1$ converges at, and oscillates around the normalized amplitude of the first ISI component, $h_1$. In a similar manner, the counters in the $180^\circ$ and $270^\circ$ slices cause $w_2$ and $w_3$ to converge to $h_2$ and $h_3$, respectively. At the end of the calibration sequence, the filter coefficients are frozen and used for equalization once the channel is returned to equalization mode. In essence, this technique directly measures the ISI and adjusts the filter coefficients to cancel it using only an up/down counter to implement the adaptation.

### 4.3.2 Reference Level Generation

To distinguish the 4-PAM symbols from each other, the DFE requires 3 comparators, each with a separate reference level. The 4-PAM symbols are members of the set $\{3, 1, -1, -3\}$, requiring reference levels of 2, 0, and -2. The zero reference is inherent to the differential slicer design, leaving the non-zero references 2 and -2 to be determined. These two references can be considered as one due to the differential nature of the design, and need to be adaptively determined.

This reference level generation scheme can be viewed as the adaptation of a cursor-symbol gain element, $w_0$. As before, in a sign-error LMS implementation it would be updated according to:
The reference-level adaptation is performed in the 0° DFE slice concurrently with the equalization filter adaptation occurring in the other 3 DFE slices. During the calibration sequence, the ‘3’ symbol is sampled by the 0° DFE slice. At the same time, all the DFE filter inputs are 0, as shown in Fig. 4.8(a). As a result, the ‘3’ symbol is applied directly to the input of the slicer, and the feedback path from the equalization filter is effectively disabled. This allows us to apply simplifications similar to those used for the filter coefficient adaptation. In this case, the cursor data, \( \hat{y}[n] \), is known to be ‘3’ from the calibration sequence. In addition, we use the ‘2’ slicer output, \( \hat{y}_2[n] \), as the sign of the error signal, although this requires an adjustment described later. As reflected in reference-level generation signal flow graph shown in Fig. 4.10(a), this allows us to simplify (4.5) to:

\[
 w_0[n + 1] = w_0[n] + 2\mu \text{sgn}(err[n])\hat{y}[n] 
\]  

In practice, it is easier to implement the gain element \( w_0 \) as a change in the slicer reference levels. Instead of adapting the input signal amplitude to match the fixed reference level, we adapt the reference level to match the input signal amplitude, as shown in Fig. 4.10(b). During adaptation, the ‘2’ reference level in the slicer is adjusted until it is equal to the amplitude of the ‘3’ pulse. This reference level is adjusted using an up/down counter controlled by \( \hat{y}_2[n] \) until the reference level equals the sampled amplitude of the ‘3’ pulse. Because this results in the reference level being adapted to ‘3’ instead of the required ‘2’, the comparator reference level needs to be scaled. This is done by implementing the reference level using 3 parallel current sources; during calibration all three are enabled, while during equalizer operation only 2 are enabled.

4.3.3 Timing Recovery

While each channel in this design has access to a source-synchronous clock, as shown in Fig. 4.6, an unknown constant phase-offset exists between this clock and the clock embedded in the received data. To compensate for this phase-offset, each channel has a phase-recovery block which creates a local clock that is synchronized with the received data. This clock is created by phase-
shifting the source-synchronous clock using a phase interpolator. This section describes the technique used to adapt the phase-code of the phase interpolator to eliminate the phase-offset.

The block diagram describing the phase recovery technique is shown in Fig. 4.11(a). Phase recovery is achieved in a similar manner to the DFE adaptation procedure, however using a separate retiming calibration sequence of ‘3 -3 -3’. This sequence is repeatedly transmitted while the local clock phase is aligned with the ‘-3’ to ‘3’ zero-crossing. This adjustment is accomplished by using

Fig. 4.10 Reference-level generation (a) signal flow graph (b) block diagram
\( \hat{y}_0[n] \) from the 0° slice as a phase detector output to drive an up/down counter. This counter produces the phase-code that controls the phase interpolator.

As shown in Fig. 4.11(b), if \( \hat{y}_0[n] = 0 \), then the clock is early and the counter is decremented. This in turn commands the phase interpolator to retard the clock, moving the sample point of \( \hat{y}[n] \) closer to the zero-crossing of the input signal. This continues until \( \hat{y}_0[n] = 1 \), where the counter increments, and the clock is advanced. From this point on, the sampling point of \( \hat{y}[n] \), will oscillate around the zero-crossing.

This timing recovery scheme aligns the clock with the transition between symbols. During data transmission, an offset of approximately \( \frac{1}{2} \) symbol period is added to the recovered phase-code to place the clock edge (and hence sampling point) in the center of the symbol period. An alternate method of timing recovery which automatically determines this offset is suggested in Section 6.2.

### 4.4 Design Implementation

This section describes the circuit-level implementation of the adaptive DFE described in the previous sections. Fig. 4.12 shows the high-level block diagram of the DFE including the up/down
counters driving D/A current sources that generate the equalization filter coefficients and the 4-PAM reference level.

As described earlier, our DFE uses a 4-phase clock to facilitate 4-way parallel interleaving. In the block diagram this is shown by the partitioning of the DFE into four separate rows. This interleaving has three advantages: First, interleaving reduces the speed requirement of the latched comparators used to implement the slicers because their reset phase can be performed during clock phases when other comparators are latching. Second, interleaving simplifies the generation of the previous decision feedback data. For example, the $\hat{y}[n-1]$ input of any of the interleaved DFEs is hardwired to the comparator output of the DFE slice operating on the previous clock phase. Third, as described in Sections 4.3.1 and 4.3.2, interleaving allows all of the DFE filter coefficients and the 4-PAM reference level to be adapted concurrently.

In addition to 4-way interleaving, each row (clock phase) in the DFE block diagram is further partitioned into three parallel paths. These three paths correspond to the three slicing levels needed to decode the 4-PAM signal. Since comparators inherently slice a signal at 0 V, we implemented the slicing levels by adding an offset to the comparator input, and combined the addition of this offset with the additions required for the equalization filter. However, because this offset addition
was combined with the filter, it is not possible to connect all three comparators to the same node, as shown in Fig. 4.7. Hence, a separate data path is required for each comparator.

At first glance, having a separate data path for each comparator appears to make redundant the stated purpose of our adaptive DFE architecture—to speed the critical path by reducing the number of comparators connected to it. In this architecture, the additional comparators required for conventional generation of the error signal would each have their own additional parallel data paths, with one comparator per path. This would, however, increase the overall size of the DFE, increasing the delay due to routing of the feedback signals from the DFE output. The net result is the same as placing many comparators on a single node—the critical path is made slower.

Due to the large number of signal additions and subtractions required by the DFE, all signal operations use current mode where they are easily performed using multiple differential pairs summing their currents into a common load. As shown in Fig. 4.12, each horizontal path in our DFE, referred to as a DFE core, can be broken down into three main blocks: The first block converts the received signal from voltage-mode to current-mode. The second block sums this signal with the DFE feedback signal and the reference levels needed for 4-PAM decoding, all in current mode using parallel differential pairs. The third block of the DFE is the slicer, implemented as a latched comparator, that uses the sum of the above mentioned currents as its input. The comparator outputs are used for DFE feedback, up/down control for the DFE adaptation counters, and also provide the decoded data that proceeds to the retiming block of Fig. 4.1. This DFE core is repeated once for each of the 3 reference level comparators \{±2, 0\}, and across each of the 4 interleaved branches, resulting in a total of 12 DFE cores.

The DFE core described above is implemented using the folded cascode architecture shown in Fig. 4.13. The differential pair on the left performs the input voltage-to-current conversion using source degeneration to set the transconductance. A zero-peaking capacitor is added in parallel to the degeneration resistor to compensate for the pole created at the current summing node, increasing the bandwidth of the input signal path.

The next three differential pairs implement the decision feedback signals using current sources weighted by \(w_1\), \(w_2\), and \(w_3\). Although only one differential pair per coefficient is shown, there are actually three differential pairs per coefficient, as the feedback data is encoded using 3-bit
thermometer coding. The reference level is implemented using three additional differential pairs, as discussed in Section 4.3.2, but these are omitted for clarity.

The column on the right side of the schematic has a current source on top, with a cascode stage below it. This PMOS current source is biased to provide approximately three times the current sunk by the input transconductance stage. Below the cascode stage is a current steering stage that directs current either into the clocked comparator below, or directly to ground during comparator reset. Without the current steering stage, the current-summing node would be pulled high during comparator reset, with the PMOS current sources entering into the triode region. This would result in incorrect current mirroring through the cascode stage when the comparator is first enabled. Below the current steering stage is the clocked comparator. This is implemented as a sense-amp latch comprised of two back to back inverters and two reset transistors.
The folded cascode design provides a low-impedance at the high-capacitance current-summing nodes which, with the added benefits of the zero-peaking capacitor, increases the bandwidth of the circuit. The settling time of this node constitutes a large fraction of the DFE’s critical path, and expanding its bandwidth increases its maximum bit-rate. In addition to the bandwidth expansion, the folded cascode architecture also provides more voltage headroom in a 1.8V supply environment.

The filter coefficient and reference-level current sources in this circuit are generated using 8-bit current DACs. They are binary weighted, using parallel current sources of unit size. The bias currents for these DACs are established using replica biasing. As shown in Fig. 4.14, a replica of the transconductance-stage degeneration resistor is used to create a reference current, $I_{\text{ref}}$, equal to the differential current resulting from the expected 300 mV maximum input signal swing. The reference currents for the decision feedback DACs are set such that the maximum currents implementing $w_1$, $w_2$, and $w_3$ are equal to $0.5I_{\text{ref}}$, $0.25I_{\text{ref}}$, and $0.25I_{\text{ref}}$, respectively. This is consistent with a pulse response with a duration that does not exceed 4 symbol periods, and reduces the size of the circuit. The maximum slicer reference-level current is set to $I_{\text{ref}}$, equivalent to the maximum pulse amplitude given a 300 mV swing.

In addition to the blocks described above, the design of this DFE receiver also includes other components, such as a VCO and numerous phase interpolators. As the design of these components is straightforward and well documented in the available literature, their design descriptions are omitted.
4.5 Simulation and Measurement Results

The full transceiver was designed and implemented in a 0.18 μm CMOS process. The die photo of the 2.3 x 2.3 mm² test chip is shown in Fig. 4.15.

Measurements of our fabricated design confirm the correct operation of the design’s low-level functions, including automatic timing recovery, adaptation of the DFE filter coefficients and 4-PAM reference level. However, the VCO was dysfunctional, and this prevented full system-level tests.

A block diagram of the VCO test features included in the design is shown in Fig. 4.16. There are two dedicated clock output pins: one directly buffered from the VCO output, and one that passes the VCO output through a phase-interpolator. Measuring the signal at these two pins (with the rest of the design disabled) revealed that the VCO was unstable, with large changes in clock period.

The VCO instability was traced to the existence of two stable oscillation modes. The VCO is a 4-stage differential ring oscillator, as shown in Fig. 4.17(a). The quadrature clock outputs of this design were taken from only 2 of the 4 VCO stages, resulting in unequal stage loading. In one oscillation mode, both the loaded and unloaded stages oscillated with full-swing outputs. However, under some bias conditions, a second oscillation mode existed where the unloaded stages remained...
full-swing, but the loaded stages became low-swing. This changed the oscillation frequency as shown in Fig. 4.17(b), where the VCO frequency is plotted against tuning voltage for both simulated and measured results. The low-slope portion of the curve represents the full-swing/full-swing oscillation mode, while the high-slope portion of the curve represents the full-swing/low-swing oscillation mode. These two modes are illustrated in Fig. 4.17(c), where the simulated output voltage of an unloaded clock node is shown settling to each of the modes for two closely-separated tuning voltages. While the implemented VCO favoured one oscillation mode over another (depending on the tuning voltage), it was obvious that, regardless of the tuning voltage, the VCO would intermittently change modes, resulting in severe clock jitter.

In addition to the VCO instability, changing the phase-control settings on the phase interpolator caused the phase interpolator output and the buffered VCO output to change their jitter behaviour, and also caused slight changes in VCO oscillation frequency. Because there is no direct feedback path from the phase-interpolator back to the VCO, we concluded that feedback through supply noise was the responsible mechanism. The likely cause of the supply noise sensitivity is insufficient on-chip power supply decoupling. This conclusion led to design changes, such as increased on-chip decoupling, that prevented the recurrence of these problems in the CDR described in Chapter 5.

The rest of this section discusses the simulation results of the DFE that demonstrate its correct operation, and measurement results from the built-in pulse-response monitor.
4.5.1 Simulated Eye Diagram

To demonstrate the correct operation of the equalizer and the adaptation scheme, we present the simulated eye-diagram of the DFE. As a result of the DFE architecture, there is no external node on which to measure an equalized signal; the equalized signal only exists at the current-summing node at the input to the slicers. The simulated eye diagram for this node in one of the interleaved DFEs is shown in Fig. 4.18 with the equalization turned both off and on. With equalization off, no eye is visible, while a clean eye opening is visible with the equalizer turned on. Because the feedback input to the interleaved DFE is only valid for one out of every four symbols, the eye pattern shows a 4-symbol cycle, where the eye is open only during one symbol period. The signal level spacing shown in the eye diagram represents a 12 µA differential current on a nominal 60 µA flowing through the latched comparator branch shown in Fig. 4.13. These simulation results are for a signaling rate of 2 Gb/s over the equivalent of 2m of FR-4 PCB trace. The transmitted signal level is 200 mV/level (differential), ranging from 1.5 V to 1.8 V.
4.5.2 Pulse Response Monitor

The DFE adaptation circuits in our design can also be used to measure the pulse response of the system’s channel using a technique similar to that of a digital sampling oscilloscope. While transmitting the calibration sequence ‘3 0 0 0’, we measure the amplitude of the received signal at each of the 256 evenly spaced clock phases available from the phase-interpolator used for phase-locking of the local clock to the received data. The phase control of the phase interpolator is adjusted using the programmable phase-offset register normally used to center the sampling point in the center of the data eye after phase recovery. At each clock phase, the sample amplitude is measured using the circuits designed to adaptively determine the 4-PAM reference level. As described in Section 4.3.2, finding the 4-PAM reference level is accomplished by finding the amplitude of a symbol ‘3’ pulse at its sampling point. If we place the DFE in adaptation mode while changing the clock phase, then the newly adapted value of the ‘reference level’ provides a measure of the sample’s amplitude over the range of adjustable phase. This measurement is monitored externally using the chip’s scan chain. Plotting this measure against clock phase produces the channel pulse response over one clock cycle (4 UI).
Fig. 4.19 shows the pulse response of a short channel on our test board. This channel is capacitively loaded at the DFE input to emulate the losses of a longer channel. The pulse response extracted using the DFE adaptation circuits is nearly identical to the direct measurement using an oscilloscope. This result also demonstrates the correct operation of most of the major elements of the design, such as the DFE, its adaptation circuits, and the phase interpolators, as they are all used to measure the pulse response. However, to compensate for the clock related problems described earlier, it was necessary to average 256 noisy measurements to produce this result.

4.6 Summary

This chapter proposed an adaptive equalization technique that uses direct measurement and cancellation of ISI during an intermittent calibration sequence to determine equalizer filter coefficients. This design eliminates the analog blocks that would be required to generate the error signal using LMS adaptation. The elimination of these blocks from the DFE critical path is essential for high-speed operation. While a malfunctioning VCO prevented the fabricated design from being fully functional, simulation results show that it is capable of operation at bit rates up to 2 Gb/s.
Semi-blind Oversampling Clock and Data Recovery

Jitter tolerance refers to the maximum amplitude of sinusoidal jitter (as a function of frequency) that can be tolerated without causing data recovery errors. A phase-tracking CDR tolerates jitter at frequencies within the bandwidth of its loop filter, but performs poorly above this frequency [60]. A blind-oversampling CDR, on the other hand, tolerates jitter in its input data stream at mid-range frequencies that are higher than that of typical loop bandwidths, but can be limited at low-frequencies by the size of its FIFO [41]. We propose a semi-blind oversampling technique that embeds a blind-oversampling CDR within a phase-tracking CDR. As shown in Fig. 5.1, the proposed Semi-Blind OverSampling technique produces a jitter tolerance, $Jtol_{SBOS}$, equal to the product of the Phase Tracking jitter tolerance, $Jtol_{PT}$, and the Blind OverSampling jitter tolerance, $Jtol_{BOS}$, thereby increasing the low-frequency jitter tolerance by the FIFO size (32 in our design). This increase allows additional system level design options. For example, increased low-frequency

Fig. 5.1  CDR jitter tolerance
The increased jitter tolerance permits enhanced spread-spectrum clocking [62] where the clock embedded in the received data can deviate from its nominal phase by as many as a few hundred UI’s. The increased jitter tolerance can also absorb jitter from both the external data source and from the internally generated VCO noise present in the increasingly noisy environment of today’s highly integrated SOCs. Finally, it also allows an additional degree of freedom in the trade-off between jitter tolerance, jitter transfer, and jitter generation. While our design increases the jitter tolerance, it does so without affecting the jitter transfer and jitter generation.

To provide context for our proposed hybrid semi-blind oversampling technique, we first describe the jitter tolerance of a phase-tracking CDR in Section 5.1, and then that of a blind-oversampling CDR in Section 5.2. Section 5.3 conceptually describes the proposed semi-blind oversampling CDR, and derives equations for its jitter tolerance. Section 5.4 describes the implementation of the semi-blind oversampling CDR. The simulated and measured results of its implementation are then discussed in Section 5.5. Finally, Section 5.6 summarizes the contributions of this work.

5.1 Jitter Tolerance in the Phase-Tracking CDR

A phase-tracking CDR employs feedback to keep the recovered clock in phase with the clock embedded in the received data, as shown in Fig. 5.2. The recovered clock is then used to sample (recover) received data using a sampler. Under ideal conditions, with no noise, ISI, or clock jitter, error-free data recovery is achieved when the received data is sampled within \( \frac{1}{2} \) UI of the nominal sampling point—the center of the data eye. In terms of the input clock phase, \( \phi_{in} \), and the recovered clock phase, \( \phi_{osc} \), this condition for ideal error-free data recovery is expressed as:

\[
K_{pd} (\phi_{in} - \phi_{osc})<\frac{3\pi}{2} \quad \text{for ideal error-free data recovery}
\]

Fig. 5.2 Phase-tracking CDR
The jitter tolerance of this CDR is the largest sinusoidal amplitude of $\phi_{in}$ (the input jitter) that satisfies (5.1). For a charge-pump CDR using a series RC filter [64], the voltage to current relationship is given by:

$$H_{lp}(s) = R + \frac{1}{sC} \tag{5.2}$$

Therefore, the phase transfer function $\frac{\Phi_{osc}(s)}{\Phi_{in}(s)}$ can be written as:

$$\frac{\Phi_{osc}(s)}{\Phi_{in}(s)} = \frac{1 + sRC}{1 + sRC + \frac{s^2C}{K_{pd}K_{osc}}} \tag{5.3}$$

where $\Phi(s)$ is the Laplace transform of $\phi(t)$.

Combining (5.3) and (5.1) and solving for twice the largest amplitude of $\Phi_{in}(s)$, we derive the peak-peak Phase-Tracking (PT) jitter tolerance:

$$J_{tol\_PT} = \left| \frac{K_{pd}K_{osc}}{-\omega^2C} + \frac{K_{pd}K_{osc}R}{j\omega} + 1 \right| UI \tag{5.4}$$

In a critically damped loop ($Q = 1/R_{pd} \sqrt{C K_{pd}K_{osc}} = 0.5$), the jitter tolerance has two poles at the origin and two zeros at $2\pi f_0 = \sqrt{K_{pd}K_{osc}/C}$, as pictorially shown in Fig. 5.3. For this type

![Fig. 5.3 Jitter tolerance of a phase-tracking CDR under ideal conditions](image-url)
of CDR, the only way to increase the jitter tolerance is to increase $f_0$, moving the tolerance curve to the right. However, as in all closed-loop feedback systems, increasing $f_0$ degrades the stability of the loop [33].

Instead of increasing $f_0$, the semi-blind oversampling CDR proposed in this chapter achieves a higher jitter tolerance by embedding a blind-oversampling CDR within a phase-tracking CDR. In the following section we examine the jitter tolerance of a blind-oversampling CDR.

### 5.2 Jitter Tolerance in the Blind Oversampling CDR

Unlike the phase-tracking CDR, the blind-oversampling CDR does not attempt to track the clock embedded in the received data. Instead, it oversamples the received data using a local clock uncorrelated with the embedded clock. As shown in Fig. 5.4, this feed-forward architecture determines the embedded clock phase from the transitions in the samples. The recovered phase is then used to select (down-sample) the data bits within a window of oversampled data. These bits then enter an elastic FIFO. For every additional bit period that the local clock lags the embedded clock, an additional bit must be delayed (stored) by the FIFO. This makes the occupied FIFO depth proportional to this phase difference with a resolution of 1 UI, a property which will be exploited later for inter-bit phase-detection.

Blind-oversampling has an important feature that we exploit to overcome the sampling restriction (5.1) of the phase-tracking CDR. Because phase detection and data selection occur after sampling, there is no longer an inherent restriction on the phase relationship between the local clock and the clock embedded in the received data. These clocks may differ by many UI, as long as the FIFO is large enough to absorb the difference. To prevent the overflow of the FIFO due to slight

---

**Fig. 5.4 Blind-oversampling CDR**

- **received signal** $\phi_{in}$
- **oversampler**
- **samples**
- **Down Sampler**
- **data**
- **Elastic FIFO**
- **recovered data**
- **Phase Detector**
- **intra-bit phase**
- **FIFO write pointer**
- **inter-bit phase**
- **local clock**
The jitter tolerance of a blind-oversampling CDR with a 5x oversampling ratio is illustrated in Fig. 5.5, where the tolerable sinusoidal jitter amplitude is plotted as a function of jitter frequency. At high jitter frequencies, the jitter tolerance is $\frac{2}{5}$ UI peak-to-peak, as described in Section 2.3.2. This occurs when the time between transitions, $t_{trans}$, (upon which the phase detector relies) exceeds half the jitter-period—the time between peak-to-peak phase values in a sinusoidal jitter source.

As the jitter-frequency decreases, the CDR begins to track phase changes in the received signal not exceeding $\frac{2}{5}$ UI between transitions, or:

$$\left| \frac{d}{dt} \Phi_{in}(t) \right| < \frac{2}{5} \frac{UI}{t_{trans}} \quad (5.5)$$

To find the resulting jitter tolerance, we apply sinusoidal input jitter, $\phi_{in}(t) = 2\pi A_j \sin(2\pi f_j t)$ rads, to the CDR, where $A_j$ is the jitter amplitude measured in UI. Inserting this into (5.5) and maximizing the derivative results in the following condition for the jitter amplitude:

$$2\pi f_j A_j < \frac{2}{5} \frac{UI}{t_{trans}} \quad (5.6)$$
This relation shows that the maximum jitter amplitude is a function of the time between transitions. We must satisfy Eq. (5.6) for all possible values of \( t_{\text{trans}} \), including the worst-case \( t_{\text{trans}} \) of \( L t_b \), where \( L \) is the runlength of the input sequence and \( t_b \) is the bit time. Therefore, the peak-peak Blind-OverSampling (BOS) jitter tolerance becomes:

\[
J_{\text{tol}, \text{BOS}} = 2 A_j = \frac{2}{5 \pi f_j L t_b} \text{UI}
\]  

(5.7)

The jitter tolerance thus increases at 20dB/decade, with decreasing frequency, until the CDR’s FIFO becomes saturated. At and below this frequency, the peak jitter tolerance remains constant and equal to the FIFO size. This occurs as the FIFO oscillates from empty to full with a swing equal to the FIFO size around its nominally half-full condition.

### 5.3 Proposed Semi-blind Oversampling CDR

The contribution of this work is the use of the blind-oversampling CDR to overcome the sampling time restriction of the phase-tracking CDR, as described by (5.1). This is illustrated in Fig. 5.6, where a blind-oversampling CDR replaces the sampler in a phase-tracking CDR. We term this hybrid design a **semi-blind** oversampling CDR, as the local clock used for oversampling is no longer blind, but instead tracks phase change within the bandwidth of the phase-tracking loop. Whereas a phase-tracking CDR has to maintain a phase difference between received and local clocks of less than \( \frac{1}{2} \text{UI} \), as specified in (5.1), the new phase detector allows a phase difference equal to the peak jitter tolerance of the blind-oversampling CDR, or:

\[
|\phi_{\text{in}} - \phi_{\text{osc}}| \leq \frac{1}{2} J_{\text{tol}, \text{BOS}}
\]  

(5.8)

![Fig. 5.6 Semi-blind Oversampling CDR](image-url)
This is similar to the embedding of a DLL inside a PLL as described in [63], in which an analog phase-shifter shifts the phase of the received signal before it is applied to a conventional phase-tracking CDR. This approach increases the allowable phase-difference between the recovered and embedded clocks by an amount equal to the tuning range of the analog phase-shifter, which is limited to approximately 2UI. In contrast, our design uses a blind-oversampling CDR as a digital phase-shifter that increases the allowable phase-difference by an amount equal to its FIFO size. Since the FIFO size can be chosen by the designer, the allowable phase-difference becomes merely another design parameter.

Combining (5.8) with the phase transfer function of the phase-tracking CDR, (5.3), and solving for twice the maximum value of $\Phi_{in}$, we obtain the peak-peak jitter tolerance of this Semi-Blind Oversampling (SBOS) CDR:

$$J_{tol_{SBOS}} = J_{tol_{BOS}} \left| \frac{K_{pd}K_{osc}}{-\omega^2 C} + \frac{K_{pd}K_{osc}}{j\omega} + 1 \right|$$

$$= J_{tol_{BOS}}J_{tol_{PT}}$$

Thus, the jitter tolerance is the product of the blind-oversampling and phase-tracking tolerances. This means that we can add the log-scale jitter tolerance curves in Fig. 5.3 and Fig. 5.5, to obtain the semi-blind oversampling CDR jitter tolerance shown in Fig. 5.7. Because the jitter tolerance of
the blind-oversampling CDR at low frequencies saturates at the FIFO size, the resulting jitter tolerance at these frequencies is a factor of the FIFO-size higher (32 in our implementation) than that of the conventional phase-tracking CDR.

5.4 Design Implementation

Fig. 5.8 shows the block diagram of our semi-blind oversampling CDR. A 20-phase 800MHz VCO is used to 5x oversample a 3.2Gbps sequence. Therefore, in one period of the 800MHz clock, we collect a total of 20 samples, corresponding to 4 UI in the received data. These 20 samples are then aligned to a single clock edge by the retiming block and passed to the fine-phase detector. The fine phase is the intra-bit component (modulo 1 UI) of the phase difference between the clock embedded in the received data, and the recovered clock. The fine-phase detector uses the location of transitions within the window of 20 samples to determine the fine phase. The fine phase is then used by the down-sampler to pick the data bits among the 20 samples. Due to the changing fine phase, the downsampled data size from a 20-sample window can be between 3 and 5 bits, as explained later. This data is then written into an 8x4 elastic FIFO where up to a total of 32 UI peak-peak jitter (the FIFO size) is absorbed.

Since the occupied depth of the FIFO is an indication of the phase difference between embedded and recovered clocks, the FIFO write pointer can be used as a measure of the coarse phase. This coarse phase is measured in integer multiples of UI and complements the fine phase, which provides the intra-bit phase. Phase tracking in this CDR is implemented by passing the FIFO write pointer (coarse phase) to a 5-bit DAC, which is then low-pass filtered to provide the control voltage for the VCO. The loop is then closed by feeding the recovered clock (from the VCO) to the blind-oversampling component, creating a hybrid CDR that is no longer blind, but tracks low-frequency jitter with higher tolerance.

To verify correct operation of the CDR, the FIFO output is sent to an internal bit-error tester (BERT). This BERT duplicates the externally generated PRBS sequence applied to the CDR, and counts the discrepancies between that sequence and the FIFO data. A scan-chain provides external access to the error count, along with other important nodes within the circuit.

The following sub-sections describe the detailed implementation of the various blocks in Fig. 5.8.
5.4.1 VCO/Samplers

Fig. 5.9(a) shows the circuit implementation of the 20-phase VCO using a 10-stage ring oscillator. Although the figure shows single-ended signals for clarity, all signals are differential. Small extra buffers cut halfway across the ring, creating sub-feedback loops that enable increased VCO frequency [61]. The CDR samplers are also integrated with the VCO, shown as the small blocks marked ‘S’ on the periphery of the ring, eliminating the need to distribute a multi-phase clock. While distributing a multi-phase clock would require multiple clock buffers, the data is an externally generated signal that is strong enough to drive all the samplers via a common net without additional buffering, thus eliminating potential clock skew problems.

Fig. 5.9(b) shows the transistor-level schematic of the shaded block in Fig. 5.9(a). The top and bottom halves of the schematic are mirror images of each other. In each half, an NMOS differential pair for a main VCO buffer and a sub-feedback buffer share common PMOS loads. The inputs of the sub-feedback differential pairs are connected to the outputs on the other side, while the inputs to the main VCO buffers are driven by the outputs of the previous stages.
The floorplan of the 10-stage VCO and samplers is organized as shown in Fig. 5.10, with the VCO stages in the center, and the samplers surrounding them. Including dummy stages at the periphery, the resulting VCO layout is 70\(\mu\)m in width and 25\(\mu\)m in height. This arrangement is chosen for two reasons: First, it minimizes the interconnection distance, and hence capacitance, at each stage. This is accomplished by placing VCO stages connected through sub-feedback loops vertically opposed to each other using a layout topologically consistent with the schematic shown in Fig. 5.9(b). The shaded pair of VCO stages indicate one instantiation of this schematic. The second reason is to desensitize the VCO to process gradients on the chip. As a result of the floorplan, any set of 5 sequential phases will traverse 5 of the 10 stages in such a way that they occupy each column once, in one of the two rows. Since the dominant layout dimension of the VCO was horizontal, we considered the delay of two vertically opposed stages to be nearly equal. As a result, while the delay of individual stages may be unequal, the total delay of any 5 phase-sequential stages (nominally 1 UI) is expected to be equal and unaffected by process gradients.
5.4.2 Retiming and Voting

The 20 samples generated by the VCO samplers are aligned to 20 evenly-spaced clock phases. The retiming block, shown in Fig. 5.11, applies the 20 samples to a voting block, discussed below, and then aligns the voted samples to a single clock phase (one of the 20). This clock is common to all other digital logic in the CDR. The retiming circuit is followed by a multiplexer used for debugging purposes. This multiplexer can bypass the retimed data, replacing it with an externally provided sequence. The external_samples data and sample_mux control signal are set using the scan chain. The scan chain is also capable of capturing and shifting out the retimed samples.
The retiming block includes a 2/3 majority sample voting block which aims to reduce the effect of noise in the input samples. For each sample, the voting block outputs the value shared by the majority (2 or more) of a 3 sample window centered on that sample. The purpose of the voting block can best be illustrated with an example: While the sequence of samples ‘0001111’ is unchanged after voting, the sequence ‘00010111’ becomes ‘00001111’ after voting. This change eliminates the extra transitions that result from noise near data transitions. These extra transitions can upset the fine-phase detector, and eliminating them improves the BER, as we will see later. The voting block can be enabled or disabled to demonstrate its effect on the BER by using the signal, vote_mode, which is set using the scan chain.

5.4.3 Fine-Phase Detector

The fine-phase detector determines the intra-bit clock phase, or fine phase, based on data transitions within a window of 20 samples spanning four nominal bit periods. In general, these transitions can occur on any of five fine phases (0 to 4, representing $0/5$ to $4/5$ UI), consistent with a 5x oversampling rate. The fine-phase detector determines which fine phase within one window is most likely to have resulted in the distribution of transitions across the 5 fine phases. This fine phase is later used to downsample the data within that window.

A block diagram of the fine-phase detector is shown in Fig. 5.12(a). The input to the fine-phase detector is the oversampled and retimed window of 20 samples, $x_n$. Each sample is XORed with its preceding sample to locate the data transitions, $t_n$. The number of transitions (0-4) occurring during each fine phase are then totalled to find the transition counts, $T_n$. The final block finds the fine phase, $\bar{n}$, which is essentially a weighted average of the transition phases, $n$, using the $T_n$ as weights. If no transitions occur, the fine phase from the previous window is retained.

Fig. 5.12(b) shows an example sequence occurring during one window. On top, the figure shows the 20 sample-periods broken down into 4 nominal bit-periods, each having 5 samples with fine phases, $n$, of 0 to 4. Under this is the sequence of samples, $x_n$. Below this are the transitions, $t_n$. The transitions occurring on each fine phase are then added to form the transition totals, $T_n$, below. This process is illustrated for $T_0$. The computation of $\bar{n}$ from these transition totals is described next.

In previous works [39], a simple majority voting scheme is used to determine $\bar{n}$ by picking the phase with the highest transition count. Such schemes, however, will be less accurate in the
presence of certain types of high-speed jitter. For example, consider an input signal consisting of a lone pulse where ISI shifts each pulse edge by one sample phase. Instead of producing a set of transition counts such as ‘0 0 2 0 0’ where both pulse edges have the same phase, we would get ‘0 1 0 1 0’. In this case, the majority voting scheme would pick the phase of one of the two edges, resulting in a phase error equal to one sample period. To produce a more accurate result, some form of phase averaging is required.

As discussed in Section 5.2, the 5x blind-oversampling component of the CDR can track phase changes in the received signal occurring at a rate of less than $\frac{2}{5}$ UI between transitions. Since our system uses a $2^{31}-1$ PRBS sequence, the maximum runlength of the sequence is 31 bits. This results in at most 32 UI between transitions, occurring over 8 windows (of 4 UI each). Over 1 window, this gives a maximum phase change of $\frac{2}{5}$ UI divided by 8, or $\frac{1}{20}$ UI. Because we measure the fine phase in discrete increments of $\frac{1}{5}$ UI, this will usually not be measurable within one window, and
as such can be approximated as constant over one window. As a result, we can treat each transition within one window as a measure of the same phase with an added component of random jitter. As a result, we can construct a fine phase probability density function (PDF) in the form of \( T_n / T \), where \( T \) is the total number of transitions. The expected fine phase, or average transition phase, can be calculated as:

\[
\bar{n} = E[n] = \frac{1}{T} \sum_{n=0}^{4} nT_n
\]  

(5.11)

There is an ambiguity, however, in the direct application of (5.11) that can result in multiple values of \( \bar{n} \) for the same set of \( T_n / T \). This ambiguity is illustrated in Fig. 5.13 where two different PDFs are associated with the same set of \( T_n / T \). In each PDF \( T_4 / T \) has a fine phase of 4, although occurring at two different coarse phases. Blindly applying (5.11) to both PDFs produces \( \bar{n} = 1.25 \). However, as shown in the figure, knowledge of the coarse phase shows that \( \bar{n} = 0 \) is the correct result for PDF2.

The above ambiguity can be resolved by applying the constraint that the change in fine phase between two transitions cannot exceed \( \frac{2}{5} \) UI. This constraint restricts the composite phase of the \( T_n \) to a 1 UI contiguous region centered around the fine phase from the previous window. In the above example, if the previous fine phase was 4, this would be represented by the shaded region in Fig. 5.13. Given a set of \( T_n \) and the previous fine phase, this allows only one interpretation of the

\[
\text{Allowable phase region if } \bar{n}=4 \text{ in previous window}
\]

as such can be approximated as constant over one window. As a result, we can treat each transition within one window as a measure of the same phase with an added component of random jitter. As a result, we can construct a fine phase probability density function (PDF) in the form of \( T_n / T \), where \( T \) is the total number of transitions. The expected fine phase, or average transition phase, can be calculated as:

\[
\bar{n} = E[n] = \frac{1}{T} \sum_{n=0}^{4} nT_n
\]  

(5.11)

There is an ambiguity, however, in the direct application of (5.11) that can result in multiple values of \( \bar{n} \) for the same set of \( T_n / T \). This ambiguity is illustrated in Fig. 5.13 where two different PDFs are associated with the same set of \( T_n / T \). In each PDF \( T_4 / T \) has a fine phase of 4, although occurring at two different coarse phases. Blindly applying (5.11) to both PDFs produces \( \bar{n} = 1.25 \). However, as shown in the figure, knowledge of the coarse phase shows that \( \bar{n} = 0 \) is the correct result for PDF2.

The above ambiguity can be resolved by applying the constraint that the change in fine phase between two transitions cannot exceed \( \frac{2}{5} \) UI. This constraint restricts the composite phase of the \( T_n \) to a 1 UI contiguous region centered around the fine phase from the previous window. In the above example, if the previous fine phase was 4, this would be represented by the shaded region in Fig. 5.13. Given a set of \( T_n \) and the previous fine phase, this allows only one interpretation of the

1. Technically, this should be called probability mass function (PMF) as it applies to a discrete random variable.
respective coarse phases. Only PDF2 in Fig. 5.13 satisfies this constraint. Once the correct PDF is constructed, its fine phases must be unwrapped to reflect their relative phases. In this example, this would mean treating the fine phases in the shaded region as \{2,3,4,5,6\}, not \{2,3,4,0,1\}. After unwrapping, we can apply (5.11) to determine \( \hat{n} \).

While the transition detector and transition counter of Fig. 5.12(a) can be implemented in a straightforward fashion, a brute-force arithmetic approach to averaging and phase-unwrapping in the average transition locator results in a circuit with a delay in excess of the short cycle times required in a CDR operating in the Gb/s regime. To speed up the computation, we use a slightly different viewpoint of averaging, as outlined next.

**Average Transition Detector Implementation**

The implementation of the average transition detector is based on the observation that (5.11) can be rewritten as:

\[
\sum_{n=0}^{4} (n - \bar{n}) T_n = 0
\]  

(5.12)

Accordingly, if we define

\[
f(\hat{n}) = \sum_{n=0}^{4} (n - \hat{n}) T_n
\]  

(5.13)

then \( f(\hat{n}) \) is a decreasing function of \( \hat{n} \) with a single zero at \( \hat{n} = \bar{n} \). Therefore, if we decrease \( \hat{n} \) from its maximum value, there will be an \( \hat{n} \) for which the sign of \( f(\hat{n}) \) changes from positive to negative. By simultaneously calculating \( f(\hat{n}) \) between all possible integer values of \( \hat{n} \) (at 0.5, 1.5, 2.5, 3.5), we can identify \( \bar{n} \) (rounded to the nearest integer value) as laying between the values of \( \hat{n} \) for which \( f(\hat{n}) \) changes polarity. For example, if \( f(1.5) = 1.5 \) and \( f(0.5) = -1.5 \), then we can conclude that \( \bar{n} = 1 \).

Unfortunately, the evaluation of (5.13) for multiple \( \hat{n} \) requires a large layout area, and results in an implementation that is still too slow for our application. However, we can achieve a simple and fast implementation by replacing \( n - \hat{n} \) in (5.13) with a monotonic function, \( g(n - \hat{n}) \):
yet maintain similar properties. The function we use is illustrated in Fig. 5.14. As shown, this function increases by a factor of 2 on each side of \( n - \hat{n} = 0 \). This reduces the calculation complexity of (5.14) as all multiplications now reduce to simple bit-shifts and, as will be shown, partial results can be shared between the calculations for the various values of \( \hat{n} \). This simplification produces almost identical results to (5.13) with a few minor differences that are discussed later.

The above substitution of \( g(n - \hat{n}) \) allows the calculation of \( \bar{n} \) with a simple implementation. Fig. 5.15 shows an example of such an implementation for PDF2 in Fig. 5.13. The implementation consists of a cascade of 5 identical blocks, one for each phase in the allowable phase region such that the phase is unwrapped with previous fine phase \( (\bar{n} = 4) \) at the center. Each of these blocks simultaneously contributes a portion of the computation required to calculate \( f(\hat{n}) \) at all 4 intermediate values of \( \hat{n} \).

Running from top-to-bottom and bottom-to-top, partial sums from the previous block are multiplied by two using a bit shift, and then added to the \( T_n \) of the local block. The resulting new partial sum is then passed on to the next block, where the process is repeated. As a result, at any point a \( T_n \) from \( m \) stages away is multiplied by \( 2^m \). Therefore, the partial sum at any point gives the contribution towards \( f(\hat{n}) \) from all the \( T_n \) on the side of the block from which that partial sum originated. To fully calculate \( f(\hat{n}) \), the partial sum running in one direction is subtracted from the partial sum from the other direction (shown by the value in brackets between partial sums). We keep only the most-significant bit (MSB) from the subtraction to indicate the polarity. The MSBs bounding each phase are then XORed to generate an average fine phase output (one-hot encoded) where the polarity of \( f(\hat{n}) \) changes.

<table>
<thead>
<tr>
<th>( n - \hat{n} )</th>
<th>-3.5</th>
<th>-2.5</th>
<th>-1.5</th>
<th>-0.5</th>
<th>0.5</th>
<th>1.5</th>
<th>2.5</th>
<th>3.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g(n - \hat{n}) )</td>
<td>-8</td>
<td>-4</td>
<td>-2</td>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

Fig. 5.14 \( g(n - \hat{n}) \) as a function of \( n - \hat{n} \)

\[
f(\hat{n}) = \sum_{n=0}^{4} g(n - \hat{n}) T_n
\]
To complete the implementation, we must ensure that the previously discussed phase-unwrapping has been performed. This can easily be done as shown in Fig. 5.16 where the five stages in the phase detector are connected in a loop. This loop is cut between blocks $1/2$ UI away from the previous fine phase output using AND gates driven by the previous one-hot fine phase. This cut creates a 1 UI contiguous region around the previous phase, as shown in Fig. 5.13(b). Once cut, the result is the 5-stage system of Fig. 5.15.

To determine the loss of accuracy of a fine-phase detector using $g(n - \hat{n})$ instead of $n - \hat{n}$, we used a behavioural simulation written in C. This simulation compared the fine phase outputs of phase-detectors using both methods, over all 126 possible unique combinations of 1 to 4 transitions. The results matched in all but 8 cases. Of these mismatches, 2 are pathological transition counts that do not happen during normal operation. Another 3 cases are boundary cases where the fine phases lie exactly halfway between discrete fine phases and could be rounded to one side or the other with identical rounding errors. The final 3 cases represent a phase-detection error of $1/30$ UI.
For these 3 cases, the tracking ability of the blind oversampling drops from $\frac{2}{5}$ UI to $(\frac{2}{5} - \frac{1}{30})$ UI between transitions. The reduction in tracking ability translates to a proportional decrease in the high-frequency jitter tolerance of our hybrid CDR, due to the blind-oversampling component described in Section 5.2. However, it has no effect on the low-frequency jitter tolerance where the FIFO is saturated and the blind-oversampler is no longer limited by the rate of phase-change. The loss at high-frequencies is a small price given the increased bit-rate permitted by the simplified implementation.

Despite its simple and fast architecture, the average transition locator shown in Fig. 5.16 is still the speed bottleneck of the CDR. The critical path starts at the register generating the old fine phase, $\bar{n}'$, then proceeds to the AND gates which cut the loop of partial-sum blocks. It then goes through these blocks and comes out as the new fine phase, $\bar{n}$, going back to the register. While the average transition locator is bracketed by pipeline stages, as shown in Fig. 5.12(a), the critical path
described above cannot be pipelined, limiting the current implementation to a bit-rate of 3.5Gb/s. A potential solution to this issue is outlined as a future work in Section 6.2.

5.4.4 Downsampler

The downsampling block, shown in Fig. 5.17, selects the data bits amongst the 20 samples in a window, guided by the fine phase. Four 5-1 multiplexers then provide 5x downsampling, using the fine phase to pick the sample in the centre of the data eye, which is 0.5 UI, or 2.5 samples, from the average transition location. Because transitions are detected by XORing a data sample with its preceding sample, the transition must be located between those two samples. On average, the transition will lie 0.5 samples before the sample with which it is identified. Hence, the phase closest to the centre of the data eye, the sample-phase, is 2 samples (2.5 - 0.5) after the fine phase. Since the fine phase is one-hot encoded, the sample-phase can be generated using only a permutation of the fine phase bits.

Under typical conditions, we downsample only 4 bits from the 20 samples. However, depending on the relationship between the sampling phases of the current and previous windows, there exists 6 cases (among a total of 25) for which we must pick an additional bit or drop a bit, ending up with 3 or 5 bits. The graph on the left side of Fig. 5.18 shows the downsampled data-size as a function of the current and previous sampling phase. On the right side of Fig. 5.18 are three examples showing the different downsampled data sizes. The different sizes arise due to the sampling phase drifting across the boundary between sample windows. When the sampling phase drifts forward

![Fig. 5.17 Downsampler](image-url)
across this boundary, as shown in the bottom example, an oversampled data-bit straddling the window boundary is demultiplexed twice—once in each window. Whenever this happens, we drop one of these bits. When the sampling phase drifts backward across this boundary, as shown in the top example, the bit straddling the window boundary is not demultiplexed in either window. In this case, we add a bit.

To deal with the additional-bit condition, the first bit in the 20-sample window is prepended to the downsampled data after the 5-1 multiplexer in Fig. 5.17, as shown by the dashed line bypassing the multiplexers. This 5-bit demux_data is then sent to the elastic FIFO. However the number of these bits used depends on the downsampled data size. When the data size is 4 bits (neither data_size_3 nor data_size_5 asserted), the 4 bits from the multiplexer are used and the prepended bit is discarded; When an additional bit results in a 5-bit data size, all 5 bits are used; When a bit is deleted, resulting in a 3-bit data size, both the prepended bit and following bit are discarded. The data size is determined based on the current and previous sampling-phase by the data-size detector of Fig. 5.17, which implements the mapping function shown in Fig. 5.18. The fine_phase, data_size_3/5 and demux_data signals are available for debugging purposes as read-only registers in the scan-chain.
5.4.5 Elastic FIFO

The elastic FIFO block accepts 3-5 bits from the downsampler block during each clock cycle, while outputing 4 bits. It has a depth of 32 bits, and hence is capable of absorbing up to 32 UI peak-peak phase difference between the clock embedded in the received data and the local sampling clock.

Fig. 5.19 shows the implementation of the FIFO block. When the downsampled data, \textit{demux\_data}, is 4 bits wide, the FIFO write address remains constant, however, when \textit{data\_size\_5} is asserted, the FIFO write address must increment. Conversely, when \textit{data\_size\_3} is asserted, the FIFO write address must decrement.

In addition to its primary function, the elastic FIFO block also provides the coarse phase output for the CDR’s loop filter. The FIFO write-pointer provides a measure of the phase offset, in UI, between the embedded clock and the local clock. As such, the FIFO write-pointer is used directly as a coarse phase output, used by the phase-tracking portion of the CDR to be described in the next section. In addition, this coarse phase output is used for frequency detection during CDR start-up. During start-up, when the embedded and local clocks are at different frequencies, the constant phase shift will cause the FIFO to repeatedly overflow or underflow. If the two or more sequential overflows or underflows occur, then the frequency detector will output \textit{frequency\_up} or \textit{frequency\_down} pulses, respectively.

![Fig. 5.19 Elastic FIFO](image-url)
For debugging purposes, the FIFO data_out is available as a read-only register in the scan-chain. The coarse phase, frequency_up and frequency_down signals are available for read, and can also be bypassed and manually controlled (written to) through the scan-chain.

5.4.6 DAC/LPF

The coarse phase and frequency up/down signals control current sources connected to the CDR’s loop filter as shown in Fig. 5.20. The coarse phase input (0-31) drives a 5-bit current DAC which is RC filtered to create the VCO control voltage, $V_{cntl}$.

The current output of DAC is $I_{DAC} = (\text{coarsephase} - 15.5)I_{\text{step}}$, where $I_{\text{step}}$ is the current resolution. A coarse phase offset of 15.5 biases the DAC around a half-full (half-empty?) FIFO. Because one coarse phase step equals 1 UI, which equals 1/4 of a clock cycle, the resulting phase detector gain is $K_{pd} = 2I_{\text{step}}/\pi$. An off-chip current reference is used to set $I_{\text{step}}$.

The frequency up/down signals control switches which source or sink a current of $I_{fd}$ directly onto the loop filter capacitor, providing quick frequency lock. Applying the current directly to the capacitor avoids saturation of $V_{cntl}$ due to high currents passing through the filter resistor. The frequency detector current switches are driven by pulse generators that turn a one clock-cycle frequency-detector pulse into a pulse 128 clock cycles in duration. This serves to reduce the magnitude of $I_{fd}$ required for quick frequency lock at start-up. An off-chip current reference is also used to set $I_{fd}$.

![Fig. 5.20 DAC and loop filter](image_url)
The loop filter resistor, $R$, serves as part of the loop filter, and also as an ESD resistor leading to the off-chip loop filter capacitor, $C$. There is a similar ESD resistor in the path from the frequency detector current sources, however, as it plays no part in the loop dynamics, it is not shown.

The loop dynamics of the CDR are those of a typical charge-pump PLL [64], characterized by:

$$\frac{V_{cntl}(s)}{\Phi_{in}(s)} = \frac{s}{K_{osc}} \frac{1 + sRC}{1 + sRC + \frac{s^2C}{K_{pd}K_{osc}}}$$

(5.15)

where $\Phi_{in}(s)$ is the phase of the embedded clock.

The CDR’s VCO was measured and found to have $K_{osc} = 30 \text{Grad/s}/V$, and by design we have $R = 200\Omega$, although it could be increased using an external series-resistor. This gives us two degrees of freedom—the choice of loop characteristic frequency, $\omega_0$, and the quality factor, $Q$. Given the loop equation above, we determine the capacitance, $C$, and the DAC current resolution, $I_{\text{step}}$, by:

$$C = \frac{1}{RQ\omega_0} \text{ and } I_{\text{step}} = \frac{\pi}{2Q^2R^2C^2K_{osc}}$$

A design flaw in the bias circuit of our test chip resulted in a fixed $I_{\text{step}} = 1.2\mu A$, constraining the above choices. For $Q = 0.85$, we are stuck with $\omega_0 = 2\pi \times 0.62$ MHz and $C = 1.5$ nF.

### 5.4.7 BERT

A bit error tester (BERT) is included in the design to verify the functionality and determine the bit-error rate of the system. The BERT processes in parallel the 4-bits per cycle output by the FIFO. Fig. 5.21 shows the structure of the BERT, simplified to a single bit. The BERT recreates the received $2^{31}-1$ PRBS sequence, but instead of feeding its XOR output back to the delay chain input, it compares the result to the input. If the FIFO output is consistent with the PRBS sequence, the bits will match; If not, an error will be flagged and recorded in an 8-bit error counter. The output of this counter is available as a read-only register in the scan-chain.

The above BERT implementation does not directly count errors, as a single error will result in multiple error counts. Two types of errors are possible in the CDR: Single-bit errors, and phase recovery errors. The single-bit error will result in an error count of 3, as it causes an initial error at the comparator input, and then once at each XOR input it reaches. However, single-bit errors are
unlikely to occur without causing additional phase recovery errors. For a single-bit error, the value of the downsampled data bit must be wrong, implying a transition that violates the $2/5$ UI phase change between transitions limit. When this happens, it is likely to cause phase recovery errors in addition to the data error.

When phase recovery errors occur, erroneous bits are added or dropped from the recovered data, generating multiple errors until these bits are flushed from the PRBS delay chain. Due to the interleaved structure the BERT uses to process 4-bits in parallel, each phase recovery error event results in 15 error counts, on average.

### 5.5 Measured Results

A testchip was designed and fabricated in a 0.11μm CMOS process to demonstrate the feasibility and performance of the CDR. Fig. 5.22 shows a die photo of the design, measuring 440μm x 340μm, excluding pads. Two iterations of this design were fabricated, as supply noise problems degraded the BER of the first version. The measured results of both iterations are discussed here.

Fig. 5.23 shows the bit-error rate of our CDR as a function of bit-rate. This figure shows the first design iteration CDR to be functional between 1.9Gb/s and 3.5Gb/s. The low-frequency limit is a result of the tuning range of the VCO, while simulation shows the high-frequency limit to be the result of a critical path in the fine-phase detector. In the mid-band the average BER is $4 \times 10^{-5}$, far worse than commercially acceptable levels of $10^{-12}$. While our measurement results show that the
Fig. 5.22 Testchip die photo

Fig. 5.23 Bit error rate vs. bit-rate
sample-voting scheme described in Section 5.4.2 provides a 15% improvement in BER, these results still fail to meet commercial standards.

During testing, two observations led us to conclude that the source of the poor BER performance was supply noise. First, it was observed that the mid-band BER was roughly $10^{-3}$ with a 50Ω load connected to the recovered clock output, and $4 \times 10^{-5}$ without. As the loading of this clock output only directly affects the final drive stage in the chip’s pad ring, we concluded that supply noise was the cause of the BER degradation. Our second observation related to the JTAG-like scan-chain that is used to configure the CDR, and to monitor the built-in BERT. In our first design iteration it was necessary to continuously shift out the BERT error-counter via the scan-chain to determine the BER. In a result reminiscent of the Heisenberg Uncertainty Principal, this BER measurement was found to be correlated with the frequency of the scan-chain clock. By lowering this frequency from 166MHz to 500KHz, we observed the BER to drop from $4 \times 10^{-5}$ to $4 \times 10^{-6}$. This scan-chain snakes through the entire design, and we suspect supply noise from the scan-chain clock buffers and the chain’s shift-registers as the cause of BER degradation in this case. The first design iteration results presented in Fig. 5.23 were obtained with an unloaded recovered clock output, and a scan-chain clock-rate of 166MHz.

To address the supply noise issues described above, we fabricated a second iteration of our CDR design that was modified to reduce the amount of supply noise, and to better suppress the remaining supply noise. To reduce the amount of generated supply noise we added an enable signal to the recovered clock output, and introduced a method to observe the BERT error-counter without clocking the scan-chain. To better suppress the remaining supply noise we increased the area of on-chip decoupling gate-capacitance from 0.002 mm$^2$ to 1 mm$^2$, and increased the average metal width connecting the circuit to the global power-grid by roughly a factor of three. In addition to these design changes, we also redesigned the testbench PCB used to improve impedance matching in the transmission-lines, as well as reducing the total length of coax-cable between the PCB and the PRBS data-source from over 3’ to 6”.

The measured results of the second design iteration are also shown in Fig. 5.23. The low- and high-frequency limits are still the same as those of the first iteration, however, the BER in the mid-band improves to better than $10^{-11}$, which was the limit of our measurement capability. In this version of the design, there was no observable difference in BER with the sample-voting scheme.
enabled or disabled. This suggests that in the first design iteration the voting scheme was compensating for samples that had been perturbed by supply noise.

Fig. 5.24 shows measured results for the jitter tolerance of the second CDR design iteration, acquired using the procedure outlined in Appendix A, and compares them against behavioural-simulation results and analytical predictions, all in close agreement. The behavioural simulations were performed in Verilog, repeated at each frequency for $10^5$ cycles with increasing jitter amplitude until an error was observed. The analytical prediction is based on an allowable phase change of $\frac{2}{5}$UI over a run length of 32 ($2^{31}$-1 PRBS), a FIFO of 32 bits, and a loop filter with an RC time constant of 300ns. With these parameters, the corner frequency of the jitter tolerance of the phase-tracking CDR exceeds that of the oversampling counterpart, producing a -60dB/decade region where both components provide increasing jitter tolerance with decreasing frequency. Below this frequency range the tolerance slope is due only to the phase-tracking CDR, while above this range it is due only to the oversampling CDR. All results are measured at 2.4Gb/s due to the limited tuning bandwidth of the VCO used to frequency-modulate the clock. The CDR, however, is fully operational from 1.9 to 3.5Gb/s.
To compare the jitter tolerance of the hybrid CDR against that of a purely phase-tracking CDR, we simulated the latter by reducing the FIFO size to 1-bit. The result confirms that the hybrid CDR has a low-frequency jitter tolerance that is 32 times that of the phase-tracking CDR, as shown in Fig. 5.24.

Note that in the above comparison it might be possible to further increase the jitter tolerance of the phase-tracking CDR by increasing its loop bandwidth. This is possible because of the lower loop latency resulting from the simpler design. The semi-blind oversampling CDR replaces the phase detector of the phase tracking CDR with a blind-oversampling CDR. This added complexity results in a higher loop latency, lowering the CDR’s maximum loop bandwidth.

In our semi-blind oversampling CDR, the total loop latency is 4 quarter-rate clock cycles. The first two cycles are consumed in the retiming block, while the last two are consumed by the complex phase detection scheme. If we postulate a simple design having no phase detection latency, then we could reduce the feedback latency in half, potentially doubling the maximum loop bandwidth. This change would result in a doubling of the low-frequency jitter tolerance. However, since our design increases the jitter tolerance by a factor of 32 over a CDR with the same loop filter, this only reduces our improvement factor from 32x to 16x. In addition, the jitter tolerance improvement of the semi-blind oversampling CDR is determined by the depth of its FIFO, and this depth has no effect on the feedback path latency. As a result, any latency penalty attributed to the complex phase detection scheme can be overcome by increasing the FIFO depth. The negative impact resulting from the reduced loop bandwidth is an increase in the CDR’s jitter generation (by a factor of 2 in the above example).

### 5.5.1 Power Consumption

The power consumption of the CDR was measured to be 55 mW from the analog supply and 60 mW from the digital supply, for a total of 115 mW from a 1.2 V supply. The analog supply feeds the VCO, samplers, and current DACs in the loop filter. Of these blocks, the VCO consumes nearly all of the analog power. The digital supply feeds the remaining circuits. The digital power is primarily due to the sense-amp flip-flops and the clock that drives them. As a result, the power consumed by individual digital blocks can be estimated by scaling the total digital power by the fraction of total flip-flops used by that component. Table 5.1 shows this breakdown of digital power.
by component. Note that 1/3 of the power is due to the BERT, which is required only for testing purposes. Potential sources of power reduction are suggested as future work in Section 6.2.

<table>
<thead>
<tr>
<th>component</th>
<th># of flip-flops</th>
<th>% of total</th>
<th>est. power</th>
</tr>
</thead>
<tbody>
<tr>
<td>retiming/voting</td>
<td>30</td>
<td>14%</td>
<td>8 mW</td>
</tr>
<tr>
<td>fine-phase detector</td>
<td>22</td>
<td>10%</td>
<td>6 mW</td>
</tr>
<tr>
<td>downsampler</td>
<td>45</td>
<td>21%</td>
<td>13 mW</td>
</tr>
<tr>
<td>elastic FIFO</td>
<td>45</td>
<td>21%</td>
<td>13 mW</td>
</tr>
<tr>
<td>BERT</td>
<td>74</td>
<td>34%</td>
<td>20 mW</td>
</tr>
<tr>
<td>total</td>
<td>216</td>
<td>100%</td>
<td>60 mW</td>
</tr>
</tbody>
</table>

Table 5.1: Digital power consumption estimates of various CDR blocks

5.6 Summary

This chapter described the design of a CDR that combines the advantages of blind-oversampling with those of traditional phase-tracking CDRs. The recovered clock in a traditional phase-tracking CDR must track the embedded clock within 1/2 UI; otherwise the CDR will recover data from the wrong symbol period, resulting in errors. By embedding a blind-oversampling CDR within a phase-tracking CDR, the recovered clock may deviate in phase from the embedded clock by the jitter tolerance of the blind-oversampling CDR. The recovered clock from the phase-tracking CDR is then used to clock the blind-oversampling CDR, resulting in a semi-blind oversampling CDR having a jitter tolerance equal to the product of the jitter tolerances of phase-tracking and blind-oversampling CDRs. At low jitter frequencies, the jitter tolerance of the blind-oversampling CDR is equal to the size of its FIFO—32 in our implementation. Measured results of our fabricated design show the low-frequency jitter tolerance of the CDR to be 32 times that of a conventional phase-tracking CDR.
Chapter 6

Conclusions and Future Directions

This thesis explores solutions to the problems of equalization and timing recovery for high-speed serial and backplane signaling. The CDR modeling work introduces modifications to the conventional event-driven CDR model to capture the jitter due to power supply noise, and bandwidth limitations in the channel and decision circuits. Our proposed model introduces these jitter sources as discrete-time perturbations of the sampling process occurring in the CDR’s phase detector. Simulations demonstrate that the jitter predicted from the proposed event-driven model is within 12% of that predicted by Hspice simulations, and that the proposed model has a simulation speed that is 1800 times higher.

The adaptation scheme for 4-PAM decision-feedback equalization (DFE) uses direct measurement of the received pulse response in order to adapt the coefficients of the equalization filter. This scheme eliminates the need for additional hardware in the critical path of the DFE. The implementation of this scheme also provides the capability to monitor the system’s pulse response, without the cost of additional hardware. Adaptation and equalization functionality were demonstrated through simulation, while the accuracy of the pulse response monitor was demonstrated through measured results of the fabricated design.

The proposed semi-blind oversampling CDR is a hybrid combination of a phase-tracking and a blind-oversampling CDR. The resulting jitter tolerance is the product of the individual jitter tolerances. This increases the jitter tolerance of a conventional phase-tracking CDR by a factor of 32 at low frequencies, while maintaining the high-frequency jitter tolerance of a 5x blind-oversampling CDR. Measured results of the fabricated design at 2.4Gb/s confirm a 200UL_{pp} jitter tolerance at 200KHz.
6.1 Thesis Contributions

This thesis contributes design tools and solutions for high-speed serial and backplane signaling. Specifically, the contributions of this thesis are:

- A semi-blind oversampling CDR architecture designed for increased jitter tolerance along with a testchip design, fabrication, and measurement. This work was presented at the *IEEE International solid-state circuits conference* (ISSCC) in Feb. 2006 [67]. An expanded version of this work has been submitted for publication to the *IEEE Journal of solid-state circuits* [68].

- An adaptation technique for 4-PAM decision-feedback equalization (DFE) along with a testchip design, fabrication, and measurement. This work was presented at the *IEEE System-On-a-Chip Conference (SOCC)* in Sept. 2004 [69].

- An enhanced modeling framework for event-driven CDR simulation which includes the effects of supply noise, and the limited bandwidth of the channel and decision-circuit. This work has been submitted for publication to the *IEEE Transactions on Circuits and Systems (TCAS)* [70].

6.2 Future Directions

In this section we examine potential areas of improvement in CDR modeling techniques presented in Chapter 3, as well as design improvements in the DFE described in Chapter 4 and the CDR described in Chapter 5.

6.2.1 CDR Modeling

Chapter 3 presented event-driven modeling techniques for CDR simulation. These techniques introduce the ability to model jitter due to power-supply noise and the finite bandwidth of the front-end decision circuits. To do this requires the extraction of modeling parameters from the CDR’s circuit description using circuit-simulation. As currently implemented, however, these simulations are time-consuming to setup and perform, and may be difficult to use by designers not intimately familiar with the techniques.

To make the proposed modeling techniques more broadly available, it would be beneficial to create an infrastructure that could accept a high-level description of a jitter source to be modeled (e.g. 50mV 100MHz sinusoidal supply noise applied to clock buffer). It would then automatically
extract the required modeling parameters from the appropriate schematic netlist and incorporate these parameters into the event-driven model to provide the desired modeling functionality.

6.2.2 Adaptive DFE

Chapter 4 presented an adaptive DFE that includes a phase-recovery scheme. We have identified this phase-recovery scheme as an area of potential improvement. The current scheme adjusts the phase of the DFE’s local clock until it locks onto the zero-crossings in the repeating calibration sequence ‘3 3 -3 -3’. In the absence of ISI, these zero-crossings represent the boundary between symbols in a fully open data eye, and the ideal sampling phase for the receiver is 1/2 UI from these zero-crossings. However, the presence of ISI will shift the timing of these zero-crossings relative to the ideal sampling phase (usually the point of maximum eye-opening). In the current implementation, a manually calibrated phase-offset is added to the zero-crossing phase to determine the sampling phase.

To adaptively determine this phase-offset, a future implementation could incorporate a scheme that identifies the phase of the maximum data-eye opening by identifying the clock phase where the peak amplitude occurs in the received pulse response during the calibration sequence ‘3 0 0 0’. This can be accomplished using the pulse monitoring scheme described in Section 4.5.2. As a side benefit, this approach would eliminate the need for the separate calibration sequence ‘3 3 -3 -3’.

6.2.3 Blind-Oversampling CDR

Two potential areas of design improvement in the CDR proposed in Chapter 5 are the reduction of power consumption, and a faster fine-phase detector.

As described in Section 5.5.1, the CDR’s digital power consumption is dominated by sense-amp flip-flops. Sense-amp flip-flops were required because of the high-speed required in some blocks, particularly the fine-phase detector. However, their exclusive use throughout the design was not necessary, but rather was done for the sake of design expediency. Substantial power and area reductions could be achieved by employing single-ended logic (we used almost exclusively a type of differential pass-gate logic [65]) and dynamic-latches where speed is not a concern. This is particularly true in the downsampler and BERT, where cascaded flip-flops act purely as delay elements and have no logic gates between them.
The speed of the design is currently limited by the fine-phase detector. The implementation of the current fine-phase detector cannot be pipelined because of the feedback path resulting from the previous fine-phase cutting the loop for the current fine-phase detection. However, a look-ahead scheme can overcome this limitation. As shown in Fig. 6.1, five copies of the fine-phase detector of Fig. 5.15 could be implemented, each assuming a different fixed value of the previous fine-phase. At their output a 5-to-1 multiplexer selects the correct result based on the previous fine-phase. In this implementation the feedback path contains only the multiplexer, allowing the fine-phase detector itself to be pipelined as deeply as necessary. In addition, because the loop is already cut in this implementation, two simplifications are made possible: First, the AND gates between stages in Fig. 5.16 can be removed. Second, the bit-widths of the data paths can be tailored to their location, cutting the required adder stages almost in half.
Appendix A

Test Setup for Jitter Tolerance Measurements

This appendix describes the measurement procedure used to test the CDR jitter tolerance as it relates to Section 5.5 of the thesis.

To find the jitter tolerance as a function of frequency, we applied sinusoidal jitter across a range of jitter frequencies (100KHz to 25MHz). At each frequency, we started with a small jitter amplitude, and increased the amplitude until BER threshold was reached. In the first spin of our chip, this threshold was $6 \times 10^{-5}$, roughly twice the jitter free BER. In the second spin of our chip, a threshold of $10^{-9}$ was selected as a compromise between measurement time and measurement accuracy. When the jitter amplitude corresponding to the BER threshold was found, it was then recorded as the jitter tolerance at the current jitter frequency.

The test setup used to identify the threshold jitter amplitude is shown in Fig. A.1. The lower three blocks create a clock with sinusoidal jitter. The block on the bottom right is a signal generator which creates a sinusoidal signal at the jitter frequency. This signal is then applied, through a bias-tee, to the control input of a VCO with a tuning range of 2165MHz to 2650MHz. The bias-tee shifts the common-mode voltage of the jitter source to 9 V, which biases the VCO around a center frequency of 2.4GHz. This jitter-source frequency-modulates the VCO, creating an FM clock which is applied to a $2^{31}$-1 PRBS data generator at the top left of the figure. The data from this block then goes through another bias-tee to shift the data’s common-mode voltage to the 1.05 V expected by the CDR. The signal from the bias-tee is then applied to the CDR input. The CDR was also connected to an Agilent 93k SOC tester which was used to poll the on-chip BERT to identify when the jitter threshold had been reached.

Once it was determined that the jitter threshold was been reached, we needed to convert the voltage amplitude of the jitter source into a jitter amplitude in UI. If the voltage amplitude of the jitter source is $A_j$ and its frequency is $f_j$, then the instantaneous frequency of the VCO is:

\[ f_{vco} = f_c + A_j K_{vco} \sin(2\pi f_j t) \quad (A.1) \]
We know from the VCO datasheet that the VCO gain is $K_{vco} = 37$ MHz/V, and we chose the FM center frequency $f_c = 2.4$ GHz in the middle of its tuning range. Integrating (A.1) to derive the instantaneous phase, we get an expression describing the normalized amplitude of the FM clock signal:

$$x_c(t) = \cos \left[ \int f_{vco} \, dt \right]$$

$$= \cos \left[ 2\pi f_c t + \frac{A_j K_{vco}}{f_j} \cos 2\pi f_j t \right]$$

(A.2)

(A.3)

Because of the equivalence between sinusoidal frequency- and phase-modulation, (A.3) can also be viewed as sinusoidal phase modulation with a phase-amplitude of $A_j K_{vco} / f_j$ [66]. This is equivalent to a peak-to-peak jitter tolerance of:

$$J_{tol} = \frac{\frac{A_j K_{vco}}{\pi f_j}}{UI}$$

(A.4)

---

![Fig. A.1 Jitter tolerance measurement setup](image-url)
References


