ABSTRACT

SU, HSUAN-JUNG. Continuous-Time Fractionally Spaced Equalization and Its Application in Capacitively Coupled Chip-To-Chip Interconnect. (Under the direction of Prof. Paul D. Franzon.)

There is an expanding gap between required bandwidth and achieved bandwidth due to the scaling mismatch between IC technology and the rest of the electronic systems. There are difficulties to increase the number of high-speed I/Os due to physical (material) reasons as well as the constraint of overall power consumption. The power efficiency, thus, has become the new metric for a high-speed I/O system.

Traditionally, AC coupled interconnect (ACCI) relies on the coupling capacitors as passive equalizer to mitigate the frequency-dependent attenuation of the channel while achieving superior power efficiency. However, the lack of flexibility of ACCI makes it less capable of compensating for various range of channel and more susceptible to variation.

This work introduces the Continuous-Time Fractionally Spaced Equalization (CT-FSE), a within-bit (high bandwidth) active equalization scheme, to complement the passive equalizer induced pulse signaling by ACCI. According to the equations derived, two major observations are presented, along with simulation result from a set of Matlab routines. The CT-FSE structure supports bandwidth and power superior to a conventional feed-forward equalizer (FFE) due to the lack of flip-flops and high-speed clock distribution.

The current-mode summation utilized in FFE has disadvantages in power consumption and linearity and thus becomes less attractive when high-speed low-power equalization is required. A Multi-Capacitor (MultiCap) structure is presented to overcome these disadvantages by supporting zero power consumption voltage-mode summation. The MultiCap structure inherits all advantages of ACCI, including high density of 200um pitch
and increased reliability. A set of equations are derived to estimate the useable value of the MultiCap, of which the range is bounded by I/O pitch, receiver sensitivity, and other parameters. The parasitics of this device are proven to be negligible. The transceiver achieved a speed of 5Gb/s with power efficiency comparable to state-of-the-art designs using 90nm technology nodes and beyond.

This work also introduces a buried capacitor (Embedded Cap) for signaling to improve cost, reliability and parasitic inductance. The smaller nominal capacitance value can be compensated by using a CT-FSE receiver equalizer. Tradeoffs in capacitance choice are explained in detail. A nominal capacitance of around 1 pF provides a good choice for the analyzed scenarios.
Continuous-Time Fractionally Spaced Equalization and Its Application in Capacitively Coupled Chip-To-Chip Interconnect

by
Hsuan-Jung Su

A dissertation submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

Electrical Engineering

Raleigh, North Carolina

2012

APPROVED BY:

________________________________  ______________________________
Dr. Paul D. Franzon  Dr. John M. Wilson
Chair of Advisory Committee

________________________________  ______________________________
Dr. W. Rhett Davis  Dr. Joseph Tracy
DEDICATION

In loving memory of my parents,

蘇玉鵬(Su, Yu-Peng) and 張丹青(Chang, Tan-Ching).
BIOGRAPHY

Hsuan-Jung Su (Bruce) was born in Taitung, Taiwan in 1979. He received B.S. degree in Electrical Engineering from National Tsing Hua University, HsinChu, Taiwan in 2001, then he joined the Army of Taiwan until 2003. He received the M.S. degree and started Ph. D. program in Electrical Engineering at North Carolina State University (NCSU) in 2006. And he worked as a research assistant in NCSU until 2010. During the summer of 2006 and 2007, he worked for IBM to develop de-embedding technique and Rambus to investigate noise analysis methodology, respectively. He has been with Rambus since 2010. His interest includes chip-to-chip communication, interconnect/package structures, and signal integrity researches.
ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Paul Franzon, for all the encouragement and inspiration. He always manages to give me very useful insights for matters inside and outside of school. I would like to thank Dr. John Wilson for technical guidance and always keeping a positive attitude toward my research, Dr. Joe Tracy for joining my committee without any hesitation, Dr. Rhett Davis and Dr. Kevin Gard for giving me advices when I first started my M.S. program and helping me make the decision to stay in NCSU for Ph. D. program, and Elaine Hardin at Graduate School for all the help provided administratively.

I would like to thank Dr. Steve Lipa for helping me with research and experiments, Dr. Jian Xu for the idea of embedded capacitor, Dr. John Wilson and Shep Pitts for suggesting me the idea of MultiCap (separately). I enjoy working with Dr. Taeyun Kim, David Winick, Dr. Julie Oh, Dr. Chanyoun Won, Dr. Hoon Seok Kim, Dr. Neil Di. Spigna, Dr. Thor Thorolffson, Akalu Lentiro, and Dr. Ravi Jenkal. Also, I will never forget the nights and days spent with Dr. Yongjin Choi and Evan Erickson in MRC429, especially before tapeout. I would like to thank Pravin Patel and Trey Greer, my mentors during internship at IBM and Rambus, respectively, my managers, Dr. Fred Heaton, Dr. John Eble, and Dr. Lei Luo for motivating me at work and school, and all the colleagues at Rambus for constantly reminding me to complete my degree.

I would like to thank my family and in-laws for giving all the heartwarming mental and financial supports although living very far away (some on the other side of the planet), my sister, Rebecca Su, and my mother-in-law, Grace Chen, for handling everything for me in Taiwan, and all my friends in Taiwan and U.S. for helps and blessings.
Last but not least, I sincerely thank my wife, Alice Yen-Fei Hsu, for always standing by my side, and I simply cannot do this without her.
# TABLE OF CONTENTS

LIST OF FIGURES ............................................................................................................................ ix

LIST OF TABLES ............................................................................................................................. xii

<table>
<thead>
<tr>
<th>Chapter 1</th>
<th>Introduction .................................................................................................................. 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.1 Motivations ........................................................................................................ 3</td>
</tr>
<tr>
<td></td>
<td>1.1.1 Bandwidth Gap .............................................................................................. 3</td>
</tr>
<tr>
<td></td>
<td>1.1.2 Power Efficiency ......................................................................................... 5</td>
</tr>
<tr>
<td></td>
<td>1.1.3 Interconnect Structures ............................................................................ 5</td>
</tr>
<tr>
<td></td>
<td>1.1.4 Equalization in Transceiver Design ..................................................... 7</td>
</tr>
<tr>
<td></td>
<td>1.1.5 Summary ...................................................................................................... 10</td>
</tr>
<tr>
<td></td>
<td>1.2 Original Contribution ..................................................................................... 11</td>
</tr>
<tr>
<td></td>
<td>1.3 Dissertation Overview .................................................................................. 13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 2</th>
<th>Background ............................................................................................................. 14</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.1 AC Coupled Interconnect (ACCI) .................................................................... 14</td>
</tr>
<tr>
<td></td>
<td>2.1.1 Physical Structure ................................................................................... 15</td>
</tr>
<tr>
<td></td>
<td>2.1.2 Schematic Models and Signaling and Method ........................................ 17</td>
</tr>
<tr>
<td></td>
<td>2.2 Equalization Schemes .................................................................................. 20</td>
</tr>
<tr>
<td></td>
<td>2.2.1 Passive Equalizations .............................................................................. 21</td>
</tr>
<tr>
<td></td>
<td>2.2.2 Active Equalization ................................................................................... 23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 3</th>
<th>Continuous-Time Fractionally-Spaced Equalization (CT-FSE) ......................... 28</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.1 Introduction .................................................................................................. 29</td>
</tr>
<tr>
<td></td>
<td>3.1.1 Transversal Equalization ........................................................................ 29</td>
</tr>
<tr>
<td></td>
<td>3.1.2 Fractionally-Spaced Equalization (FSE) ................................................. 30</td>
</tr>
<tr>
<td></td>
<td>3.2 Derivation of the CT-FSE Pair .................................................................... 32</td>
</tr>
<tr>
<td></td>
<td>3.3 Analysis of the CT-FSE Characteristic ..................................................... 36</td>
</tr>
</tbody>
</table>
Chapter 4 Transmitter-Side CT-FSE Utilizing MultiCap Voltage Summer ..........51
  4.1 A Zero-Power Consumption MultiCap Structure ...........................................52
    4.1.1 Parameter Analysis of the MultiCap Structure ..............................54
    4.1.2 Parasitic Capacitance of the MultiCap Structure ...........................59
  4.2 Voltage-Mode CT-FSE Transmitter Utilizing MultiCap Voltage Summer ...62
  4.3 Receiver Design ......................................................................................64
  4.4 Design Flow .............................................................................................66
  4.5 Measurement Results ................................................................................66

Chapter 5 Receiver-Side CT-FSE with Embedded Capacitor ..........................73
  5.1 Embedded Capacitor and Channel Model ..................................................73
    5.1.1 Analysis of Embedded Capacitor and Signaling Modes ....................76
    5.1.2 Stub Length of the ACCI with Embedded Capacitor ............81
  5.2 Transmitter Design ..................................................................................82
  5.3 Receiver Design ......................................................................................83
    5.3.1 Current-Mode CT-FSE .................................................................84
    5.3.2 High-Speed Latch ..........................................................................87
  5.4 Design Flow .............................................................................................87
  5.5 Simulation Results ...................................................................................89

Chapter 6 Conclusions and Future Works ......................................................93
  6.1 Conclusions ............................................................................................93
  6.2 Future Works ..........................................................................................96
    6.2.1 CT-FSE with Adjustable Delay $\tau$ ..................................................96
    6.2.2 CT-FSE with Non-Uniform Delay ....................................................97
    6.2.3 Method to Design High-Speed Latch with Adjustable Hysteresis 98
Appendix A  CT-FSE Synthesizer: Instructions and Matlab Programs (.m files).. 106

A.1 Frequency-Domain Analysis then Synthesis: Sweep Filter Corner Freq. 106
   A.1.1 CT_FSE_Sweep_Filter.m ................................................................. 107

A.2 Frequency-Domain Analysis then Synthesis: Sweep Tdelay ................. 110
   A.2.1 CT_FSE_Sweep_Tdelay.m ............................................................... 111

A.3 Time-Domain Parameter Optimization (for Pulse Signaling) ............ 114
   A.3.1 Output files .................................................................................. 117
   A.3.2 CT_FSE_Optimizer_Pulse.m ......................................................... 118
   A.3.3 smartlatch_new.m ................................................................. 125

A.4 Time-Domain Parameter Optimization (for NRZ Signaling) ............ 128
   A.4.1 Output files .................................................................................. 129
   A.4.2 GenerateChannelModel.m ......................................................... 130
   A.4.3 CT_FSE_Optimizer_NRZ.m ......................................................... 135
   A.4.4 smartlatch_NRZ.m ................................................................. 145
   A.4.5 LFSR.m ...................................................................................... 148
   A.4.6 Bit2Waveform.m ....................................................................... 149
   A.4.7 Delay_5Taps.m ......................................................................... 150
LIST OF FIGURES

Figure 1.1. CMOS scaling trends and bandwidth gap [5]. ................................................................. 4
Figure 1.2. Cross-section of a capacitive coupling interface with high-k under-fill [15].... 5
Figure 1.3. Top view of a high density ACCI [20]............................................................................ 6
Figure 1.4. S21 plot of an ideal PCB microstrip with various lengths. ........................................... 8
Figure 1.5. Comparing time-domain waveforms of two sets of channel input/output to demonstrate the effect of ISI ...................................................................................... 8
Figure 1.6. Block diagram of Feed-Forward Equalization ................................................................. 10
Figure 1.7. Comparing various criteria between different equalization schemes and the expectation of this work. ................................................................................................ 11
Figure 2.1. Cross-section of a capacitive coupling interface with high-k under-fill [20].... 15
Figure 2.2. Schematic showing different configurations of CCI and LCI....................................... 17
Figure 2.3. ACCI with small coupling capacitors generates pulse signal at RX input .... 18
Figure 2.4. Equalization inserted into high-speed link to boost the high frequency signaling component. ........................................................................................................... 20
Figure 2.5. Schematic of a R-C Filter ............................................................................................. 21
Figure 2.6. Schematic of a T-Junction (RLC) Filter ......................................................................... 22
Figure 2.7. Schematic of (Left) R-L filter and (Right) R-TLine filter. ........................................... 22
Figure 2.8. Schematic of a continuous-time linear equalizer (CTLE) ........................................... 24
Figure 2.9. Block diagram of the FFE (or FIR) structure .................................................................. 25
Figure 2.10. Block diagram of the Decision Feedback Equalization (DFE) structure .... 26
Figure 2.11. Different ways of combining equalizers. ................................................................. 27
Figure 3.1. Block diagram of the conventional transversal filter structure ........................................ 29
Figure 3.2. Block diagram of the CT-FSE ....................................................................................... 32
Figure 3.3. Similarity between the CT-FSE Pair and the Fourier Series in complex form. 35
Figure 3.4. The effects of $k$ and $\tau$ on when linearly combining $e^{-j\omega k\tau}$ in Eq. (3.7) ....... 35
Figure 3.5. Procedure of synthesizing $H_N(\omega)$ ...................................................................... 37
Figure 3.6. Swept $H_1(\omega)$ corner frequency from 1 to 5GHz and follow procedure in Figure 3.5 to synthesize (Left) ideal LPF and (Right) ideal HPF .................................................. 38
Figure 3.7. Swept $\tau$ from 50 to 200ps and follow procedure in Figure 3.5 to synthesize (Left) ideal LPF and (Right) ideal HPF. ................................................................. 39
Figure 3.8. Different value of \( \tau \) changes the available time point (indicated by arrows) upon which equalization is applied. ................................................................. 40

Figure 3.9. The CT-FSE synthesizer implemented in Matlab. (Upper) Frequency-domain behavioral simulation. (Lower) Time-domain parameter optimization................................................................. 42

Figure 3.10. CT-FSE synthesizer optimizes \( \{a_k\} \) to achieve open eyes when \( \tau \) varies. .... 44

Figure 3.11. Cross domain operation to verify the optimized \( k, a_k, \) and \( \tau \) are valid. ......... 45

Figure 3.12. \( S_{21} \) of various channel lengths................................................................. 46

Figure 3.13. CT-FSE Inputs (Channel outputs) vs. CT-FSE Outputs (Time-Domain Optimizer outputs). ............................................................................................................. 47

Figure 3.14. Target \( H_d(\omega) (=1/ S_{21}) \) vs. Synthesized \( H_N(\omega) \) according to \( k, a_k, \) and \( \tau \) from Time-Domain Optimizer ................................................................. 48

Figure 3.15. System overall frequency response = \( S_{21} \times H_N(\omega) \) ........................................... 50

Figure 4.1. The cross-section showing MultiCap replace one pad in the ACCI structure. . 52

Figure 4.2. Schematic symbol of MultiCap and its usage as a voltage summer. ............... 52

Figure 4.3. TX-side pre-emphasis realized by CT-FSE utilizing MultiCap voltage summer. ................................................................. 53

Figure 4.4. Top view of the MultiCap structure includes a matrix of silicon top metal plates and a shared big bottom plate on package. ................................................... 55

Figure 4.5. The parasitic capacitances between a pair of \( C_C \) and ground............................... 59

Figure 4.6. Schematic model for the pair of \( C_C \) shown in Figure 4.5................................. 60

Figure 4.7. Calculation results using (4.9) vs. Sonnet simulation result .............................. 61

Figure 4.8. All-digital complementary CT-FSE TX............................................................... 62

Figure 4.9. Detail of a slice of TX gain and driver stages. .................................................... 63

Figure 4.10. The MultiCap structure implemented by on-chip VNCAP............................... 63

Figure 4.11. Detail of high-speed latch RX ........................................................................... 64

Figure 4.12. Block diagram of the design flow for CT-FSE TX utilizing MultiCap voltage summer .............................................................................................................. 65

Figure 4.13. Die photo with TRX area and PCB photo with channel setup............................ 67

Figure 4.14. PCB cross section and the real measurement setup on probe station............... 68

Figure 4.15. The overall test setup when displaying waveform on Oscilloscope. ............... 69

Figure 4.16. The overall test setup when measuring for BER ............................................. 70

Figure 4.17. Comparing the RX inputs before and after the TX-Side CT-FSE is enabled. .. 71
Figure 4.18. RX output: recovered 5Gb/s NRZ bit-stream .......................................................... 71
Figure 5.1. Structure of package cross-section, including embedded capacitors for both power supply decoupling and ACCI signaling .......................................................... 74
Figure 5.2. Simplified schematic of the ACCI channel with embedded capacitor .................. 75
Figure 5.3. Simulated S21 plots of the channel shown in Figure 5.2 ........................................ 76
Figure 5.4. The center frequency and bandwidth of the channel-equivalent BPF change as Cc increases ................................................................................................................. 77
Figure 5.5. The step response of the ACCI channel with various Cc. (Bottom right) The way to calculate time for maximum run length ................................................................. 79
Figure 5.6. Simulated S21 plots of the channels with various stub length. The difference in peak is around 2dB ......................................................................................................... 82
Figure 5.7. TX is made of a pair of complementary progressively-sized inverter chains ... 83
Figure 5.8. RX block diagram ............................................................................................... 83
Figure 5.9. Block diagram of the RX CT-FSE ....................................................................... 84
Figure 5.10. Matlab simulation results shows the open 5Gb/s eyediagrams of the CT-FSE with tap delay (τ) varies from 75ps to 225ps ................................................................. 85
Figure 5.11. Detail of the building blocks in RX CT-FSE ........................................................ 86
Figure 5.12. The common-mode feedback circuit utilizing Gilbert cell to maintain CT-FSE output common-mode voltage .......................................................................................... 86
Figure 5.13. The high-speed latch is a non-clocked regenerative comparator with common-source pre-amplifier ............................................................................................... 87
Figure 5.14. Design flow of the system (RX CT-FSE with embedded capacitor) ................. 88
Figure 5.15. Time-domain waveforms of the delayed RX inputs .......................................... 89
Figure 5.16. Comparison of the waveforms before and after CT-FSE is turned ON ............. 89
Figure 5.17. The CT-FSE eliminate most ISI in RX input, and the latch recovers the signal back to 3.33Gb/s NRZ signals ................................................................. 90
Figure 5.18. Detail current and total power consumption of each RX block ......................... 91
Figure 6.1. Comparison of power efficiency (mW/Gb/s) and topology .................................. 94
Figure 6.2. Block diagram of the CT-FSE with adjustable delay elements ............................ 96
Figure 6.3. Block diagram of the CT-FSE with non-uniform delay τk .................................... 97
Figure 6.4. The input/output voltage transfer function of a latch with adjustable hysteresis ................................................................................................................................. 98
LIST OF TABLES

Table 3.1. Optimized tap weights, $a_k$, and eye opening for 5 different channel length. ....... 48
Table 4.1. Comparison of Performance. ................................................................................ 72
Table 5.1. Investigation of available materials for embedded capacitor. ......................... 74
Table 5.2. Timing and Circuit Requirement for Various $C_C$. ........................................... 80
Table 5.3. Comparison of the TRX Performance. ................................................................. 92
Chapter 1

Introduction

Since the invention of integrated circuit (IC) technology in the 1950’s, there have been significant progresses in CMOS manufacturing, packaging, testing, material, and system design. Observing from higher level, the progresses in the electronic components enable rapidly improvement in high-performance electronic products, such as gaming platform, portable devices, server, etc. In every new generation, these products are expected to have exponential growth in performance.

One of the most critical factors for the exponential performance improvement is the communication between electronic components. For instance, as much information as a microprocessor can process in a short amount of time, or as much computing power as a microprocessor can have, it still needs proper I/Os to exchange information with the outside world in the same short amount of time. Otherwise, the microprocessor would be idle waiting bits to be transmitted and received. In other words, the performance of a system would ultimately be limited by the I/O bandwidth.
Unfortunately, the issues of I/O bandwidth can easily be solved were it not for the system power limitation. Assume there is no issue for heat dissipation; for the gaming platform, the system power limit is the maximum power rating for a household wall socket; for a portable device, the time lapse between each battery recharge; and for the server, the total available output in megawatt (MW) provided by a dedicated power plant. To overcome the performance bottleneck in system I/Os while consuming as less power as possible, this dissertation explores solutions to be applicable on majority of high-speed chip-to-chip communication. The following chapters show some design considerations involved with circuit and interconnect co-design, and the combination of different concept in each field resulting in state-of-the-art power efficiency.

Due to the multidisciplinary nature of the chip-to-chip communication, the angle of researches in this dissertation is twofold: interconnect structure and transceiver (TRX) design. One cannot only optimize the interconnect structure while expecting the TRX to remain unchanged. On the other hand, a different topology for TRX design would affect the compatibility with the interconnect structure. That is, a successful chip-to-chip communication requires the combination and tradeoff between good power efficiency for the TRX and a high-density scalable interconnect structure.
1.1 Motivations

1.1.1 Bandwidth Gap

In 1965, Moore predicted that the number of transistor counts in a single wafer would double every year [1] (that was more from the viewpoint of cost per component dropping exponentially). Later this so called “Moore’s Law” was altered for better accuracy to “doubling transistor counts in a die for every two years” or “doubling performance per chip every 18 months if the performance of the transistor itself is counted”. In fact, almost 30 years later, Moore himself compared the history in transistor counts per die against the prediction to show the reality lagging mildly [2]. Nonetheless, the Moore’s Law has governed the development of the modern CMOS technology. The semiconductor foundries and the International Technology Roadmap for Semiconductors (ITRS) [3] try to follow the exponential performance growth described by Moore’s Law. For each newer generation, it is expected that the chip performance would increase exponentially.

The number of interconnects per die, in contrast, has a different story. As predicted by Rent in the 1960’s and described by Landman and Russo in 1971 [4], the number of pins with regarding to the number of logic gates should also follow an exponential law, with the exponent related to the level of complexity of the logic: Number of pins = K × Gates^r. As shown in Figure 1.1, the blue dots on top shows the trend of performance per die (Gates × GHz), the red dash line in the middle shows the prediction of required bandwidth per die (Pins × GHz) assuming K=0.82 and the exponent r=0.45 for early microprocessors, and the
green dots at the bottom shows the history of real bandwidth. It is clear that there is an expanding gap between required bandwidth and achieved bandwidth.

Figure 1.1. CMOS scaling trends and bandwidth gap [5].

During the past decade, more and more literatures in high-speed chip-to-chip communication (or serial link)\(^1\) paid attention to this issue of bandwidth gap. For example, Kuroda and Miura in 2006 [6] addressed this issue and compared the existing trend with the prediction by Rent’s rule (with slightly modified K and r). They also pointed out the gap between real annual bandwidth growth and required annual growth of is 28% vs. 45%. Also for example, Palermo et al [7] in 2008 mentioned ITRS predicted that that trend of aggregate I/O bandwidth (per pin data rate × I/O number) was required to grow exponentially.

\(^1\) Throughout this dissertation, “high-speed chip-to-chip communication” and “serial link” are used interchangeably. The term “serial link” was first used in the field of backplane communication but a number of literatures in high-speed chip-to-chip communication imply the two terms are equivalent.
1.1.2 Power Efficiency

The newer update of ITRS in 2009 [3] maintains the same point of view on the increasing I/O bandwidth gap while stating the difficulties to increase the number of high-speed I/Os due to the constraint of overall power consumption. It also mentioned the recent trend of integrating memory controller inside microprocessors further exacerbate the bandwidth gap since the memory interface is going to have higher and higher bandwidth requirement (DDR4 is predicted to hit 4Gbps [9]). Therefore, we know that the real bottleneck in chip-to-chip communication, instead of just the I/O bandwidth, is the I/O bandwidth per Watt. Poulton et al [8] presented one of the first serial link designs that targeted at better power efficiency (mW/Gb/s) instead of pure high bandwidth. In terms of power efficiency, they achieved performance an order of magnitude better than the rest of the high-speed serial link designs. To satisfy the requirement of better power efficiency, a serial link design needs not only support high bandwidth but also consume least amount of power possible.

1.1.3 Interconnect Structures

Figure 1.2. Cross-section of a capacitive coupling interface with high-k under-fill [15].
Figure 1.2 shows the cross-section of a capacitively coupled interconnect (CCI) structure with high-k dielectric underfill. Luo et al. [21] demonstrated the possibility to apply CCI with PCB channels up to 45cm, besides enabling high per-pin data rate.

![Diagram of CCI structure](image)

Figure 1.3. Top view of a high density ACCI [20].

As shown in Figure 1.3, Wilson et al. [20] presented the possible setup for ACCI to support high-density high-speed I/O at the peripheral of the die, low inductance power/ground distribution via the centrally located DC buried bumps, and the underfill inlet from the corners of the die. The high-k underfill material was developed by Kim et al. [50] to reduce the area requirement for the capacitor arrays, relax the constraints on the dielectric, and provide stress relief between die and substrate. This leads to potential improvement in yield, and the increase in long-term reliability. Therefore, ACCI not only supports high aggregate bandwidth by enabling high-density I/O structure and high per-pin data rate, but it also shows some physical advantages over other interconnect structures.

As noted by Luo et al. [19], the main difference between ACCI and traditional capacitive coupling signaling is the size of capacitors. Traditional coupling capacitors have size at least
1000 times of those in ACCI. The series capacitors in ACCI are small because they function as passive equalizers. When pushed toward highest bandwidth possible, as described in detail later in section 2.1.2, the series capacitance could become so small that the RX input signal amplitude drops below the receiver (RX) sensitivity threshold. Therefore, the only solution to this shortcoming is to devise a new I/O structure that inherit all the advantages of ACCI and supports a novel active equalization scheme to complement ACCI.

1.1.4 Equalization in Transceiver Design

The main challenge for a transceiver (TRX) to support high data rate is the ability to compensate for the frequency-dependent attenuation of the channel. A typical PCB channel has a low-pass response due to the skin effect ($\propto f^{1/2}$) and dielectric loss ($\propto f$) [10]. As Dally and Poulton pointed out in [37], the frequency-dependent attenuation causes inter-symbol interference (ISI). Take the microstrip channel shown in Figure 1.4 for example, with Nyquist frequency of 5GHz, the S$_{21}$ plot shows the attenuation difference between DC and 5GHz could increase from 3.3dB for a 20cm microstrip line to 16.6dB for a 100cm one. For backplane applications, such as the tens of inches backplane channel in [13], the loss for 6Gb/s bit stream (Nyquist rate of 3GHz) could range from 20dB to over 30dB. The time-domain plot in Figure 1.5 compares the inputs/outputs between a short channel with less high-frequency attenuation (left) and a long channel with more high-frequency attenuation (right). It further emphasizes the effect of inter-symbol interference (ISI) caused by the long tail of consecutive bits. In addition to the various channel length, hence wide range of high
frequency loss, the presence of process-voltage-temperature (PVT) variation could also significantly affect the TRX performance.

Figure 1.4. $S_{21}$ plot of an ideal PCB microstrip with various lengths.

Figure 1.5. Comparing time-domain waveforms of two sets of channel input/output to demonstrate the effect of ISI.
To mitigate the problem of frequency-dependent attenuation (or ISI) and PVT variation, the TRX requires equalizations to add flexibility to the system. There are two categories of equalization: 1.) passive equalization utilizing passive components as filters, and 2.) active equalization consisting of active circuitry to synthesize the filter response.

Passive equalization schemes, benefiting from simple circuits, enable high-speed serial link with low power consumption. Luo et al. [19] and Wilson et al. [20] demonstrated that AC Coupled Interconnect (ACCI) has one of the best power efficiency (1.97mW/Gb/s) among various passive equalization schemes. However, there are two major problems associated with passive equalization schemes: flexibility and limited channel length. The passive equalizer in ACCI – capacitors, for example, are formed between silicon and package; thus, the capacitors are fixed during manufacturing process, after which there is no adjustability on the capacitor value, hence the fixed amount of equalization. Further, passive equalization can only support PCB channel up to certain length (about 30cm for ACCI [21]). Although the use of a smaller capacitor can shift the filter peak toward higher frequency and compensate more high-frequency attenuation from longer channel, it also decreases the signal swing. Therefore, there is a limit beyond which the signal magnitude will be attenuated below the noise floor of the system; i.e., the system is swing limited.

Active equalization utilizes the transistor (amplifier) characteristic or the digital filtering technique to improve the flexibility of the TRX, compensate for the frequency dependent loss, and extend the upper limit of the channel length. The most commonly used active equalization scheme is a Finite Impulse Response (FIR) filter, or a feed-forward equalizer.
(FFE, Figure 1.6), such as those used in [24] and [25]. The FFE structure is a good compromise between complexity, power consumption, and equalization performance.

Although the FFE structure is straightforward and flexible, unfortunately, this scheme usually dissipates a significant amount of power due to the use of 1.) current summation block (combined with transmit driver), 2.) high-speed flip-flops, and 3.) high-frequency clock distribution network. Therefore, a novel active equalization scheme is required to keep the flexibility of the TRX while eliminating these three issues to save power consumption.

![Figure 1.6. Block diagram of Feed-Forward Equalization.](image)

### 1.1.5 Summary

In order to close the gap between required aggregate bandwidth and achieved bandwidth while supporting low power operation, we need innovation in equalization scheme as well as interconnect structure. The overall system should be able to integrate interconnect and equalization seamlessly so that, as shown in Figure 1.7, the presented work has performance
better than a traditional FFE and power efficiency close to passive equalization. Notice the
detail of each equalization scheme is described in section 2.2.

Figure 1.7. Comparing various criteria between different equalization schemes and the
expectation of this work.

1.2 Original Contribution

The research in this dissertation achieves further improvement in the power efficiency
and extends the capability of the passive equalization in ACCI by complementing it with a
within-bit high-bandwidth active equalization. The intention of the dissertation is to
demonstrate a consistent methodology for system design, by deducing analytical equations
for the fundamental design parameters first, and then build components on top of the
equations. The original contributions in this dissertation include:
1. The CT-FSE theory to mathematically explain why and how feed-forward equalization (FFE) with continuous-time delay would operate. Chapter 3 presents the derivation of the equations that describes the CT-FSE theory. The equations can be generally applied to a FFE with any continuous-time delay, in addition to the conventional unit-delay case. Appendix A shows a group of Malab programs developed to verify the theory. Integrating CT-FSE structure into high-speed transceiver design would eliminate the use of high-speed clock distribution and save power.

2. A novel multi-capacitor (MultiCap) structure to enable zero power consumption in voltage summation for voltage-mode CT-FSE (or any FFE) in transmitter (TX). Chapter 4 analyzes the MultiCap structure and provides equations for searching the usable range of dimension and material parameters. The combination of MultiCap and voltage-mode CT-FSE solves several problems of conventional current-mode FFE structure and results in power efficiency comparable to TX designs implemented in more advanced CMOS process nodes.

3. A method to combine the passive equalization realized by the embedded capacitor in printed circuit board (PCB) and the active equalization realized by current-mode CT-FSE at the RX end. The capacitor embedded in PCB has potential to improve cost, reliability and parasitic inductance. Chapter 5 implements current-mode CT-FSE compatible with ACCI pulse signaling, analyzes the embedded capacitor structure for usable value, and investigates the material of embedded capacitor to ensure compatibility with developing materials in backplane communication channel in the short-term future.
1.3 Dissertation Overview

Chapter 2 reviews the backgrounds of two topics separately, including ACCI and equalization schemes implemented in high-speed link design. Since both topics are very broad in nature, this dissertation only covers the qualitative introduction and comparison with proper references. Chapter 3 derives the theory of CT-FSE in equations and proves the analysis with Matlab programs (source codes in Appendix A). Chapter 4 presents a novel Multi-Capacitor (MultiCap) structure to be utilized as the voltage summation block when combined with TX-side CT-FSE. Chapter 5 demonstrates the idea of incorporating active equalization (CT-FSE) with passive equalization (PCB-embedded capacitor). Both Chapter 4 and Chapter 5 have design flow sections dedicated to show how the overall system is put together, including navigating through different domains of the multi-disciplinary work. Finally, Chapter 6 concludes the dissertation.
Chapter 2

Background

2.1 AC Coupled Interconnect (ACCI)

ACCI was introduced in the early 1990’s to address the problem of limited space and aggregate bandwidth for I/Os on a multichip module (MCM) [14]. In early 2000’s, Mick et al. [17] demonstrated the feasibility of using buried bump in flip-chip configuration to realize ACCI. The possibility of high density interconnect with high structural reliability were presented later by Wilson et al. [20]. And the power efficient transceiver design is also by Luo et al. [19]. Therefore, this technology is also well known for its versatility in connector designs and superior power efficiency.

There are two aspects of ACCI (or CCI) that makes it more attractive than conventional chip-to-chip channel:

- Physical Structure
- Signaling and Passive Equalization
2.1.1 Physical Structure

The fundamental concept of ACCI is to replace the DC connections (that transfer AC signals) of typical chip I/Os and inserts non-contacting passive components in series with the chip-to-chip channel. Since there is no hard requirement on where and how those connections should be replaced, there are a variety of structures that utilize the concept of ACCI. A more complete description regarding the ACCI technology in more detail can be found in the dissertation by Xu [15], but in general ACCI can be broken down into two major categories:

- Capacitively-Coupled Interconnect (CCI): As presented by Mick et al. [18], coupling capacitors are created between die and substrate top metal in a flip-chip environment. As shown in Figure 2.1, one plate of the capacitor is the silicon top metal, and the other plate of the capacitor is located on the substrate, which could be package or PCB. The signal is coupled through the electric field (or voltage) from one plate to another. The dielectric underfill can be high-k material with
thermal-stress relief characteristic [50] to increase the long-term reliability. It is found that CCI is capable of signaling at higher speed [19] due to less parasitic in the interconnect structure as well as low power due to the nature benefit of voltage-mode signaling.

- Inductively-Coupled Interconnect (LCI): This configuration is similar to CCI with the capacitors replaced with coupled coil (inductors). Between the two inductors, the signal is transferred through the magnetic field (or current). Xu [15] found that LCI is capable of transfer signal through larger gaps (tens of µm as opposed to single digit of µm in CCI) due to the penetrating magnetic field as well. LCI also tends to consume more power due to the nature of the current-mode signaling. For more detail related to LCI, the dissertation by Chandrasekar [16] can be referred.

Depending on the scenario in which the interconnect structure is implemented in, there are other applications for CCI and LCI. For instance, there have been examples of ACCI utilized in 3-dimensional integrated circuit (3DIC) as presented by Davis et al. [31]. Also, there are advanced variations for CCI and LCI in connectors, including daughtercard connectors proposed by Chandrasekar [16] and sockets proposed by Xu [15]. This dissertation only focuses the application related to CCI.
2.1.2 Schematic Models and Signaling and Method

Several generalized schematic models for the ACCI system are shown in Figure 2.2, including one capacitor without channel, two capacitors on two side of the channel, one pair of inductors without channel, and two pairs of inductors on two side of the channel. The requirements for signaling imply different designs of TRX circuitry and different ways of connecting to ACCI; i.e., voltage-mode signaling for CCI and current-mode signaling for LCI. As long as the signal integrity analysis satisfies the design budget, it is the choices of designers to determine either single-ended or differential signaling is more suitable for the specific application.
A most significant difference between ACCI and other type of interconnect, besides the structure, is the signaling mode. The passive components of small value in series with the channel in ACCI normally induce pulse signaling. The effect is illustrated very well by Salzman et al. [14] and Luo et al [19], and we redraw the concept in Figure 2.3. The voltage coupled across the capacitors would be differentiated, leaving only the edge information as pulses at the RX input. As opposed to NRZ (non-return-to-zero) signaling, the pulse signaling have zero DC component (due to AC coupled) and have positive pulse when there is a low-to-high transition at TX output and negative pulse when there is a high-to-low transition. This time-domain behavior implies that the data is carried by the edge information. Therefore, the data rate can be pushed higher if each edge is narrower (or the narrower pulses). It also verifies the effect of passive equalization in ACCI (further detail in section 2.2.1).

The only way to make pulses narrower in ACCI is to decrease the size of series capacitors or inductors, which also has a side effect of decreasing the pulse amplitude. This leads to a fundamental shortcoming of passive equalization in ACCI, or in any interconnect.
technologies that uses off-chip passive components to perform equalization: the lack of flexibility (or less degree of freedom).

As Luo et al. presented in [19], the high-passing coupling capacitors need to matches to the channel for extreme performance. When pushed toward highest bandwidth possible, the more attenuation of the channel, the smaller the series capacitance or inductance need to be. In other words, the appropriate value of the AC components depends on the channel attenuation. To compensate for a very long channel with large amount of frequency-dependent attenuation, the ACCI system would require series capacitance or inductance so small that the signal amplitude goes below the RX sensitivity threshold. In other words, it is possible to have a perfect open eyediagram at the RX input, but the RX latch is not sensitive enough to recover the pulse signal.

To add the flexibility (or more degrees of freedom) to the ACCI technology, proper equalization scheme needs to be adopted. The nature of the pulse signaling in ACCI leaves only edge information at the RX input. Thus, it is more suitable to have the equalization operating within one bit period. After a short introduction to general equalization schemes in section 2.2, this dissertation presents a novel CT-FSE scheme in Chapter 3 to be combined with ACCI structures in later chapters.
2.2 Equalization Schemes

The bandwidth bottleneck for a high-speed chip-to-chip link is usually the channel. As described in section 1.1.4, the frequency-dependent attenuation of the channel causes ISI and high-speed serial links requires equalizations of some sort. The upper half of Figure 2.4 indicates the two possible locations to insert the equalization block: TX-side or RX-side. The frequency domain plot at the lower half of Figure 2.4 shows that high-speed link utilizes pre-emphasis and/or de-emphasis to obtain an equalized channel with flat response within Nyquist frequency. Equalization in general is able to add flexibility to a chip-to-chip communication system, compensate for PVT variations, and increase system bandwidth. This
chapter separates equalization schemes into two categories: passive equalization and active equalization.

2.2.1 Passive Equalizations

Passive equalization achieves filtering function by utilizing passive components, such as capacitors, inductors, transmission line, or any combination of them.

Figure 2.5. Schematic of a R-C Filter.

R-C filter: As shown in Figure 2.5, it consists of a resistor in parallel with a capacitor. It is the most basic implementation for passive equalization. Shin et al [32] demonstrated an on-package discrete implementation of the R-C filter and it supported data rate up to 10Gbps. Although ACCI is a special case of this filter (R = ∞), R-C filter cannot be integrated with CCI since it is less likely to manufacture the resistor between silicon die and substrate as a typical CCI structure.
T-Junction (R-L-C): As shown in Figure 2.6, Sun et al [33] presented a “T” configuration that consists of resistors, inductors, and capacitors to form a high-pass filter (HPF). The differential version of this filter supported data rate up to 20Gbps.

R-L Termination: As shown in Figure 2.7, Guo et al [34] presented a method to insert an inductor on the RX termination to deliberately cause mismatch and thus achieve equalization. In the same literature they also presented a variation that replaces the inductor with a section of high-impedance transmission. It is demonstrated to support data rate of 5Gbps.
ACCI (Coupled-C/L): As shown in Figure 2.2, ACCI use series capacitors (or transformers) as signal filter. More detail is already described in section 2.2.1. Note the major difference between ACCI and other passive equalization scheme are the small value of the passive component and the loss of DC information due to lack of DC path. The former generates pulse signaling and the later implies a requirement of signaling bias generation built into the RX circuitry.

All passive equalization schemes have one shortcoming in common: the lack of flexibility. To complement the passive equalization that already performed de-emphasis on the low frequency component, an active equalization only needs to boost the really high frequency component. This relaxes the sharpness requirement of the filter, but the bandwidth requirement of the filter is still as high as the case without passive equalization. Take the pulse signaling in ACCI for example, at RX input, most of the frequency-dependent attenuation induced ISI is mitigated, leaving only minor “tail” that is mostly within one bit period for the active equalization to deal with. Therefore, the bandwidth requirement for active equalization in ACCI is even higher without the effect of the coupling capacitors.

2.2.2 Active Equalization

Although passive equalization is simple and does not consume power, sometimes, a high-speed serial link needs to sacrifice in system complexity and power consumption for better flexibility and/or higher performance. As the fundamental limitation of ACCI described at the end of section 2.1.2, in order to improve system performance, a serial link requires an
active equalization that handles within-bit ISI (later in Chapter 3, the equations proves the within-bit capability is equivalent to high-bandwidth). There are many different topologies of active equalizations, among which, some utilizes the knowledge in analog/RF circuit design, and others adapt the digital filter structure from digital signal processing (DSP).

![Schematic of a continuous-time linear equalizer (CTLE).](image)

As shown in Figure 2.8, continuous-time linear equalization (CTLE), or sometime simply linear equalizer (LE), is a capacitive source-degenerated differential amplifier. At low frequency, the two tail current work independently and the gain is low. At high frequency, the capacitor acts as a short and the structure work as a differential amplifier work boost the gain. The tunability can be achieved by adjusting the value of the resistor and/or capacitor. This circuit can be fairly efficient since it was used by Palmer et al. [51], one of the most power efficient serial link designs. Lee [22] presented equations to describe the transfer function of LE. He also proposed an improved version with inductor load that supports 20Gbps data rate.
In the domain of digital signal processing, the finite-impulse response (FIR) is one of the most well-known filters. As shown in Figure 2.9, the $x[n]$ and $y[n]$ are the input and output signals, respectively. And $a_0$ to $a_3$ represent the tap weights. The output, $y[n]$, is a weighted sum of all the delayed version of input, $x[n]$. The unit delay is normally implemented with clocked flip-flop. Dally and Poulton [37] integrated the FIR algorithm into a current-mode transmitter and allow for pre-emphasis right on the high-speed driver node. This topology is very straightforward and linear. I.e., it will boost/equalize signal, ISI, and noise. This topology is at least as useful as LE since they both have more linear and predictable default filter response (some adaptive algorithm requires a default setting that works without calibration). FFE is more flexible and potentially higher bandwidth than LE when it has more taps (generally more than 2 taps). Therefore, it can be seen in many literatures that involves with more practical researches, such as [24] and [25]. However, FFE consumes a lot of power because the tap-delay-weighted chain needs high-speed flip-flop, high-speed clock distribution, and the use of current-mode logic (CML) summation. The detail of the shortcomings is described in the beginning of Chapter 4.
Figure 2.10. Block diagram of the Decision Feedback Equalization (DFE) structure.

The DFE, shown in Figure 2.10, has the reverse mechanism as FFE. As described by Belfiore and Park [12], assume the output of the summation block is the original signal without any ISI; the tap-delay-weighted chain should emulate the negated channel response and feedback this information (which is actually the residue of the ISI added onto the following bits) to the summer. Therefore, when the following bit arrive at RX input, the ISI on x(t) can be subtracted out at the summer state. This topology is nonlinear – it will not boost the noise added to x(t) because the comparator and the flip-flop (which are nonlinear themselves) saturate the signal all the way to supply rail voltage level. However, the DFE structure consumes even more power than FFE due to tap-delay-weighted chain (just like FFE), besides the auxiliary circuitry to support the DFE operation. In addition, DFE has a known issue of error propagation. I.e., when an error occurs, the likelihood of making another error on the next bit is increased drastically.
In addition to all the blocks being implemented individually, many literature combine two or more equalization schemes together. As shown in Figure 2.11, multiple equalizers (whether different or the same) can be combined in parallel, in series, or one integrated into another as a building block. For example, two modified T-junction passive equalizer were combined in parallel in [23]; Payne et al. [24] combined the FFE at TX and DFE at RX; and Higashi et al. [25] in the RX utilized the split-path approach to combine signals from analog equalizer blocks, which integrated LE as part of the RX pre-filter. The complexity of the overall system is a direct result of trading-off between performance and power.
Chapter 3

Continuous-Time Fractionally-Spaced Equalization (CT-FSE)

This chapter introduces the concept of utilizing continuous-time delay, instead of a unit delay or fractional delay, in a conventional FIR structure. The novel equalization structure is name Continuous-Time Fractionally-Spaced Equalization (CT-FSE). According to the equations derived for the CT-FSE theory, two major observations are presented along with select examples from a set of Matlab programs (more examples and source codes in Appendix A). The CT-FSE has bandwidth and power superior to conventional FIR structure and serves the foundation of the system designs in Chapter 4 and Chapter 5.
3.1 Introduction

3.1.1 Transversal Equalization

\[ y[n] = \sum_{k} a_k x[n-k] \]  \hspace{1cm} (3.1)

, where \( a_k \) is the tap weight. Perform Fourier Transform on both sides of (3.1) and observe the equation, the filter transfer function is the discrete Fourier transform of the tap weights.

Figure 3.1. Block diagram of the conventional transversal filter structure.

One simplest way of realizing an active equalizer is the conventional transversal filter structure, shown in Figure 3.1, which is also known as finite-impulse response (FIR) filter in the DSP field, or feed-forward equalizer (FFE) if implemented in transceiver design. Given a discrete delay of one bit period, the output, \( y[n] \), of the transversal filter is the weighted-sum of the input, \( x[n] \):
\[ H(\omega) = \sum_k a_k e^{-j\omega k} \]  

(3.2)

Therefore, given a known filter response and follow the classic mathematics derivation for Fourier series, the tap weights can be calculated as:

\[ a_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(\omega)e^{j\omega k} d\omega \]  

(3.3)

The pair of (3.2) and (3.3) enables direct analysis and synthesis \(^2\) when designing a transversal filter. An early example of the high-speed link implementation of this structure was presented in [37].

### 3.1.2 Fractionally-Spaced Equalization (FSE)

The concept of using a fractional tap delay (Figure 3.2) instead of unit symbol interval in a transversal filter was first introduced in the 1970s. The mathematic theory, analysis, and the fractionally spaced equalization (FSE) system implementation were realized in the sampling domain, i.e., there are asynchronous sampling actions at the input, output, or the tap delay line, due to the original intention of integrating FSE into digital adaptation hardware as presented by Ungerboeck [27]. In the 1980s, Gitlin and Weinstein [26] demonstrated the performance improvement of FSE over a conventional transversal equalization. Qureshi [28] compared the conventional transversal equalization scheme and the FSE scheme, and

---

\(^2\) The terms “analysis” and “synthesis” follows the definition in Mathematics. Analysis means to decompose the original transfer function into an orthogonal set and calculate the coefficients for the linear combination, while synthesis means to re-assemble the approximated transfer function using the derived information from analysis.
discussed the characteristics of FSE in detail, including behaviors of different adaptation algorithms.

Recently, just as Dally and Poulton [37] brought the concept of FIR to TRX designs, several literatures adopted the concept of continuous-time delay element to FSE designs. These newer design uses continuous-time delay element, such as the current-mode biquad [29], the inductively peaked inverter[30], the LC-delay lines in [35], and the combination of LC-delay line and buffer in [36]. Although these designs could function based on the experiment result, there was lack of direct mathematic proof on how and why the FSE structure would work with continuous-time delay element not exactly at fractions of a period. Therefore, most of these designs select a delay of half-unit interval (T/2) without knowing that the FSE structure would also function properly as long as the delay is within a proper range. Therefore, the rest of this chapter derives the equations of Continuous-Time Fractionally-Spaced Equalization (CT-FSE), analyzes of the filter behavior in detail, gives key observations, proposes method to do design tradeoffs, and provides Matlab programs for behavioral simulation.
3.2 Derivation of the CT-FSE Pair

Figure 3.2. Block diagram of the CT-FSE.

Figure 3.2 shows the block diagram of the CT-FSE structure. This structure uses the continuous-time tap delay, \( \tau \), which can be any analog delay besides unit delay. In continuous-time domain, the output, \( y(t) \), can be written as the linear combination of the input, \( x(t) \), of which each delayed version is scaled by corresponding tap weights:

\[
y(t) = \sum_{k} a_k x(t - k\tau)
\]  

(3.4)

Here \( x(t-k\tau) \) represents the delayed version of input, \( x(t) \), while \( k \) is the number of taps, and \( a_k \) is the set of tap weights for tap branches used in CT-FSE. Perform continuous-time Fourier transform on both sides of (3.4) and we obtain:

\[
Y(\omega) = \sum_{k} a_k X(\omega)e^{-j\omega k\tau}
\]  

(3.5)

Move \( X(\omega) \) out of the summation term, (3.5) becomes:
Therefore, the transfer function, $H(\omega)$, of the CT-FSE structure shown in Figure 3.2 can be defined as:

$$H(\omega) = \sum_{k} a_k e^{-j\omega k\tau} \quad \text{(3.7)}$$

Looking at right hand side of (3.7), it is clear that $e^{-j\omega k\tau}$ forms a set of orthogonal functions by which the transfer function, $H(\omega)$, is expanded just as those in the classical Fourier series in complex form. The following mathematic manipulation, therefore, follows classical derivation for calculating the coefficients of the Fourier series. If $k=0$, integrate both sides of (3.7) over a period, the equation is written as:

$$\int_{-\pi/\tau}^{\pi/\tau} H(\omega) d\omega = \int_{-\pi/\tau}^{\pi/\tau} a_0 e^{-j\omega 0\tau} d\omega \quad \text{(3.8)}$$

The right hand side of (3.8) equals $a_0(2\pi/\tau)$. Therefore (3.8) becomes:

$$a_0 = \frac{\tau}{2\pi} \int_{-\pi/\tau}^{\pi/\tau} H(\omega) d\omega \quad \text{(3.9)}$$

If $k\neq 0$, multiply both sides of (3.7) by $e^{j\omega m\tau}$, where $m$ is an arbitrary integer, and integrate over a period, we obtain:
\[
\int_{-\pi/\tau}^{\pi/\tau} H(\omega)e^{j\omega m\tau} d\omega = \int_{-\pi/\tau}^{\pi/\tau} \sum_{k} a_k e^{j\omega k\tau} e^{-j\omega k\tau} d\omega \tag{3.10}
\]

Separate the \(k=0\) term out of the summation term in the right hand side of (3.10), we obtain:

\[
\int_{-\pi/\tau}^{\pi/\tau} H(\omega)e^{j\omega m\tau} d\omega = a_0 \int_{-\pi/\tau}^{\pi/\tau} e^{j\omega m\tau} d\omega + \sum_{k \neq 0} a_k \int_{-\pi/\tau}^{\pi/\tau} e^{j\omega(m-k)\tau} d\omega \tag{3.11}
\]

Carry out the integral on the right hand side, we obtain

\[
\int_{-\pi/\tau}^{\pi/\tau} H(\omega)e^{j\omega m\tau} d\omega = \begin{cases} 
0 & , k \neq m, k \neq 0 \\
\frac{2\pi}{\tau}a_m & , k = m, k \neq 0
\end{cases} \tag{3.12}
\]

Since (3.12) shows the \(k \neq m\) term can be eliminated, leaving the \(k=m\) term, (3.12) can be rearranged:

\[
a_{k,k \neq 0} = \frac{\tau}{2\pi} \int_{-\pi/\tau}^{\pi/\tau} H(\omega)e^{j\omega k\tau} d\omega \tag{3.13}
\]

Equations (3.9) and (3.13) can be combined for both \(k=0\) and \(k \neq 0\) cases:

\[
a_k = \frac{\tau}{2\pi} \int_{-\pi/\tau}^{\pi/\tau} H(\omega)e^{j\omega k\tau} d\omega \tag{3.14}
\]

Here we arrive at a pair of equations, (3.7) and (3.14), that forms the CT-FSE pair.
Figure 3.3. Similarity between the CT-FSE Pair and the Fourier Series in complex form.

\[ H(\omega) = \sum_{k} a_k e^{-j\omega k\tau} \]

\[ a_k = \frac{\tau}{2\pi} \int_{-\pi/\tau}^{\pi/\tau} H(\omega) e^{j\omega k\tau} d\omega \]

k: number of taps
a_k: tap weights
\( \tau \): tap delay

Complex Fourier Series

\[ f(t) = \sum_{n=-\infty}^{\infty} a_n e^{j\omega t} \]

\[ a_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) e^{-j\omega t} d\omega \]

Figure 3.4. The effects of k and \( \tau \) on when linearly combining \( e^{-j\omega k\tau} \) in Eq. (3.7).
3.3 Analysis of the CT-FSE Characteristic

The CT-FSE pair, (3.7) and (3.14), has similar properties as the conventional FIR filter. Given number of taps, $k$, tap weights, $a_k$, and the tap delay, $\tau$, Eq. (3.7) enables straightforward synthesis$^3$ of the transfer function, $H(\omega)$, in the frequency domain. On the other hand, if the assumed tap delay, $\tau$, and the targeted transfer function, $H(\omega)$, are given, Eq. (3.14) allows direct analysis of the tap weights.

Careful examination of (3.7) shows that the continuous-time delay, $\tau$, adds one more degree of freedom on the orthogonal expansion set, $e^{-j \omega \tau}$, while the conventional FIR structure only has one fixed set of orthogonal functions. Take the right-most column ($\tau = 200$ ps in blue) drawn in Figure 3.4 for example, conventional FIR structure only has the freedom to change the number of taps, $k$, and corresponding tap weights, $a_k$. As number of taps, $k$, increases, the available orthogonal set goes down to allow for higher varying rate of the filter. However, as many taps as a design can incorporate, the bandwidth of the overall FIR filter is limited by the unit-delay ($\tau$ fixed to 200ps). Whereas Eq. (3.7) introduces the continuous-time delay, $\tau$, hence extending the availability of the orthogonal expansion set from only one fixed column in Figure 3.4 to different columns. Therefore, the bandwidth of the CT-FSE filter increases when the available orthogonal set in Figure 3.4 goes toward left. This leads to the first key observation of this chapter: Less $\tau$ means more bandwidth.

---

$^3$ As stated in the footnote in page 19, the term “synthesis” used is not the same as the definition used in digital logic design. Here, “synthesis” is defined as the process of obtaining the transfer function, $H(\omega)$, by linearly combining the orthogonal set, $e^{-j \omega \tau}$, given the $k$, $a_k$, and $\tau$. 
To apply the CT-FSE theory in equalizer design, Figure 3.5 shows the procedure to synthesize an arbitrary transfer function: First, from a preliminary simulation, an ideal transfer function can be calculated, for example, $H_1(\omega) = 1/\mathcal{S}_{21,\text{channel}}$. Second, $\{a_k\}$ can be directly analyzed using (3.14), given the ideal $H_1(\omega)$, $k$, and $\tau$. Finally, the transfer function (non-ideal $H_N(\omega)$) of the CT-FSE can then be synthesized by substituting $\{a_k\}$ into (3.7). To prove synthesizability of CT-FSE, the Matlab plot in Figure 3.6 shows that, assuming $\tau = 74\text{ps}$ and $|k| \leq 4$, $H_N(\omega)$ can be synthesized to match ideal LPF and HPF with different cutoff frequency ($f_c = 1\text{GHz}$ to $5\text{GHz}$). Each unique cutoff frequency would have a corresponding set of $\{a_k\}$. 

Figure 3.5. Procedure of synthesizing $H_N(\omega)$. 

\[
H_i(\omega) : \text{ideal filter response} \\
H_N(\omega) : \text{non-ideal filter response}
\]
Figure 3.6. Swept $H_I(\omega)$ corner frequency from 1 to 5GHz and follow procedure in Figure 3.5 to synthesize (Left) ideal LPF and (Right) ideal HPF.

Depends on how the continuous-time delay, $\tau$, is realized in a system, it is possible that $\tau$ varies with PVT corners, hence affect the robustness of the system. To prove robustness of CT-FSE, the Matlab plot in Figure 3.7 shows that, assuming $|k| \leq 4$ for each different value of $\tau$ swept from 50 to 200ps, $H_N(\omega)$ can be synthesized to match ideal LPF and HPF with cutoff frequency of 1GHz. Again, each unique $\tau$ would correspond to a set of $\{a_k\}$. In other words, (3.7) and (3.14) prove that adjusting tap weights can compensate for the negative effects of varying $\tau$. 

* Plots shifted vertically for comparison
Figure 3.7. Swept $\tau$ from 50 to 200ps and follow procedure in Figure 3.5 to synthesize (Left) ideal LPF and (Right) ideal HPF.

It is clear shown in Figure 3.3 that the main difference between the conventional FIR and CT-FSE systems lies in the variable tap delay, $\tau$. When synthesizing for an ideal $H(\omega)$, not only $k$ and $a_k$ can be varied, $\tau$ also has effects on the bandwidth and sharpness of the filter. By observing the limits of integral from (3.14), it is shown that the bandwidth of the CT-FSE system is inversely proportional to the tap delay, $\pm \pi/\tau$. However, a $\tau$ that is too small indicates that the set of $e^{-j\omega k\tau}$, by which $H(\omega)$ is to be expanded in (3.7), has inadequate varying rate (notice this is in the frequency domain), due to a finite $k$. This in turn reduces the sharpness of the filter, and the effect is demonstrated by identifying the $\tau=10$ps curve as a outlier along with other synthesized $H_N(\omega)$ in both LPF and HPF of Figure 3.7. This leads to the second key observation: Less $\tau$ means less filter sharpness.
The waveform in Figure 3.8 further explains the second observation in time domain. Assume the physical design of the filter only allows for 4 taps (k=4) to be implemented, to eliminate the long tail of the signal with ISI, the configuration with moderate \( \tau \) would be the most suitable choice, whereas the one with large \( \tau \) would give a coarse time resolution (smaller bandwidth), and the one with small \( \tau \) would stop equalizing before the long tail appears (cannot filter out the low frequency component of ISI, hence the low sharpness in frequency domain).

Figure 3.8. Different value of \( \tau \) changes the available time point (indicated by arrows) upon which equalization is applied.

From the two key observations bring this analysis into a dilemma: When a sharper filter response is needed, a larger delay, \( \tau \), or more taps, \( k \), are preferred. In contrast, when a higher filter bandwidth is required, a smaller delay, \( \tau \), should be implemented. A proposed way to solve this dilemma is to prioritize the most critical design parameters. If satisfying a power budget is top priority for a system, then the first step should be determining the number of taps, \( k \), that is affordable in the physical design, since \( k \) directly impact the number of
branches in the real circuitry hence the power implication. All other parameters, including number of taps, $k$, can be determined later. On the other hand, if the system has a performance expectation more important than anything, the higher bandwidth requirement would imply the use of small $\tau$. And the number of taps has to be enough to cover enough timespan for proper filtering sharpness in frequency domain.

### 3.4 CT-FSE Synthesizer in Matlab

The CT-FSE synthesizer$^4$ (Figure 3.9) is a group of Matlab programs that analyze CT-FSE parameters and synthesize the CT-FSE filter response in both frequency-domain and time-domain.

The upper block of Figure 3.9 shows the frequency-domain function of this CT-FSE synthesizer. It is implemented by coding the CT-FSE pair from (3.7) and (3.14) into Matlab, also accounting for the non-ideality of the parameters, such as finite $k$. Furthermore, this Matlab program allows the behavioral characterization of the CT-FSE system. For example, the plots in Figure 3.6 and Figure 3.7 demonstrate the influence of $\tau$ on CT-FSE response.

The lower block of Figure 3.9 shows the time-domain function of CT-FSE synthesizer. Besides the most straightforward function of direct processing waveform through CT-FSE algorithm in time-domain, it is also capable of parameter optimization, including $k$, $a_k$, and $\tau$. Sometimes the target ideal filter response, $H_I(\omega)$, is not readily available, especially in ACCI, when pulse signaling is used and an unknown bandpass response is required. Therefore, the

---

$^4$ The Matlab source code of CT-FSE synthesizer is in Appendix A.
time-domain search (sweep and sort) function bypasses the steps shown in Figure 3.5 and allows direct optimization of the CT-FSE parameters and estimation on the hardware requirement.

\[
\tau, a_k, k
\]

\[
\frac{a_k}{2\pi} \int e^{j\omega k\tau} d\omega
\]

\[
H_N(\omega) = \sum_k a_k e^{-j\omega k\tau}
\]

Figure 3.9. The CT-FSE synthesizer implemented in Matlab. (Upper) Frequency-domain behavioral simulation. (Lower) Time-domain parameter optimization.

The Time-Domain Optimizer (the time-domain portion of the CT-FSE Synthesizer) consists of a MAIN control unit, a CT-FSE core, an ideal latch implemented by a PWL amplifier (necessary to support pulse signaling), and a summer generating the error to be stored. After fed with a signal with ISI, the MAIN unit performs a blind search on parameters and sorts the result with normalized LMS algorithm by: 1) sweeping one or more parameters
in the parameter sets: \{k, a_k, \text{and } \tau\} for CT-FSE, 2) storing the errors corresponding to each set of parameters, and 3) sorting the normalized error. According to the normalized errors, the program outputs a list of the ranked parameters (corresponding to different \(\tau\)), as well as eyediagram before and after CT-FSE is turned ON. This program took about 1 hour to run with the depict flow but was optimized to run under 1 minute by flattening the “for” loops into very large matrices and utilizing the speedy matrix multiplication in Matlab.

### 3.4.1 Time-Domain Example (Pulse-Signaling)

This example demonstrates the operation of the Time-Domain Optimizer of the CT-FSE Synthesizer shown in Figure 3.9.

In HSPICE simulation, an ideal Pseudo-Random Bit Sequence (PRBS) is passed through a long 5 cm + 1pF capacitor + 115cm PCB channel, and the channel output (CT-FSE input) is a 5Gbps close eye, as shown in the left of Figure 3.10. After importing this time-domain waveform into Matlab, the time-domain optimizer is setup to search for the best combination of the three tap weights, \{a_k\}, for each possible \(\tau\) between 75ps and 225ps. This proves that, for each \(\tau\) within proper range, there exist tap weights so that the CT-FSE output is an open eye diagram, as shown in the right of Figure 3.10.

Notice from the difference of a pulse signal eyediagram – the fuzz band at voltage around zero. This is the main reason why the Ideal PWL Amplifier is required, as denoted in Figure 3.9, to recover the NRZ signal for error calculation. More detail information and source code of the Time-Domain Optimizer for pulse signaling is in A.3.
Figure 3.10. CT-FSE synthesizer optimizes \( \{a_k\} \) to achieve open eyes when \( \tau \) varies.
3.4.2 Cross Frequency- and Time-Domain Example (NRZ Signaling)

Figure 3.11. Cross domain operation to verify the optimized $k$, $a_k$, and $\tau$ are valid.

This example modifies the error calculation part in bottom of Figure 3.9, replacing it with a NRZ Eye Opening Measurement block, to accommodate NRZ signaling and measure vertical opening based on certain percentage of a unit interval (UI) in horizontal opening (50% assumed in this example). This example also demonstrates the method to verify the parameters optimized by Time-Domain Optimizer.

As shown in Figure 3.11, a Stimulus function is added to the CT-FSE Synthesizer. This Stimulus function includes 1) a configurable linear feedback shift registers (LFSR) block at bottom left corners to generate ideal input signals in time-domain and 2) a time-domain simulation to pass the ideal input signals through a channel imported from s-parameter files.
(Touchstone format). The Stimulus function assumes ideal driver with 50 ohm matching impedance, simulates in Matlab, then output signals with ISI to the Time-Domain Optimizer.

In this example, there are 5 channels for result comparison. These channels are the same PCB microstrip with 5 different lengths (from 20cm to 100cm) with $S_{21}$ plots shown in Figure 3.12. It is shown that the longer the channel, the higher attenuation at high frequency. For model validity, this Matlab $S_{21}$ plot matches the HSPICE plot in Figure 1.4. The channel output eyediagrams simulated by the Stimulus function are shown in left column of Figure 3.13, which shows the decreasing eye opening when channel length is increased.

![Comparison of $|S_{21}|$](image.png)

Figure 3.12. $S_{21}$ of various channel lengths.
Figure 3.13. CT-FSE Inputs (Channel outputs) vs. CT-FSE Outputs (Time-Domain Optimizer outputs).
Table 3.1. Optimized tap weights, $a_k$, and eye opening for 5 different channel length.

<table>
<thead>
<tr>
<th>Channel Length</th>
<th>$a_0$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>Vertical Eye Opening</th>
</tr>
</thead>
<tbody>
<tr>
<td>20cm</td>
<td>7</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1.400</td>
</tr>
<tr>
<td>40cm</td>
<td>5</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1.129</td>
</tr>
<tr>
<td>60cm</td>
<td>1</td>
<td>-6</td>
<td>2</td>
<td>0</td>
<td>1.013</td>
</tr>
<tr>
<td>80cm</td>
<td>1</td>
<td>-7</td>
<td>3</td>
<td>0</td>
<td>0.933</td>
</tr>
<tr>
<td>100cm</td>
<td>1</td>
<td>-6</td>
<td>3</td>
<td>0</td>
<td>0.845</td>
</tr>
</tbody>
</table>

Figure 3.14. Target $H_I(\omega)$ (=1/$S_{21}$) vs. Synthesized $H_N(\omega)$ according to $k$, $a_k$, and $\tau$ from Time-Domain Optimizer

Assuming a $\tau$ of 80ps and 4 taps ($k=4$), the Time-Domain Optimizer generates 1) five top ranked CT-FSE output eyediagrams, one for each channel length, as shown in the right column of Figure 3.12 and 2) five sets of tap weights, $a_k$, and vertical eye opening of the top ranked CT-FSE output eyediagrams, one set for each channel length, as shown in Table 3.1.
To do cross domain verification, as indicated by the red arrow in Figure 3.11, all five sets of tap weights, $a_k$, along with $\tau$ of 80ps and $k$ of 4, are also fed into the Frequency-Domain Synthesizer. As shown in Figure 3.14, the five synthesized non-ideal transfer functions, $H_N(\omega)$, are plotted on top of the target ideal transfer functions $H_I(\omega)$, which are calculated by the inverse of channel response ($H_I(\omega) = 1/S_{21}$). Figure 3.14 verifies the trend of the non-ideal transfer functions, $H_N(\omega)$, showing more high-pass characteristic as the channel length increases. The little mismatch between $H_N(\omega)$ and $H_I(\omega)$ due to three reasons: 1) The final goal of the Time-Domain Optimizer is to maximize eye opening, instead of matching transfer functions, 2) the CT-FSE has limited number of taps, 3) coarsely quantized tap weight resolution (3-bit). Nonetheless, the mismatch does not affect the final outcome, as confirmed by the eyediagrams in Figure 3.13 and the eye opening measurement in Table 3.1.

Multiplying the channel $S_{21}$ with the synthesized $H_N(\omega)$, respectively for each channel, results in the overall system response in Figure 3.15. The over-boost effects on the 60cm, 80cm, and 100cm channels can be confirmed by observing the slightly overshoot of corresponding eyediagrams in Figure 3.13. Again, this is acceptable since the goal of the Time-Domain Optimizer is to maximize eye opening instead of matching transfer functions.

More detail information and source code of the Time-Domain Optimizer for NRZ signaling is in A.4.
Figure 3.15. System overall frequency response = $S_{21} \times H_N(\omega)$
Chapter 4

Transmitter-Side CT-FSE Utilizing

MultiCap Voltage Summer

Traditionally, current-mode summing, or current-mode digital-to-analog converter (DAC) has been the only choice when implementing the summing function in high-speed FFE, such as those used in [24] and [25]. There are several problems associated with current-mode summation: First, it consumes a significant amount of power due to the steady bias/tail current. Secondly, a large parasitic capacitor (dominant pole) exists on the node where all current summing branches are joined together. Thirdly, the output currents are summed non-linearly due to channel-length modulation. Finally, current summation has limited dynamic range because of constraints on output swing to keep transistors in saturation. This chapter presents a novel multi-capacitor (MultiCap) structure to eliminate these problems all together. Using MultiCap as a voltage-summer, a new form of voltage-mode CT-FSE achieve power efficiency significantly better than the current-mode CT-FSE as well as conventional FFE.
4.1 A Zero-Power Consumption MultiCap Structure

A MultiCap structure compatible with the ACCI technology is presented in Figure 4.1. This passive structure enables a simpler yet more efficient way to perform the summing.
function. Traditionally, the coupling capacitors in ACCI are formed by the top metal plates on the flipped chip and the package (left pad of Figure 4.1). CMOS technology, however, has the advantage of much higher fabrication resolution than the package; therefore, it is straightforward to utilize this advantage toward creating the MultiCap structure (middle pad of Figure 4.1) for signal coupling as well as voltage summing. The difference between the traditional coupling capacitor and the MultiCap structure is that the former has a single top plate, while the later has a matrix of discrete top metal plates. Each individual capacitor of the MultiCap, as illustrated in Figure 4.2, is connected to separate branch (1 LSB) of the voltage-mode output driver, replacing both the voltage-to-current and current summing block in conventional FFE.

![MultiCap as Σ block](image)

Figure 4.3. TX-side pre-emphasis realized by CT-FSE utilizing MultiCap voltage summer.

The full system block diagram is shown in Figure 4.3. Instead of current being summed on a single branch in the conventional current-mode FFE, voltage is summed on the shared bottom plate. This structure solves the problems of the conventional current summation all
together. First, simple voltage-mode circuits can be used throughout the TX, and the need for steady tail currents is eliminated. Secondly, each branch of the FFE is independently coupled to one common bottom plate, eliminating the presence of a single dominant parasitic capacitor. Finally, the MultiCap linearly sums the charges at the bottom plate regardless of the magnitude of the voltage on the top plates. Therefore, unlike current-mode summing, the MultiCap voltage summer does not have the constraints on dynamic range of the output swing nor the need of a common-mode feedback circuit. Additionally, This ACCI with MultiCap system is identical to traditional ACCI from the bottom plate in package to the RX [21]. And it inherits the advantages of ACCI, including high I/O density, potential to improved yield, and increased long-term reliability [20].

The concept of MultiCap can also be used on-chip whenever voltage-summing function is feasible, but this work is focused mostly on the application to ACCI technology.

4.1.1 Parameter Analysis of the MultiCap Structure

The top of the MultiCap structure, as shown in Figure 4.4, is an array of individual capacitances ($C_C$) connected to the FFE voltage-mode driver. The number of rows equals the number of taps of the FFE ($N_{tap}$), and the number of columns equals $2^R$, where each tap weight has R-bit resolution. All individual capacitances are of identical geometries; therefore, the tap weights of the FFE are set by whether each branch has a signal or not. It is important for each individual $C_C$ to be identical to keep the circuit simple and linear. The following sections provide equations to help choosing useable values of the MultiCap. The variables
used in the following equations are the plate geometries (w, h, and s), individual plate area (A_{CC}), shared bottom pad area (A_{pad}), and dielectric constant (k), also shown in Figure 4.4. Note that the spacing (s) is neglected in most of the equations when calculating the MultiCap geometries.

The values of C_C, like all parallel-plate capacitors, are determined by the size of the overlapping plate area (A_{CC}), the dielectric thickness (d), and the dielectric constant (k):

\[ C_C = k \varepsilon_0 \frac{A_{CC}}{d} \]  \hspace{1cm} (4.1)

The upper-bound of the area of each top plate, A_{CC,max}, is limited so that the total area of the MultiCap is smaller than the bottom plate:

Figure 4.4. Top view of the MultiCap structure includes a matrix of silicon top metal plates and a shared big bottom plate on package.
In special cases, ACC may be more than indicated in (4.2). For instance, the total number of top plates is no longer \((2^R - 1)\) when the FFE has different maximum tap weight for each tap. Nonetheless, \(A_{CC,\text{max}}\) can still be calculated by assuming each top plate is identical, and the following equations can be modified accordingly.

The minimum area of each top plate, \(A_{CC,\text{min}}\), is implicitly limited by the minimum RX sensitivity. That is, the total capacitance of one row of the MultiCap, \(C_{C,\text{row, min}}\), should be large enough such that the RX is able to detect the non-equalized signal after attenuation over the longest channel. The value of \(C_{C,\text{row, min}}\) is treated as a given number and should be estimated from a preliminary simulation, which includes an ideal driver and an ACCI channel. The total equivalent area of the entire row of top plates \(A_{CC,\text{row, min}}\) can be calculated:

\[
A_{CC,\text{row, min}} = \frac{A_{pad}}{N_{\text{ap}} \times (2^R - 1)} \tag{4.3}
\]

Thus, the corresponding minimum area of each top plate, \(A_{CC,\text{min}}\), can be derived:

\[
A_{CC,\text{min}} \geq \frac{A_{CC,\text{row, min}}}{(2^R - 1)} = \frac{d}{k\varepsilon_0 (2^R - 1)} C_{C,\text{row, min}} \tag{4.4}
\]

A larger \(A_{CC}\) is preferable because, unlike the non-equalizer version where \(C_C\) larger than necessary causes ISI [21], the equalizer is capable of either tuning down the driving power.
for shorter channels or generating more high-frequency components for longer channels, thereby eliminating ISI.

The range of the MultiCap area is bounded mainly by two variables, $A_{pad}$ and $C_{C, row, min}$, from (4.2) and (4.4), respectively. The former is related to I/O density and the later to circuit capability. By separately substituting (4.2) and (4.4) into (4.1), then combining the results, the upper-bound and lower-bound of $C_C$ can be calculated:

$$\begin{align*}
C_{C, min} & = \frac{C_{C, row, min}}{(2^k - 1)} \\
\leq C_C & \leq \frac{k \varepsilon_0 A_{pad}}{d \times N_{tap} \times (2^k - 1)} = C_{C, max}
\end{align*}$$

(4.5)

Equation (4.5) is essential in estimating the available value of $C_C$. For example, given a set of variables; $A_{pad} = 175 \times 175 \mu m^2$ (200µm pitch and 25µm spacing on package from [20]), $C_{C, row, min} = 500 fF$ (from preliminary simulation with TRX in 0.13µm standard CMOS and 75cm microstrip on FR4 PCB), $d = 1 \mu m$ (assumed), $k = 18$ (from [50]), $N_{tap} = 4$, and $R = 3$, substituting into (4.5) results in:

$$71.5 fF = \frac{500 \times 10^{-15}}{(2^3 - 1)} \leq C_C \leq \frac{18 \times 8.85 \times 10^{-12} \times (175 \times 10^{-6})^2}{1 \times 10^{-6} \times 4 \times (2^3 - 1)} = 174.2 fF$$

(4.6)

The usable range of MultiCap calculated from (4.6) helps the early stage of the design process (as further detailed in section 4.4).

Further analysis of the capacitance range can be achieved by equating both sides of (4.5):

$$C_{C, row, min} \leq \frac{k \varepsilon_0 A_{pad}}{d \times N_{tap}}$$

(4.7)
Equation (4.7) shows the trade-offs between three of the most dominant groups of variables for the MultiCap structure: the geometries ($A_{pad}$ and $d$), the property of the dielectric filling ($k$), and the circuit complexity ($N_{tap}$ and, implicitly, $C_{C,row,min}$). In order to satisfy (4.7), when driven toward higher-density I/O, the area of the bottom plate is limited; thus, a higher dielectric constant is needed. Alternatively, the same goal can be achieved by decreasing the dielectric thickness, decreasing the circuit complexity, such as lower $N_{tap}$, or improving the RX sensitivity, hence lowering $C_{C,row,min}$. Take the same example used in (4.6), if all the parameters are fixed except for $k$,

$$500 \times 10^{-15} \leq \frac{k \times 8.85 \times 10^{-12} \times (175 \times 10^{-6})^2}{1 \times 10^{-6} \times 4} \Rightarrow k \geq 7.4$$  \hspace{1cm} (4.8)$$

Equation (4.8) shows that $k$ needs to be at least 7.4 for a useable MultiCap structure. If the high-$k$ dielectric ($k=18$) used in the example is not available, and an ordinary oxide ($k=4$) is used instead, equation (4.8) will not be satisfied unless the geometric constraints are relieved ($A_{pad} \geq 238 \times 238 \mu m^2$ or $d \leq 0.54 \mu m$), circuit complexity is decreased ($N_{tap} \leq 2$), or the RX sensitivity is improved ($C_{C,row,min} \leq 271 fF$).
4.1.2 Parasitic Capacitance of the MultiCap Structure

Figure 4.5. The parasitic capacitances between a pair of \( C_C \) and ground
As shown in the 3D illustration (Figure 4.5) and schematic plot (Figure 4.6), there are two types of parasitic capacitances in the MultiCap structure: vertical parasitic ($C_{p,TG}$, and $C_{p,BG}$, coupled to ground) and horizontal parasitic ($C_p$, coupled to next row of MultiCap). The vertical capacitances, $C_{p,TG}$ and $C_{p,BG}$, are between the top plate and the on-chip ground plane, and between the bottom plate and the package ground plane, respectively. They are treated as constants from manufacturing process, due to the fixed height between plates and ground planes, and can be minimized by placing a cutout in the ground plane where the MultiCap is located. The other significant parasitic component is the horizontal fringe capacitance, $C_p$, between two rows of the MultiCap. The ratio between $C_p$ and $C_C$ can be derived:

$$
\begin{align*}
C_C &= k \varepsilon_0 \frac{wh}{d} \\
C_p &= k \varepsilon_0 \frac{wd}{s}
\end{align*}
\Rightarrow R = \frac{C_p}{C_C} = \frac{d^2}{sh}
$$

(4.9)

It is shown that the ratio’s dependency on $w$ is eliminated in (4.9) because of the one shared width. The parasitic capacitance between columns can be calculated in the same manner (with eliminated $h$) but is ignored due to both terminals of the capacitor being
connected to the same tap of FFE. Also, R is independent of k only when the dielectric filling is homogeneous – the dielectric filling is not patterned, nor layered; otherwise, R will be smaller than indicated in (4.9). Under normal circumstances, when \( d \approx s \), the ratio R is about \( d/h \). If \( h \gg d \), \( C_p \) is negligible.

Figure 4.7. Calculation results using (4.9) vs. Sonnet simulation result

Figure 4.7 shows a comparison between calculated results using (9) and EM simulation results using Sonnet Suites (3D planar and model extraction). It is shown from both curves that \( C_p \) is less than 10% for a horizontal spacing larger than 1µm. EM simulation shows more parasitic capacitance because it takes into account the fringe capacitance.
4.2 Voltage-Mode CT-FSE Transmitter Utilizing MultiCap Voltage Summer

Figure 4.8. All-digital complementary CT-FSE TX.

Figure 4.8 depicts the TX design that integrates the MultiCap structure into the CT-FSE structure. The all-digital complementary TX has four tap branches \((k=4)\), while each tap branch includes a sign switch followed by 7 slices (3-bit resolution) of voltage-mode tap weight controllers (gain stage) and drivers. The tap weight of each slice is controlled via 7-bit thermometer code, \(a_{k,s}\).
The detail of each slice is shown in Figure 4.9; the gain stage is a simple NAND gate switch, and the NMOS-only multiplexer-type driver uses a low voltage power supply for further reducing power consumption. The driver outputs are individually connected to the top plates of the MultiCap structure shown in Figure 4.10.

Figure 4.9. Detail of a slice of TX gain and driver stages.

Figure 4.10. The MultiCap structure implemented by on-chip VNCAP

$$\text{On-Chip implementation}$$
$$\text{Using VNCAP (M3-M6)}$$
$$\text{Capacitance} = 150\text{fF}$$

$$\text{#Columns} = 2^{(N-1)} - 1 = 2^{(4-1)-1} = 7$$
(4-bit resolution for each tap branch)
Due to availability issues of the ACCI technology, the MultiCap structure in the summing node of Figure 4.8 is realized by a $4\times(2^3-1)$ matrix of on-chip Vertical Natural Capacitor (VNCAP, for metal layers M3-M6). Each unit capacitor utilizes the lateral capacitances between interdigitated metals. Each row of the MultiCap is connected to one tap branch, and each column corresponds to one LSB. The individual sub-capacitor of the MultiCap has a capacitance value of 150fF, selected according to (4.6), and yields a total of 1.05pF per tap branch.

### 4.3 Receiver Design

![Figure 4.11. Detail of high-speed latch RX](image)

The RX (Figure 4.11) latch used here is a clockless current-mode regenerative latch. It is modified from a non-clocked version of a low-power differential NMOS latch [19]. The
A high-speed latch normally has a weaker cross-coupled latch, in order to function at high-speed. However, a weak latch would be easily affected by PVT variation, such as a mismatch between the amplifying and the latching transistors, when the cross-coupled latch is weaker than desired. To compensate for this problem, a stronger cross-coupled latch is used, but this would drive the differential output voltage further apart and makes it difficult to switch, hence the less sensitivity. Therefore, a resistive transistor across both output nodes is added to clamp the output voltage and enable a stronger latch with moderate sensitivity.

Figure 4.12. Block diagram of the design flow for CT-FSE TX utilizing MultiCap voltage summer
4.4 Design Flow

Figure 4.12 shows the design procedure for the system in this chapter. It is similar to the one shown in previous chapter; the only difference is the preliminary work due to different physical constraints. The most important parameters (capacitance) are related to the geometric constraints via (4.5) due to the implementation of the ACCI-compatible MultiCap structure. If on-chip MultiCap structure is used instead, procedure similar to section 4.1.1 can be followed. In this case, unlike the dielectric constant (k) and dielectric thickness (d) used in the equations in section 4.1.1, capacitance density (fF/µm²) is the dominant constraint. Typical, on-chip capacitors have density of 2 to 4 fF/µm².

4.5 Measurement Results

A test chip was fabricated in the IBM 0.13µm standard digital CMOS technology. Figure 4.13 shows PCB configuration and silicon area of the TRX, including MultiCap located underneath the bondpad. Figure 4.14 shows the PCB cross section and the real measurement setup on the probe station. The PCB has silicon chip directly epoxied onto the top copper pad and the chip is connected to PCB through wire-bonding. The overall test setup, including one setup for displaying waveform and the other for measuring BER, is shown in Figure 4.15 and Figure 4.16.
Figure 4.13. Die photo with TRX area and PCB photo with channel setup.
Figure 4.14. PCB cross section and the real measurement setup on probe station.
Figure 4.15. The overall test setup when displaying waveform on Oscilloscope.
Figure 4.16. The overall test setup when measuring for BER.

The delay cell made of a pair of inverters has a measured delay of 84ps. Figure 4.17 shows the effects of the changing tap weights by comparing RX inputs before and after the CT-FSE is enabled. Figure 4.18 shows the 5Gb/s NRZ data recovered by the RX. The system has a bit-error-rate (BER) of less than $10^{-12}$ for a PRBS $2^{23}-1$ input pattern. For each channel, the TX and RX consume 6.5mW and 1.1mW, respectively. Table 4.1 compares the TRX performance with state-of-the-art designs [51] [52] [53]. It is demonstrated that the TRX,
benefitting from the CT-FSE scheme combined with MultiCap, has an area of only
0.007mm² and achieves a power efficiency of 1.7mW/Gb/s.

Figure 4.17. Comparing the RX inputs before and after the TX-Side CT-FSE is enabled.

Figure 4.18. RX output: recovered 5Gb/s NRZ bit-stream.
Table 4.1. Comparison of Performance.

<table>
<thead>
<tr>
<th>Process</th>
<th>This work(^1)</th>
<th>[51]</th>
<th>[52]</th>
<th>[53]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel Length</td>
<td>10cm(^2)</td>
<td>40cm</td>
<td>80cm(^2)</td>
<td>75cm(^3)</td>
</tr>
<tr>
<td>TX Power</td>
<td>5.2mW</td>
<td>6.5mW</td>
<td>7.6mW</td>
<td>4.9mW</td>
</tr>
<tr>
<td>RX Power</td>
<td>1.1mW</td>
<td>1.1mW</td>
<td>1.1mW</td>
<td>8mW</td>
</tr>
<tr>
<td>TRX Total Area</td>
<td>.007mm(^2)</td>
<td>.307mm(^2)</td>
<td>.054mm(^2)</td>
<td>.023mm(^2)</td>
</tr>
<tr>
<td>TRX Power Efficiency (mW/Gb/s)</td>
<td>1.3</td>
<td>1.5</td>
<td>1.7</td>
<td>2.0</td>
</tr>
<tr>
<td>TRX Config.</td>
<td>TX CT-FSE (k=4,R=3)</td>
<td>Passive EQ</td>
<td>RX half-rate Cap. Degen.</td>
<td>RX Cap. Degen. (\times 4)</td>
</tr>
</tbody>
</table>

\(^1\)No multiplexer incorporated. \(^2\)Simulation result. \(^3\)Estimated from loss. \(^4\)Only RX was reported.
Chapter 5

Receiver-Side CT-FSE

with Embedded Capacitor

This chapter presents a RX-side CT-FSE realized in conventional CML. In addition to the active equalization provided by the current-mode CT-FSE, the package is inserted an embedded capacitor in series with the signaling channel to compensate for high frequency loss of channel. This interconnect design verifies the operation of current-mode CT-FSE combined with ACCI’s pulse signaling generated by capacitor embedded in PCB channel.

5.1 Embedded Capacitor and Channel Model

Some interconnect standards, such as FiberChannel, require series capacitors for DC decoupling. These are usually built using surface mount technology (SMT). Embedded capacitor technology has proven better performance than surface mount technology (SMT) capacitors in power supply decoupling [38] - [46]. Embedded capacitor not only saves area for on-chip capacitor and the mount surface of SMT capacitor on PCB, but also has better reliability due to its embedded structure [41]. It also significantly reduces parasitic inductance [40]. The question addressed in this chapter is whether buried capacitors can be
used to replace surface mount capacitors in interconnect circuits, and thus bring these advantages to that application.

Table 5.1. Investigation of available materials for embedded capacitor.

<table>
<thead>
<tr>
<th></th>
<th>Structure</th>
<th>Dk ($\varepsilon_r$)</th>
<th>Cap. Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sanmina-SCI FaraFlex[42]</td>
<td>Planar BC-xx</td>
<td>4.4-30</td>
<td>~ 1 - 17aF/µm</td>
</tr>
<tr>
<td>3M C-PLY19[43]</td>
<td>Planar ECM</td>
<td>21</td>
<td>~ 9.6aF/µm</td>
</tr>
<tr>
<td>DuPont Interra[44]</td>
<td>Planar HK-04</td>
<td>3.6</td>
<td>~ 2.4aF/µm</td>
</tr>
<tr>
<td>DuPont Interra[45]</td>
<td>Discrete Ceramic</td>
<td>&gt;2000</td>
<td>~ 2 - 10pF/µm</td>
</tr>
<tr>
<td>On Chip</td>
<td>Variety of Caps</td>
<td>&gt;2000</td>
<td>~ &lt;25 fF/µm</td>
</tr>
</tbody>
</table>

Figure 5.1. Structure of package cross-section, including embedded capacitors for both power supply decoupling and ACCI signaling.

The key problem is that buried capacitors provide capacitance values smaller than those called for in the standards. With capacitance density of up to tens of nF/cm², if combined with the circuit design technique of AC coupled interconnect (ACCI) [21], the embedded capacitor actually is suitable for replacing the DC blocking capacitor in backplane chip-to-chip communication.
A typical cross-section of a package-embedded capacitor structure is shown in Figure 5.1. The layer inside the laminate packaging for the embedded capacitors was initially created for power supply decoupling purposes [47], as shown on the left hand side of Figure 5.1, but eventually signals have to be routed from top to bottom and pass through the embedded capacitor layer. Therefore, it is straightforward to make use of the capacitor structure and insert a series capacitor in the signaling channel for the purpose of passive equalization, as shown on the right hand side of Figure 5.1. Notice that the embedded capacitor can also be manufactured in the PCB but, as long as the channel model matches Figure 4.2, the analysis in the following sections remains valid.

![Figure 5.2. Simplified schematic of the ACCI channel with embedded capacitor.](image)

The simplified schematic in Figure 5.2 creates model for the presented interconnect system. The stubs indicate the transmission line in package, while $C_P$ is the dominant parasitic capacitance. As in typical CCI [21], small series capacitor inserted into the signal path would differentiate the input NRZ bit stream into pulse signals; positive edge to positive pulse, and negative edge to negative pulse. This passive equalization is combined with the
active equalization in RX, which requires a CT-FSE to eliminate ISI and a high-speed latch to recover pulse signals back to NRZ bit stream.

5.1.1 Analysis of Embedded Capacitor and Signaling Modes

![Simulated S21 plots of the channel shown in Figure 5.2.](image)

Figure 5.3. Simulated $S_{21}$ plots of the channel shown in Figure 5.2.

S-parameter simulation results in Figure 5.3 helps analyze the channel shown in Figure 5.2; assuming the channel consists of a 10cm stub, a variable $C_c$, a 100cm PCB trace, and another 10cm stub. Figure 5.3 compares the simulated $S_{21}$ plots of the channel by sweeping $C_c$ over a large range (100fF to 10nF). The peak of $S_{21}$ shifts toward higher frequency as $C_c$ decreases. It is clearly shown that this ACCI channel acts as an equivalent band-pass filter (BPF). And there is more than 30dB difference in peak of $S_{21}$ between channels with
different \( C_C \). Although the magnitude of the \( S_{21} \) is also affected by channel length, once the value of \( C_C \) is determined during design process, the maximum acceptable attenuation can be calculated according to the circuit performance, such as RX sensitivity, noise, and active filter response. And then, maximum channel length can be subsequently set. Therefore, only \( C_C \) is treated as main variable for the analysis here.

Either one of the two types of signaling modes – pulse signaling and NRZ signaling – is supported by ACCI. Whether the channel is suitable for pulse or NRZ signaling depends on the size of the \( C_C \). The peak of the channel equivalent BPF (Figure 5.3) shifts toward lower frequency when the value of \( C_C \) increases, allowing for more low frequency components passing through the channel. The amount of low frequency components being passed through will determine whether NRZ signaling is supported.

![Figure 5.4: The center frequency and bandwidth of the channel-equivalent BPF change as \( C_C \) increases](image)

Figure 5.4. The center frequency and bandwidth of the channel-equivalent BPF change as \( C_C \) increases
Figure 5.4 is generated by measuring center frequency and bandwidth of the channel-equivalent BPF in Figure 5.3. Ideally, the pulses in pulse signaling consist of mostly high-frequency components; therefore, the most suitable $C_C$ value is the one that makes the channel BPF peak at highest frequency points. That is, according to Figure 5.4, the smaller the value of $C_C$ the higher center frequency of the BPF, hence higher bit-rate supported. Figure 5.4 also shows that the bandwidth is higher when $C_C < 10\text{pF}$.

The unit step response of the ACCI channel with various $C_C$, as shown in Figure 5.5, further explains this trend in the time-domain. It shows that smaller values of $C_C$ results in narrower pulse widths. However, smaller values of $C_C$ also reduce the pulse height. For $C_C < 1\text{pF}$, the voltage pulse drops rapidly after the initial coupling. Thus, ideally, a very small $C_C$ could make the pulse narrower than the target bit period and eliminate the need for additional equalization. However, the minimum reasonable value for $C_C$, which also corresponds to the lowest signal amplitude, is limited by the circuit performance and signal integrity; i.e., RX sensitivity, crosstalk, and noises. For example, in standard 0.18um CMOS technology, in order to generate a 100mV swing plus 20mV margin for noise/crosstalk at receiver side, $C_C=500\text{fF}$ is too small to satisfy the sensitivity requirement, and a suitable value is $C_C=1\text{pF}$. 
If larger values of $C_c$ are used (>10pF), NRZ signal is supported by the ACCI channel. Unlike SMT DC-blocking capacitors with sizes of tens of nF or more, the embedded capacitor technology will only support capacitances up to the nF-range. The benefits of relatively smaller (compared with SMT) capacitance are relaxed constraints on the TX driving strength and less power consumption. But the downside comes from the pulse response produced in a channel with these smaller values of capacitance. In Figure 5.5, larger $C_c$ (>5pF) generate larger pulses, as well as much higher and longer tails. Those tails decay depending on the size $C_c$ but with proper constraints they can be used to support NRZ signaling.
Table 5.2. Timing and Circuit Requirement for Various $C_C$.

<table>
<thead>
<tr>
<th>$C_C$</th>
<th>$t_{\text{MaxRunLength}}$</th>
<th>TX Requirement</th>
<th>RX Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>500fF</td>
<td>$-$</td>
<td>Voltage Mode</td>
<td>Pulse Receiver</td>
</tr>
<tr>
<td>1pF</td>
<td>455ps</td>
<td>Voltage Mode</td>
<td>Pulse Receiver</td>
</tr>
<tr>
<td>5pF</td>
<td>2.4ns</td>
<td>Voltage Mode</td>
<td>Pulse Receiver</td>
</tr>
<tr>
<td>10pF</td>
<td>4.3ns</td>
<td>Voltage Mode</td>
<td>NRZ Receiver</td>
</tr>
<tr>
<td></td>
<td>(Or Pulse Receiver)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50pF</td>
<td>21.0ns</td>
<td>Voltage Mode</td>
<td>NRZ Receiver</td>
</tr>
<tr>
<td></td>
<td>(Or Pulse Receiver)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100pF</td>
<td>37.8ns</td>
<td>Voltage Mode</td>
<td>NRZ Receiver</td>
</tr>
<tr>
<td></td>
<td>(Or Pulse Receiver)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2 lists the time for maximum run length in the bit stream when various $C_C$ is used. The maximum run length is calculated as indicated in Figure 5.5. The maximum run length would determine whether NRZ signaling is supported. For example, if 10pF is to be used, the maximum number of consecutive 1’s or 0’s for a 5Gb/s (200ps period) bit stream will be $4.3\,\text{ns}/200\,\text{ps} = 21$; thus, a 8B/10B coding scheme would be a fine choice for NRZ signaling. However, based on a capacitance density of $1705\,\text{pF/cm}^2^{[39]}$, this requires a minimum area of $5.86\times10^5\,\mu\text{m}^2$, or $765\times765\,\mu\text{m}^2$, which is too large for the scheme to be implemented.

A capacitor value between 1pF and 10pF might generate a hybrid (pulse AND NRZ) signaling, which returns to zero for longer consecutive 1’s or 0’s but remains NRZ for short duration of 1’s and 0’s. This seems un-suitable for either pulse or NRZ signaling but in fact can still be adopted as long as the RX has the latch block. In other words, the latch in the pulse receiver (comparator with hysteresis) is capable of recovering not only pulse but also NRZ signal, including the hybrid. The only shortcomings of capacitance within the range
between 1pF and 10pF are the relatively larger area and higher performance requirements on the active equalization. Therefore, a smaller capacitor value (around 1pF) with pulse signaling, which requires a latch block following a moderate strength active equalizer at the RX-side, is a more suitable choice.

5.1.2 Stub Length of the ACCI with Embedded Capacitor

One major difference between decoupling capacitors like [48] and signaling capacitors presented in this paper are the parasitics. In the case of power supply decoupling, all the fringe capacitance are beneficial, while the parasitic capacitance for ACCI signaling will degrade the signal edge rate and swing, as shown in Figure 5.5. Especially when larger $C_C (>10\text{pF})$ is used, the rise time increases as much as 50% and amplitude decreases around 10% due to the presence of parasitic capacitance.

The presented scheme is assumed to share the same embedded capacitor structure with decoupling capacitors, which will be placed in the “die shadow” due to its sensitivity to the relative location to the actual circuit (silicon die) [49]. Therefore, for a next generation package of 10,000 pins with 0.5mm pitch BGA, the routing channel from the chip/package interface to the embedded capacitor could be as long as 5cm. Figure 5.6 compares the $S_{21}$ of channels for different stub lengths. The stub is modeled as a 40ohm 30um-wide channel inside the organic laminate package. It is shown that when $L_{\text{stub}}$ increases from 1cm to 5cm given that the total channel length is fixed at 120cm, the $S_{21}$ does NOT vary by more than
2dB. In other words, the length of the stub does not affect the channel response unless the impedance is badly controlled.

![Simulated S21 plots of the channels with various stub length. The difference in peak is around 2dB.](image)

Figure 5.6. Simulated $S_{21}$ plots of the channels with various stub length. The difference in peak is around 2dB.

### 5.2 Transmitter Design

This scheme focuses on the compatibility with the embedded capacitor and correct operation of the RX-side CT-FSE. The only requirement for the TX is to generate voltage-mode NRZ bit stream. Figure 5.7 shows the schematic of the TX, which is made of progressively sized complementary inverter chains.
5.3 Receiver Design

Figure 5.7. TX is made of a pair of complementary progressively-sized inverter chains.

Figure 5.8. RX block diagram.

Figure 5.8 shows the block diagram of the RX, which consists of a CT-FSE and a high-speed latch. Differential current-mode circuit is used throughout the RX. The CT-FSE is capable of equalizing analog signals, such as pulse signals in this case, and the latch recovers pulse signals into NRZ signals.
5.3.1 Current-Mode CT-FSE

The RX CT-FSE (Figure 5.9) is designed according to the CT-FSE structure (Figure 3.2). The number of taps is 3 (k=3), which means there are 4 delay cells, of which the first one is for self-biasing and the last one is a dummy load for uniform delay. The delay cell has non-fixed delay due to PVT variation although the nominal $\tau$ value is around 120ps from preliminary simulation. Therefore, it is important to prove that the CT-FSE with 3 taps and 3-bit resolution is capable of cancelling the effects of the varied $\tau$. As demonstrated in Figure 5.10, the CT-FSE Synthesizer in Matlab optimizes the tap weights ($a_k$) for different tap delays ($75\text{ps} < \tau < 225\text{ps}$) and shows open 5Gb/s eyediagrams for all of them.

![Block diagram of the RX CT-FSE.](image)

Figure 5.9. Block diagram of the RX CT-FSE.
Figure 5.10. Matlab simulation results shows the open 5Gb/s eyediagrams of the CT-FSE with tap delay ($\tau$) varies from 75ps to 225ps.

Figure 5.11 and Figure 5.12 depicts the current-mode logic (CML) realization of the CT-FSE in transistor-level schematic. The delay cell, as shown in Figure 5.11, is a differential pair with source degeneration. On the other hand, although resembling a differential amplifier, the current-mode digital-to-analog converter (DAC) is actually a simple common-source amplifier with source degeneration because only one of the two NMOS switches would be ON, which is set by controlled logic (function shown in bottom right of Figure 5.11), where $k$ is for branch number, and $i$ indicates the $i^{th}$ bit to set the $k^{th}$ tap weight. The common-mode feedback (CMFB) block, as shown in Figure 5.12, utilizing Gilbert cell to
control current source and current sink for each output nodes, consequently, maintaining the common-mode voltage and biasing for the following latch.

![Diagram of Delay Cell and Current-Mode DAC](image)

**Delay Cell**
- Diff. pair with source degen. and resistor load
- 3 taps → 4 delay cells (inc. Bias)

**Current-Mode DAC**
- C.S. amp. w/ source degen.
- 3-bit resolution (i=0-7)
  → 7 DAC per tap
- $\text{Sign}_k/a_k$ change thru switch
  
  $$f'(A, B) = (A \oplus B) \bar{B}$$

Figure 5.11. Detail of the building blocks in RX CT-FSE.

![Diagram of CMFB and Gilbert Cell](image)

**CMFB**
- Gilbert Cell
- $(\text{OutP}+\text{OutN})/2=\text{Vcm}$

Figure 5.12. The common-mode feedback circuit utilizing Gilbert cell to maintain CT-FSE output common-mode voltage.
5.3.2 High-Speed Latch

Figure 5.13. The high-speed latch is a non-clocked regenerative comparator with common-source pre-amplifier.

The high-speed latch adopted from [19] is a non-clocked regenerative comparator with common-source pre-amplifier, as shown in Figure 5.13. The input biasing is set by the Vcm via the CMFB block (Figure 5.12).

5.4 Design Flow

The design flow of this research task is depicted in Figure 5.14. There are three research domains in which analysis and design are done: physical domain, electrical domain, and behavior domain; the major working environments for these domains are electromagnetic (EM) simulation software and calculator, HSPICE, and Matlab, respectively.

The first step is to generate channel models for package and PCB, and estimate available capacitor value from literature. The system level environment is designed according to , which also needs to be setup while keeping its flexibility, including variable stub length,
channel length, and capacitor value. Then, a preliminary simulation in HSPICE, including only an ideal TX driver and the channels, would generates RX inputs (with ISI), which is subsequently processed by the CT-FSE Synthesizer in Matlab to obtain necessary parameters (k, R, and acceptable τ) for the CT-FSE design (Figure 5.9). After determining the CT-FSE structure and the system level parameters from the preliminary results, the circuitry can be designed block by block, from back to front (RX to TX). Several iterations between preliminary work and actual circuit design are needed because of the non-ideality of the current-mode circuit.

Figure 5.14. Design flow of the system (RX CT-FSE with embedded capacitor)
5.5 Simulation Results

Figure 5.15. Time-domain waveforms of the delayed RX inputs.

Figure 5.16. Comparison of the waveforms before and after CT-FSE is turned ON.
Figure 5.17. The CT-FSE eliminate most ISI in RX input, and the latch recovers the signal back to 3.33Gb/s NRZ signals.

This section shows the post-layout simulation results. The waveforms of the delayed RX inputs (through delay cells) are shown in Figure 5.15. The tap delay (τ) is around 120ps according to post layout simulation. The delayed cell has a gain slightly less than 1, but this is compensated by the adjustable gain (tap weights) in DAC. To demonstrate the operation of the RX CT-FSE, Figure 5.16 shows that CT-FSE is capable of eliminating long tails and generating pure pulse signals for the following high-speed latch. The eyediagrams of the RX input, CT-FSE output (pulse signaling) and the latch output (NRZ signaling) are shown in Figure 5.17.
The detail steady current consumption for each RX block is indicated in Figure 5.18. The comparison between this work and other TRX performances is listed in Table 5.3. Combining the passive equalization from the embedded capacitor and the active equalization from the RX side CT-FSE, this work has lower power consumption than designs with similar structure [29] and [30]. Comparing the RX-side CT-FSE used here with conventional TX-side FFE implementation, there are two aspects to look at. First, the analog delay cells in RX-side CT-FSE consumes more power than their counterpart in TX-side FFE, which are only flip-flops. But the analog delay cell in RX-side CT-FSE enables higher transmission bandwidth. Second, the TX-side FFE usually consumes significant amount of power due to the major parasitic capacitance at the current summing node since it combines current-mode digital-to-analog converter (DAC) with the channel output driver and (separating these two functions would result in a penalty of increased circuit complexity and power consumption).
On the other hand, the RX-side CT-FSE enables a very simple and efficient voltage-mode driver and moves the current summing node to the RX side, just before the pulse latch. Last but not least, a CT-FSE does not require clock distribution. In contrast, the TX-side FFE do need to distribute high-speed clock to drive the flip-flops and subsequent predrivers. To summarize, overall this work has much better power efficiency than other designs with TX-side FFE, due to simple circuitry, separation of current-mode DAC and TX drivers, and the topology the requires no high-speed clock distribution.

<table>
<thead>
<tr>
<th>Technology</th>
<th>This work</th>
<th>[25]</th>
<th>[24]</th>
<th>[29]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel Loss@ half bit-rate</td>
<td>14dB</td>
<td>40dB</td>
<td>21.3dB (by TX)</td>
<td>Est. 18dB</td>
</tr>
<tr>
<td>TX Power</td>
<td>5mW</td>
<td>150mW</td>
<td>221mW</td>
<td>–</td>
</tr>
<tr>
<td>RX Power</td>
<td>19.2mW</td>
<td>90mW</td>
<td>210mW</td>
<td>22.25mW</td>
</tr>
<tr>
<td>Total Power Consumption</td>
<td>24.2mW</td>
<td>240mW</td>
<td>431mW</td>
<td>–</td>
</tr>
<tr>
<td>Speed</td>
<td>3.33Gb/s</td>
<td>6.4Gb/s</td>
<td>6.25Gb/s</td>
<td>1Gb/s</td>
</tr>
<tr>
<td>Power Efficiency</td>
<td>7.26mW/Gb/s</td>
<td>23.43mW/Gb/s</td>
<td>35.36mW/Gb/s</td>
<td>22.25mW/Gb/s <em>1</em>3</td>
</tr>
<tr>
<td>TRX Config.</td>
<td>RX CT-FSE (k=3, R=3)</td>
<td>TX FIR (k=5, R=?)</td>
<td>TX FIR (k=4, R=4)</td>
<td>RX CT-FSE (k=6, R=5)</td>
</tr>
<tr>
<td></td>
<td>Passive EQ</td>
<td>RX 2nd order EQ</td>
<td>RX DFE</td>
<td></td>
</tr>
</tbody>
</table>

*1No adaptive logic. No multiplexer. *2Only TX is considered. *3Only RX is considered.
Chapter 6

Conclusions and Future Works

6.1 Conclusions

The theory of CT-FSE was introduced in Chapter 3 with equations to describe the behavior of the equalizer. This theory shows the relationship between the continuous-time delay ($\tau$), the number of taps ($k$), tap weights ($a_k$), and the final synthesized transfer function, $H(\omega)$. The continuous-time delay ($\tau$) adds one more degree of freedom on the orthogonal set on which the transfer function is expanded on. The two key observations for designing with $\tau$ as a parameters are 1.) Smaller $\tau$ will result in more CT-FSE bandwidth and 2.) Larger $\tau$ means the CT-FSE can achieve sharper filtering response. The dilemma of a small or large $\tau$ can be solved the same way as the tradeoff between power and performance is done. Moreover, the Matlab CT-FSE synthesizer was developed to perform simulations in frequency- and time-domain to prove the synthesizability and robustness of replacing high-speed flip-flops with delay cells with non-fixed delay in FFE. The Matlab programs also allow for the estimation of the circuit parameters.
Chapter 4 introduces MultiCap, a zero power consumption voltage-summing structure. The analysis for geometries and other design parameters shows equations to determine the useable range of the capacitance as well as the impact of the parasitic. With the realistic value of the capacitance, this MultiCap structure is combined with CMOS tap-delay branches to realize voltage-mode TX CT-FSE. This novel concept solves several problems of current-summing in conventional FFE. The voltage-mode circuit is by its nature robust and easily scalable.

Figure 6.1. Comparison of power efficiency (mW/Gb/s) and topology.

Chapter 5 investigated a interconnect scheme combining the active equalization from CT-FSE and the passive equalization from the package-embedded capacitor. The investigation
and model analysis of the embedded capacitor shows that it is feasible for embedded capacitor as part of a signaling scheme. The challenge of matching a precise capacitor value to channel response in conventional ACCI is relaxed by introducing a CT-FSE current-mode receiver to the system. Although implemented in CML, the CT-FSE implemented at RX-side has the benefit of saving power consumption when compared with conventional CML FFE. The power savings are mainly from simple circuitry, elimination of the clock related circuit, and the separation of the summing block from the driver.

The bar chart in Figure 6.1 compares the power efficiency (mW/Gb/s) of the designs presented in this dissertation and other literature. It shows the TX CT-FSE design presented in Chapter 4 has power advantage of 92% and 95% when compared with the current-mode FSE receiver in [29] and TX FFE by Payne et al. [24], respectively. Figure 6.1 also shows the RX CT-FSE design presented in Chapter 5 has a power savings of 67% when compared with similar design by Lin and Liu [29]. In general, the power efficiency of the CT-FSE design implemented with 130nm technology, are comparable to state-of-the-art designs using 90nm technology nodes and beyond.
6.2 Future Works

6.2.1 CT-FSE with Adjustable Delay $\tau$

From Chapter 2 we know that the tap delay ($\tau$) has influence on both the bandwidth and the sharpness of the filter response. For the TX-side CT-FSE, the tap delay is realized with a pair of back-to-back inverters and leaves no control over the actual value of delay. It might be beneficial to add delay control functions for the digital delay elements, as shown in Figure 6.2, by utilizing simple topologies from the ring oscillator in PLL or CDR, such as the current starved design presented by Redman-White et al. [56].

Figure 6.2. Block diagram of the CT-FSE with adjustable delay elements.
6.2.2 CT-FSE with Non-Uniform Delay

To further generalize the CT-FSE theory in Chapter 3, one can assume a structure that has non-uniform delay. As shown in Figure 6.3, the delay of each stage can be denoted as $\tau_1$, $\tau_2$, to $\tau_k$.

![Block diagram of the CT-FSE with non-uniform delay $\tau_k$.](image)

The time domain output, $y(t)$, can be written as:

$$y(t) = \sum_{k} a_k x(t - \tau_k) \quad (6.1)$$

Here $\tau_k$ represents the different delay for each taps. Perform continuous-time Fourier transform on both sides of (6.1) and we obtain:

$$Y(\omega) = \sum_{k} a_k X(\omega) e^{-j\omega \tau_k} \quad (6.2)$$

Moving $X(\omega)$ out of the summation term and we can observe that the transfer function, $H(\omega)$, of the CT-FSE structure shown in Figure 6.3 can be defined as:
\[ H(\omega) = \sum_k a_k e^{-j\omega \tau_k} \]  \hfill (6.3)

It is clear there is more complicated mathematics involved in calculating the inverse relationship between \(a_k\) and \(H(\omega)\). If this can be done, however, there will be a set of equations to describe any FIR filter, with any delay at each tap. And this might allow for more efficient design in the FFE.

### 6.2.3 Method to Design High-Speed Latch with Adjustable Hysteresis

The most critical component for pulse signaling is the RX latch, of which the input/output transfer function is shown in Figure 6.4. There has not been a well-organized method about how to design a latch. Nor was there a good comparison between all the non-clocked regenerative latches. The relationship between the circuit topology, transistor parameters, and the latch characteristic, such as the sensitivity (\(\delta\)) shown in Figure 6.4. Although it is intuitive to choose appropriate topology and adjust the transistor sizing, the

Figure 6.4. The input/output voltage transfer function of a latch with adjustable hysteresis.
whole process is not fully explained. There is a need to: 1.) formulate performance matrices for a latch, including at least power consumption, sensitivity (δ), and speed, and 2.) further investigate the operation of the latch, including how the hysteresis is formed and through what parameter it can be adjusted. For the latch used in the section 5.3.2, a first order analysis should be performed following the process used to analyze a Schmitt trigger.
REFERENCES


103


APPENDICES
Appendix A  CT-FSE Synthesizer: Instructions and Matlab Programs (.m files)

A.1  Frequency-Domain Analysis then Synthesis: Sweep Filter Corner Freq.

In the following program, the corner frequency of the ideal transfer function $H_f(\omega)$ is swept from 1GHz to 5GHz (for `FilterEdgeFreq = [1e9:1e9:5e9]`). Following the process shown in the top of Figure 3.9, the result would be a plot with $H_f(\omega)$ (black dash lines) overlaying $H_N(\omega)$ (red solid line), as shown below. Notice here the continuous-time delay ($\tau$) is assumed to be 74ps.

![Plot](image.png)
A.1.1 CT_FSE_Sweep_Filter.m

%% Calculate and compare
%%% Ideal Filter Response <===> Synthesized CT-FSE Coefficients
%%% Use numerical integral to calculate the coefficients
%%%        f(x) =             \sum_{-\infty}^{\infty} \{c_N \cdot \exp(jNx)\}
%%%        c_N  = 1/(2\pi)* \int_{-\pi}^{\pi} \{f(x)\cdot\exp(-jNx)\}
%%% I call it "Fourier-like" expansion because
%%% 1. It's in frequency domain
%%% 2. It uses \exp(-j\omega T) instead of \exp(j\omega T) as orthogonal functions
%%% 3. It has the Tdelay term to scale up/down frequencys
%%% Here the ideal filter response scale with 1/Tdelay
%%%

clear
clc
%%% Choose the transfer function to be expanded
FunctionNumber=4;

%%% Set number of terms of the Fourier Expansion
Ntaps  = 4;
Tdelay = 74e-12; % pico-seconds

%%% Set \omega-resolution
Npoints = 1000001;

figure(111) %Mag
set(gca,'xscale','log')
% figure(222)
% set(gca,'xscale','log')
% set(gca,'yscale','log')
LW=0;
for FilterEdgeFreq = [1e9:1e9:5e9]
%%% Set \omega-index vector
w = linspace(-pi/Tdelay,pi/Tdelay,Npoints);

%%% Define \text{H}(\omega) %others might call it \text{H}(j\omega), same thing
if FunctionNumber==1 %HPF@f=0
    H_w(1,1:(Npoints-1)/2)           = -1*ones(1,(Npoints-1)/2);
    H_w(1,(Npoints-1)/2+1)           = 0;
    H_w(1,((Npoints-1)/2+2):Npoints) = ones(1,(Npoints-1)/2);
elseif FunctionNumber==2%Differentiator
    H_w(1,1:(Npoints-1)/2+1)         = linspace(-1,0,(Npoints-1)/2+1);
    H_w(1,((Npoints-1)/2+1):Npoints) = linspace(0,1,(Npoints-1)/2+1);
elseif FunctionNumber==3 %HPF. Even function
    [min1 loc1]=min(abs(w(1:(Npoints-1)/2))/(2*pi)+FilterEdgeFreq);
    H_w=zeros(1,Npoints);
    H_w(1,1:loc1)=ones(1,loc1);
    H_w(1,(end-loc1+1):end)=ones(1,loc1);
end
elseif FunctionNumber==4 %LPF. Odd function
    [min1, loc1]=min(abs(w(1:(Npoints-1)/2)/(2*pi)+FilterEdgeFreq));
    H_w=ones(1,Npoints);
    H_w(1,1:loc1)=zeros(1,loc1);
    H_w(1,(end-loc1+1):end)=zeros(1,loc1);
elseif FunctionNumber==5 %BPF
    FilterEdgeFreq1=1e9; %set the location of the filter transition edge
    FilterEdgeFreq2=0.5e9;
    [min1, loc1]=min(abs(w(1:(Npoints-1)/2)/(2*pi)+FilterEdgeFreq1));
    [min2, loc2]=min(abs(w(1:(Npoints-1)/2)/(2*pi)+FilterEdgeFreq2));
    H_w=zeros(1,Npoints);
    H_w(1,loc1:loc2)=ones(1,loc2-loc1+1);
    H_w(1,(end-loc2+1):(end-loc1+1))=ones(1,loc2-loc1+1);
end

%% Calculate w-bin width
dw_wid = (2*pi/Tdelay)/(Npoints-1);
dw = dw_wid*ones(Npoints,1);

Stmp=zeros(Ntaps+1,Npoints);
c=zeros(1,Ntaps);
c_N=zeros(1,Ntaps);
% c_N_backup=zeros(1,Ntaps);

%% Calculate CT-FSE coefficients and Synthesize
for i=(0:1:Ntaps)
    if i==0
        c0 = Tdelay/(2*pi)*H_w*dw;
        exp_N_jwNT = exp(-1i*w*i*Tdelay);
        Stmp(i+1,:) = c0*exp_N_jwNT;
    else
        exp_N_jwNT = exp(-1i*w*i*Tdelay);
        exp_jwNT = exp(1i*w*i*Tdelay);
        c(1,i) = Tdelay/(2*pi)*H_w.*exp_jwNT*dw;
        c_N(1,i) = Tdelay/(2*pi)*H_w.*exp_N_jwNT*dw;
        Stmp(i+1,:) = c(1,i)*exp_N_jwNT + c_N(1,i)*exp_jwNT + Stmp(i,:);
    end
end
S = abs(Stmp); %%%%%%%%%%% Important
%S(1,:) ==> c0 term
%S(2,:) ==> c_+1 and C_-1 term
%S(3,:) ==> c_+2 and C_-2 term
%S(N+1,:) ==> c_+N and C_-N term
figure(111)
hold on
plot(w/(2*pi),S(2,:),'c', 'LineWidth',2)
plot(w/(2*pi),S(4,:),'g', 'LineWidth',2)
LW=LW+1;
plot(w/(2*pi),db(abs(S(end,:))/max(abs(S(end,:)))),'r', 'LineWidth',1)
plot(w/(2*pi),H_w*40-40,'k--','LineWidth', 1)
legend('H(w)', 'S_1', 'S_3', 'S_7')
axis([1e7, 1e10, -20.5, 0.5])
hold off
figure(222)
plot(w(1:end-1)/(2*pi),diff(S(:,end)),'r', 'LineWidth',2)
hold on
end
figure(111)
axis([1e7, 1e10, -40.5, 0.5])
set(gcf, 'Color','white')
hold on
plot(w/(2*pi),H_w,'--K','LineWidth', 2)
hold off

index_series=(-1*Ntaps:1:Ntaps);
c_series=[flip(c_N), c0, c];
figure
hold on
bar(index_series, real(c_series),'r','LineWidth', 2)
%bar(index_series, imag(c_series),'b','LineWidth', 2)
bar(index_series(1:length(c_N_backup)), flip(c_N_backup))%,'--r','LineWidth',2)
%legend('Approximated{C}', 'Imag{C}', 'Leftover{C}')
hold off
A.2 Frequency-Domain Analysis then Synthesis: Sweep Tdelay

In the following program, the continuous-time delay (τ) of CT-FSE is swept from 50ps to 200ps, besides 10ps (for $T_{delay} = [10e-12, 50e-12:10e-12:200e-12]$). Following the process shown in the top of Figure 3.9, the result would be a plot with $H_I(\omega)$ (black dash lines) overlaying $H_N(\omega)$ (red solid line), as shown in Figure 3.7 (left). Notice here the target ideal transfer function, $H_I(\omega)$, is an ideal low-pass filter with corner frequency at 1GHz.
A.2.1  CT_FSE_Sweep_Tdelay.m

clear
clc

%% Choose the transfer function to be expanded
FunctionNumber=4;

%% Set number of terms of the Fourier Expansion
Ntaps  = 4;
Tdelay = 74e-12; % 100 pico-seconds

%% Set w-resolution
Npoints = 1000001;

figure(111) %Mag
set(gca,'xscale','log')
figure(222)
set(gca,'xscale','log')
set(gca,'yscale','log')

LW=0;

for FilterEdgeFreq = [1e9:1e9:5e9]

%% Set w-index vector
w = linspace(-pi/Tdelay,pi/Tdelay,Npoints);

%% Define H(w) %others might call it H(jw), same thing
if FunctionNumber==1 %HPF@f=0
    H_w(1,1:(Npoints-1)/2)           = -1*ones(1,(Npoints-1)/2);
    H_w(1,(Npoints-1)/2+1)           = 0;
    H_w(1,((Npoints-1)/2+2):Npoints) = ones(1,(Npoints-1)/2);
elseif FunctionNumber==2 %Differentiator
    H_w(1,1:(Npoints-1)/2+1)         = linspace(-1,0,(Npoints-1)/2+1);
    H_w(1,((Npoints-1)/2+1):Npoints) = linspace(0,1,(Npoints-1)/2+1);
elseif FunctionNumber==3 %HPF. Even function
    [min1 loc1]=min(abs(w(1:(Npoints-1)/2)/(2*pi)+FilterEdgeFreq));
    H_w=zeros(1,Npoints);
    H_w(1,1:loc1)=ones(1,loc1);
    H_w(1,(end-loc1+1):end)=ones(1,loc1);
elseif FunctionNumber==4 %LPF. Odd function
    [min1 loc1]=min(abs(w(1:(Npoints-1)/2)/(2*pi)+FilterEdgeFreq));
    H_w=ones(1,Npoints);
    H_w(1,1:loc1)=zeros(1,loc1);
    H_w(1,(end-loc1+1):end)=zeros(1,loc1);
elseif FunctionNumber==5 %BPF
    FilterEdgeFreq1=1e9; %set the location of the filter transition edge
    FilterEdgeFreq2=0.5e9;
    [min1 loc1]=min(abs(w(1:(Npoints-1)/2)/(2*pi)+FilterEdgeFreq1));
    [min2 loc2]=min(abs(w(1:(Npoints-1)/2)/(2*pi)+FilterEdgeFreq2));
    H_w=zeros(1,Npoints);
    H_w(1,loc1:loc2)=ones(1,loc2-loc1+1);
    H_w(1,(end-loc2+1):end)=ones(1,loc2-loc1+1);
end
%% Calculate w-bin width
dw_wid = (2*pi/Tdelay)/(Npoints-1);
dw     = dw_wid*ones(Npoints,1);

Stmp=zeros(Ntaps+1,Npoints);
c=zeros(1,Ntaps);
c_N=zeros(1,Ntaps);
% c_N_backup=zeros(1,Ntaps);

%% Calculate CT-FSE coefficients and Synthesize
for i=(0:1:Ntaps)
  if i==0
    c0          = Tdelay/(2*pi)*H_w*dw;
    exp_N_jwNT  = exp(-1i*w*i*Tdelay);
    Stmp(i+1,:) = c0*exp_N_jwNT;
  else
    exp_N_jwNT  = exp(-1i*w*i*Tdelay);
    exp_jwNT    = exp(1i*w*i*Tdelay);
    c(1,i)    = Tdelay/(2*pi)*(H_w.*exp_jwNT)*dw;
    c_N(1,i)  = Tdelay/(2*pi)*(H_w.*exp_N_jwNT)*dw;
    Stmp(i+1,:) = c(1,i)*exp_N_jwNT + c_N(1,i)*exp_jwNT + Stmp(i,:);
    % Stmp(i+1,:) = c(1,i)*exp_N_jwNT + Stmp(i,:);
    %force the k<0 term to be zero
  end
end

S = abs(Stmp);  % Important
%S(1,:) --> c0 term
%S(2,:) --> c_+1 and C_-1 term
%S(3,:) --> c_+2 and C_-2 term
%S(N+1,:) --> c_+N and C_-N term

figure(111)
hold on
%      plot(w/(2*pi),S(2,:),'c', 'LineWidth',2)
%      plot(w/(2*pi),S(4,:),'g', 'LineWidth',2)
LW=LW+1;
plot(w/(2*pi),db(abs(S(end,:))/max(abs(S(end,:)))),'r', 'LineWidth',1)
plot(w/(2*pi),H_w*40-40,'k--','LineWidth', 1)
%legend('H(w)', 'S_1', 'S_3', 'S_7')
ax = axis([1e7, 1e10, -40.5, 0.5])
hold off
% figure(222)
% plot(w(1:end-1)/(2*pi),diff(S(end,:)),'r', 'LineWidth',2)
% hold on

end

figure(111)
axis([1e7, 1e10, -40.5, 0.5])
set(gcf, 'Color','white')
% hold on
% plot(w/(2*pi),H_w,'--K','LineWidth', 2)
% hold off
index_series=(-1*Ntaps:1:Ntaps);
c_series=[fliplr(c_N), c0, c];
figure
hold on
bar(index_series, real(c_series), 'r', 'LineWidth', 2)
% bar(index_series, imag(c_series), 'b', 'LineWidth', 2)
bar(index_series(1:length(c_N_backup)), fliplr(c_N_backup), '--r', 'LineWidth', 2)
% legend('Approximated{C}', 'Imag{C}', 'Leftover{C}')
hold off
A.3 Time-Domain Parameter Optimization (for Pulse Signaling)

This program follows the process shown in bottom of Figure 3.9 to find the optimized tap weights, \( a_k \), for up to 10 different channel outputs.

The inputs to the main function (A.3.2) are time domain waveforms with two columns; first column indicates time, and second column indicates signal amplitude (voltage in our case). The format is the .print output of HSPICE listing file (.lis). The waveform files are read into the main function by:

```matlab
VoutName(1,1,:) = {'L15cm.lis'}; % using new extracted TX. (old PDK patch)
VoutName(2,1,:) = {'L30cm.lis'}; % using new extracted TX. (old PDK patch)
VoutName(3,1,:) = {'L45cm.lis'}; % using new extracted TX. (old PDK patch)
...
```

As an example, the first 5 rows of the “L15cm.lis” looks like:

0.7526392e-05
1.000000e-12 7.526392e-05
2.000000e-12 7.526392e-05
3.000000e-12 7.526392e-05
4.000000e-12 7.526393e-05

The second column (voltage) remains close to zero because the first bit has not arrived at the channel output. Notice for each channel output, the corresponding sub-directory needs to be created beforehand and match those names assigned by:

```matlab
TLLength(1,1,:) = {'L15cm'};
TLLength(2,1,:) = {'L30cm'};
TLLength(3,1,:) = {'L45cm'};
```

In order for successful operation, the trip time of each channel need to be entered, so that the main function knows when the first bit starts. The trip time is assigning values to the matrix, `TripTime[]`, by:

```matlab
TripTime(1,1) = 1.37e-9; % Trip time across the transmission line (15cm)
TripTime(1,2) = 2.36e-9; % Trip time across the transmission line (30cm=1.96e-9)
TripTime(1,3) = 3.13e-9; % Trip time across the transmission line (45cm)
```
A vector TL_Len_index[] defines which channel output to work on. In this example, only channel output 4 is picked (which is the 'L60cm.lis'):

\[
\text{for TL_Len_index=(4:1:4)}
\]

More channel outputs can be selected. For instance, to enable optimization on all 10 channel outputs:

\[
\text{for TL_Len_index=(1:1:10)}
\]

For each channel output, before optimizing tap coefficients, the main function first varies the continuous-time delay (τ) by assigning values to the array Tdelay. In this example, only one value is assigned.

\[
\text{for Tdelay=(84:5:84) \% Sweep Tdelay}
\]

More values of τ can be assigned. For instance, to sweep τ from 75ps to 225ps in 50ps step:

\[
\text{for Tdelay=(75:50:225)}
\]

The core of the main function sweeps every possible combination of tap coefficients, then sorts the result according to the error term. There is actually no “for loop” used the coefficient sweep, instead, it uses the “ndgrid” command to expand every combination into a giant matrix. Therefore, the delay-weight-sum is performed in one step of matrix multiplication:

\[
\text{FIR_out=Coef_Comb*Delay_Out;}
\]

A sub-function, smartlatch_new.m (A.3.3), is called before calculating the error term. This function acts as an ideal RX and decodes the CT-FSE output into digital bit streams.

The outputs of this program, including output waveforms, eyediagrams, and coefficients ranking, are saved in the sub-directory assigned by TLLength.

More details are in the comments.
A.3.1 Output files

The output of this program includes:

- Top ranked eyediagram annotated with tdelay and tap coefficients. For example:

![Eyediagram](image)

- A list of top ranked tap coefficients. The first few rows look like:

<table>
<thead>
<tr>
<th>Rank</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>-4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>-4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>-2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>-7</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>-5</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
A.3.2  CT_FSE_Optimizer_Pulse.m

clc
clear
tic

% This routine optimize the combination of tap coefficients given only
% the receiver pulse-signal inputs
%
% This routine can optimize tap coefficients for up to 10 channels (for TL_len_index=(1:1:10))
%
% [Continuous-Time Fractionally-Spaced Equalization (CT-FSE) = a FIR-like structure]
% The Finite-Impulse-Response (FIR) filter implemented in this routine
% actually has adjustable tap delay (Tdelay)
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % System Block Diagram %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% \
% | \ | \-------|-----------------|---|-----|--/
% | /   | Delay|
% | /   | & Sum|
% | /   |-----|-----------------|---|-----|--/
% ^   ^<-----------^ ^
% |   | Tline length |   |   |
% Vin | Vout     |   |   |
% % Delay&Sum block (FIR core) has 4 taps = 0,1,2,3
% %
% % This routine
% % (1) Calculate Vout_fir based on the given FIR coefficients
% % (1.1) Create "Delayed versions"
% % (1.2) Expand the combination of Tap Coefficients
% % (1.3) Calculate all the FIR_out in one matrix multiplication
% % (2) Pass Vout_fir thru an ideal rx (smart_latch)
% % (3) Compare the results between Vout_fir and smart_latch
% %
% % Note that each channel length has a number assigned to it.
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Constants %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ** Important Constants
% TimeResolution=1e-12; % This should match the HSPICE .print time resolution
% Nsamples = 100; % Number of samples per period

PERIOD = Nsamples*TimeResolution;
RX_sensitivity=0.02;

Latch_tolerenc=20; % in percentage, at sampling point, if Vout>+- Latch_tolerenc, the
latch changes
PulseSwitch=1; %1: Ideal Pulse for Ideal Latch Output = Half Sine Wave
PulseSwitch=2; %2: Ideal Pulse for Ideal Latch Output = Square Wave

RankingList=30; % According to error term, op 30 coefficient sets saved.

% ** Not So Important Constants for plotting
OFFSETin = 0; % Input offset in number of samples, for Eyediagram
OFFSETout = 0; % Output offset in number of samples, for Eyediagram

Tdelay=100; % for IBM Aug 2008 run

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ====== Set Tap Coefficient Sweep Ranges ==
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ** CoefMax indicates the Max/Min value of Tap Coefficients
%CoefMax=zeros(1,10,'int8'); %Data type = integer

% CoefMax(1,1) = 2;
% CoefMax(1,2) = 2;
% CoefMax(1,3) = 4;
% CoefMax(1,4) = 4;
% CoefMax(1,5) = 5;
% CoefMax(1,6) = 7;
% CoefMax(1,7) = 7;
% CoefMax(1,8) = 7;
% CoefMax(1,9) = 7;
% CoefMax(1,10) = 7;
% CoefMax(1,1:10)=CoefMax(1,1:10)/7;

CoefMax(1,1:10)=ones(1,10,'single'); % Overwrite all of them

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% =============== Set Trip Time ============
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ** TripTime is an array used to store corresponding TLine trip time
% ** This value is set so that the beginning of the bit stream can align with
%TripTime = zeros(1,10,'single'); % Initialize

TripTime(1,1) = 1.37e-9; % Trip time across the transmission line (15cm)
TripTime(1,2) = 2.36e-9; % Trip time across the transmission line (30cm=1.96e-9)
TripTime(1,3) = 3.13e-9; % Trip time across the transmission line (45cm)
TripTime(1,4) = 4.03e-9; % Trip time across the transmission line (60cm)
TripTime(1,5) = 4.90e-9; % Trip time across the transmission line (75cm)
TripTime(1,6) = 6.37e-9; % Trip time across the transmission line (100cm)
TripTime(1,7) = 7.83e-9; % Trip time across the transmission line (125cm)
TripTime(1,8) = 9.296e-9; % Trip time across the transmission line (150cm)
TripTime(1,9) = 10.76e-9; % Trip time across the transmission line (175cm)
TripTime(1,10) = 12.23e-9; % Trip time across the transmission line (200cm)

TripTime = round(TripTime/TimeResolution)*TimeResolution; % Round the trip time to pico-second
%%------------------------------------------
% ============= Set File Name ==============
% ------------------------------------------
% ** VoutName is a string matrix for load-in filename(may include path).
% The file linked has two columns - time and voltage - that comes from
% HSPICE simulation (.print VRX_IN_Diff)
VoutName(1,1,:) = {'L15cm.lis'}; % using new extracted TX. (old PDK patch)
VoutName(2,1,:) = {'L30cm.lis'}; % using new extracted TX. (old PDK patch)
VoutName(3,1,:) = {'L45cm.lis'}; % using new extracted TX. (old PDK patch)
VoutName(4,1,:) = {'L60cm.lis'}; % using new extracted TX. (old PDK patch)
VoutName(5,1,:) = {'L75cm.lis'}; % using new extracted TX. (old PDK patch)
VoutName(6,1,:) = {'100cm_extracted2.lis'}; % new design 2008
VoutName(7,1,:) = {'125cm_extracted.lis'}; % using new extracted TX. (old PDK patch)
VoutName(8,1,:) = {'L150cm.lis'}; % using new extracted TX. (old PDK patch)
VoutName(9,1,:) = {'L175cm.lis'}; % using new extracted TX. (old PDK patch)
VoutName(10,1,:) = {'L200cm.lis'}; % using new extracted TX. (old PDK patch)

%%------------------------------------------
% ============= Load in Vin ================
% ------------------------------------------
% ** This Vin is the ideal bit stream
Vin=load('IN_TLIN_TLOUT_30cm_500f_10G.txt'); %10Gbps, 1.2V
Vin=[Vin(:,1), Vin(:,2)]; % Delete extra rows
Vin(:,2)=-Vin(:,2); % for correct polarity

%%------------------------------------------
% ========= Alter TLine Lengths ============
% ------------------------------------------
% ** Load different RX inputs waveforms generated from different TL length
% for TL_Len_index=(1:1:10)
% warning('MATLAB:Reminder', ...
% ['\n Please make sure subdirectory: \', char(TLLength(TL_Len_index,1,:)), ' is
created'])
% =********* Load RX_IN into Vout =*********
Vout=load(char(VoutName(TL_Len_index,1,:))); % Delete extra rows
Vout=single([Vout(:,1), Vout(:,2)]);
% =********* Performing time offset =*******
% Locate the location of the row that matches the TripTime for that specific length
locshift=find(Vout(:,1)==TripTime(1,TL_Len_index));
% Clear the zero data, and trim to period
Vout=Vout(locshift:end,:);
Vout=Vout(1:Nsamples*floor(size(Vout,1)/Nsamples),:);

% Shift in time-domain, make first point correspond to zero time
TimeAxis = single(Vout(:,1)-Vout(1,1));
TimeAxis = round(TimeAxis/TimeResolution)*TimeResolution;

% ========= Initial Plots =========
% Plot In(t) and Out(t) on same figure
% h1=figure;
% plot(Vin(:,1), Vin(:,2)/1.8*max(abs(Vout(:,2))), 'g');  % Plot Vin with normalized amplitude
% hold on
% plot(Vout(:,1), Vout(:,2)/max(abs(Vout(:,2))), 'b');  % Plot Vout with normalized amplitude
% hold off

% Plot Eyediagram of TxIN and RxIN
% eyediagram(Vin(:,2), Nsamples, PERIOD, OFFSETin)
% eyediagram(Vout(:,2), Nsamples, PERIOD, OFFSETout)

%========================================
% ============ Sweep Tdelay =============
% =========================================
for Tdelay=(84:5:84) % Sweep Tdealy
  % (1) Delay & Sum (FIR)
  % Initializing Variables: [VoutX] are column vectors
  Len_vout=size(Vout,1);
  Vout0=zeros(Len_vout,1,'single');
  Vout1=zeros(Len_vout,1,'single');
  Vout2=zeros(Len_vout,1,'single');
  Vout3=zeros(Len_vout,1,'single');

  % (1.1) Perform Delay
  Vout0=Vout(:,2); % No delay
  Vout1(1+Tdelay:end,1)=Vout0(1:end-Tdelay); % Delay of 1xTdelay
  Vout2(1+Tdelay:end,1)=Vout1(1:end-Tdelay); % Delay of 2xTdelay
  Vout3(1+Tdelay:end,1)=Vout2(1:end-Tdelay); % Delay of 3xTdelay

  % Combined into one Delay_Out matrix (for multiplication later)
  Delay_Out=[Vout0'; Vout1'; Vout2'; Vout3'];

  clearvars Vout0 Vout1 Vout2 Vout3

  % (1.2) Expand Tap Coefficient Matrices
  % Define Tap Coefficients
  A0=(CoefMax(1,TL_Len_index):1/7:CoefMax(1,TL_Len_index));
  A1=(-CoefMax(1,TL_Len_index):1/7:0));
% A2=(-CoefMax(1,TL_Len_index):1/7:CoefMax(1,TL_Len_index));
% A3=(-CoefMax(1,TL_Len_index):1/7:CoefMax(1,TL_Len_index));
A0=(0:1/7:CoefMax(1,TL_Len_index));
A1=(-CoefMax(1,TL_Len_index):1/7:CoefMax(1,TL_Len_index));
A2=(-CoefMax(1,TL_Len_index):1/7:CoefMax(1,TL_Len_index));
A3=(-CoefMax(1,TL_Len_index):1/7:CoefMax(1,TL_Len_index));
% A0=(CoefMax(1,TL_Len_index):1/7:CoefMax(1,TL_Len_index));
% A1=(-CoefMax(1,TL_Len_index):1/7:-CoefMax(1,TL_Len_index));
% A2=(-2/7:1/7:2/7);
% A3=(-2/7:1/7:2/7);

% Create Combinations of A0, A1, A2, A3
[Tmp0,Tmp1,Tmp2,Tmp3]=ndgrid(A0,A1,A2,A3);
% Each row of Coef_Comb = one combination of A0, A1, A2, A3
Coef_Comb=[Tmp0(:) Tmp1(:) Tmp2(:) Tmp3(:)];
Coef_Comb=single(Coef_Comb);

% Print out number of combinations
sprintf(['
 A0 sweep range: from ', num2str(A0(1)), ' to ', num2str(A0(end)), ...
 ' A1 sweep range: from ', num2str(A1(1)), ' to ', num2str(A1(end)), ...
 ' A2 sweep range: from ', num2str(A2(1)), ' to ', num2str(A2(end)), ...
 ' A3 sweep range: from ', num2str(A3(1)), ' to ', num2str(A3(end)), ...
 ' Combinations of all the coefficients are created and saved in 
[Coef_Comb]', ...
 ' Total number of combinations: ',num2str(size(Coef_Comb,1))
])
clearvars A0 A1 A2 A3 Tmp0 Tmp1 Tmp2 Tmp3

%(1.3) Perform all FIR (Delay & Sum) calculation (on all combinations of taps)
FIR_out=Coef_Comb*Delay_Out;
sprintf(['
 All FIR calculations (Delay & Sum) are performed', ...
 ' The results are saved in [FIR_out]'])

% Calculate the maximum amplitude of each row
FIR_out_max=max(abs(FIR_out),[],2);

% If the whole length of one row of FIR_out is less than RX_sensitivity, delete that row
KeepVector=find(FIR_out_max>=RX_sensitivity);
Coef_Comb=Coef_Comb(KeepVector,:);
FIR_out=FIR_out(KeepVector,:);
FIR_out_max=FIR_out_max(KeepVector,:);

sprintf(['
 Delete rows of [FIR_out] that have amplitude < RX_sensitivity (=', num2str(RX_sensitivity), ')', ')
 ',num2str(size(Coef_Comb,1))])

%(2) Pass Vout_fir thru an ideal rx (smart_latch)
sprintf('"
 [FIR_out]--> Ideal Latch Function(smartlatch_new.m)"
')
[Latch_out,FIR_out, BestSamplingPoint]=smartlatch_new(FIR_out, PulseSwitch, FIR_out_max, Latch_tolerence, Nsamples);
sprintf('Ideal Latch Function(smartlatch_new.m) generates:
- Ideal latch output [Latch_out]
- Time shifted FIR_out [FIR_out]

% (3) Compare the results between Vout_fir and smart_latch
error=sum(abs(Latch_out-FIR_out),2)./FIR_out_max;

% Sort rows according to error, then return vector [SortIndex]
[error, SortIndex]=sortrows(error);
% Sort matrices according to SortIndex
Coef_Comb=Coef_Comb(SortIndex,:);
FIR_out=FIR_out(SortIndex,:);
FIR_out_max=FIR_out_max(SortIndex,:);
Latch_out=Latch_out(SortIndex,:);

sprintf('Sort rows of [Coef_Comb], [FIR_out], [FIR_out_max], [Latch_out]
according to [error]=sum(abs([Latch_out]-[FIR_out]))

% ================ Save all coefficient set ================ %
% Coef_Comb_Integer=int16([1:1:size(Coef_Comb,1); round(Coef_Comb*7).'']);
% fid = fopen([char(TLLength(TL_Len_index,1,:)), '\', char(TLLength(TL_Len_index,1,:)), '_Tdelay', num2str(Tdelay), 'ps_', 'CoefComb.dat'], 'wt');
% fprintf(fid, ' Rank  A0  A1  A2  A3
%          %3.0f %2.0f %2.0f %2.0f
', Coef_Comb_Integer);
% fclose(fid);

sprintf('Coefficient combinations [Coef_Comb] (sorted according to [error]) are
saved into file:
char(TLLength(TL_Len_index,1,:)), '_Tdelay', num2str(Tdelay), 'ps_', 'CoefComb.dat')

% ================ Verify the FIR ================ %
sprintf('Top ', num2str(RankingList), ' eyediagrams and time-domain waveforms are
being saved under directory \char(TLLength(TL_Len_index,1,:)), ''
for i=1:RankingList
% ========= Plot eyediagram of FIR_out =========
    h1=eyediagram(FIR_out(i,:),Nsamples,PERIOD,round(Nsamples/2));
    set(h1,'Position',[1, 1, 1920/2, 1200/2])
    title(['Tdelay=', num2str(Tdelay), '#', num2str(i), ',a0=', num2str(Coef_Comb(i,1)), ',a1=', num2str(Coef_Comb(i,2)), ',a2=', num2str(Coef_Comb(i,3)), ',a3=', num2str(Coef_Comb(i,4))])
    saveas(h1, char(TLLength(TL_Len_index,1,:)), '_Tdelay', num2str(Tdelay), 'ps_', num2str(i),'_eye', 'fig'); % Save
    saveas(h1, char(TLLength(TL_Len_index,1,:)), '_Tdelay', num2str(Tdelay), 'ps_', num2str(i),'_eye', 'jpg'); % Save
    close(h1); % Close

% ========= Plot Time-domain waveforms =========

samp=ones(Len_vout,1,'single')*(-FIR_out_max(i));
samp(round(Nsamples/2):Nsamples:end)=FIR_out_max(i);
h2=figure;
plot(TimeAxis, samp, 'r');
hold on
plot(TimeAxis, FIR_out(i,:), 'g');
hold on
plot(TimeAxis, Latch_out(i,:), 'b');
hold on
title(["Tdelay=", num2str(Tdelay), '#', num2str(i), ',a0=', num2str(Coef_Comb(i,1)), ',a1=', num2str(Coef_Comb(i,2)), ',a2=', num2str(Coef_Comb(i,3)), ',a3=', num2str(Coef_Comb(i,4))])
legend('t_{sampling}','FIR_{out}','Latch_{out}')
hold off
saveas(h2, [char(TLLength(TL_Len_index,1,:)), '\',char(TLLength(TL_Len_index,1,:)), '_Tdelay', num2str(Tdelay), 'ps_', num2str(i), '_plot'], 'fig'); % Save
close(h2); % Close
toc
end
end
end
A.3.3 smartlatch_new.m

function [Latch_out, Latch_in, BestSamplingPoint]=smartlatch_new(Latch_in, PulseSwitch, IN_max, tolerance, Nsamples)
% SmartLatch transfer function:
% Vout
% ^
% |--------------<= IN_max
% | |
% ThresholdNEG | | |
% ---------------|-------------->Vin
% | | ^
% | | ThresholdPOS
% =IN_max--> -------|
%
% Latch parameters:
% IN_max - scale factor for the modulated signal, which is used to
% compared with FIRout
% Tolerance - To set the thresholds
% ThresholdP= IN_max * Tolerance
%This is the latch that does:
% 1-a. Locating the zero-crossing points by majority vote
% (or) 1-b. Find the best sampling point (ONE for each bit period)
% 2-a. Sample the output of TapSum (FIRout) and judge it's 1, 0, or -1 by
% thresholds, OF WHICH THE AMPLITUDE IS SET BY IN_max, which is a
% scale factor and in turn affects the thresholds and the demodulated
% signal that the FIRout is used to compared with.
% 2-b. Demodulate(Create ideal output from the sampling output)

%IN_max=IN_max*1.2;
tolerance=0.15;
PulseSwitch=1;%1: Ideal Pulse = Half Sine Wave
PulseSwitch=2;%2: Ideal Pulse = Square Wave

% Find best sampling point
NumRows=size(Latch_in,1);
NumCols=size(Latch_in,2);
% Re-shuffle [Latch_in] so that
% [ReshapeTmp] = [ R_{1,1};
% R_{1,2};
% ....
% R_{1,M};
% R_{2,1};
% R_{2,2};
% ....
% R_{2,M};
% ...
% R_{N,1};
% R_{N,2};
% ...
% ...
% R_{N*M}
% where M = number of tap combinations (size(Latch_in,1))
% n = 1 to N and N = number of period consisted in Latch_in (or size(Latch_in,2)/Nsamples)
% and each R is a row vector that represent the group of time points
% that has same relative offset against each period
ReshapeTmp = reshape(Latch_in, NumRows*Nsamples, NumCols/Nsamples);
% By summing ReshapeTmp on each row, the result can be used to do majority vote
ReshapeTmpSum = sum(abs(ReshapeTmp), 2);
% Reshaping again so that row number = M and column number = Nsamples
SumOfAmplitude = reshape(ReshapeTmpSum, NumRows, Nsamples);

[mm, BestZeroXing] = min(SumOfAmplitude, [], 2); % 1-a
BestSamplingPoint = mod(BestZeroXing + Nsamples/2, Nsamples);

% Alternatively, find best sampling point (peaks) by majority vote
% [mm, BestZeroXing] = max(SumOfAmplitude, [], 2); % 1-b
% BestZeroXing = mod(BestZeroXing + Nsamples/2, Nsamples);

sprintf([' Function "smartlatch_new"', ...
'\n Samples at ideal timing then outputs:', ...
'\n - Ideal positive pulse when V > ThresholdPOS,', ...
'\n - Ideal negative pulse when V > ThresholdNEG,', ...
'\n - Zeros when ThresholdNEG < V < ThresholdPOS'])

% Create templates for ideal demodulated output
if PulseSwitch == 1
  % Half sine wave
  IndexTmp = (0:pi/(Nsamples-1):pi)';
  PulseTmp = sin(IndexTmp).'; % This is a row vector

  warning('MATLAB:Reminder2', '...
'\n  Here ideal pulse is a half sine-wave. Can be changed to square wave or anything else')
elseif PulseSwitch == 2
  % Alternatively, create square wave
  PulseTmp = ones(1, Nsamples); % This is a row vector

  warning('MATLAB:Reminder2', '...
'\n  Here ideal pulse is a square wave. Can be changed to half sine wave or anything else')
else
  warning('MATLAB:Reminder2', '...
'\n  Ideal Pulse needs to be correctly set')
end

% Set Threshold, different for each row
ThresholdPOS = IN_max*tolerance/100;
ThresholdNEG = -IN_max*tolerance/100;
Latch_out = zeros(size(Latch_in), 'single');

% Shift each row of Latch_in with shiftsize=BestZeroXing
for i = 1:NumRows

126
% Shift each row of Latch_in according to corresponding BestZeroXing
Latch_in(i,1:end)=circshift(Latch_in(i,:),[0,-BestZeroXing(i)]);
% Pick the sampling points
tmpA=Latch_in(i,round(Nsamples/2):Nsamples:end);
% Creat the ideal bit-sequence (consists of [1,0,-1], instead of [1,-1])
tmpA(tmpA>=ThresholdPOS(i))=1;
tmpA(tmpA<=ThresholdNEG(i))=-1;
tmpA(tmpA<ThresholdPOS(i) & tmpA>ThresholdNEG(i))=0; %2-a
% Generate ideal Latch_out
Latch_out(i,1:NumCols)=kron(tmpA,PulseTmp*IN_max(i)); %2-b

% eyediagram(Latch_in(i,1:end),Nsamples)
% figure
% plot(Latch_in(i,1:end),'g')
% hold on
% plot(Latch_out(i,1:NumCols),'b')
% hold on
% samp=zeros(NumCols,1); samp(round(Nsamples/2):Nsamples:end)=IN_max(i);
% plot(samp,'r')
% hold off
end
A.4 Time-Domain Parameter Optimization (for NRZ Signaling)

This program is built on top of the program in A.3.1. Before running the main function (A.4.3), put existing channel S-parameter files (touchstone format) into sub-directory “/SnP”, and Run “GenerateChannelModel.m (A.4.2)”. It will convert those SnP files into Matlab object format to be used later.

Instead of taking the channel output waveforms directly as done in A.3, the main function is capable of generating the channel output. First, it generates ideal PRBS bit stream via a LFSR function (A.4.5 then A.4.5). Second, it runs the PRBS bit stream through the channel model converted by GenerateChannelModel.m.

The overall process of the main function remains similar to the one in A.3.1, the only exception being the smartlatch_new.m is now replaced with smartlatch_NRZ_new.m (A.4.4). And the error term is now replaced with the “eye opening”. Additional eye opening information will also show on the output eyedigram. Notice the definition of “eyeopening” here is to fix the horizontal opening at certain percentage of UI (UIpercent), and measure the vertical eyeopening.

More details are in the comments.
### A.4.1 Output files

The output of this program includes:

- Top ranked eyediagram annotated with tdelay and tap coefficients. For example:

![Eye diagram with annotations](image)

- A list of top ranked tap coefficients with vertical eyeopening. The first few rows look like:

<table>
<thead>
<tr>
<th>Rank</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>VopenInPercent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-7</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0.612</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-6</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0.599</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-7</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0.558</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.540</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.540</td>
</tr>
</tbody>
</table>
A.4.2 GenerateChannelModel.m

% This routine generate the rfmodel.rational structure of channels
% Each channel was individually verified by running
% 'test_import.S2P.m' or 'test_generate_S2P.m'
%
% The variable "FunctionSwitch" is set so that later the saved channel
% can be loaded in without running "rationalfit" again, which might be
% time-consuming.
%
% The rationalfit channel models are saved and can later be loaded
% to do a function:"timeresp".
% *Note: Trouble fitting==> increase freq. points & npoles
clc
clear

% FunctionSwitch=1; % S2P==> Fit ==> Save & Compare
FunctionSwitch=2; % Load Saved Fit Date ==> Compare & Verify

%% Channel 1 - Load measured S2P
%(-3.4dB @ 2.5GHz)
%(-5dB @ 5GHz)
%(-7.2dB @ 10GHz)
channel1 = read(rfdata.data,'SnP\LocustLine_j441_j433_balanced.s2p');
freq = channel1.Freq;
data = channel1.S_Parameters(2,1,:);
if FunctionSwitch==1
    channel1_fit = rationalfit(freq,data); %rational fit
    save channel1.mat channel1 % -7.2dB @ 10GHz. Real measured channel.
    save channel1_fit.mat channel1_fit
else
    load channel1.mat channel1 % -7.2dB @ 10GHz. Real measured channel.
    load channel1_fit.mat channel1_fit
end

% ** Verify the fit
[resp,freq] = freqresp(channel1_fit,freq);

figure()
plot(channel1, 'S21','dB');
hold on
plot(freq/1e9,db(resp),'r');
title('|S_21| of Measured channel 1')
hold off

figure()
plot(channel1,'S21','Angle (radians)');
hold on
plot(freq/1e9,unwrap(angle(resp)),'r');
hold off

130
%% Channel 2 - Load measured S2P
%(3dB  @ 2.5GHz)
%(4.6dB @ 5GHz)
%(13.4dB @ 10GHz)
channel1 = read(rfdata.data,'SnP\MothLine_j1100_j102_balanced.s2p');
freq = channel1.Freq;
data = channel1.S_Parameters(2,1,:)
if FunctionSwitch==1
    channel1_fit = rationalfit(freq,data); %rational fit
    save channel1.mat channel1 % -13.4dB @ 10GHz. Real measured channel.
    save channel1_fit.mat channel1_fit
else
    load channel1.mat channel1 % -13.4dB @ 10GHz. Real measured channel.
    load channel1_fit.mat channel1_fit
end

% ** Verify the fit
[resp,freq] = freqresp(channel1_fit,freq);

figure()
plot(channel1, 'S21','dB');
hold on
plot(freq/1e9,db(resp),'r');
title('|S_21| of Measured channel 2')
hold off

figure()
plot(channel1,'S21','Angle (radians)');
hold on
plot(freq/1e9,unwrap(angle(resp)),'r');
hold off

%% Channel 3 - Load HSPICE simulated S2D21
%(2.1dB  @ 2.5GHz)
%(3.3dB @ 5GHz)
%(5.4dB @ 10GHz)
channel1 = read(rfdata.data,'SnP\20cm.s4p');
freq = channel1.Freq;
data = channel1.S_Parameters(2,1,:)
if FunctionSwitch==1
    channel1_fit = rationalfit(freq,data); %rational fit
    save channel1.mat channel1 % -5.4dB @ 10GHz. Real measured channel.
    save channel1_fit.mat channel1_fit
else
    load channel1.mat channel1 % -5.4dB @ 10GHz.
    load channel1_fit.mat channel1_fit
end

% ** Verify the fit
[resp,freq] = freqresp(channel1_fit,freq);
figure()
plot(channel1, 'S21','dB');
hold on
plot(freq/1e9,db(resp),'r');
title('|S_21| of Measured channel 3')
hold off

figure()
plot(channel1,'S21','Angle (radians)');
hold on
plot(freq/1e9,unwrap(angle(resp)),'r');
hold off

%% Channel 4 - Load HSPICE simulated SDD21

if FunctionSwitch==1
  channel1_fit = rationalfit(freq,data, -0.1, [], 0, 1, 200);
  save channel4.mat channel1 % -10.8dB @ 10GHz.
  save channel4_fit.mat channel1_fit
else
  load channel4.mat channel1 % -10.8dB @ 10GHz.
  load channel4_fit.mat channel1_fit
end

% ** Verify the fit
[resp,freq] = freqresp(channel1_fit,freq);

figure()
plot(channel1, 'S21','dB');
hold on
plot(freq/1e9,db(resp),'r');
title('|S_21| of Measured channel 4')
hold off

figure()
plot(channel1,'S21','Angle (radians)');
hold on
plot(freq/1e9,unwrap(angle(resp)),'r');
hold off

%% Channel 5 - Load HSPICE simulated SDD21

if FunctionSwitch==1
  channel1_fit = rationalfit(freq,data, -0.1, [], 0, 1, 200);
  save channel5.mat channel1 % -16.1dB @ 10GHz.
  save channel5_fit.mat channel1_fit
else
  load channel5.mat channel1 % -16.1dB @ 10GHz.
  load channel5_fit.mat channel1_fit
end

% ** Verify the fit
[resp,freq] = freqresp(channel1_fit,freq);
```matlab
data = channel1.S_Parameters(2,1,:); if FunctionSwitch==1
    channel1_fit = rationalfit(freq,data); % rational fit
    save channel1_fit.mat channel1_fit
else
    load channel1.mat channel1 % -16.1dB @ 10GHz.
end

% ** Verify the fit
[resp,freq] = freqresp(channel1_fit,freq);

% Channel 6 - Load HSPICE simulated SDD21
% (-8.5dB @ 2.5GHz)
% (-13.2dB @ 5GHz)
% (-21.5dB @ 10GHz)
channel1 = read(rfdata.data,'SnP\80cm.s4p');
freq = channel1.Freq;
data = channel1.S_Parameters(2,1,:); if FunctionSwitch==1
    channel1_fit = rationalfit(freq,data); % rational fit
    save channel1_fit.mat channel1_fit
else
    load channel1.mat channel1 % -21.5dB @ 10GHz.
end

% ** Verify the fit
[resp,freq] = freqresp(channel1_fit,freq);
```

133
figure()
plot(channel1,'S21','Angle (radians)');
hold on
plot(freq/1e9,unwrap(angle(resp)),'r');
hold off

%% Channel 7 - Load HSPICE simulated SDD21 (-26.8dB @ 10GHz)
channel1 = read(rfdata.data,'SnP\100cm.s4p');
freq = channel1.Freq;
data = channel1.S_Parameters(2,1,:);
if FunctionSwitch==1
    channel1_fit = rationalfit(freq,data); %rational fit
    save channel7.mat channel1 % -26.8dB @ 10GHz.
    save channel7_fit.mat channel1_fit
else
    load channel7.mat channel1 % -26.8dB @ 10GHz.
    load channel7_fit.mat channel1_fit
end

% ** Verify the fit
[resp,freq] = freqresp(channel1_fit,freq);

figure()
plot(channel1, 'S21','dB');
hold on
plot(freq/1e9,db(resp),'r');
title('|S_{21}| of Measured channel 7')
hold off

figure()
plot(channel1,'S21','Angle (radians)');
hold on
plot(freq/1e9,unwrap(angle(resp)),'r');
hold off
clear all
clc
tStart=tic;

% Delay&Sum block (FIR core) has 5 taps - 0,1,2,3,4
% This routine
% (1) Generate Vin
% (1.1) Generate LFSR bit sequence
% (1.2) Convert LFSR bit sequency to non-ideal waveform
% (2) Generate Vout by passing Vin thru imported channel S2P
% (3) Calculate Vout_fir based on the given FIR coefficients
% (1.1) Create "Delayed versions"
% (1.2) Expand the combination of Tap Coefficients
% (1.3) Calculate all the FIR_out in one matrix multiplication
% (4) Pass Vout_fir thru an ideal rx (smart_latch)
% (5) Compare the results between Vout_fir and smart_latch
%
% Note that each channel length has a number assigned to it.

%% --------------- Initializing Constants ------------------------------
% Important Constants
SeqLen=2250; % Bit count of the inputs
TimeResolution=1e-12;
Nsamples = 100; % Number of samples per period (Implied bit rate)
EdgeSamples=10; % Number of points of the edges for the input bit stream waveforms
TdelayRange=10:10:150;
TdelaySamples=round(TdelayRange./(TimeResolution*1e12)); %=84 for IBM Aug 2008 run
ChannelRange=1:1:7;

PERIOD = Nsamples*TimeResolution;

RX_sensitivity=0.02;

% Latch_tolerance=20; % in percentage, at sampling point, if Vout> +- Latch_tolerance, the latch changes
UIpercent=50; % Used to determine the magnitude of the eye opening

RankingList=50; % The number of coefficient sets saved in the end

NumSaveFigs=3; % The number of figs (eye and time-domain plot) saved

RankingListMultiplier=10; % Each sweep of A0 saves RankingList multiplied by this number

% Not So Important Constants
OFFSETin = 0; % Input offset in number of samples, for Eyediagram
OFFSETout = 0; % Output offset in number of samples, for Eyediagram
% % ------------------------ Setup Channel Dependent Parameters ------------------------
% ------------------------ Set Tap Coefficient Sweep Ranges ------------------------
% CoefMax indicates the Max/Min value of Tap Coefficients
% CoefMax(each row,:) = [A0min A0max A1min A1max A2min A2max A3min A3max A4min A4max);
% The A4 taps=0 so the 5th tap is disabled. This is a 4-tap system
CoefMax(1,:) = [1 7 -7 -7 7 -7 7 0 0];
CoefMax(2,:) = [1 7 -7 7 -7 7 7 0 0];
CoefMax(3,:) = [1 7 -7 -7 7 -7 7 0 0];
CoefMax(4,:) = [1 7 -7 -7 7 -7 7 0 0];
CoefMax(5,:) = [1 7 -7 7 -7 -7 7 0 0];
CoefMax(6,:) = [1 7 -7 -7 -7 -7 7 0 0];
CoefMax(7,:) = [1 7 -7 -7 7 -7 0 0 0];
CoefMax(8,:) = [1 7 -7 -7 7 -7 7 0 0];
CoefMax(9,:) = [1 7 -7 -7 -7 -7 7 0 0];
CoefMax(10,:) = [1 7 -7 -7 7 -7 7 0 0];
CoefMax(11,:) = [1 7 -7 -7 -7 -7 7 0 0];
CoefMax = single(CoefMax);
% CoefMax=ones(11,8,'single'); %Overwrite all of them

% % ------------------------ Set Trip Time ------------------------
% ** TripTime is an array used to store corresponding TLine trip time
% ** This value is set so that the beginning of the bit stream can align with
% TripTime = zeros(11,1,'single'); % Initialize
TripTime(1,1) = 6.0700e-010; % Trip time across the transmission line
TripTime(2,1) = 1.1410e-009; % Trip time across the transmission line
TripTime(3,1) = 1.1930e-009; % Trip time across the transmission line
TripTime(4,1) = 2.3940e-009; % Trip time across the transmission line
TripTime(5,1) = 3.5980e-009; % Trip time across the transmission line
TripTime(6,1) = 4.8040e-009; % Trip time across the transmission line
TripTime(7,1) = 6.0120e-009; % Trip time across the transmission line
TripTime(8,1) = 6.1020e-009; % Trip time across the transmission line
TripTime(9,1) = 6.0870e-009; % Trip time across the transmission line
TripTime(10,1) = 6.0850e-009; % Trip time across the transmission line
TripTime(11,1) = 6.1020e-009; % Trip time across the transmission line
TripTime = round(TripTime/TimeResolution)*TimeResolution; % Round the trip time to pico-second
TripTime = single(TripTime);

% % ------------------------ Set Corresponding File Name ------------------------
% ** Channel_fit is a string matrix for load-in filename (may include path).
Channel_fit(1,1,:) = {'Channel1_fit.mat'};
Channel_fit(2,1,:) = {'Channel2_fit.mat'};
Channel_fit(3,1,:) = {'Channel3_fit.mat'};
Channel_fit(4,1,:) = {'Channel4_fit.mat'};
Channel_fit(5,1,:) = {'Channel5_fit.mat'};
Channel_fit(6,1,:) = {'Channel6_fit.mat'};
Channel_fit(7,1,:) = {'Channel7_fit.mat'};
Channel_fit(8,1,:) = {'Channel8_fit.mat'};
Channel_fit(9,1,:) = {'Channel9_fit.mat'};
Channel_fit(10,1,:) = {'Channel10_fit.mat'};
Channel_fit(11,1,:) = {'Channel11_fit.mat'};

% ** ChannelName is a string matrix for creating file for record
ParentFolderName = ('4Taps');
ChannelName(1,1,:) = {'L1'};
ChannelName(2,1,:) = {'L2'};
ChannelName(3,1,:) = {'L3'};
ChannelName(4,1,:) = {'L4'};
ChannelName(5,1,:) = {'L5'};
ChannelName(6,1,:) = {'L6'};
ChannelName(7,1,:) = {'L7'};
ChannelName(8,1,:) = {'L8'};
ChannelName(9,1,:) = {'L9'};
ChannelName(10,1,:) = {'L10'};
ChannelName(11,1,:) = {'L11'};

%% ================= (1) Generate Vin =====================================
% ========== Create PRBS 2^10-1 bit stream ==========
Taps=[10, 9, 8, 5, 0]; %Set the LFSR tap configuration
Seed=[1 0 1 0 1 0 1 0 1 0];
% (1.1) Generate LFSR bit sequence
(Seq1)=LFSR(Taps, SeqLen, Seed);
% (1.2) Convert LFSR bit sequency to non-ideal waveform
[Vin_square Vin]=Bit2Waveform(Seq1, Nsamples, EdgeSamples); %Generate the input time-domain waveform
TimeAxisIn = (0:TimeResolution:TimeResolution*(length(Vin)-1));

%% ---------------------- (1) Generate Vin -------------------------------
% --- Create PRBS 2^10-1 bit stream ---
Taps=[10, 9, 8, 5, 0]; %Set the LFSR tap configuration
Seed=[1 0 1 0 1 0 1 0 1 0];
% (1.1) Generate LFSR bit sequence
(Seq1)=LFSR(Taps, SeqLen, Seed);
% (1.2) Convert LFSR bit sequency to non-ideal waveform
[Vin_square Vin]=Bit2Waveform(Seq1, Nsamples, EdgeSamples); %Generate the input time-domain waveform
TimeAxisIn = (0:TimeResolution:TimeResolution*(length(Vin)-1));

% % % sprintf(['
 PRBS 2^', num2str(Taps(1)), '-1 input bit stream [Seq1] is generated.' ...
 % % %     '
 Length(Seq1)= ', num2str(SeqLen), 'bits'])

% % % % ** Verify: Compare inputs with ideal edges
% % figure()
% % plot(TimeAxisIn,Vin_square,'b')
% % hold on
% % plot(TimeAxisIn,Vin,'r')
% % legend(['Edge rate=1*', num2str(TimeResolution)], ['Edge rate=',
% % num2str(EdgeSamples), '**', num2str(TimeResolution)])
% % title('Time-domain waveforms of the input bit-stream')
% % hold off

%% ------------------------ Alter the channels ---------------------------
for Name_index=ChannelRange

% % Import the Channel Rational Fit Model % %
load(char(Channel_fit(Name_index,1,:)));

% % Pass the input signal thru channel % %
% (2) Generate Vout by passing Vin thru imported channel S2P
[Vout0] = timeresp(channel1_fit, Vin, TimeResolution); %Channel output
[Vout0_square, TimeAxisOut] = timeresp(channel1_fit, Vin_square, TimeResolution);
%Channel output if Vin is ideal
% Channel output [Vout0] is generated by passing [Vin] thru channel:
% sprintf(['
 Channel output [Vout0] is generated by passing [Vin] thru channel:
',
char(Channel_fit(Name_index,1,:))])

% ** Verify: Compare the outputs with ideal and non-ideal inputs
% h99=figure();
% hold on
% plot(TimeAxisIn,Vin, 'c')
% hold on
% plot(TimeAxisIn,Vin_square, 'm')
% hold off

% ========== Performe time offset on Vout ==========
TimeAxisOut=single(TimeAxisOut);
% ** Locate the location of the row that matches the TripTime for that specific TLine
locshift=find(TimeAxisOut==TripTime(Name_index,1));
% ** Clear the zero data, and trim total time to multiple of period
Vout0=Vout0(locshift:end, :);
Vout0=Vout0(1:Nsamples*floor(size(Vout0,1)/Nsamples),:);
Vout0=single(Vout0);
Vout0_square=Vout0_square(locshift:end, :);
Vout0_square=Vout0_square(1:Nsamples*floor(size(Vout0_square,1)/Nsamples),:);
Vout0_square=single(Vout0_square);

% ** Make first point correspond to zero time
TimeAxisOut = (0:TimeResolution:TimeResolution*(length(Vout0)-1));
TimeAxisOut = round(TimeAxisOut/TimeResolution)*TimeResolution;

% ** Verify: Compare the outputs with ideal and non-ideal inputs
figure(h99)
hold on
plot(TimeAxisOut,Vout0, 'r')
hold on
plot(TimeAxisOut,Vout0_square, 'b')
title('Compare channel outputs')
legend('Channel Inputs', 'with non-ideal inputs', 'with ideal (square wave) inputs')
hold off

% ** Verify: plot eyediagram of channel inputs and outputs
eyediagram(Vin, Nsamples, PERIOD, OFFSETin)
eyediagram(Vout0, Nsamples, PERIOD, OFFSETout)

% ================= Sweep Tdelay =====================================
for Tdelay=TdelaySamples % Sweep Tdelay
    sprintf(['
 Tdelay = ', num2str(round(Tdelay*TimeResolution*1e12)), 'ps'])
    % (3) Delay & Sum (FIR)
    % Initializing Variables: [VoutX] are column vectors
    Len_vout=size(Vout0,1);
    % (3.1) Perform Delay
    Delay_Out=Delay_5Taps(Vout0,Tdelay);
    % (3.2) Expand Tap Coefficient Matrices

%%% ------------------------ Sweep Tdelay ------------------------------
A0=(CoefMax(Name_index,1):1:CoefMax(Name_index,2));
A1=(CoefMax(Name_index,3):1:CoefMax(Name_index,4));
A2=(CoefMax(Name_index,5):1:CoefMax(Name_index,6));
A3=(CoefMax(Name_index,7):1:CoefMax(Name_index,8));
A4=(CoefMax(Name_index,9):1:CoefMax(Name_index,10));

% Print out combinations
sprintf(['
 A0 sweep range: from ', num2str(A0(1)), ' to ', num2str(A0(end)), ...
 ' A1 sweep range: from ', num2str(A1(1)), ' to ', num2str(A1(end)), ...
 ' A2 sweep range: from ', num2str(A2(1)), ' to ', num2str(A2(end)), ...
 ' A3 sweep range: from ', num2str(A3(1)), ' to ', num2str(A3(end)), ...
 ' A4 sweep range: from ', num2str(A4(1)), ' to ', num2str(A4(end))]);

if RankingList*RankingListMultiplier>length(A0)*length(A1)*length(A2)
  if RankingList>length(A0)*length(A1)*length(A2)
    RankingList=length(A0)*length(A1)*length(A2);
    RankingListMultiplier=1;
  else
    RankingListMultiplier=floor(length(A0)*length(A1)*length(A2)/RankingList);
  end
end

%Initializing variables
Rdimensions=length(A4)*length(A3)*RankingList*RankingListMultiplier;
vopen_all=zeros(Rdimensions,1,'single');
vopenLoc_all=zeros(Rdimensions,1,'single');
PosMin_all=zeros(Rdimensions,1,'single');
NegMax_all=zeros(Rdimensions,1,'single');
Coef_Comb_all=zeros(Rdimensions,5,'single');
%% FPGA configuration
FIR_out_all=zeros(Rdimensions,Len_vout-10*Nsamples,'single');

%% ================ Sweep A4 (not enough memory) =================
for k=1:1:length(A4)
  %% ================ Sweep A3 (not enough memory) ================
  for i=1:1:length(A3)
    sprintf(['
 Run section:', num2str(i+length(A3)*(k-1)), ' of ', num2str(length(A3)*length(A4)), ...
 ' A4=', num2str(A4(k)), ', A3=', num2str(A3(i))]);

    % Create Combinations of A0, A1, A2, A3
    [Tmp0,Tmp1,Tmp2,Tmp3,Tmp4]=ndgrid(A0,A1,A2,A3(i),A4(k));
    % Each row of Coef_Comb = one combination of A0, A1, A2, A3
    Coef_Comb=[Tmp0(:) Tmp1(:) Tmp2(:) Tmp3(:) Tmp4(:)];
    Coef_Comb=single(Coef_Comb);
    clearvars Tmp0 Tmp1 Tmp2 Tmp3

    % (3.3) Perform all FIR (Delay & Sum) calculation (on all combinations of taps)
    FIR_out=Coef_Comb*Delay_Out;
    % % %
    sprintf(['
 All FIR calculations (Delay & Sum) are performed', ...
 ' The results are saved in [FIR_out]']);
end
end

139
% Calculate the maximum amplitude of each
FIR_out_max=max(abs(FIR_out),[],2);

% % %
    sprintf(['\n Original number of combinations:
', num2str(size(Coef_Comb,1))])

% If the whole length of one row of FIR_out is less than RX_sensitivity,
delete that row
KeepVector=find(FIR_out_max>=RX_sensitivity);
Coef_Comb=Coef_Comb(KeepVector,:);
FIR_out=FIR_out(KeepVector,:);
FIR_out_max=FIR_out_max(KeepVector,:);

% % %
    sprintf(['\n Delete rows of [FIR_out] that have amplitude <
RX_sensitivity (-', num2str(RX_sensitivity), '), ...
% % %         \n Reduce number of combinations to: ',
num2str(size(Coef_Comb,1))])

% % % Normalize FIR_out
FIR_out=FIR_out./repmat(FIR_out_max,1,size(FIR_out,2));

% % %
    sprintf('\n [FIR_out] is normalized to [FIR_out_max]')

% Trim off the first and last 5 periods
FIR_out(:,1:5*Nsamples)=[];
FIR_out(:,end-5*Nsamples+1:end)=[];

% (4) Pass Vout_fir thru an ideal rx (smart_latch)
% % %
    sprintf('\n [FIR_out] ==> Ideal Latch Function(smartlatch_NRZ.m)')

    [FIR_out vopen, vopenLoc, PosMin, NegMax]=smartlatch_NRZ(FIR_out, FIR_out_max,
UIpercent, Nsamples);

clearvars FIR_out

% % %
    sprintf('\n Ideal Latch Function(smartlatch_NRZ.m) generates:', ...
    '\n - Time-shifted FIR_out[FIR_out]', ...%
    '\n - Eye Opening [vopen]', ...
    '\n - Eye measurement window location [vopenLoc]', ...
    '\n - Eye measurement window top [PosMin]', ...
    '\n - Eye measurement window bottom [NegMax]')

% Sort rows according to vopen, then return vector [SortIndex]

    [vopen, SortIndex]=sortrows(vopen,-1);

% Sort matrices according to SortIndex
Coef_Comb=Coef_Comb(SortIndex,:);
    FIR_out=FIR_out(SortIndex,:);
FIR_out_max=FIR_out_max(SortIndex,:);
vopenLoc=vopenLoc(SortIndex);
PosMin=PosMin(SortIndex);
NegMax=NegMax(SortIndex);

% Keep the first RankingList*RankingListMultiplier
vopen=vopen(1:RankingList*RankingListMultiplier,:);
Coef_Comb=Coef_Comb(1:RankingList*RankingListMultiplier,:);
% FIR_out=FIR_out(1:RankingList*RankingListMultiplier,:);
FIR_out_max=FIR_out_max(1:RankingList*RankingListMultiplier,:);
vopenLoc=vopenLoc(1:RankingList*RankingListMultiplier,:);
PosMin=PosMin(1:RankingList*RankingListMultiplier,:);
NegMax=NegMax(1:RankingList*RankingListMultiplier,:);

% Concatenate [vopen], [Coef_Comb], [FIR_out]
vopen_all(((i-1)+length(A3)*(k-1))*RankingList*RankingListMultiplier+1:(i+length(A3)*(k-1))*RankingList*RankingListMultiplier,:)=vopen;
Coef_Comb_all(((i-1)+length(A3)*(k-1))*RankingList*RankingListMultiplier+1:(i+length(A3)*(k-1))*RankingList*RankingListMultiplier,:)=Coef_Comb;
% FIR_out_all((i-1)*RankingList*RankingListMultiplier+1:i*RankingList*RankingListMultiplier,:)=FIR_out;
FIR_out_max_all(((i-1)+length(A3)*(k-1))*RankingList*RankingListMultiplier+1:(i+length(A3)*(k-1))*RankingList*RankingListMultiplier,:)=FIR_out_max;
vopenLoc_all(((i-1)+length(A3)*(k-1))*RankingList*RankingListMultiplier+1:(i+length(A3)*(k-1))*RankingList*RankingListMultiplier,:)=vopenLoc;
PosMin_all(((i-1)+length(A3)*(k-1))*RankingList*RankingListMultiplier+1:(i+length(A3)*(k-1))*RankingList*RankingListMultiplier,:)=PosMin;
NegMax_all(((i-1)+length(A3)*(k-1))*RankingList*RankingListMultiplier+1:(i+length(A3)*(k-1))*RankingList*RankingListMultiplier,:)=NegMax;
end
end

vopen=vopen_all;
Coef_Comb=Coef_Comb_all;
% FIR_out=FIR_out_all;
FIR_out_max=FIR_out_max_all;
vopenLoc=vopenLoc_all;
PosMin=PosMin_all;
NegMax=NegMax_all;

% clearvars vopen_all Coef_Comb_all FIR_out_all FIR_out_max_all vopenLoc_all
PosMin_all NegMax_all
clearvars vopen_all Coef_Comb_all FIR_out_max_all vopenLoc_all PosMin_all NegMax_all clearvars vout0 vout0_square

% Sort rows according to vopen, then return vector [SortIndex]
[vopen, SortIndex]=sortrows(vopen,-1);
% Sort matrices according to SortIndex

141
Coef_Comb=Coef_Comb(SortIndex,:);
%         FIR_out=FIR_out(SortIndex,:);
FIR_out_max=FIR_out_max(SortIndex,:);
vopenLoc=vopenLoc(SortIndex);
PosMin=PosMin(SortIndex);
NegMax=NegMax(SortIndex);

%%%        sprintf(['\n Sort rows of [Coef_Comb], [FIR_out], [FIR_out_max], [vopenLoc]
[PosMin] [NegMax]',...  
%%%         '\n according to eye opening \%', num2str(UIpercent),'\percent UI'])

%%%         Coef_Comb_Integer=[1:1:size(Coef_Comb,1); round(Coef_Comb*7).';]; % Take all rows of
[Coef_Comb]
%%%         Coef_Comb_Integer=[1:1:RankingList; round(Coef_Comb(1:RankingList,:)*7).';]; % Only
take the first "RankingList" rows of [Coef_Comb]
mkdir(char(ParentFolderName), char(ChannelName(Name_index,1,:))); % Make new folder
Coef_Comb_Integer=[1:1:RankingList; round(Coef_Comb(1:RankingList,:));].';
vopen(1:RankingList).';]; % Only take the first "RankingList" rows of [Coef_Comb]
fid = fopen([char(ParentFolderName), '\', char(ChannelName(Name_index,1,:)), '\',
char(ChannelName(Name_index,1,:)), '_Coef_List_Tdelay',
num2str(round(Tdelay*TimeResolution*1e12)), 'ps_EyeOpenAt_', num2str(UIpercent),
'\Percent_UI.dat'], 'wt');
fprintf(fid, ' Rank  A0 A1 A2 A3 A4  VopenInPercent
');
fprintf(fid, '%5.0f %3.0f %2.0f %2.0f %2.0f %2.0f   %2.3f
', Coef_Comb_Integer);
fclose(fid);

%%%         Coef_Comb_Integer=[(1:1:size(Coef_Comb,1)).',round(Coef_Comb), vopen]; % Take all
rows of [Coef_Comb]
save([char(ParentFolderName), '\', char(ChannelName(Name_index,1,:)), '\',
char(ChannelName(Name_index,1,:)), '_Coef_List_Tdelay',
num2str(round(Tdelay*TimeResolution*1e12)), 'ps_EyeOpenAt_', num2str(UIpercent),
'\Percent_UI.mat'], 'Coef_Comb_Integer')

%%%         sprintf(['\n Coefficient combinations [Coef_Comb, Vopen] (sorted according to [Vopen])
are saved into file:\', ...  
%%%         '\', char(ParentFolderName), '\', char(ChannelName(Name_index,1,:)), '\',
char(ChannelName(Name_index,1,:)), '_Coef_List_Tdelay',
num2str(round(Tdelay*TimeResolution*1e12)), 'ps_EyeOpenAt_', num2str(UIpercent),
'\Percent_UI.dat', ...  
%%%         '\ and \', char(ParentFolderName), '\', char(ChannelName(Name_index,1,:)),
'\', char(ChannelName(Name_index,1,:)), '_Coef_List_Tdelay',
num2str(round(Tdelay*TimeResolution*1e12)), 'ps_EyeOpenAt_', num2str(UIpercent),
'\Percent_UI.mat'])

%%%         Generating FIR_out (again) ---------------------
% -- to save memory during the for loop
Coef_Comb=Coef_Comb(1:NumSaveFigs,:);
FIR_out_max=FIR_out_max(1:NumSaveFigs,:);  
FIR_out=Coef_Comb*Delay_Out;
clearvars Delay_Out
% Normalize FIR_out
FIR_out=FIR_out./repmat(FIR_out_max,1,size(FIR_out,2));

% Trim off the first and last 5 periods
FIR_out(:,1:5*Nsamples)=[];
FIR_out(:,end-5*Nsamples+1:end)=[];

[FIR_out, vopen, vopenLoc, PosMin, NegMax]=smartlatch_NRZ(FIR_out, UIpercent, Nsamples);
[FIR_out, a, b, c, d]=smartlatch_NRZ(FIR_out, FIR_out_max, UIpercent, Nsamples);
clearvars a b c d

% --------- Verify the FIR -------------------------------

Vopen=Vin(:,5*Nsamples+1:end-5*Nsamples);

for i=1:NumSaveFigs
  h1=eyediagram(FIR_out(i,:),Nsamples,PERIOD,round(Nsamples/2));
  hold on
  set(h1,'Position',[1, 1, 1920/2, 1200/2])
  title(['Tdelay=', num2str(round(Tdelay*TimeResolution*1e12)), '#', num2str(i), ...'
    ',a0=', num2str(Coef_Comb(i,1)), ',a1=', num2str(Coef_Comb(i,2)), ...'
    ',a2=', num2str(Coef_Comb(i,3)), ',a3=', num2str(Coef_Comb(i,4)), ',a4=', ...'
    num2str(Coef_Comb(i,5)), ',Vopen@', num2str(UIpercent),'%UI=', num2str(vopen(i))]);
  plot([(vopenLoc(i)-Nsamples/2-1),(vopenLoc(i)+round(Nsamples*UIpercent/100)-1- ...'
    Nsamples/2-1)]*TimeResolution,PosMin(i)*ones(1,2), 'r')
  plot([(vopenLoc(i)-Nsamples/2-1),(vopenLoc(i)+round(Nsamples*UIpercent/100)-1- ...'
    Nsamples/2-1)]*TimeResolution,NegMax(i)*ones(1,2), 'r')
  plot(ones(1,2)*(vopenLoc(i)-Nsamples/2-1)*TimeResolution,
    [NegMax(i), PosMin(i)], 'r')
  plot(ones(1,2)*(vopenLoc(i)+round(Nsamples*UIpercent/100)-1-Nsamples/2-1)*TimeResolution,
    [NegMax(i), PosMin(i)], 'r')
  saveas(h1, [char(ParentFolderName), '/', char(ChannelName(Name_index,1,:)), '/EyeOpenAt_1_Tdelay', num2str(Tdelay), 'ps', num2str(i), '_EyeOpenAt_', num2str(UIpercent), 'PercentUI_1', '.jpg']);
  close(h1);

  samp=ones(size(FIR_out,2),1,'single')*(-FIR_out_max(i));
  samp(round(Nsamples/2):Nsamples:end)=FIR_out_max(i);
h2=figure;
plot(TimeAxisIn, Vin, 'b');
hold on

plot(TimeAxisOut, samp, 'r');
hold on

plot(TimeAxisOut, FIR_out(i,:), 'g');
hold on

plot(TimeAxisOut, Latch_out(i,:), 'b');
hold on

title(['Tdelay=', num2str(Tdelay), '#', num2str(i), ',a0=', num2str(Coef_Comb(i,1)), ',a1=', num2str(Coef_Comb(i,2)), ',a2=', num2str(Coef_Comb(i,3)), ',a3=', num2str(Coef_Comb(i,4)), ',a4=', num2str(Coef_Comb(i,5))])

hold off

legend('t_{sampling}', 'FIR_{out}', 'Latch_{out}')

legend('V_{in}', 'FIR_{out}')

hold off

saveas(h2, [char(ParentFolderName), '\', char(ChannelName(Name_index,1,:)), '\', char(ChannelName(Name_index,1,:)), '_Tdelay', num2str(Tdelay), '_ps_', num2str(i), '_plot'], 'fig'); % Save

close(h2); % Close
end
end

tElapsed=toc(tStart)
A.4.4 smartlatch_NRZ.m

function [Latch_in vopen vopenLoc PosMin NegMax]=smartlatch_NRZ(Latch_in, Latch_in_max, UIpercent, Nsamples)
% SmartLatch transfer function:
% Latch parameters:
%   Latch_in - waveform before latch sampling
%   PulseSwitch - select output waveform (so far only square wave)
%   IN_max  - scale factor for the modulated signal, which is used to
%            compared with FIRout
%   Nsamples - number of points per period
%This is the latch that does:
% 1. Auto-align:
% 1-a. Locating the zero-crossing points by majority vote
% (or) 1-b. Find the best sampling point (ONE for each bit period)
% 1-c. Bit-Rotate (cirshift) each row of Latch_in individually. Normalize.
% 2. Calculate Eye Opening: (row by row)
% 2-a. Sweep the UIpercent Window across the period
% 2-b. Find the max Eyeopening
% Requirement:
% The eye must be vertically centering at '0'; i.e., pure differential
% Latch_in=FIR_out(1:53, :) ; %for test

%% ---------------- 1. Auto-align: ---------------------------
% Find best sampling point
% % % NumRows=size(Latch_in,1);
% % % NumCols=size(Latch_in,2);
[NumRows,NumCols]=size(Latch_in);
% Re-shuffle Latch_in so that
% [ReshapeTmp] = [ R_(1_1);
%               R_(1_2);
%               ...
%               R_(1_M);
%               R_(2_1);
%               R_(2_2);
%               ...
%               R_(2_M);
%               ...
%               R_(N_1);
%               R_(N_2);
%               ...
%               R_(N*M)]
% where M=number of tap combinations (size(Latch_in,1))
% n=1-N and N-number of period consisted in Latch_in (or size(Latch_in,2)/Nsamples)
% and each R is a row vector that represent the group of time points
% that has same relative offset against each period
% Look into the "reshape" function to find out the detail!

ReshapeTmp=reshape(Latch_in,NumRows*Nsamples,NumCols/Nsamples);
% By summing ReshapeTmp on each row, the result can be used to do majority vote
ReshapeTmpSum=sum(abs(ReshapeTmp),2);

145
% Reshaping again so that row number = M and column number = Nsamples
SumOfAmplitude=reshape(ReshapeTmpSum,NumRows,Nsamples);

[mm,BestZeroXing]=min(SumOfAmplitude,[],2); %1-a
% BestSamplingPoint=mod(BestZeroXing+Nsamples/2,Nsamples); % Best Sampling Point

% Alternatively, find best sampling point (peaks) by majority vote
[mm,BestSamplingPoint]=max(SumOfAmplitude,[],2); %1-b
%BestZeroXing=mod(BestSamplingPoint+Nsamples/2,Nsamples);

for idx_row=1:NumRows % Processing each row at a time
    % Shift each row of Latch_in according to corresponding BestZeroXing
    Latch_in(idx_row,1:end)=circshift(Latch_in(idx_row,:),[0,-
    BestZeroXing(idx_row)])./Latch_in_max(idx_row); %1-c
    % Create temporary matrix [WaveTmp]
    % where each row of it is one period (#cols=Nsamples)
end

% Trim off the last 5 periods
% discontinuity might happen during the circshift process
Latch_in=Latch_in(:,1:end-5*Nsamples);
[NumRows,NumCols]=size(Latch_in);

%% --------------------- 2. Calculate Eye Opening: (row by row) ---------------------
WaveTmp=(reshape(Latch_in.',Nsamples,NumRows*NumCols/Nsamples)).';

PosHalfRowsIdx=WaveTmp(:,round(Nsamples/2))>=0;
NegHalfRowsIdx=WaveTmp(:,round(Nsamples/2))<0;

NumOfWindow=floor(Nsamples*(1-UIpercent/100));
PosMin=zeros(NumRows,NumOfWindow);
NegMax=zeros(NumRows,NumOfWindow);

for WindowLoc=1:NumOfWindow % 2-a
    CurrentWindow=WaveTmp(:,WindowLoc:round(Nsamples*UIpercent/100)-1));
    for idx_row=1:NumRows
        CurrentRows=CurrentWindow((idx_row-1)*NumCols/Nsamples+1:idx_row*NumCols/Nsamples,:);
        CurrentPosIdx=PosHalfRowsIdx((idx_row-1)*NumCols/Nsamples+1:idx_row*NumCols/Nsamples,:);
        CurrentNegIdx=NegHalfRowsIdx((idx_row-1)*NumCols/Nsamples+1:idx_row*NumCols/Nsamples,:);
        PosMin(idx_row,WindowLoc)=min(min(CurrentRows(CurrentPosIdx,:)));
        NegMax(idx_row,WindowLoc)=max(max(CurrentRows(CurrentNegIdx,:)));
    end
end

vopen=PosMin-NegMax;
[vopen vopenLoc]=max(vopen,[],2);

PosMin=PosMin(sub2ind([NumRows,NumOfWindow],(1:NumRows).',vopenLoc));
NegMax=NegMax(sub2ind([NumRows,NumOfWindow],(1:NumRows).',vopenLoc));
% % Use sub2ind to replace the following command
% PosMin=reshape(PosMin',1,[])';
% PosMin=PosMin(vopenLoc+(0:NumOfWindow:NumOfWindow*NumRows-1)');
%
% NegMax=reshape(NegMax',1,[])';
% NegMax=NegMax(vopenLoc+(0:NumOfWindow:NumOfWindow*NumRows-1)');
function [Seq] = LFSR(Taps, SeqLen, Seed)
% Change the Tap[] vector and get a list of LFSR sequence
% Need to set SeqLen and Seed
% http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_lfsr.htm
%
% Example:
% [Seq1] = LFSR([10, 7, 0], 1023, ones(1,10));
%
% Note:
% Reversing the tap order for each feedback set results in another valid
% m-sequence feedback set. The sequence produced with the reversed taps
% will be the mirror image of the sequence produced with the original taps.
% If the original feedback set is [m, A, B, C], the reversed feedback set
% is described by [m, m-C, m-B, m-A], where m is the number of LFSR stages.
%
% Taps = [10, 9, 8, 5];
% Taps = [23 18 0];
% Taps = [26, 25, 24, 20]; % Define Tap location
%
% Main Routine
n = Taps(1); % Number of LFSR stages = Taps(1)
L = 2^n - 1; % Set length of the original LFSR sequence
LFSRout = zeros(1, L);

% Generating the bit sequence
for i = 1:L
    LFSRout(i) = Seed(n);
    Seed = [mod(sum(Seed(Taps(1:end-1))), 2), Seed(1:n-1)]; % Next iteration
end

% Fit the length of the sequence to SeqLen
if SeqLen < L
    Seq = LFSRout(1:SeqLen);
else
    Seq = repmat(LFSRout, 1, floor(SeqLen/L));
    Seq = cat(2, Seq, LFSRout(1:mod(SeqLen, L)));
end
function [Waveform_square, Waveform]=Bit2Waveform(BitStream,Nsamples,EdgeSamples)
% This function generates the time-domain waveform from the given bitstream

%EdgeSamples=50; %Fixed default: Number of points per edge
if EdgeSamples>=Nsamples
    error('Error when generating waveforms:EdgeSamples>=Nsamples!!! ')
else
    %Find the Rising/Falling Edges
    BitStream_d1=cat(2,BitStream(1),BitStream(1:end-1)); %Delayed by 1 bit
    EdgeLoc=xor(BitStream,BitStream_d1); %Find the edge location
    RiseEdgeLoc=and(BitStream,EdgeLoc); %Find the rising edges
    RiseEdgeLocIdx=find(RiseEdgeLoc==1); %Find the index of the rising edges
    FallEdgeLoc=and(not(BitStream),EdgeLoc); %Find the falling edges
    FallEdgeLocIdx=find(FallEdgeLoc==1); %Find the index of the falling edges

    %Create square wave templates for the waveform (for 1s)
    PulseTmp=ones(1,Nsamples,'single'); %A row vector template for the square wave

    %Generating square waveform
    BitStreamShifted=single(sign((BitStream-0.5)));%Y-Shift the bit sequency to be symmetrical to y=0
    Waveform_square=kron(BitStreamShifted,PulseTmp); %Square waves

    %Reshape the square waveform for replacing Rise/Fall edges later
    Waveform=reshape(Waveform_square,Nsamples,[]); %Each row of Vin is one bit period

    %Create rising edge template (falling edge is the mirror image of this)
    EdgeTmp=single(linspace(-1,1,EdgeSamples)); %A row vector for the rising edge

    %Replacing the edges according to the location (via location index found earlier)
    Waveform(RiseEdgeLocIdx,1:EdgeSamples)=repmat(EdgeTmp,length(RiseEdgeLocIdx),1); %Replace the rising edges
    Waveform(FallEdgeLocIdx,EdgeSamples:-1:1)=repmat(EdgeTmp,length(FallEdgeLocIdx),1); %Replace the falling edges

    %Reshape the waveform back to row vector
    Waveform=reshape(Waveform',1,[]);
end
A.4.7  Delay_5Taps.m

function Delay_Out=Delay_5Taps(Vout0,Tdelay)

% (3.1) Perform Delay
Vout1=[Vout0(1)*ones(Tdelay,1); Vout0(1:end-Tdelay)]; % Delay of 1xTdelay
Vout2=[Vout1(1)*ones(Tdelay,1); Vout1(1:end-Tdelay)]; % Delay of 1xTdelay
Vout3=[Vout2(1)*ones(Tdelay,1); Vout2(1:end-Tdelay)]; % Delay of 1xTdelay
Vout4=[Vout3(1)*ones(Tdelay,1); Vout3(1:end-Tdelay)]; % Delay of 1xTdelay

% Combined into one Delay_Out matrix (for multiplication later)
% - [Delay_Out]'s size should be NumberOfTaps x LengthOfVout
Delay_Out=[Vout0; Vout1; Vout2; Vout3; Vout4];