# ELECTRONIC DISPERSION COMPENSATION FOR INTERLEAVED A/D CONVERTERS IN A STANDARD CELL ASIC PROCESS

A Dissertation
Presented to
The Academic Faculty

By

Matthew David Clark

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in
Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
August 2007

# ELECTRONIC DISPERSION COMPENSATION FOR INTERLEAVED A/D CONVERTERS IN A STANDARD CELL ASIC PROCESS

Approved by:


Dr. Doug Williams, Advisor
*Professor, School of ECE*
*Georgia Institute of Technology*

Dr. Mark Richards
*Professor, School of ECE*
*Georgia Institute of Technology*

Dr. John Peatman
*Professor, School of ECE*
*Georgia Institute of Technology*

Dr. Nick Feamster
*Asst. Professor, School of Computer Science*
*Georgia Institute of Technology*

Dr. Joseph Hughes
*Professor, School of ECE*
*Georgia Institute of Technology*


Date Approved: June 21, 2007

*For Angela*

# ACKNOWLEDGMENTS

While it might be my name on the front page, I would not be here without the support of many people:

- Dr. Doug Williams, whose advice to an unconventional student has been perfect, every time.
- Wes Smith, whose encouragement and enthusiasm could always be counted upon.
- Dr. Allen Dotson, of St. Andrews Presbyterian College, who placed my feet on this path many years ago. The fact that it took me this look to get here is my fault, not his.
- Dr. Giorgio Casinovi, whose advice has been relied on since my first day at Georgia Tech.
- Wil and Jackie, for all of their encouragement and support.
- My parents, whose love and support have enabled me to become who I am.
- My wife, without whom, nothing would be possible.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# SUMMARY

The IEEE 802.3aq standard recommends a multi-tap decision feedback equalizer be implemented to remove inter-symbol interference and additive system noise from data transmitted over a 10 Gigabit per Second (10 Gbps) multi-mode fiber-optic link (MMF). The recommended implementation produces a design in an analog process. This design process is difficult, time consuming, and is expensive to modify if first pass silicon success is not achieved.

Performing the majority of the design in a well-characterized digital process with stable, evolutionary tools reduces the technical risk. ASIC design rule checking is more predictable than custom tools flows and produces regular, repeatable results. Register Transfer Language (RTL) changes can also be relatively quickly implemented when compared to the custom flow. However, standard cell methodologies are expected to achieve clock rates of roughly one-tenth of the corresponding analog process.

The architecture and design for a parallel linear equalizer and decision feedback equalizer are presented. The presented design demonstrates an RTL implementation of 10 GHz filters operating in parallel at 625 MHz. The performance of the filters is characterized by testing the design against a set of 324 reference channels. The results are compared against the IEEE standard groups recommended implementation. The linear equalizer design of 20 taps equalizes 88% of the reference channels. The decision feedback equalizer design of 20 forward and 1 reverse tap equalizes 93% of the reference channels. Analysis of the unequalized channels in performed, and areas for continuing research are presented.

# CHAPTER 1

# INTRODUCTION

In this chapter, the research topic is defined and an introduction to the research area is given. The sections contained in this chapter include the research objective, the problem statement, and an overview of the current research in the literature.

## 1.1 Research objective

The objective of this research is to find and implement a method to recover data transmitted through a 10 Gbps Ethernet (10 GbE) fiber, sampled with interleaved, low resolution 2.5 Gbps analog-to-digital converters (ADC) using a standard cell application-specific integrated circuit (ASIC) process. The specific implementation process considered in the examples in this thesis is that of a 90 nm complementary metal-oxide semiconductor (CMOS) ASIC tool set. The process of recovering data that has been corrupted by channel modal imperfections and additive noise is called electronic dispersion compensation (EDC). Electronic dispersion compensation is normally performed in the analog domain at symbol rate prior to a monolithic analog-to-digital converter.

There are many theoretical advances that have been proposed by other EDC researchers, but their common theme is the requirement for the implementation to be performed in a full custom digital or analog design flow. A full custom digital or analog design flow adds significant technical risk to the project. Sources of this risk include increased complexity, additional gate-level simulations, increased design and layout time, and the increased schedule impact of late design changes.

The goal of this research is to perform the EDC design in a well-characterized digital process with stable, evolutionary tools that will reduce the risk from a timing closure aspect. The design process is referred to as “standard-cell” process because the physical layout macros are chosen from a limited selection of pre-built designs. If the digital logic is

implemented using a standard-cell methodology, the only remaining part of the circuit to be implemented in an analog process is the ADC. By re-using an older, well-characterized ADC, the custom layout work is significantly reduced.

The standard-cell methodology has some significant advantages over the custom layout methodology. Standard-cell design rule checking is more predictable than custom tool flows and produces regular, repeatable results. Register transfer level (RTL)[1] code changes can also be relatively quickly implemented when compared to the custom flow.

While the ease of use and faster turn around time are hallmarks of the standard-cell process, the custom design flow is generally able to reach much faster speeds. The rule of thumb is that custom layout can achieve up to an order of magnitude improvement over tool-driven layout. For example, EDCs currently on the market are implemented in an analog 90 nm process and operate at 10 GHz. Personal experience with this 90 nm digital library has shown that the fastest clock speed achievable is in the 800 MHz - 1 GHz range.

The majority of the adaptive equalizers on the market today are implemented via custom design flows simply because of the performance a custom design flow can achieve. Up to this point, there has not been an equalizer design that could process data at 10 GHz and still be implemented in a standard-cell process. Until such a design was found, performance and engineering trade-off studies could not be performed. The research presented here demonstrates that 90 nm is the first process step where a 10 GHz EDC can be implemented in a standard-cell methodology.

The key result of this research is a description of a pair of adaptive filters fully implemented in an RTL methodology for a 10 Gbps optical fiber communications link. In addition, the proposed design's performance is characterized and compared against the theoretical performance specified by the IEEE 8082.3aq study group.

[1] RTL is a coding methodology for hardware description languages (HDLs). The circuit designer explicitly defines the registers (also referred to as flip-flops) and how the outputs of one register become the inputs of another register. This type of coding methodology tends to produce the most highly timing-optimized results, yet still remains readable.

## 1.2 Problem statement

The goal of the IEEE802.3aq group (10Gb/s Ethernet Over Fiber Distributed Data Interface class Multi Mode Task Force) is to define a physical medium standard such that Ethernet frames can be transmitted across a variety of inexpensive multi-mode fibers (MMF) with a maximum bit error rate (BER) of 1E-12 for a minimum distance of 220 to 300 meters, depending on type of fiber.

The research performed by the IEEE study group concentrates on a physical medium device (PMD) performing data recovery at the A/D at a rate of 10 GHz using a decision feedback equalizer (DFE) to remove the inter-symbol-interference (ISI). Initial analysis [1] of the representative MMF models reported that in order for a finite impulse response (FIR) filter to invert some of the channels, a very large fractionally spaced equalizer (FSE) would be required.

The recommendation from the IEEE study group was a 20 tap T/2 (two samples per symbol) FSE along with a DFE composed of 4 taps spaced at the symbol rate. Analysis showed that this configuration could equalize 95% of the theoretical fiber models. The IEEE study group had proposed that this filter design can run at 10GHz in either a custom analog, SiGe, or CMOS process in a blind equalization mode. Once companies began to design to the standard, it was found that blind equalization would not work reliably. Research teams that have implemented products for this standard have added an eye opening monitor to provide an approximation of a training sequence. (J. Peeters Weem [2], 2006, Personal Communication)

An analog or custom implementation has several benefits compared to a register transfer level (RTL) design. First, it is much more power efficient. Second, a custom implementation typically results in a physical layout that can be clocked at rates that are an order of magnitude higher than what can be expected for a synthesis tool, thus reducing power and

[2]Dr. Peeters Weem is employed by Intel as a research engineer, working on various fiber-optic PHY implementations. I met Dr. Peter-Weems when he was an architect for Intel's 802.3aq PHY project, where he was responsible for modeling the behavior of Intel's proposed PHY design.

area as well.

By using a custom design flow, the design team assumes some technical risk. The custom design flow is significantly more difficult than an RTL flow and making changes to the design can be very expensive. A change to even a minor part of the circuit may require that the entire design be re-laid out by hand. After the change is implemented, all the design rule checks and physical simulations that were previously performed must be repeated, a lengthy and expensive process. If a process shrink occurs in a custom process, the entire IC layout may have to be repeated in order to take advantage of the smaller feature size.

On the other hand, if the number of filter taps is changed in an RTL design, the change can be implemented very quickly. RTL generics can be changed and the code re-synthesized with very little effort. The same ability to react quickly to design changes also benefits the RTL design in the process shrink scenario. The previous synthesis scripts are easily modified to use the new library and then can be re-executed. Only additional CPU time is required for the new digital layout to be created. The amount of redesign work in an RTL flow is limited to the physical ADCs. The physics of a process shrink guarantee that routing and propagation time decrease. Therefore, once a design has made timing in a larger, slower processes, the expectation is that the design will easily make timing in the smaller, faster process.

The IEEE 802.3aq study group is interested in providing a solution for the vast majority of the installed fiber at the maximum distance of 300 m. The problem under investigation is if a reasonable subset of fibers can be equalized without having to resort to custom or analog layout.

## 1.3 Background: Adaptive filters for equalization

The prior art for this research can be classified into three main areas:

1. The theoretical derivation of architectures and methods that invert an unknown channel. In particular, those methods that minimize some performance measurement in

order to converge to a steady-state approximation of the channel. Examples of using adaptive filters for ISI cancellation, such as FIR-LMS, DFE, and various other architectures, are reviewed in Section 1.3.1 on page 5.

2. Research into methods for pipelining or re-arranging a DSP-type algorithm without changing the underlying theory. In other words, how to make the theory more tractable for implementation. Section 1.3.2 on page 13 reviews this prior art.

3. Research into methods for implementing high-speed ADCs in silicon, either as monolithic high speed converters or as a collection of lower speed interleaved units, is covered in Section 1.3.3 on page 19.

### 1.3.1 Adaptive filter theory, algorithms, and implementations.

Researchers have been working to solve the problem of inter-symbol interference (ISI) since the days of the first modems. By 1985, the volume of research published on the subject of adaptive filters and how to remove ISI was sufficient for Qureshi [2] to publish a codified summary of the previous two decades. The source material for his review included over 110 distinct papers and extended back to the late 1960s. In this publication, Qureshi utilized a common mathematical framework for all of the reviewed methods so that the reader could concentrate on the conceptual and architectural differences, not on those caused by different derivation and notational styles. Some of the items presented in this paper were as follows:

- The definition of ISI and how it affects different media.
- Linear equalization and the differences between symbol spaced and fractionally spaced equalizers (FSE).
- Decision feedback equalizers (DFE), and how they differ from linear equalizers (LE).
- The difference between filters implemented in a direct versus transposed form.

- Implementing least mean square (LMS) equalizers.
- The performance trade-offs of binary integer math.
- The implementation of analog and digital equalizers, as well as programmable equalizers.

The essential theory of an LMS adaptive filter is conceptually quite simple. Data that is recovered from an unknown channel is filtered with an FIR filter. The output of the filter is "sliced" by a quantizer, and the pre- and post-quantized values are compared with each other. The difference between the desired and the actual result is defined as the "error." After the error has been determined, the tap weights of the filter are updated in a direction that would have reduced the error of the previous data sample. The process then repeats with another set of input data. If the proper performance metric is used and the amount of error adjustment that is performed at each step is the right order of magnitude, over time the filter should contain an approximation of the inverse of the transmission channel. Figure 1 describes the basic steps of an adaptive FIR, direct form filter.

The objective of an adaptive filter is to minimize a performance measurement by adjusting the filter's coefficients in response to an error calculation. In the LMS case, the performance measurement to be minimized is the square of the error term. The mechanics of the LMS derivation, which is where the weight update calculation is defined,is best left to other sources, for example, see Barry et al. Chapter 9 [3]. For our purposes, we will simply report the LMS equations in (1) so that they can be referred to later. The formula notation used in [4] is adopted as our standard notation.

$$c_k = c_k[n-1] + \mu e[n]x[n-k] \qquad \text{Tap Update} \qquad (1a)$$

$$e[n] = d[n] - y[n] \qquad \text{Error Calculation} \qquad (1b)$$

$$y[n] = \sum_{k=1}^{N} c_k[n]x[n-k] \qquad \text{Output Calculation} \qquad (1c)$$

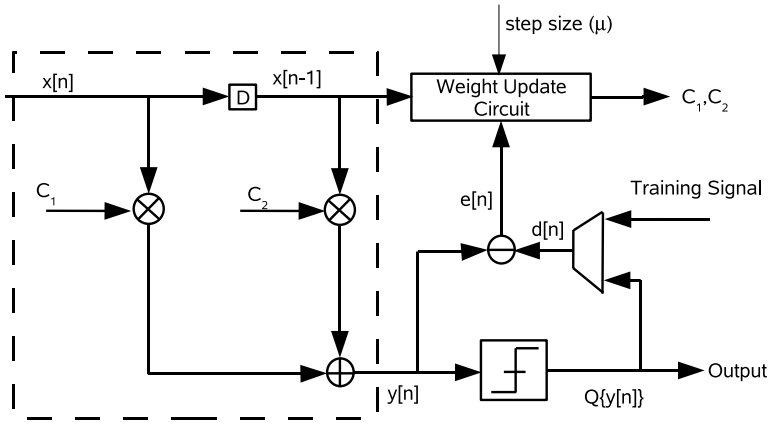


**Figure 1. Expanded view of an adaptive filter showing the composition of the two tap forward filter.**

The *N* in (1c) represents the number of taps in the feed forward filter. The remaining terms in (1) are defined in Figure 1.

Qureshi reviews up through the development of LMS and some of its variants, but does not describe either the block or delayed variant. Clark et al. [5] proposed a method for calculating a set of LMS outputs given a set of data inputs. The implementation method proposed was to convert a serial data stream into a parallel set of data and then calculate a “block set” of data either in the time or frequency domain. Once all the data sets were calculated, the weight update was calculated by averaging the individual weight updates from the corresponding serial LMS implementation. Figure 2 shows a conceptual diagram of this proposal.

Equation 2 shows the modifications that are made to the serial LMS equations for the block LMS algorithm. [5] showed that the miss-adjustment and conversion rate are equivalent to the standard LMS algorithm if the step size used in the block algorithm is equal to the number of blocks multiplied by the standard LMS step size. ($Block\ Step\ Size\ =$

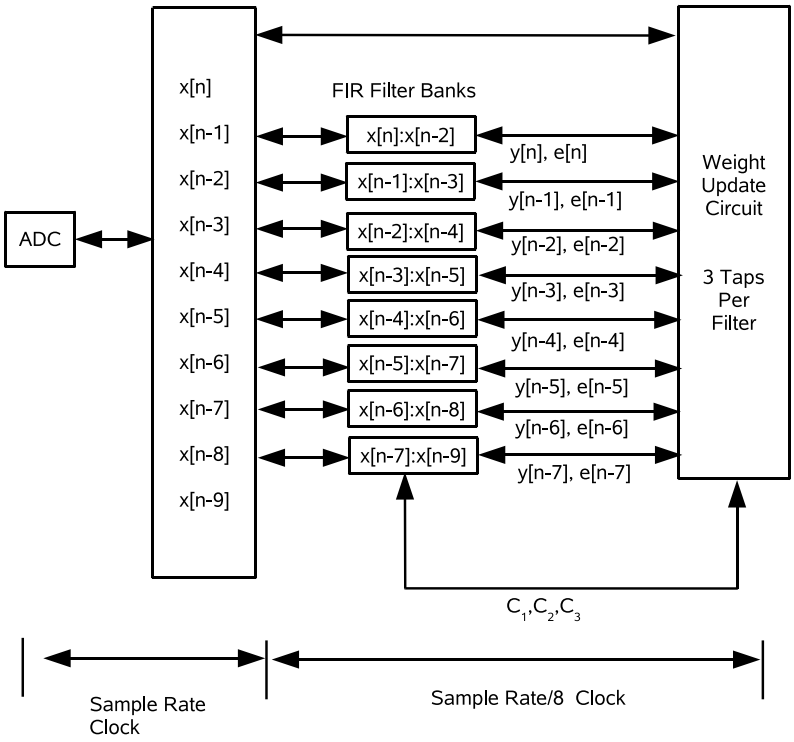


**Figure 2. Example of the block LMS architecture with block size of eight and three filter taps originally proposed by Clark et al.[5]This example calculates eight filter outputs for every tap update calculated. Eight data samples are collected between filter calculations, allowing the filter clock to be run, in theory, at $\frac{1}{8}$ the data symbol rate. In reality, the filter clock rate most likely is faster than $\frac{1}{8}$ the symbol rate because the error and weight update must be calculated before the next set of filter results is begun.**

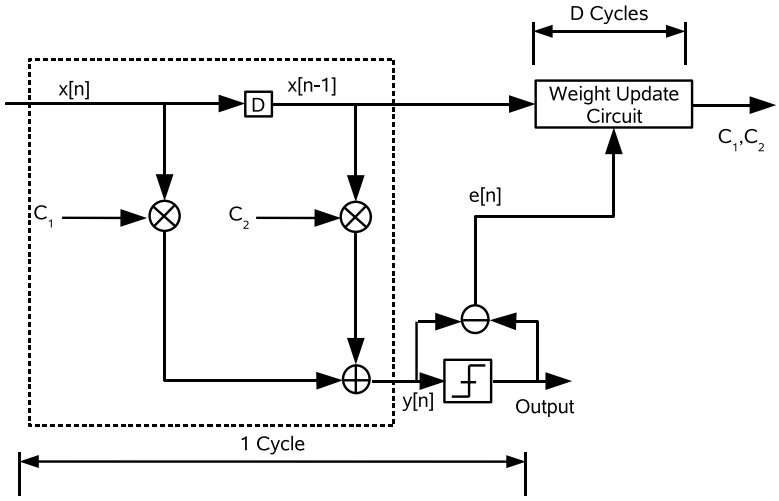


**Figure 3. An example of the delayed LMS architecture proposed by Long et al. [4]. The forward filter and slice operation occur within a single data clock cycle, but the calculation of the weight update is allowed to be delayed as long as the delay does not exceed the length of the filter. Thus the tap weights being used to calculate the filter output could be several cycles old.**

$L * LMS\ Step\ Size$.)[5, pg 28] The tap update equation variable $r$ is used to retrieve the sample values that were multiplied by a particular tap when an error calculation occurred.

$$m = \lfloor \frac{n}{L} \rfloor \qquad \text{Define the coefficient time index} \qquad (2a)$$

$$c_k[m] = c_k[m-1] + \frac{2}{L}\mu \sum_{r=(m-1)L+1}^{mL} e[r]x[r-k] \qquad \text{Tap update} \qquad (2b)$$

$$e[n] = d[n] - y[n] \qquad \text{Error calculation} \qquad (2c)$$

$$y[n] = \sum_{k=1}^{N} c_k[m]x[n-k] \qquad \text{Output calculation} \qquad (2d)$$

The coefficients are updated after every $L$ output calculations; hence the definition of the separate time index. (2b) shows that every tap is updated with an average of the $error * data$ product that occurred during block $m$.

In [4, 6], the authors show that introducing a delay in the coefficient adaptation has only a minor effect on the steady-state behavior ([6], page 1403), as long as the step size is chosen with sufficient care. This result led them to define the "delayed LMS" (DLMS) algorithm. Equation 3 shows the modification made to the standard LMS algorithm notation to account for the delay. The $D$ in (3) and Figure 3 represents the total amount of delay in

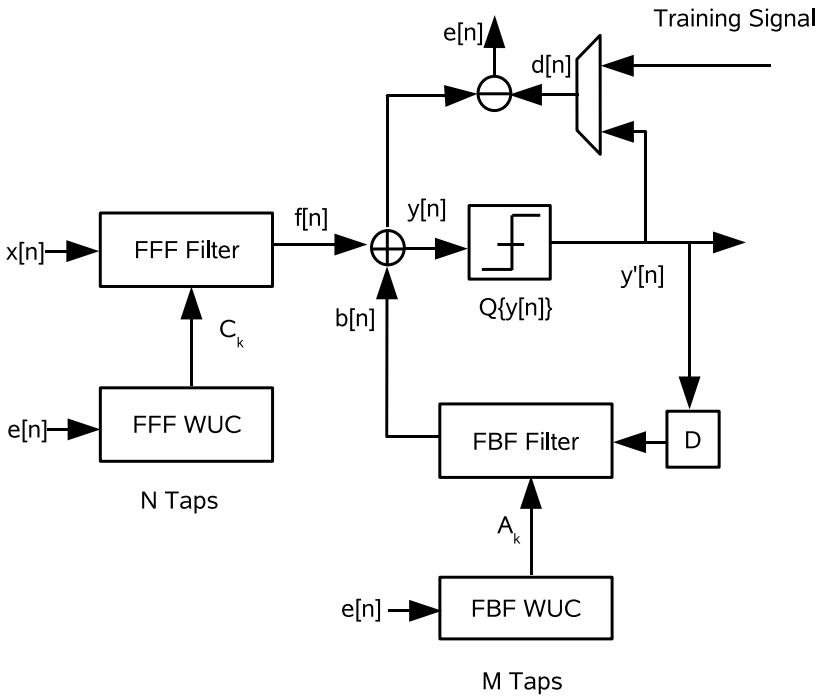


**Figure 4. A block diagram of the decision feedback equalizer data path. The DFE differs from the forward equalizer by the addition of an FIR filter in the feedback path between the output and input of the slicer. The feedback filter uses past decisions to predict the ISI affecting the current symbol. The error calculation is used to update both the feed-forward and feedback filters.**

the feed forward and coefficient update paths.

$$c_k[n] = c_k[n-1] + \mu e[n]x[n-k-D] \qquad \text{Tap update} \qquad (3a)$$

$$e[n-D] = d[n-D] - y[n-D] \qquad \text{Error calculation} \qquad (3b)$$

$$y[n-D] = \sum_{k=1}^{N} c_k[n-D-1]x[n-k-D] \qquad \text{Output calculation} \qquad (3c)$$

The discussion of the block and delayed LMS algorithms has focused on adaptive filters that are composed of a single FIR filter bank and that operate on the input data from the ADC. This method is equivalent to looking at a long length of the fiber at an instant in time, and then trying to decide what the bit in the middle position is, based on the waveform ahead and behind the bit of interest. This approach does not allow the system to take advantage of a powerful piece of information, knowledge of the bits that directly preceded

the bit of interest in time. In a dispersive fiber channel, the most significant impact on a given bit will be these data bits that were transmitted immediately prior to its own transmission. The decision feedback equalizer (Figure 4) uses previous decisions to predict the ISI contribution at the current sample time.

The DFE algorithm does not change the format of the LMS equations, but simply adds a few more to the set (4). The DFE algorithm adds an additional FIR filter in the feedback path, along with the associated weight update circuit. The feed back filter (FBF) is normally much smaller than the FFF, but there is no requirement for this to be so. The notation used in (4) matches that of Figure 4.

$$f[n] = \sum_{k=1}^{N} c_k[n]x[n-k] \qquad \text{Forward filter output calculation} \qquad (4a)$$

$$b[n] = \sum_{k=1}^{M} a_k[n]\hat{y}[n-k] \qquad \text{Feedback filter output calculation} \qquad (4b)$$

$$c_k[n] = c_k[n-1] + \mu e[n]x[n-k] \qquad \text{Forward filter tap update} \qquad (4c)$$

$$a_k[n] = a_k[n-1] + \mu e[n]y'[n-k] \qquad \text{Feedback filter tap update} \qquad (4d)$$

$$y[n] = b[n] + f[n] \qquad \text{Input to slicer} \qquad (4e)$$

$$d[n] = select\,(training\ input, y'[n]) \qquad \text{Selection of training signal} \qquad (4f)$$

$$e[n] = d[n] - y[n] \qquad \text{Error calculation} \qquad (4g)$$

$$y'[n] = Quantize\,(y[n]) \qquad \text{Slicer output} \qquad (4h)$$

In [7], the authors review and derive statistical and numerical methods for setting the FFF and FBF coefficients. Unfortunately, these methods require absolute knowledge of the transmitter and data channel characteristics which are not available a'priori in this solution space. However, this work makes an additional contribution by demonstrating the equivalence of the "predictive" architecture to the one shown above. Unfortunately, this alternative model still suffers the feedback issue that will be discussed in section 1.3.2.

The adaptive version of the DFE is derived in Barry et al.[3, pg 446]. The method the DFE uses to remove ISI is essentially the same as the linear equalizer. An error signal is used to adjust the filters so as to produce less error on the next data cycle. The DFE has a deficiency when compared to the linear equalizer. When a LE makes an incorrect decision, a single error is possible. Since a DFE assumes that all past decisions are correct, an incorrect decision by a DFE tends to induce a burst of errors.

Up to this point, the term LMS has been used to describe linear equalizers. However, the term does not define an architecture, but rather a method of cost minimization. Therefore, the DFE algorithm may also be described as an LMS algorithm. Since the publication of [5, 6], there has been a plethora of proposals for modifications to the LMS algorithm. These proposals suggest new ways to make LMS converge faster[8], reduce complexity and area[9], fit the algorithm into a regular array[10], or improve steady-state behavior for a particularly difficult channel [11]. The majority of these proposals become no more than academic references, as they are rarely implemented.

For most commercial implementations, block, delayed, or serial LMS is used because the algorithms are simple and easily partitioned into hardware, software, or mixed implementations. More importantly, the simple LMS algorithms work for the vast majority of channels. (W. Smith[3] 2006, private communication.) For those cases where a particularly difficult channel is required to be equalized, if the initial LMS implementation does not give sufficient performance, then one of the methods proposed in an academic study might be attempted.

The use of the LMS algorithm for channel equalization has remained a primary implementation choice for over 30 years because of its low complexity and its performance over a wide variety of practical channels. There have been many modifications suggested, but only two have become prevalent: the block and delayed algorithms [5, 4]. These two are

[3]Wesley Smith is a Principal Engineer with Intel's Software and Solutions Group. He is currently a communications architect designing systems supporting Voice over IP. He has over 25 years experience designing and implementing adaptive LMS equalizers, cancelers and other adaptive systems for voice band and DSL modems.

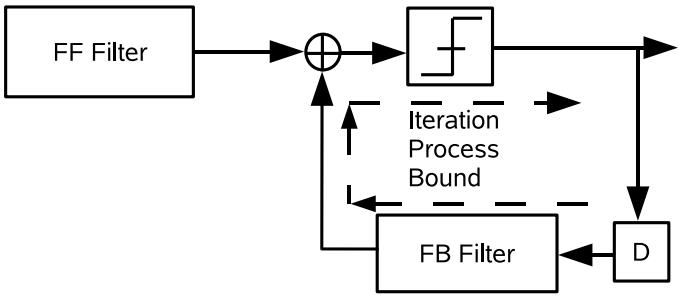


**Figure 5. Illustration of the critical iteration process bound (IPB) in a serial DFE system. The IPB is the path around the loop through the slicer. For a single tap feedback filter the IPB includes a full multiplier and adder.**

used as starting points for many proposals, but there have been no reported implementations of a block, delayed LMS algorithm.

### 1.3.2 Pipelining for implementation tractability.

Standard wisdom says that in order to equalize a high-speed, low-dispersion serial data channel, some type of decision feedback equalizer is required. Some channels may require a feed forward filter (FFF) as well. The forward filter section can remove the pre-cursor ISI, but, to remove the post-cursor ISI, a DFE must be implemented.

The implementation issue with a DFE is the single-cycle iteration bound through the feedback filter and the adder, as shown in Figure 5. At first glance, the iteration process bound (IPB) loop in the serial DFE shown in Figure 5 does not appear to be a critical impediment. If the feed forward filter can be pipelined to achieve arbitrary speeds, why not the feedback filter? The primary architectural hazard is that pipelining of the FFF was performed at the cost of increased latency through the filter. If the FBF is pipelined to have one cycle of latency, then the most significant post-cursor term will not be equalized. Instead, only the ISI contribution from two cycles beforehand will be equalized. For every additional clock cycle of latency in the FBF implementation, one additional post-cursor term is not equalized. The parallel implementation suffers the same structural hazard as the serial case; the feedback loop (Figure 6); the current output value relies on the decisions

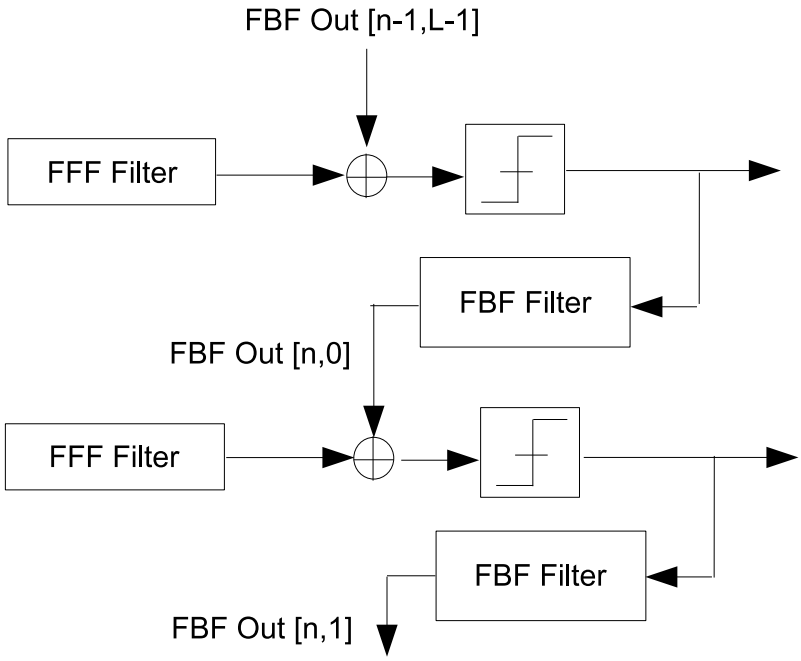


**Figure 6. Illustration of how the IPB affects attempts to implement a parallel DFE. Although the feed-forward filters are placed in parallel, the requirement for the DFE outputs to be resolved in a serial fashion results in no net change to the required timing. A single output must resolve in a 10 GHz clock cycle.**

made for the immediately previous data values. In the FFF, the output of the filter does not depend on the output of the previous data values.

Finding ways to speed up or completely eliminate this loop has been the goal of many researchers. Solutions for this problem fall into two categories, which will be discussed in detail later:

- Pipeline the original algorithm in some way so as to reduce the raw iteration bound time, but do not try to eliminate the hazard.
- Preserve the intent of the original algorithm, but re-characterize it so that the loop is unrolled, or some other method to remove the hazard is employed.

#### *1.3.2.1 Pipelining to reduce the iteration process bound*

References [12–19] are all examples of recent methods that attempt to insert pipeline stages into a filter in order to increase the sample rate. References [12–14] insert register delays into a flat FIR filter in an attempt to increase the sample time, while [15–19] focus on a DFE system. Each of these methods has some interesting contributions to the art, but they all have a critical liability in that they can only produce one result per clock cycle.

In [12], Karkada, Chakrabarti and Spanias propose a two-dimensional matrix of processing engines to perform the block LMS (BLMS) algorithm. This structure reduces the total execution time by dividing the work into parts that can be spread across processing engines. Regardless of how the work is divided, expanding the solution to incorporate a DFE still requires the the result of the previous calculation before the current sample calculation can be started. The penalty of only being able to calculate one sample at a time precludes use of such a DFE in our method, but the authors do mention two items that could be of potential use to us.

- Define $x[n]$ as the sampled data and let N represent number of items in the "block." Then, if $x[n]$ is stationary, the error summation step of $e[n] = \sum_{m=1}^{H<N}(d_m[n] - y_m[n])$ does not need to include all of the individual error calculations. As $N$ increases, the stability bound on $\mu$ (step size) becomes tighter. For a block LMS implementation, this potentially allows area and power savings.

- If $x[n]$ is stationary and slowly time-varying, the delay in the feedback loop will affect the convergence speed [12, pg 132], but not the final result. For optical fiber, the impulse response drift has been specified as 100 Hz to 1 KHz [1], which is 200,000 times slower than the worse case coefficient update time of our proposed system. Therefore, our system may be considered to meet the slowly varying criteria, allowing maximum pipelining of the implementation for maximum speed.

Douglas, Zhu, and Smith [13] propose pipelining a transposed form FIR filter, with the

addition of an LMS correction term first proposed by Poltmann [20]. Poltmann's correction term compensates for the delayed error update relative to the serial, non-delayed LMS algorithm. Poltmann defines a correction term to the weight update calculation step so that the DLMS algorithm converges at the same rate as the original LMS algorithm. This is the only implementation found that actually implements Poltmann's correction term. One of the trade-offs with this algorithm is that the output delay is equal to the number of taps in the filter.

Although the Douglas algorithm includes the error term update as part of the FIR filter implementation, the error correction term consumes a significant amount of resources and does not appear to be required for the types of channels being studied. The standard LMS requires $2N$ multiplies in the FIR filter implementation. ($N$ tap multiplies to generate the output of the filter, and $N$ multiplies to generate tap weight update equation.) The Douglas, Zhu and Smith algorithm consumes $5N + 1$ multiplies. When multiplied by 16 to form a block LMS algorithm, these extra resources make the algorithm untenable for our purposes. Douglas, Zhu, and Smith identify two areas that are "difficult" to implement with adaptive delayed LMS filters:

1. The authors claim that the adjustment of the optimum step size is problematic.

2. The authors claim that implementing a binary tree adder for the error update step is difficult to perform in VLSI.

In [14], Yi et al. propose a re-timed version of both a direct and transformed FIR filter for an FPGA implementation of an adaptive equalizer. Their solution is scalable in hardware (in terms of adding/deleting taps) but only produces one output per clock cycle and cannot be modified for parallel operation.

Chakraborty and Pervin [15] perform some innovative loop unrolling of the DFE equation so that the most significant term is produced first, which allows zero latency for the DFE. The only drawback from our perspective with this implementation is that it can only

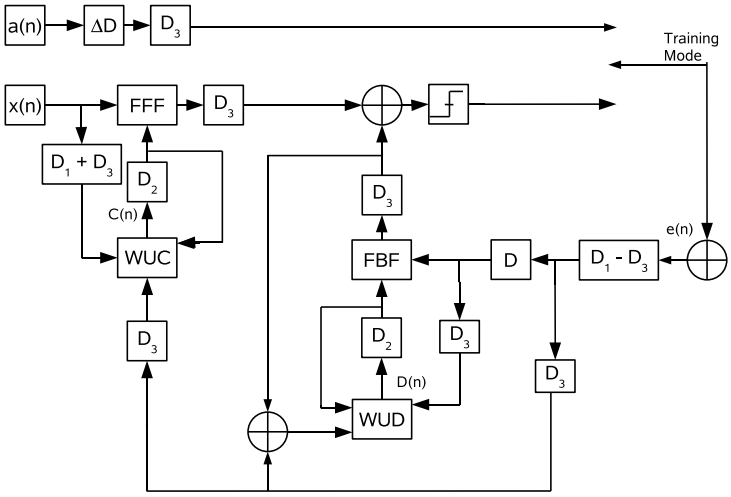


**Figure 7. Flow diagram of the pipelined adaptive DFE algorithm proposed by Shanbhag and Parhi. The algorithm is fully pipelined to maximize the clock speed. The circuit has not removed the feedback loop around the quantizer, which consists of a full adder before the slicer, and an adder after the slicer from which the error is calculated. The feedback loop is still too long for operation at a 10 GHz clock rate. The $D_x$ values represent delays added during different steps of the pipelining process. All the $D_1$'s were added during the same step. If the delay blocks are replaced with wires, the block diagram reduces to the same form as Figure 4.**

produce a single output per clock cycle, regardless of how fast it operates.

In [16], Wang et al. show a "hybrid" form of a direct form DFE. They make the claim that because the taps that are further away from the cursor are less important, they can be updated at a slower rate, thus sharing hardware which reduces the area, circuit complexity, and power consumption.

Like [16], [17] proposes a method of partial coefficient updates to reduce the complexity (and thus the power and area consumption). This paper builds on previous works by providing a theoretical justification for the minimization criteria that had previously been only quantitatively derived. In essence, the prior art reviewed and theoretically justified in [17] shows that by properly selecting the next block of coefficients to be updated, a normalized LMS (NLMS) algorithm converges with a properly chosen step size.

Shanbhag and Parhi [18] suggest a heavily pipelined method of the "predictor" DFE architecture that was originally shown to be equivalent to the standard DFE architecture in

[7]. The input to the slicer is a two input adder, which seems to be the perfect IPB.

Regardless of how much the DFE feedback path is pipelined, it still requires one clock cycle between every output produced. The IPB has been reduced for very fast execution, but it still requires a two input adder to execute at 10GHz, which is not possible for a standard-cell process.

In [19], Gatherer and Meng provide a parallel implementation of an ADFE circuit that self-corrects for incorrect decisions. In this case, the "parallel" label indicates that blocks of data are being executed in parallel but within the blocks of data, the individual data samples are still calculated individually. The proposed method inserts a preset code in between every block so that every block DFE knows the previous data values. Depending on the block size, this approach could create up to 20% of coding loss.

#### *1.3.2.2 Re-characterize the algorithm*

In a paper describing how to implement CDMA algorithms in digital logic [21], Parhi proposes a way to unroll a quantizer loop. In this paper, the unrolling theory is developed and several several examples of how to unroll a two- and four-level quantizer loop are shown. With some manipulation, Parhi's two-level quantizer loop can be shown to be equivalent to the DFE feedback loop that has been the subject of so much research. The chief benefit of Parhi's proposal is that the mathematics are all calculated as outputs from registers, and the final result is selected from two pre-calculated inputs. This selection process can be performed by a digital multiplexer. The loop unrolling is tractable because for most DFE implementations, the number of taps in the FBF is small. In this investigation only two taps will be used so the four-level quantizer is appropriate.

A derivation of Parhis proposal is made in [22]. The authors suggest that rough estimates of a channel's characteristics can be made in advance and used to pre-set the most significant taps of the DFE. The pre-calculated values and the forward filter can be added in parallel, while the lessor DFE taps are calculated. This reduces the number of adders in the unrolled pipeline. In this implementation however, the channels can not be estimated

apriori. The critical timing path remains the same for both algorithms.

### 1.3.3 Interleaved analog-to-digital converters and monolithic high speed EDCs.

When experienced engineers hear the term "interleaved ADC," many times their first thought is of gain and offset mismatches and how to reduce the effect of using non-matched ADCs to sample a single waveform. This view is most likely residual institutional knowledge of an effect that was first reported by Hodges and Black [23] in 1980. In this paper, the authors analyze and implement a quad array of smaller, slower analog-to-digital converters in an attempt to reduce the area of a fully parallel $2^n$ comparison circuit. They found that by interleaving smaller ADC's, the conversion time can be reduced by 1/2, and the required area can by reduced by 2/3, when compared with a full parallel ADC. In addition, metrics for the sensitivity of different array non-linearities were reported:

- The array is eight times as sensitive to phase mismatch on the sample clock as gain mismatch.
- The array is 1.3 times as sensitive to offset mismatch as gain mismatch.

Hodges and Black also derive expressions for calculating the reduction of the system SNR for a set of design parameters. Overall, array mismatches caused a decrease of 2 dB (from 41.5 db SNR) in the output SNR, as measured by error power versus pure tone signal power. They did postulate that in general, phase, gain, and offset mismatches between the individual converters in the array would be manifested as increased non-linearities in a monolithic analog-to-digital converter.

In our application, we are not trying to digitize an eight or ten bit value that has unique properties (i.e, a video signal) for each bit. We are sampling a serial data stream and trying to determine if the value is a '0' or a '1'. We may be receiving a multi-bit representation that we will first have to evaluate for the 0/1 threshold, but quantization effects will not have the impact that they would in a Nyquist rate converter for an application like a video stream.

We have not found prior art that quantifies the effects of offset and gain differences between sub-rate ADCs in a serial data stream application. However, Milijevic and Kwasniewski [24] measured less than one dB of signal loss due to gain mismatch when 10 interleaved ADCs were calibrated via a “Digital Reference Calculation” step. Varzaghani and Yang measured a loss of roughly 2.5 dB due to a clock jitter of 20 ps peak-to-peak. This clock jitter corresponds to roughly 1.2% of the sampling clock rate of 600 MHz. Varzaghani and Yang tested their device in a 0.18uM CMOS process.

As a comparison point to the ASIC method that we are investigating, the current state of the art in a full custom process needs to be described. Several recent papers have shown that if the design is implemented in an analog or full custom digital process, the chip designer has several trade-offs available in the complexity versus performance arena.

Milijevic and Kwaniewski [24] describe a method for a 4 Gb/s receiver that uses eight interleaved ADCs with a single tap DFE to implement a blind adaptive bit receiver. Each copy of the equalizer calculates two speculative outputs, assuming that the previous sample was either a one or a zero. When the previous value is known, a mux selects from between the two speculative choices, and the result is output on the negative edge of the sampling clock phase. One of the more interesting claims from this paper is that the coefficient update does not have to be performed every cycle but can be reduced by a factor of the number of interleaved equalizers. The timing requirement for the half cycle output from the previous decision makes this design unsuitable for an ASIC implementation. At 10 GHz, the spacing between the negative edge of one clock phase and the following clock phase is 5 ps, which is too fast for an ASIC mux and latch setup time constraint.

The three architectures for a custom DFE presented by Li, Wang, and Kwasniewski [25], extend the single tap DFE lookup architecture to two taps. If current mode logic (CML) is available as part of the process and the equalizer tap and CML latch can be combined into a single instance, 10 Gb/s operation is possible across a significant amount (18”) of standard circuit board material (FR4.) This architecture presents the same difficult

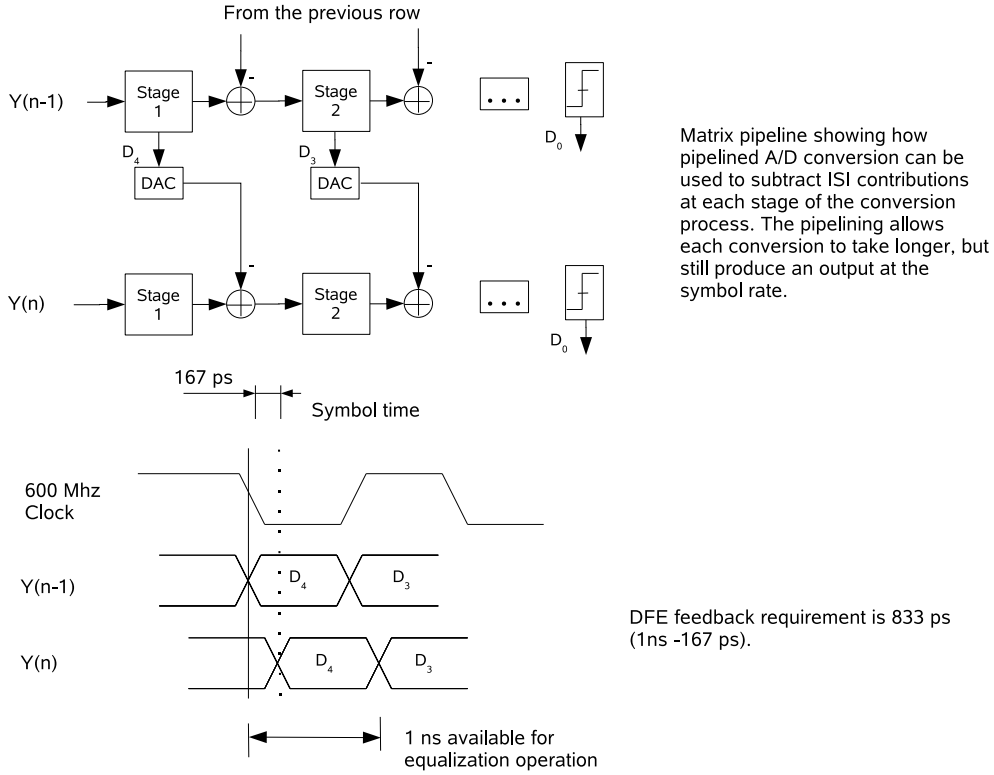


**Figure 8. Varzaghani and Yang's pipelined ADC and ISI equalizer circuit. As the ADC resolves one bit at a time during the conversion process, the ISI is removed. There are actually two pipelines in the circuit. The first pipeline is between successive symbols; the second pipeline is between the stages of the ADC resolution process. Figure reproduced from [27, Fig.6].**

requirement as other proposals; the switching frequency requires analog components. In addition, this is a pure DFE implementation. Our method requires pre- and post-cursor ISI cancellations, adding an additional full-adder to the equalizer critical timing path.

Like [25], the follow-on work [26], requires that the data bits be equalized one bit at a time, implying operation at 10 GHz. Even if the equalizer part of the circuit is split into even and odd components so that it can run at half rate, the clock requirement is still too fast for our methodology. In our operation, we will be receiving data 16 bits at a time, thus invalidating this architecture as a possible solution for the problem we are investigating.

Varzaghani and Yang [27] proposed an architecture where a single tap DFE is implemented inside the analog-to-digital converter. Their method is to formulate ISI as a per bit multiplicative error of the previous sample and then remove the ISI as each bit is resolved

in the current ADC data sample. What is of particular interest is that their method might be applied to the two tap DFE filter to allow pipelining. The most significant challenge in porting this algorithm will be to recast it in terms of using a five bit quantized value as the input, as opposed to the analog input signal the authors propose.

Xia, Ajgaonkar, and Rosenkranz [28] report on an equalizer design for 10 Gbps Ethernet. In this paper, the authors develop an FIR-DFE equalizer that uses non-linear elements where the input data is squared or multiplied against other data samples for some of the taps. Significantly improved results were obtained compared to the linear FIR-DFE adaptive filter. The issues with this architecture are (1) data samples still have to be calculated in series and (2) the non-linear operators make the timing issues experienced by a standard DFE implementation even worse. The authors also admit that this implementation is significantly complex and will impact area and power.

All of these methods assume that the equalization is being done one bit at a time, and only the Rosenkranz paper proposes both a pre- and post-cursor ISI solution. The other common assumption of these papers is that the analog components and custom layout tasks will allow the designs to run at the clock speeds necessary. In fact, the assumption of an analog process is required if a 10 Gbps signal is to be processed serially, one bit at a time.

For the design being proposed here, the ADCs produce data samples at a rate of 20 GSpS. In each clock cycle, 32 data samples are consumed, and 16 decision outputs are produced. The 802.3aq study group proposal requires a pre-cursor filter, which then places a requirement on any DFE addition that is added to the FSE design. Since there are no alternative FSE architectures that exist with the required performance, any DFE addition must produce 16 samples in a single 625 MHz clock cycle. Of the items reviewed to date, only the Varzaghani architecture shows any promise of executing at this rate, and then only for a single tap DFE system.

### 1.3.4 Literature search summary

Several types channel equalization methods have been examined and their relative merits reported upon. The class of equalizers known as linear equalizers has been shown to be fairly delay tolerant allowing arbitrary insertion of pipeline registers in order to achieve desired clock rates. The classic architectures of block and delayed LMS, along with several other linear equalizer variations, have been reviewed and their pertinent contributions characterized. Most of the recent linear equalizer architectures attempt to increase the algorithm speed when implemented on a general purpose processor. The techniques are generally not useful for a hardware implementation when compared with the ability to arbitrarily create duplicate, parallel calculation cores.

The decision feedback equalizer architecture has been shown to use the knowledge of previous decisions to improve equalizer performance on many channels. The drawback of the DFE is the feedback loop, and the timing requirements it imposes on the implementation. Several proposed methods of reducing the impact of the feedback loop have been reviewed, their uniform results are to reduce the calculation delay imposed by the feedback loop, not eliminate it. The majority of the reviewed methods concentrate on novel CPU array architectures or methods to reduce the calculation time of the feedback filter. For a hardware implementation, the loop unrolling method reduces the feedback loop to the smallest time delay. None of the reviewed proposals suggest a method to calculate more than one DFE output per clock cycle.

The final topic area reviewed was that of interleaved analog-to-digital converters. Initial publications indicated that phase and gain offset of the array components inserted noise into the frequency spectrum of the sampled signals. More recent papers have suggested that these sources have been eliminated for monolithic integrated circuit implementations by new techniques of on-chip calibration. The third noise source defined in the original papers was that of clock skew between the array components. Several recent research papers have reported the achievable clock skew at both 130 and 90 nm process steps. Similar clock

skew will be inserted into our research methodology.

# CHAPTER 2

# SIMULATION MODELING ENVIRONMENT

## 2.1 Introduction

The most deterministic method by which to characterize the performance of the proposed design would be to fabricate the design in the chosen 90 nm standard-cell process and physically test the design under real-world conditions. The primary impediment to this plan is cost. The expense of obtaining a “slot” in a fab and the non-refundable engineering cost of building multiple mask layers exceeds the budget for this project.

Even for a large commercial company, these costs are non-trivial and can exceed one million dollars. Before a company will commit such a large amount of fiscal resources, the design will normally undergo an extensive testing process. The testing regimen is divided into several segments.

1. Functional verification exercises every state in the design against all possible input vectors. The bulk of this testing is performed before the design is synthesized and converted to a gate representation.

2. Once the synthesis and layout task is complete, the internal paths are checked to ensure that the setup and hold timing margins are met between every source and destination node. This task is referred to as ”closing timing.”

3. Once the design has passed all of the static timing checks, several models of the design’s operating performance at different voltage, process, and temperature corners are generated. These models are then simulated using a sub-set of the verification suite that was executed in step 1. This process is referred to as ”back annotated timing simulations” because the actual path delays are annotated into the simulation model. These tests verify that the synthesis engine has correctly routed the design

with sufficient margin for various operating conditions. Once these tests are complete, the design is released to the factory for production.

4. When the parts are returned after fabrication, the functional tests are repeated in the lab using the actual hardware.

In the author's experience, pre-silicon verification tasks consume 50-60% of the project man-hours, while the design tasks consume only 25%. Because the pre-silicon verification process is used in industry to prove that the design is ready to be released to production, it will be used in this project to prove that the design is robust and operates correctly. In order to avoid performing an analog-digital co-simulation, the effects of noise and modal dispersion will be simulated as digital effects and applied to the data before the digital simulation is executed on the signal plus noise data set.

The static timing checks in step 2 verify that the design will operate at the desired clock rate. The static timing checks require a completely synthesized design, models for the analog pads, a clock tree, and the built in self test (BIST) logic to be inserted. This level of design preparation requires specialized tools and fab specific models that are not available.

For this project, we are not required to prove that the proposed design is ready for fabrication, but rather that the design has a reasonable probability of being successfully fabricated. The timing closure process is an iterative one, culminating in the checklist enumerated above. At each stage of the design, timing checks are performed to ensure that when the design enters the final timing check phase, there are no egregious timing offenders. Therefore, instead of completing the entire timing checklist, the first several stages of the checklist will performed, stopping when the next step requires tools or models that are not available.

The detailed procedure that will be used for validating this design is as follows:

1. The digital design will be synthesized using the target standard-cell library with a 7% timing margin, which will account for optimistic routing estimates. Timing margin

is normally added for block level synthesis. The purpose of the margin is to make sure that if input or output delays have been incorrectly estimated the neighbor block may have enough excess margin to enable the overall path to meet the flip-flop setup timing requirements.

Routing congestion is another reason why timing margin is used during block level synthesis. When all the design blocks are placed into the ASIC, routing congestion in dense areas of the design is common. Timing margin allows less than optimal routing to be used to route around the congestion.

The amount of margin used for block synthesis depends on the character of the design and the surrounding blocks. A small design with a regular data-path structure might only assign a 5% timing margin, whereas a very large mux block might use a timing margin of 10% or higher. The complex logic in this design is concentrated in the multipliers and adders, which operate on only two inputs per instance. Therefore, the amount of routing congestion is expected to be minimal. For these reasons, a margin of 7% was chosen for this design.

2. Using the known characteristics of the interleaved ADCs, a model of each channel will be created with which to filter the digital data. The analog effects of clock jitter, transmitter and receiver noise, and the channel will be imposed during the data generation step. By moving all of the channel effects into the data set, the purely digital RTL simulation is sufficient. Once the input data set contains all of the analog and optical imperfections, the operation of the digital receiver can be simulated. By using worst-case assumptions for those metrics that have not been characterized before, and measured real-world performance data for the portions of the design that have been characterized in a lab, this step will ensure that the analog noise effects are accounted for in the simulation. The performance of the analog-to-digital converters has already been validated in existing products.

3. A self checking RTL testbench will be used to characterize the performance of the design under test (DUT) against each fiber model. This testing will validate the architecture and detailed design of the proposed adaptive filter.

This chapter discusses the architecture and detailed design of the simulation environment, including the methodology used to account for the analog noise.

## 2.2 Analog noise sources

The world outside the digital portion of the adaptive equalizer imposes most of the signal corruption. Therefore, the raw digital data operated on by the digital filter must be corrupted in the same fashion in order for an accurate measurement of the design performance to be made. For each noise contributor or corruption source, a corresponding step in the data generation or simulation environment has been added, in the order that the noise occurs in the physical world. Figures 9 and 10 show the physical and noise models used to simulate the fiber optic communications link from the laser transmitter to the array of interleaved ADCs in the receiver. (Section 2.4 on page 33 discusses how the sub-rate interleaved ADCs are used to sample a 20 GHz signal.)

The physical characteristics are either specified as worst case values by the IEEE standard committee or are measured performance metrics of fielded 90 nm products. The IEEE standard specifies the pulse rise time. The transmitter and receiver noise figures are measured performance values for a product[1] that implemented the 2.5 GHz ADCs being used in this proposed design.

Figure 9 illustrates the physical data path of the optical channel. Data is launched from a transmitter with a 47 ps rise time and is injected into an FDDI class multi-mode optical fiber. At the other end of the optical fiber, the light is converted into an electrical signal by the PIN diode/trans-impedance amplifier that comprises the optical receiver. An array of eight analog to digital converters produces eight, five-bit values for every four data symbols

[1]The product referenced was a 2.5 Gbps ADC/EDC pre-production test shuttle produced by Intel in one of their fabs.

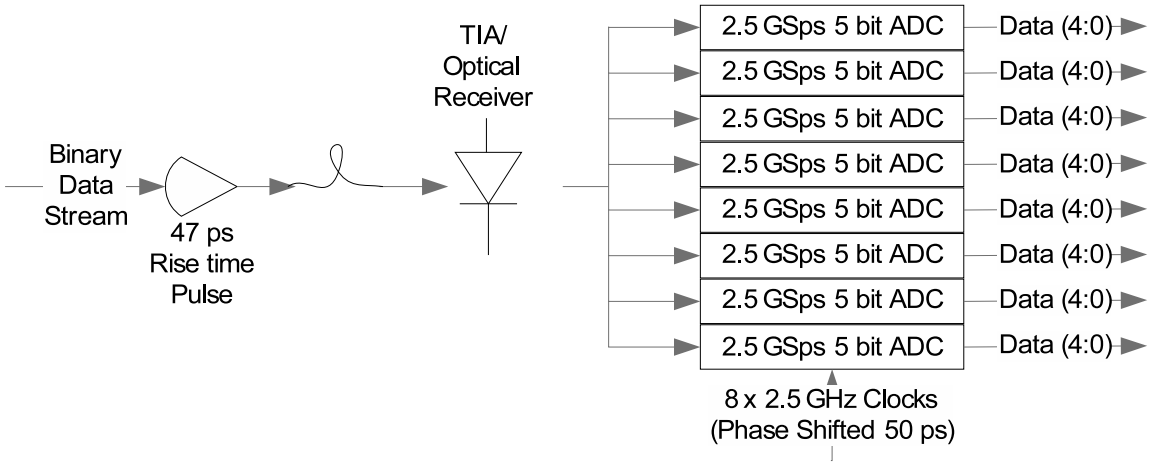


**Figure 9. Identification of the components that make up the physical transmission system. Binary data is launched into an optical fiber by a driver with a 47 ps rise time, transmitted over a multi-mode optical fiber with certain modal dispersion characteristics, converted back to an electrical signal by a PIN diode/TIA, and is finally sampled by a set of eight interleaved analog-to-digital converters.**

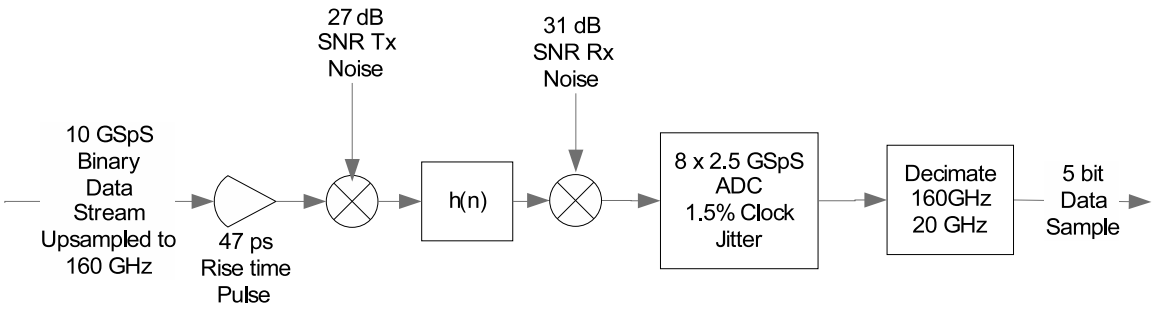


**Figure 10. A system diagram identifying the noise sources, and how they are modeled. The noiseless channel model for the optical fiber is represented by $h(n)$, while the Tx and Rx noise represent the additive Gaussian noise that simulates the effects of the physical transmitter and receiver.**

received.

Figure 10 illustrates the signal processing model of the noise sources in the physical system. The transmission laser induces relative intensity noise (RIN) noise. In the reference product, RIN has been characterized to be a maximum of 27 dB of additive[2] white Gaussian noise (AWGN). The data plus AWGN is corrupted by filtering the signal plus noise with the impulse response of the chosen channel. The optical channel shapes the signal and transmitter noise, which is then received by the PIN diode/trans-impedance amplifier (TIA). The PIN/TIA produces a voltage output proportional to the received signal. The PIN/TIA induces noise that is a function of the receiver's input sensitivity. The TIA noise contribution has been characterized as adding additional colored noise at 32 dB SNR.

## 2.3 Digital noise sources

The final noise contributor in the transmission system is caused by clock jitter between the eight phases of the ADC 2.5 GHz clock. Figure 11 shows the desired, perfect phase relationship between the different clock phases.

The digital clock layout is performed with the goal of achieving a perfect phase relationship between the phases. However, there will be some static phase differences caused by routing differences and manufacturing tolerances. Of additional concern, the PLL that generates the different clock phase inside the IC is not a perfect generator and will cause some phase and period differences on a dynamic basis within a part because of heat and age. The dynamic differences will also vary from part to part based on process and manufacturing tolerances.

Others [27] have reported achieving roughly 1.5% clock jitter in 0.13 um processes. Similar performance has been reported in the 90 nm literature [29]. To simulate the effects of clock jitter caused by the various routing, manufacturing, and process variations, the sampling of the data will be affected by simulated jitter.

[2] Another way to express this would be to say that after the noise was added, the output SNR was 27 dB.

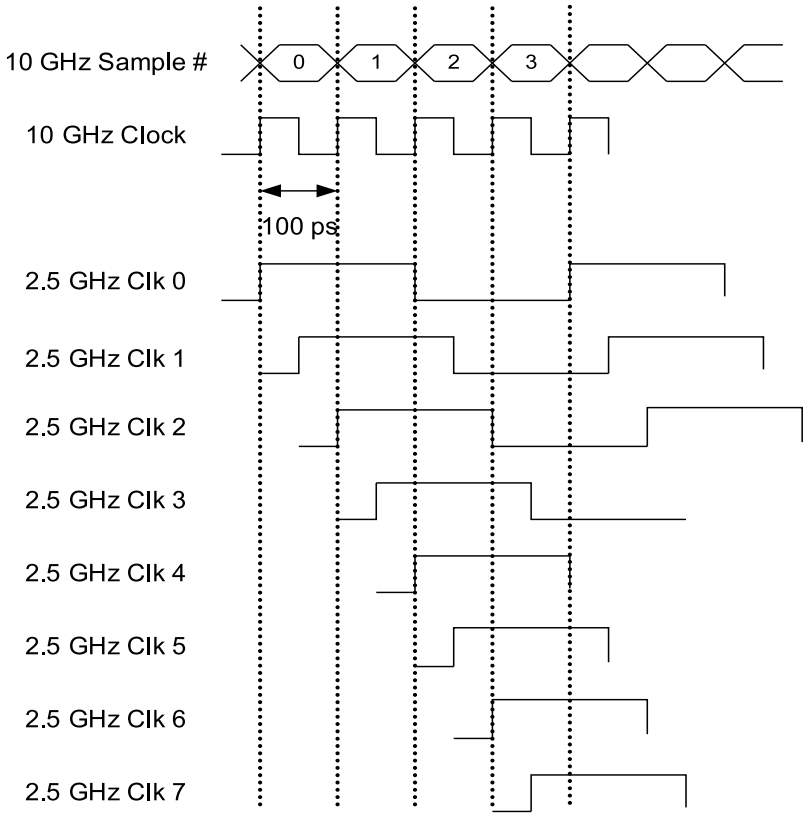


**Figure 11. A timing diagram demonstrating how eight interleaved ADCs operating at 2.5 GHz can sample a symbol train transmitted at 10 GHz with two samples per symbol.**

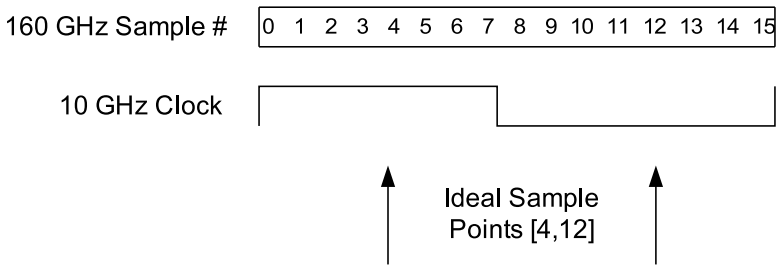


**Figure 12. A diagram demonstrating the relationship of the channel models over-sampled sequence numbers to the symbol rate of 10 GHz. The impulse response of the channel model was produced with a sample rate of 160 GHz, or an over-sampling rate of 16. If the sample sequence was to be decimated to symbol rate, sample number eight would be saved. Since a $T/2$ FSE is being implemented, two samples per symbol are saved, for a decimation factor of eight. The ideal sampling locations would be at [4,12], but, by choosing adjacent samples, ADC sample clock jitter can be simulated.**

### 2.3.1 Clock jitter simulation

The impulse responses for the various fibers were generated with a sampling resolution of 160 GHz to yield an over-sampling rate of 16. If the system being developed used a single sample per symbol, the ADCs would be configured to sample in the middle of the symbol period at sample eight. For a $T/2$ FSE, where two samples per symbol are utilized, the obvious choice would be to take the $4^{th}$ and $12^{th}$ samples. One method to model clock jitter would be to take neighboring samples from the perfect [4,12] case. Adding 1.5% of clock jitter to the 2.5 GHz sampling clock would require jittering the 400 ps clock by 6 ps. The smallest available sampling resolution is that of the 160 GHz time samples, which has a period of 6.4 ps. Offsetting the filter output by one of the 160 GHz samples results in a jitter of 1.6%. Therefore, the clock jitter may be simulated by sampling the output signal at an offset of $\pm 1$.

The entire simulation is performed at the channel model sample rate of 160 GHz. Once the channel output is calculated at this rate, two samples of the 16 samples per symbol are converted to five bit precision and saved with the remaining samples being discarded. The decimated data is saved into equivalent Matlab and text file formats. The text file is used by the VHDL simulation. Figure 12 shows the clock relationship between how the data samples are counted in the 160 GHz channel model and how the decimation occurs.

## 2.4 Matlab test bench

The Matlab test bench generates the data sets for the RTL simulation to process. This process is completed in three major steps: (1) Generation of the random stream of data; (2) generation an impulse response from the statistical channel model; and (3) filtering of the data set with the channel model, followed by addition of the noise contributions and decimation of the filtered data set.

Generation of the random data stream is performed by using the `randn` function in Matlab such that $100,000$ data samples of [-1, 1] are generated. The random data sequence is up-sampled by a factor of 16 to simulate transmitting the data sequence at 160 GHz in order to match the impulse response sample rate. The uncorrupted data is saved to become the reference data for error detection and training later in the simulation. This single raw data set is then used to generate the data set for every channel being tested. Every channel is tested with the same raw data set so that performance comparisons may be made.

The channel models used in this research are referred to as "The Cambridge Data Set" [30]. This data set was originally designed by a group at Cambridge University to simulate the worst case modal delays of multi-mode fibers. The data set comprises models of 108 fibers, each with various defects. Each fiber has three models for launch offsets of 17, 20, and 23 $\mu$m. The launch offset measures how far into the fiber the light source was inserted. These fiber models were selected by the IEEE 802.3aq working group as the fiber references to which the equalizer would be designed against.

Once the impulse response for a channel is calculated, several scaling operations are performed before using the result to filter the raw data. The impulse response is scaled so that the minimum value in the impulse response is zero, the max value is one, and the energy in the impulse response totals one.

The next step in the data generation is to use the generated data and impulse responses along with white and colored noise to create the simulation of a transmitted and recovered signal for each channel being tested. The simulated RIN noise to be added to the signal

is created by generating a random data sequence at 160 GHz and scaling it by the ratio of the energy in the data signal and the desired TX SNR (27 dB). The RIN noise is added to the data signal to create a model of the signal at the output of the transmission pin. The data signal and TX noise is then filtered with the channel impulse response. This colors the white noise that was added at the transmitter, as well as the data signal, with the response of the channel.

At the receiver, additional colored noise is added to the filtered signal to simulate the receiver imperfections. Raw white noise is generated using the same method that was used to generate the TX noise with the exception that the receiver SNR is used (31 dB). Once generated, the white noise is colored by filtering with a $4^{th}$ order Bessel filter with a 7.5 GHz cutoff frequency. The $4^{th}$ order Bessel filter has been shown to approximate the coloration of the noise that the receiver adds to the digitized signal [31].

Once these operations have been completed, the data vector is 1.6 million data samples long and contains the over-sampled data signal as it should appear at the input of a monolithic ADC. The final two steps of the data generation process are to simulate the effects of using interleaved ADCs instead of a monolithic ADC and to convert the infinite precision data vector into a five bit, twos complement representation.

Simulation of the interleaved ADCs occurs during the decimation step of the data set from a 160 GHz sampling rate down to the nominal 10 GHz symbol rate. Because the FIR filter uses two samples per symbol, the decimation step should theoretically take the $4^{th}$ and $12^{th}$ data samples from each symbol. These sampling points are equidistant from each other and are offset from the rising and falling clock edges in the raw data signal. (See Figure 12). By decimating at points other than [4,12] however, the effects of clock jitter between the interleaved ADC's can be simulated. The pattern used in the simulation is [4,11,5,11,4,11,5,11], which gives a repeating jitter pattern of [0 ps,-6 ps, +6 ps, -6 ps]. For example, the first symbol is sampled by ADCs zero and one and samples the 160 GHz data model at points four and eleven. The second symbol is sampled at model points five

and eleven. Symbol five is sampled at the same place that symbol one was sampled.

After decimation, the data is rounded, quantized to 5 bit resolution, and saved as Matlab and text files. The data file intended for the RTL simulation is scaled to a five bit integer representation, where the five values are coded as (Sign, $2^0$, $2^{-1}$, $2^{-2}$,$2^{-3}$). The coding is pre-set by the architecture of the ADC. The conversion from floating point to integer is performed so that the RTL can read integer values from a text file and convert them to bit vectors. Performing this conversion in Matlab is much easier and more efficient than performing the format conversion in VHDL.

Once all 324 data sets have been generated and saved into separate text files, the RTL testbench is run on each data file. The testbench records various data points inside the design and exports the data to text files at the end of the simulation. Some of the exported data is used for calculating the bit error rate of the filter. The bit error rate calculation could be performed in RTL, but a less error prone method is to read the simulation output records into Matlab and calculate the BER using the built in functionality that Matlab provides.

### 2.4.1 Bit error rate calculation

In a presentation to the IEEE study group, Bhoja et al. [32] specified the adaptive equalizer performance metric target as $1E-12$. Bhoja, Voois, and Shanbhag also specified an equation by which to calculate the BER:

$$BER = \frac{1}{2}\,\texttt{erfc}\left(\frac{V_1 - V_0}{2\sigma\sqrt{2}}\right). \tag{5}$$

Equation 5 actually calculates the probability of a bit error given statistics of the signal. The `erfc` function is defined as:

$$erfc(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}\,dt. \tag{6}$$

By examining the argument to the `erfc` function in (5), a instinctive understanding of how the bit error probability is calculated can be gained. The $\frac{V_1-V_0}{2}$ term is the distance of the symbols from the slice point. $V_1$ and $V_0$ correspond to the centers of the PDFs at +1 and

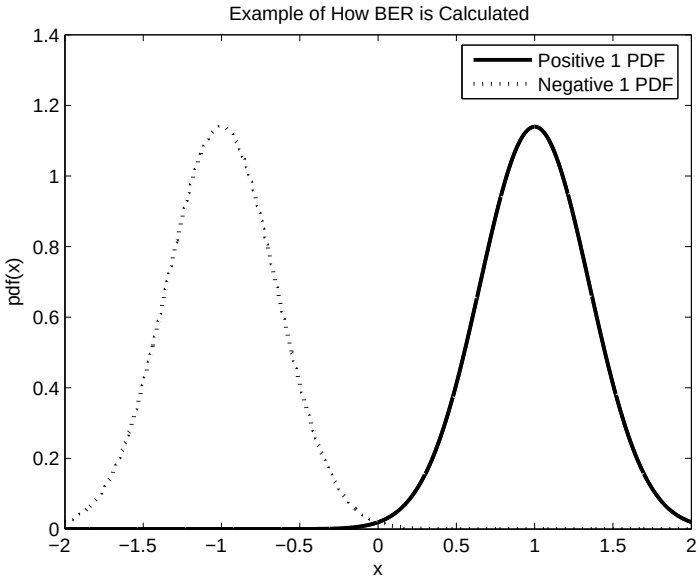


**Figure 13. The probability density functions (PDF) of the decisions at the input to the quantizer determine the BER. By recording the value that is presented to the slicer and the correct decision, the PDF for the $+1$ and $-1$ symbols can be created. The means and standard deviations of the symbols are used to calculate the argument to the erfc function, which then gives the bit error rate for the simulation. The erfc function calculates the error under the curve for the tails of the symbols that cross the slicer line into the other half of the PDF graph. Essentially the BER calculation determines the probability that a received symbol will be incorrectly classified by the slicer. The slicer's decision point is zero Volts. A symbol corresponding to $+1$ that is modified by the channel to have a value of less than 0 Volts will be incorrectly classified as a $-1$ and a single bit error will have occurred.**

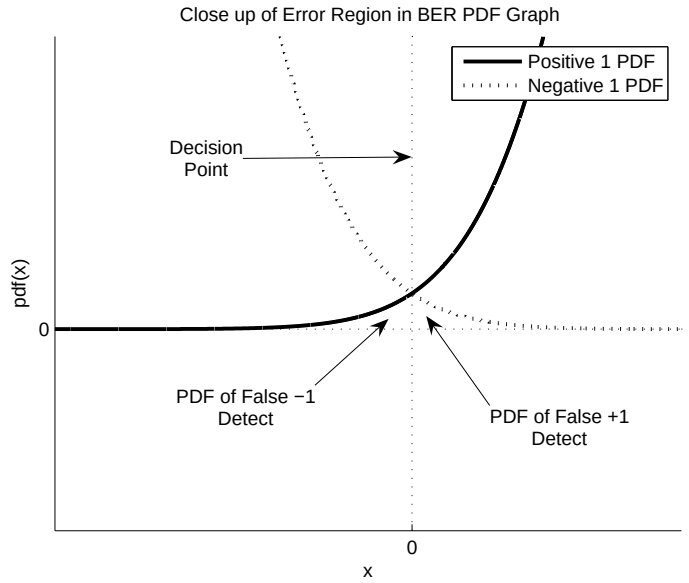


**Figure 14. A close-up view of the tails of the PDF in the false-detect region. The BER is one-half the area under the "$+1$" curve from negative infinity to zero plus one-half the corresponding area for the "$-1$" curve.**

$-1$ in Figure 13. The average of $+1$ and $-1$ is zero, which is the value the quantizer uses as a decision point.

The $\frac{1}{\sigma\sqrt{2}}$ term divides the distance between the slice point and the symbol center point by the standard deviation. Thus, the `erfc` function calculates the probability of bit error using the number of standard deviations that the symbol is from the slice point. The BER can be increased by decreasing the standard deviation or by moving the symbols further away from the slice point. Figure 14 shows an expanded view of the area around the slice point for a case where the standard deviation is large compared to the symbol separation. To obtain a BER greater than $1.03E-12$, then $\left(\frac{V_1 - V_0}{2\sigma}\right) \geq 7.02$.

The formula suggested by Bhoja et al. assumes that the slice point is the midpoint between the two symbol values and the standard deviation is the same for each symbol. Agrawal provides an alternative formulation that allows the standard deviations to be different for the two symbols and the symbol placement to be asymmetric around the slice point [33]:

$$BER = \frac{1}{4}\left(\texttt{erfc}\left(\frac{\mu_1 - 0}{\sigma_1 * \sqrt{2}}\right) + \texttt{erfc}\left(\frac{0 - \mu_0}{\sigma_0 * \sqrt{2}}\right)\right). \tag{7}$$

The zeros in the numerator of the `erfc` argument represent the slice point of zero Volts. Notice that instead of using predetermined symbol values, (7) uses the mean values of observed data. If the equalizer applies a DC offset to the quantizer input or if the two symbols had different standard deviations, (7) would correctly calculate the BER, whereas (5) would ignore these important differences. The BER calculation defined by Bhoja et al. (5) can be obtained by substituting the `erfc` definition (6) into Agrawal's equation (7) and

making some assumptions.

$$BER = \frac{1}{4}\left( \frac{2}{\sqrt{\pi}} \int_{\frac{\mu_1 - 0}{\sigma_1 \sqrt{2}}}^{\infty} e^{-t^2}\, dt + \frac{2}{\sqrt{\pi}} \int_{\frac{0 - \mu_0}{\sigma_0 \sqrt{2}}}^{\infty} e^{-t^2}\, dt \right) \tag{8a}$$

Assume that $\sigma_0 = \sigma_1 \;==\; \sigma$ and that $\mu_0 \;=\; -\mu_1 \;==\; \mu$ (8b)

$$BER = \frac{1}{4}\left( \frac{2}{\sqrt{\pi}} \int_{\frac{\mu}{\sigma \sqrt{2}}}^{\infty} e^{-t^2}\, dt + \frac{2}{\sqrt{\pi}} \int_{\frac{\mu}{\sigma \sqrt{2}}}^{\infty} e^{-t^2}\, dt \right) \tag{8c}$$

$$BER = \frac{1}{4}\left( \frac{2}{\sqrt{\pi}} (2) \int_{\frac{\mu}{\sigma \sqrt{2}}}^{\infty} e^{-t^2}\, dt \right) \tag{8d}$$

$$BER = \frac{2}{4}\left( \frac{2}{\sqrt{\pi}} \int_{\frac{\mu}{\sigma \sqrt{2}}}^{\infty} e^{-t^2}\, dt \right) \tag{8e}$$

$$BER = \frac{1}{2}\, \mathtt{erfc}\left( \frac{\mu}{\sigma \sqrt{2}} \right) \tag{8f}$$

In (8b) we assumed symmetric means. Therefore $\mu \;=\; \dfrac{V_1 - V_0}{2}$ (8g)

$$BER = \frac{1}{2}\, \mathtt{erfc}\left( \frac{V_1 - V_0}{2\sigma \sqrt{2}} \right) \tag{8h}$$

In addition to using all four of the calculated statistics, Agraval's formulation [33] allows the slice slice point to be moved relative to the symbol values in order to optimize the BER performance. This feature is not needed in this investigation because the equalizer does not appear to inject an DC offset. Choosing zero as a slice point significantly reduces the implementation complexity. The RTL slicer can be implemented by inverting the sign bit at the input to the slicer. A sign bit of '1', which indicates a negative number, is inverted to become a binary zero. Likewise, a sign bit of '0' is inverted to become a binary '1'.

During the simulation of each channel, the VHDL model records and exports to a text file the value of the input to the quantizer. Once imported into Matlab, the data is separated into logical values corresponding to 1s and 0s. The mean and standard deviation is calculated for the '1' and '0' data sets, and the BER is calculated using (7).

## 2.5 RTL test bench

The RTL test bench consists of several functions including input file management, instancing of the adaptive filter, test case management (clock and reset management), and data recording functions.

### 2.5.1 Input file management

The test bench opens two input files at the beginning of the simulation. The data input file contains $200,000$ five-bit samples corresponding to ADC samples spaced at 50 ps increments. The ADC samples are stored in the text file as repeating single samples from ADCs zero through seven. The test bench aggregates 32 samples, or four samples per ADC, into a vector and sends a single vector to the adaptive equalizer every 625 MHz clock cycle.

The second data file contains the reference data in a vector of $100,000$ binary values. The single bit values are collected into 16 bit vectors and sent to the adaptive filter bank every 625 MHz clock cycle. The reference values are used at the beginning of the simulation to train the filter and to simulate the use of an explicit training sequence or an eye-opening monitor. The eye-opening monitor has been used on the only available 10 GbE product [31, pg 8].

### 2.5.2 Instancing the adaptive filters

The test bench instances sixteen copies of the adaptive filter and controls the movement of data between each instance. Each copy of the adaptive filter receives a unique subset of the input data vector every clock cycle. The unique subsets contain 20 of the 32 data values in the input data vector. The test bench controls the generation of the sub-vectors as well as the alignment of the filter outputs, error calculation outputs, and data.

### 2.5.3 Test case management

The test bench performs several management functions that normally would be controlled by software running on an external processor.

1. **Tap bump**. An LMS adaptive filter uses the method of steepest decent to find the

optimum filter coefficients. By definition, the optimum filter coefficients may require more taps than what are implemented. If this occurs, the outlying taps will continue to accumulate energy in an effort to make the error zero, which can cause other taps to become stuck at local minimas. One common method to counteract this phenomenon is to multiply each tap by a fraction close to one which then causes the other taps to move off their local minima if they were indeed stuck there. [W. Smith, personal communication, 2006]

The test bench may command each tap to multiply its current value by the ratio of 254/255. Taps zero through nineteen are commanded in series to 'bump' their tap weight every 256 clock cycles. 256 clock cycles are sufficient for the other taps to settle out to a new minima before the next tap is bumped. Each tap is 'bumped' every $N * 256$ clock cycles.

2. **Tap centering**. In order to maximize the energy in the equalizer taps, the center tap should have the largest magnitude. The location of the center tap is set by aligning the reference bit and the output of the filter. If two fiber's group delays or physical lengths are different, the filter output may have to be delayed forwards or backwards in time by several bit intervals. The filter output may be shifted, or the reference/training signal may be shifted the opposite direction by the same amount.

   The test bench has a management function that looks for the largest tap magnitude and shifts the reference signal until the center taps contain the most energy. This function would normally be contained in an external processor, but, by placing the hardware to support the calculation in the design, the load on the external processor can be significantly reduced and the response time significantly increased. For example, by adjusting the cursor backwards for three bit intervals, the BER for fiber 34, offset 17, was improved from $1E-9$ to $1E-20$.

3. **Reset coordination**. A DFE circuit, when present as part of the equalizer, must be

held in reset until the FSE portion of the circuit produces valid outputs. If the DFE attempts to adapt to the string of zeros that the FSE produces on start-up, the series of tap updates can bias the FSE taps to an unrecoverable solution by the time the first FSE output is produced. The test bench coordinates two separate reset circuits, one for the FSE filters and a second reset for the DFE, the error calculation block, and the weight update calculation circuit.

4. **Error counting and statistics gathering**. The test bench monitors and records certain key intermediate values in the design. The recorded values can be written to an external text file for analysis by external programs. The data used to calculate the BER is gathered via a data recorder.

   The other operation that this module performs is counting the occurrence of errors. The test bench tracks incorrect results and from this, determines when the filter taps have converged. Upon convergence, this module commands the filter to shift from training mode to decision directed mode. In training mode, the reference data is used to form the error calculation. In the decision directed mode, the output of the slicer is used to form the error signal.

5. **Simulation execution scripts**. The RTL test bench is able to control the operation of a single test run, but VHDL is not a batch processing language. Therefore, a series of PERL language scripts were created to automate test operation. The RTL test bench assumes predefined values for the input and output file names. The PERL execution script creates filesystem file links between the generic input/output file name and the actual directory/file name. Thus allowing multiple input data files to be created for different amounts of additive noise and the results to be saved to separate directories.

## 2.6 Simulation summary

The simulation environment is composed of Perl, Matlab and VHDL software. Matlab is used to create a data set for each fiber optic cable under investigation and to simulate the effects of the analog channel and additive noise on the recovered data. The designs were described using VHDL and a test bench was created to help exercise the design. The test bench is self checking and exercises the data path as well as the various utilities that are present in the digital portion of the design. Perl scripts are used to automate the test bench so that regressions can be run over the entire data set, as opposed to a single test at a time. Finally, various Matlab scripts are used to analyze the performance of the VHDL design using data files generated during the test bench runs.

The correct operation of the test bench is a primary requirement for proper assessment of the design performance. The test bench itself has been subjected to tests in order to verify that the test bench detects all possible error conditions and correctly reports all gathered statistics.

# CHAPTER 3

# PARALLEL LINEAR EQUALIZER RESULTS

This chapter presents the architecture, design and performance of a parallel digital linear equalizer. The equalizer is shown to combine the delayed LMS and block LMS algorithms, be synthesizable in a 90 nm process, and to execute with a minimum clock speed of 625 MHz, resulting in an aggregate data rate of 10 Gbps.

## 3.1 Current 10 GHz analog EDC methodology

The IEEE 802.3aq standards body (10 Gb/s on FDDI-grade Multi-Mode Fiber Study Group) has reported a linear equalizer with approximately 30 T/2 spaced taps can equalize 95% of the fibers in the Cambridge data set, albeit with a larger EDC optical power penalty than desired by the IEEE [1, pg. 11]. In addition, a combination of 40 FFE and 3 DFE taps can equalize ($BER < 10^{-12}$) approximately 97% of the Cambridge channels. The study group's recommended implementation is an adaptive filter with 20 fractionally spaced equalizer (FSE) taps and four DFE taps. This configuration is predicted to equalize 95% of the channels with a 6 dB optical power penalty.

The recommended implementation is a serial one, where each individual symbol is processed by an equalizer with a clock rate of 10 GHz. By specifying that the equalizer would operate with a 10 GHz clock, the study group essentially specified that the equalizer and analog to digital converter would be constructed in an analog process. Since a digital implementation is proposed in this research, the round off noise added by performing the equalization using fixed-point math must be controlled as an additional source of error.

A 10 GHz clock is not feasible in a standard-cell 90 nm ASIC process given current capabilities. Such a design consumes too much power, and a clock of such a high rate is too difficult to route to a large number of standard cells. In addition, a monolithic 10 GHz ADC is not available, forcing the use of interleaved 2.5 GHz ADCs. As a result of these

restrictions, a 10 GHz clock cannot be used for the digital logic. The decision not to use a monolithic ADC adds additional sources of noise to those identified by the study group. The architecture proposed in this chapter accounts for the noise/error sources identified by the IEEE, as well as the additional sources imposed by the targeted system.

## 3.2 Converting analog algorithms to digital implementations.

When converting an analog implementation into a digital one, there are several purely digital considerations that must be managed. These considerations might be viewed as side-effects to the main effort of converting the target algorithm. However, not having a plan to mitigate these "secondary" effects can be a serious mistake. If untreated, these effects can cause an otherwise valid algorithm to fail.

As an example, the primary effort in this proposal was to convert a serial 10 GHz analog algorithm into a parallel algorithm that could operate at a lower clock rate, but still perform at an aggregate 10 Gbps rate. The sample rate algorithm must be converted into one that operates in parallel at $\frac{1}{16}$ the clock rate. The secondary effects to be managed are a mix of the standard issues that arise when implementing math in digital logic and signal processing algorithm-specific timing issues. The specific digital implementation issues that were managed or mitigated during this design are as follows:

- **Bit width versus the impact on closing timing:** As bit width increases, adders and multipliers become more complex and, thus, operate slower. In addition, there is a significant impact on routing complexity. Routing five bits between two nodes such that the path delay variation between the end points is small is much easier than trying to perform a similar action on a 12 bit value. In general, the more bits used to store information, the more difficult it is to meet the timing objectives.

- **Bit width versus algorithm performance:** A signal processing algorithm is ultimately precise when infinite precision math is used. In a fixed point, fixed bit width design, every calculation that involves truncating a signal adds noise to the result.

The magnitude of the noise can be as large as one half the value of the next lowest bit. Some algorithms do not perform well with a large amount of noise from this source.

- **Pipelining for timing closure versus the effects of pipeline delay on a feedback algorithm and power consumption:** Adding pipeline registers allows faster clock rates but also consumes additional power. A flip-flop consumes significantly more power than an "and" gate and does not perform any calculations. Adding pipeline stages adds logic that does not perform any "work", consumes power, and adds latency.

- **Trading off dynamic range versus precision for a fixed bit width implementation:** When implementing binary math with a fixed number of bits, the decision of where to place the binary point is of primary concern. Setting the binary point to the far left allocates more of the limited number of bits to the fractional portion of the number. This method biases the implementation towards greater mathematical precision, as the magnitude of the error caused by rounding or truncation is reduced by a power of two for every additional place that the binary point is shifted to the left.

  If the binary point is shifted too far to the left, and the mathematical operation being performed is a summation or multiplication whose magnitude grows quickly, the magnitude of the result might grow larger than what can be stored in the bit vector. When this occurs, the operation is said to have "overflowed." The difference between the smallest and largest values that can be represented is called the dynamic range of the bit vector. Shifting the binary point to the right to provide storage for the growth of intermediate results reduces the possibility of overflow but also reduces the precision of the fractional storage. The majority of the time the most significant bits are not used, wasting the storage that could be used for precision. Various methods

exist in the digital designer's toolbox to handle this issue, but care must be taken so as to not affect the main algorithm's performance.

The conversion of the serial adaptive equalizer algorithm into a parallel one has some unique challenges, but there are additional difficulties caused by replacing the high-speed monolithic ADC with an interleaved bank of slower rate ADCs.

### 3.2.1 Slower interleaved versus faster monolithic ADC

The conversion rate of an ADC is controlled by the longest of two time constraints: the sampling window and the conversion time. The sampling window is the amount of time that the sampling capacitor takes to charge to the input signal and determines how quickly the capacitor can track the changing input sequence. The conversion time is the amount of time needed to convert the value stored on the input capacitor to a digital value. A 10 GHz ADC must be able to charge the capacitor at 10 GHz and resolve the digital output before the next signal time.

In this project, a 10 GHz ADC is not available and has been replaced with eight 2.5 Gsps ADCs. The interleaved ADCs have a conversion rate of 2.5 Gsps, but a sampling window in excess of 20 GHz. With a $T/2$ FSE, the effective sampling rate is 20 GHz. Thus, the chosen ADCs have the required sample-and-hold performance but are too slow in their conversion from an analog signal to a digital signal. By using eight interleaved ADCs, the conversion time can be mitigated. As with any engineering trade off, replacing the monolithic ADC with an interleaved bank of ADCs introduces a new set of complications that must be managed.

In a standard analog 10 Gbps EDC the system only requires one ADC and EDC. This results in an implementation with the smallest area and power. Additional advantages of the analog methodology include only having to calibrate a single ADC and only having to mitigate the jitter on a single clock. In a system with interleaved ADCs, there are multiple calibration circuits required and the designer must be concerned with the post-calibration

output relationship between the interleaved ADCs. The jitter on a single clock can be characterized and controlled to fall within an acceptable parameter. When interleaved ADCs are driven by different phases of a clock, the jitter on each phase can combine to create a phase-to-phase timing relationship that is outside the design jitter parameters. The routing of the clock phases to the ADCs must be performed with care.

At the beginning of this research, the use of multiple ADCs was expected to require the modeling of each ADC as a separate entity, with individual bias and offset parameters and that doing so would add a substantial source of noise, as described in [23]. However, as further research was conducted into the state of the art of integrated circuit ADC design, recent techniques [27] in laser trimming and on-chip calibration circuits were found to have eliminated many of the problems reported in [23] and are now commonplace in the industry. The one difficulty that has not been eliminated is the ADC reliance on the clock phase between the members of the interleaved array. To simulate this effect, the simulated clock phases have been jittered by 1.5%, which is equivalent to a single 160 GHz time slot. The details of this implementation are covered in section 2.3.1 on page 32.

## 3.3 Derivation of the block delayed LMS algorithm

The algorithm proposed in this section is a combination of the delayed LMS (DLMS) and block LMS (BLMS) algorithms which are well known and originally described in [4, 5]. The algorithm introduced in this paper (referred to as “Block-Delayed LMS”, or BDLMS) applies the relaxed coefficient update timing rules defined in DLMS to the BLMS algorithm and results in a parallel, scalable, high symbol-rate digital adaptive filter.

The block LMS algorithm [5] proposes a calculation reduction mechanism whereby outputs are calculated in the standard serial fashion with the resulting error terms saved until some number have been accumulated. The number saved is called the “block size.” In one cycle, “block size” number of outputs and error terms are calculated and used to calculate a single update to the filter. This update is the average of the individual updates

that would have been applied in the serial algorithm.

In the BLMS algorithm, the updated coefficients are applied as soon as the block size is finished. For example, if the block size is 16, then as soon as 16 outputs are calculated, the tap weights are updated before calculation starts for the $17^{th}$ output sample. The worst case delay between an error term being calculated and the error update being applied is *Block Size* – 1 samples, and the average is one-half the block size. By averaging the error terms, the instantaneous coefficient updates are smoothed, but over time the BLMS algorithm delivers performance equivalent to the standard serial LMS algorithm. Figure 15 re-introduces the BLMS algorithm that was originally discussed on page 8.

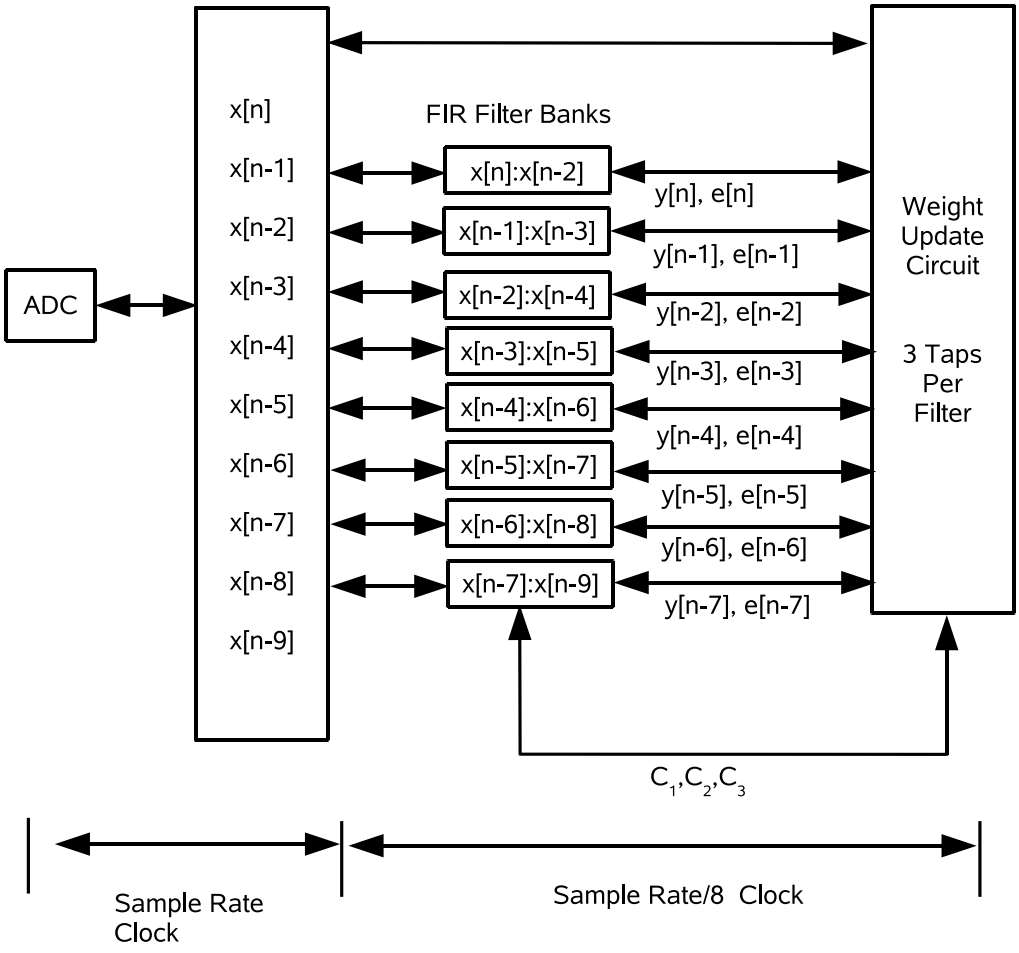


**Figure 15. A reminder of the BLMS architecture. See Figure 2 on page 8 for a full discussion of this architecture.**

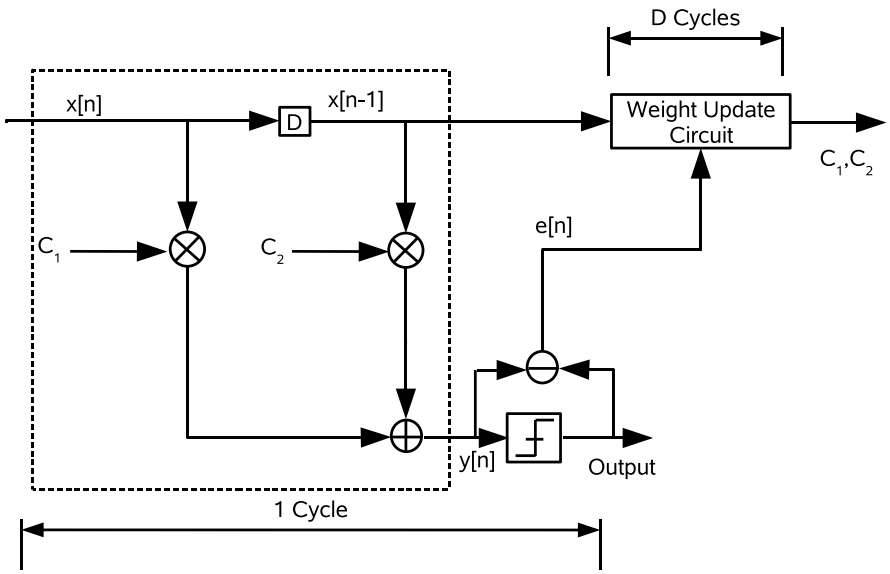


**Figure 16. A reminder of the DLMS architecture. See Figure 3 on page 9 for a full discussion of this architecture.**

In the delayed LMS algorithm [4], every output data sample is calculated in a serial fashion identical to the regular LMS method. Rather than immediately updating the tap weights based on the previous error before calculating the next output, the algorithm allows the tap update to be delayed by a fixed amount for every update. Thus, the DLMS method allows the filter output and error term calculation to be pipelined.

Figure 16 summarizes the delayed LMS algorithm, which calculates the updates one sample at a time. After the filter operation has begun and D samples have been calculated (where D is the pipeline delay of the filter output and the error calculation step), an update will be applied to the filter set based on a single error result at the end of every sample time. The DLMS research defined several boundary conditions that can be used in the current research to place a limit on possible solutions [4]. The conditions specified were (1) how much delay can be tolerated between the use of the tap weight and the error update from that sample and (2) an upper boundary on the step size that could be used in the steepest decent algorithm.

The guideline defined for maximum delay is that if the channel being equalized changes at some frequency, then the maximum delay in the error calculation must be less than the

period between changes. If the channel characteristics change while the first output is being calculated, the filter update could be in the wrong direction.

In the BLMS algorithm, the amount of delay between coefficient use and tap update is variable but constrained to be fairly small. The delay is fixed in the DLMS algorithm but can be longer than the BLMS algorithm. The BDLMS algorithm will need to extend the update time of the BLMS into the range supported by the DLMS.

The maximum expected rate of change in fiber, short of something destructive like a fiber kink or disconnect, is in the range of 100 Hz to 1 KHz [1]. At a symbol rate of 10 GHz and a clock rate of 625 MHz, a DLMS design could withstand up to 625,000 clock cycles of delay and still converge. Therefore, a standard DLMS implementation should be able to converge given a fairly long pipeline for the 10 Gbps fiber optic system.

A 10 GHz serial rate DLMS algorithm cannot be implemented in a standard cell process because that would require running digital multipliers and adders at 10 GHz. The clock buffers in the chosen process cannot operated at this rate. Realistic rates for this digital process are less than 1 GHz. Other experiences with this 90 nm process have shown a reasonable logic cone at a clock rate of 625 MHz. The proposed clock rate of 625 MHz is a power of two multiple of 10 GHz, which makes the interleaving and digital design easier. The method of the proposed algorithm is to integrate the block and delayed algorithms, resulting in an algorithm where sixteen filter results are calculated in parallel and their outputs fed into parallel error calculation and weight update circuits.

In the standard BLMS algorithm, the average delay between sample input and weight update is $BLOCK\_SIZE\ /2$. In the implemented BDLMS, the average delay is 27 times as large as the BLMS algorithm because of the pipelining required. The pipeline delay caused by the digital implementation must be analyzed to ensure that the delay does not prevent the circuit from converging.

The BDLMS algorithm (Figure 17) consists of eight interleaved ADCs operating at a combined sample rate of 20 Gsps. This provides two data samples per symbol. The data

samples are collected into a 32 sample data vector at 625 MHz. From the data vector, 16 sub-vectors are generated, and passed to 16 different FSE filters. Each sub-vector contains 20 of the 32 data samples. Each FSE produces a single result per clock cycle, which is then passed to 16 parallel error calculation blocks. The error calculation blocks determine the error between the FSE output and the desired output at a rate of one output per clock cycle. The weight update circuit calculates the amount that each FSE tap should be changed in order to reduce the average error for each tap. Figure 17 gives an overview of the algorithm and shows the clock cycle delay associated with each sub-part.

The next section details the impact of process limitations in the architecture of the resulting design and discusses the details of the proposed design.

## 3.4 BDLMS architecture and design

The architecture being proposed is a combination of the block and delayed LMS algorithms. BLMS suggests that a set of 16 filter outputs can be calculated with the same set of filter coefficients and the updates resulting from the calculation can be averaged and applied before the next set of filter output calculations begin. In the BDLMS method, the filter operation, error calculation, and tap update operations have been pipelined for maximum speed. Beginning with the performance analysis performed by the IEEE study group, a 20 tap $T/2$ FSE was chosen as the implementation target. The proposed BDLMS method scales to any size FSE without any impact on the algorithm other than resources and pipeline delay. If a different size FSE were desired, the circuit could be quickly modified by changing the VHDL generic statements that control the filter widths. The flexibility gained by using VHDL is offset by its slower performance. As previously discussed, the trade-off between bit width and operation speed is one of the primary design data points in a digital design. Before the BDLMS algorithm could be decided upon, the capabilities of the synthesis library had to be characterized so that maximum bit widths for each mathematical operation could be set. This data would allow trade-offs in the algorithm implementation

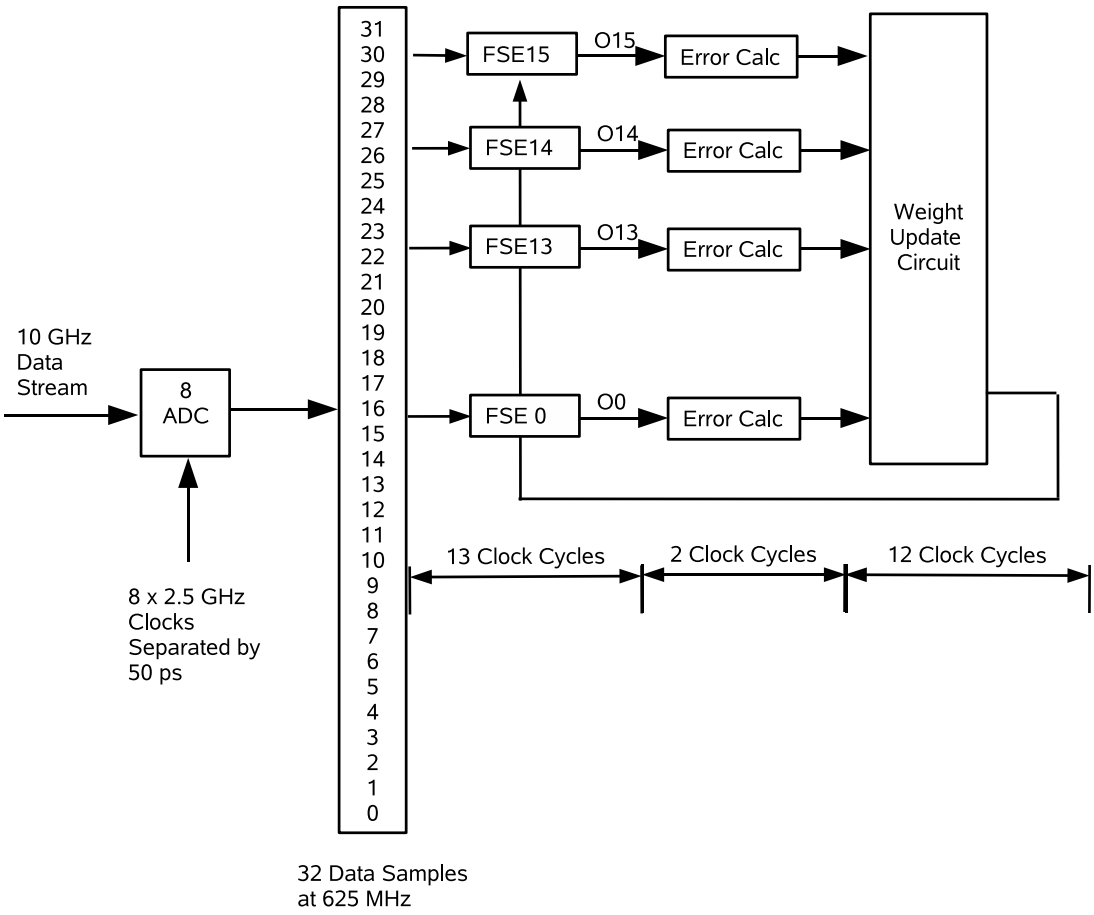


**Figure 17. The data flow diagram of the proposed block delayed LMS (BDLMS) algorithm. The data signal is sampled by eight interleaved ADCs, buffered into 32 data samples at 625 MHz, and processed by 16 instances of the BDLMS adaptive filter. Each parallel filter consists of a 20 tap FSE, error calculation block, and a shared weight update calculation circuit. The clock cycle labels show the latency through each part of the circuit.**

to be decided upon.

### 3.4.1 Synthesis derived restrictions on the architecture

One of the challenges of implementing theoretical DSP algorithms in hardware is the wide dynamic range that intermediate results tend to have. For a fixed point implementation, one cycle may find an intermediate result overflowing, while one cycle later the result is so small that it gets truncated to zero or worse, negative one. This issue is essentially an MSB versus LSB (most significant bit versus least significant bit) trade-off. If the fixed binary point is biased to provide sufficient bits at the MSB end, then overflows are prevented because the dynamic range is large enough. If the binary point is biased towards the LSB end, then the precision of the algorithm is improved. In an algorithm that uses an error term to decrease the steady state error, the LSB value sets the residual error. Once the error term drops below the LSB, the algorithm is done converging. If the LSB is set too large, the adaptive equalizer will not converge to a small enough residual error, and the implementation will not extract data at the desired data rate.

To find the largest size operations that could be performed in the chosen process, a series of synthesis experiments were conducted. Starting with the largest inputs supported by the synthesis library, the adder and multiplier circuits were synthesized, reducing the input width at every timing failure. The largest components that could operate at 625 MHz were found to be an 11 bit, two stage multiplier, and a 25 bit, one stage, full adder.

The results of the synthesis test create the boundaries within which the implementation must live. For example, anything that must be used as an input to a multiplier such as the error, sample data, and coefficient must be 11 bits or less. The leaf nodes of any adder trees must be less than 25 bits. Given that the multiplier is limited to inputs of 11 bits, the maximum adder tree depth is either limited to three stages[1] or overflow protection must be included in the adder tree.

[1] In order to avoid overflow, a fixed point multiplier with $N$ bit inputs produces an output of size $2N$ bits. A fixed point adder of size $N$ produces size $N + 1$ outputs. Therefore, starting with 11 bits as the input to the multiplier produces an output of 22 bits, which then leaves three bits for adder growth.

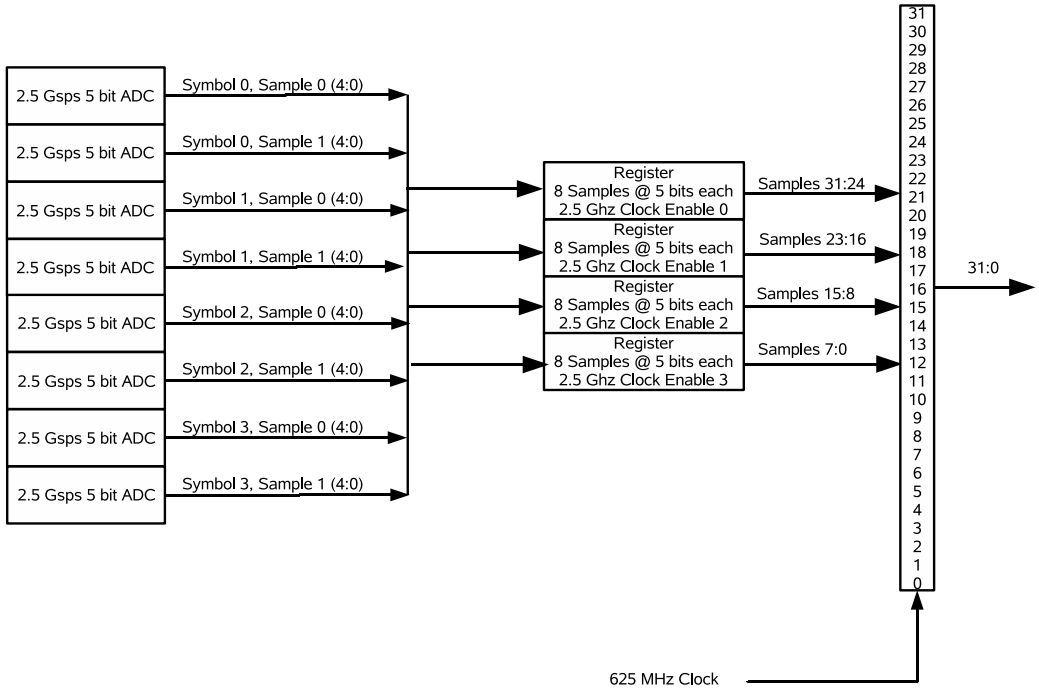


**Figure 18. This data flow diagram demonstrates the mapping of the ADC samples into registers at a rate of 2.5 GHz and then into a 32 sample wide register at 625 MHz.**

### 3.4.2 Converting serial data to a parallel format

The first step in the process is to convert the incoming 10 GHz serial data stream into a 625 MHz parallel data stream. Figure 18 shows how the data is received from the fiber in a serial fashion and converted into a 32 sample wide parallel structure.

Figure 19 explains how the parallel data is distributed to the 16 FSEs. Each of the filters receives a data vector to operate on. Data vectors zero through nine require parts of their data vector to be copied from the previous data vector.

### 3.4.3 Forward filters

Each forward (FSE) filter operates on ten symbols, each symbol composed of two samples, for a total input to the filter of twenty, 11-bit data values and twenty, 11-bit coefficients. The next filter in line receives nine of the same ten symbols, dropping the oldest symbol and adding a newer symbol, but receiving the same 20 coefficients. The multipliers and adders that comprise the filter have registers inserted after every intermediate operation.

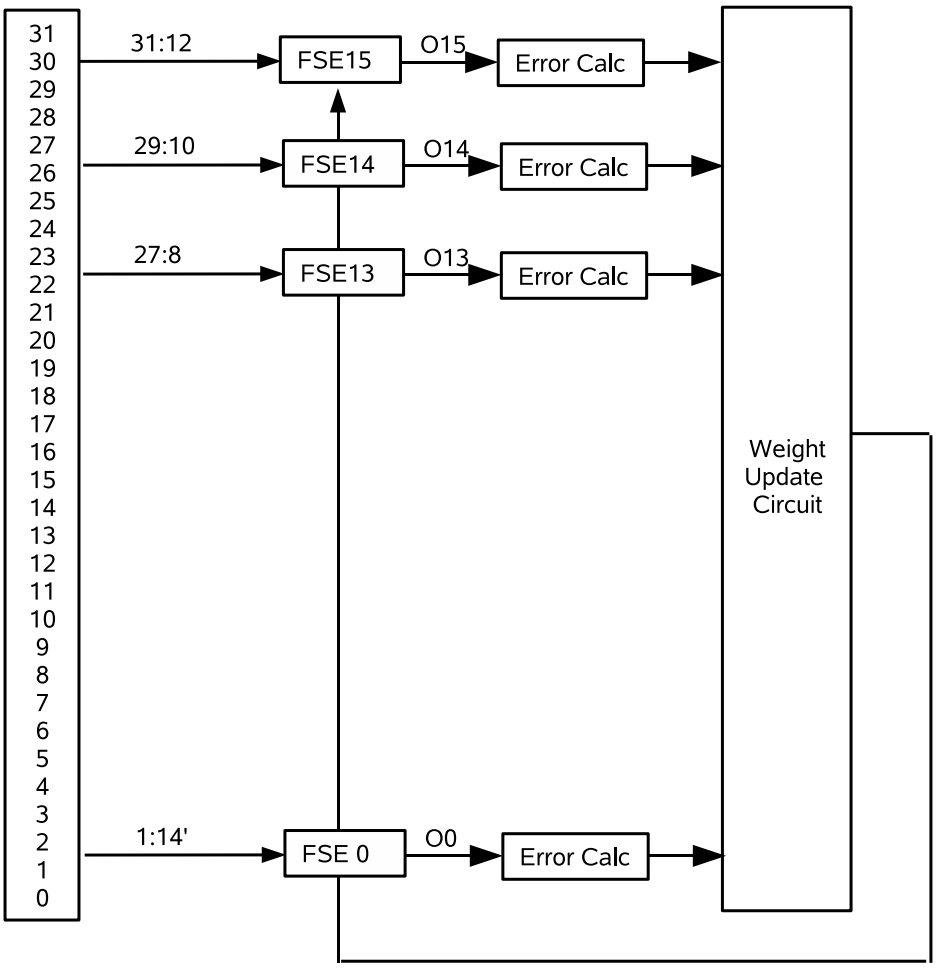


**Figure 19. This data flow diagram demonstrates how each FSE operates on 20 of the 32 data samples in the parallel register. Those FSEs that are older use a mix of data from the current parallel register and the immediately previous register to generate a 20 sample vector.**

This structure allows the filter to accept a new batch of inputs every 625 MHz clock cycle. Figure 20 details the bit growth and the register placement for the FSE filter.

Notice that although the bit growth is shown in Figure 20, the physical wires are a constant 25 bit value. The simulation analysis showed no cases of overflow resulting from not expanding the adder tree to 27 bits, as would theoretically be required. The dynamic range of the system is such that the filter output never grew enough to require the extra bits.

### 3.4.4 Weight update circuit

The weight update circuit (WUC) block is a wrapper for several sub-blocks. The logic in the WUC calculates the error for each filter, calculates the weight update for each tap, and applies the update to the tap holding register. The WUC is also responsible for delaying the input and reference data so that the weight update calculation block has the correct data samples. Figure 21 shows the component parts of the WUC block.

#### *3.4.4.1 Error calculation*

The first step of the WUC is to calculate the error for the just calculated output. The error calculation step is straightforward and is simply the subtraction of the FSE filter result from [+1, -1]. The selection of ±1 is based on the combination of training mode, the sliced FSE value, and the reference signal. The design of the error calculation block is shown in Figure 22

If the circuit is operating in training mode, the reference value used is selected from ±1, else the inverted sign bit of the filter output is used. The filter output is in twos complement notation, a sign-bit of '1' represents a negative number while a sign-bit of '0' represents a positive number. A binary '0' chooses a $-1$ as the input into the subtraction circuit. The error output is 25 bits. There are 16 of the error calculation blocks, one for each FSE filter.

#### *3.4.4.2 Weight update calculation circuit*

The standard LMS equations were defined on page 6. For conceptual simplicity, the tap update equation (part c) has been redefined in simpler terms so that the block modifications

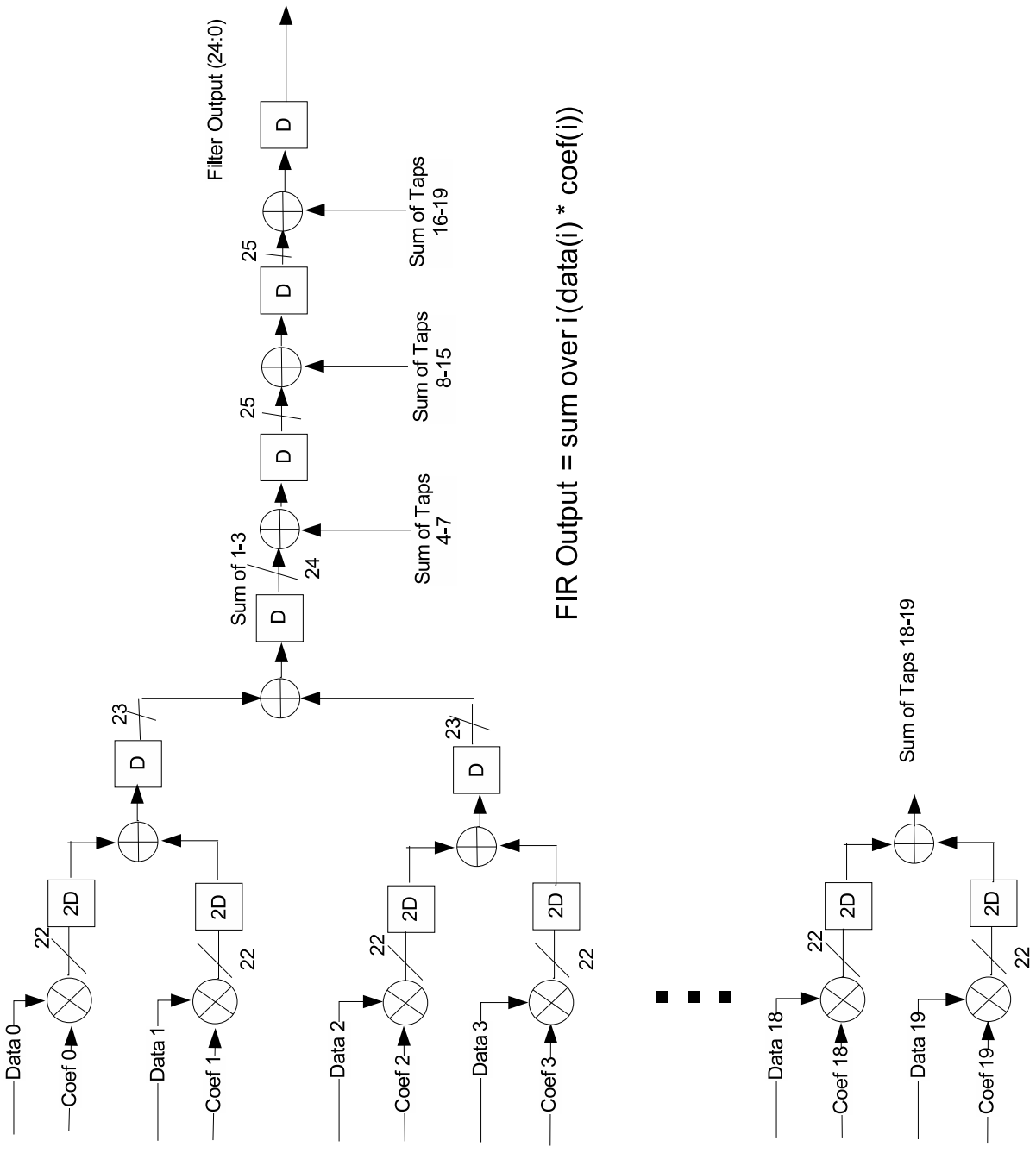


**Figure 20. This data flow diagram indicates the locations of the pipeline registers in the 20 tap FSE filter. The signals between mathematical operations are all 25 bits. The bit width labels in the Figure demonstrate the theoretical bit growth of the signal. Extensive simulation demonstrated that providing more than 25 bits for the signals in the filter was not necessary.**

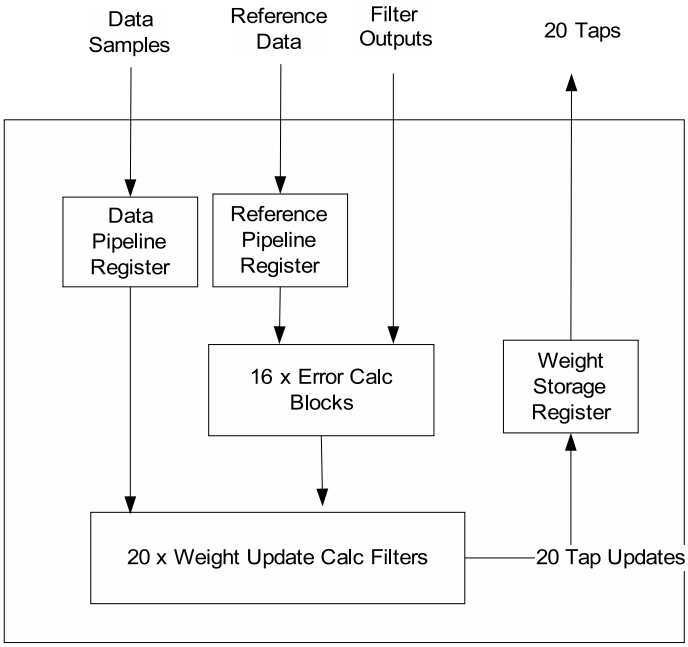


**Figure 21. The weight update circuit (WUC) consists of several sub-blocks, including the error calculation block and the tap weight storage registers. The WUC also delays the data from the ADCs so that the tap weight update calculation filter uses the proper data for the current error result.**

can be more intuitively discussed. The simplified notation for the serial tap update equation is

$$C_k[n+1] = StepSize * Error_n * Data_{k,n} + C_k[n]. \quad (9)$$

In Equation 9, $C_k[n]$ and $C_k[n+1]$ represent the value of the $k^{th}$ coefficient at the current time $n$, and at the next time cycle $n + 1$, respectively. The $StepSize$ is the adaptation parameter $\mu$. $Error_n$ represents the difference between the desired and actual filter output for the current time $n$. $Data_{k,n}$ represents the data value that was in the filter at time $n$ in tap location $k$. For a given tap $C_k$ at time $n$, the update is the error for the entire filter multiplied by the data value that was in the tap at the time. Therefore, the delta amount added to the coefficient is relative to the product of the error magnitude and the data magnitude.

For the block delayed LMS algorithm, the error update equation is very similar but is now composed of the average of the error-data product:

$$C_k[n+1] = C_k[n] + StepSize * \frac{1}{N}\sum_{i=0}^{N-1} Error_{n-i} * Data_{k,n-i}. \quad (10)$$

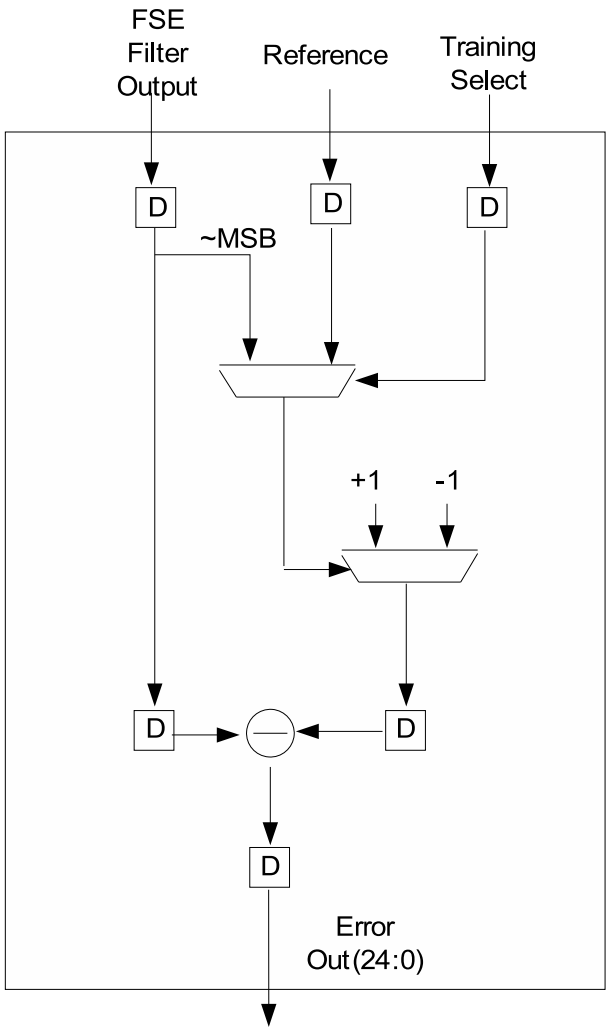


**Figure 22. The error calculation block subtracts the output of the FSE filter from either the reference signal or the quantized version of the FSE signal.**

**Table 1. Example of how the data sample indices are numbered in a small filter.**

| Tap | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Output | | | | |
| 0 | $d_0$ | "0" | "0" | "0" |
| 1 | $d_1$ | $d_0$ | "0" | "0" |
| 2 | $d_2$ | $d_1$ | $d_0$ | "0" |
| 3 | $d_3$ | $d_2$ | $d_1$ | $d_0$ |
| 4 | $d_4$ | $d_3$ | $d_2$ | $d_1$ |
| 5 | $d_5$ | $d_4$ | $d_3$ | $d_2$ |
| 6 | $d_6$ | $d_5$ | $d_4$ | $d_3$ |

This equation explains that every tap has 16 error-data products that contribute to its update. The key to implementing (10) is to recognize the summation as an FIR filter in which the error term replaces the normal coefficient. Once equation 10 has been recognized as an FIR filter equation, the next difficulty lies in determining which data values should be used in the calculation.

Determining the causal relationship between 16 filter errors, 20 tap weights and 36 data values can be challenging. The first step in specifying this relationship is to determine the location of every data value relative to every tap. To explain the steps, the solution to a four tap filter problem will be demonstrated.

1. Number the input data samples from 0 to 6, with 0 being the first (oldest) data sample.

2. Create a table with one column for each tap in the filter, and one row for each filter output to be calculated.

3. Fill the table with the data sample numbers that would be present in each tap at the output time. At time 0, only tap zero will contain a data value. All following taps will contain their reset/power-up values. Table 1 demonstrates how the table should look for a four tap filter at time $n = 6$.

4. To calculate the filter outputs, perform a series of multiply-accumulate operations starting with tap 0. The input to each multiply step is the value in the tap number

**Table 2. The example table re-labeled to demonstrate the tap weight update calculation.**

| Output Time | 3 | 4 | 5 | 6 |
|---|---|---|---|---|
| Tap Number | | | | |
| 0 | $d_3$ | $d_4$ | $d_5$ | $d_6$ |
| 1 | $d_2$ | $d_3$ | $d_4$ | $d_5$ |
| 2 | $d_1$ | $d_2$ | $d_3$ | $d_4$ |
| 3 | $d_0$ | $d_1$ | $d_2$ | $d_3$ |

at the top of the column and the data sample number listed at the intersection of the output row and tap number column. At time $n = 1$, the filter output would be $Tap(0)*Data(1)+Tap(1)*Data(0)$. Store the filter output in the variable `Output[n]`.

5. By substituting $k = 2$ and $n = 3$ into Equation 9, we can see the error that is calculated at time 3 is multiplied by the data sample that was in tap 2. Table 1 is used to find $d_1$ at the intersection of tap 2 and output time $n = 3$.

$$C_2[4] = Step\ Size * Error_3 * Data_{2,3} + C_2[3]. \tag{11}$$

This example has shown that Table 1 can be read in two different manners. When calculating the filter output, the table can be used to look up which tap number should be multiplied by which data sample. When calculating the tap update equation, the table can be used to match the error with the data sample number for a given tap number. Calculating the inputs for the tap update equation using Table 1 might be considered to be using the table "backwards", as the table is used by starting with the row, and reading the output along the column. Transposing the table places the outputs in their usual location. Table 2 transposes a portion of Table 1 as a demonstration.

The previous example demonstrated how to determine the relationship between the equation inputs for the case where a single filter output was calculated for every tap update. For the adaptive equalizer being discussed in this paper, 16 outputs are calculated for every tap update. We will revisit our four tap filter example as a demonstration, with the modification that four outputs (3 through 6) will be calculated from the same tap weights.

The filter calculation step is the same as in the previous example; use Table 1 to align the tap value and proper data value. However, the tap weight calculation step must now take into account the errors calculated at times 3 through 6. Using the example of Equation 11, determine the values for each tap for outputs $3, 4, 5, 6$. The tap update equation for tap two can be shown to be:

$$\begin{aligned} New\ Tap\ 2\ Weight\ =\ StepSize\ *\ (&Error(3) * Data(1) + Error(4) * Data(2)\ + \\ &Error(5) * Data(3) + Error(6) * Data(4))\ +\ (12) \\ &Current\ Tap\ Weight. \end{aligned}$$

From Table 1, the summation can be seen to be the vector inner product of the error vector and the tap column vector. Given this result, we can easily determine the causal alignment of the error output calculation, the data sample number, and the tap number, from the unrolled FIR calculation table.

1. Write out the FIR calculation equations for each of the FIR filters being implemented, as shown in Table 1.

2. Calculate the FIR filter's outputs and the corresponding errors.

3. Create a row vector, populated with the result of the error calculation steps.

4. Build a column vector with the data samples that are enumerated in the column beneath each of the tap labels in Table 1.

Using this procedure, the tap update for tap 2 is found to be:

$$\begin{aligned} New\ Tap\ Weight\ &= Current\ Tap\ Weight + StepSize * \\ &(ErrorRowVector\ *\ Tap2ColumnVector). \qquad (13) \\ New\ Tap\ Weight\ &= Current\ Tap\ Weight + StepSize * \\ &([e_3, e_4, e_5, e_6]\ *\ [d_2, d_3, d_4, d_5]'). \qquad (14) \end{aligned}$$

The preceding two examples have shown how an easily generated table of FIR filter equations can be used to determine the relationship between error calculations, data inputs, and tap weights. For the FSE filters that are being investigated in this project, the relationship is a bit more complicated. Because each data symbol is composed of two samples, the distance between data samples in the tap column vector is two. In the examples, the distance was one. Table 3 demonstrates how the data sample values align relative to the output calculation number and tap weight for the FSE filters.

Once the input data, tap weights, and error values are aligned, the tap update equation may be calculated by reusing the RTL that was used to calculate the filter outputs. The tap weight vector is replaced with the error vector, and the input data vector is rearranged. The FSE filter bank was sixteen filters of twenty taps, where as the WUC filter bank is twenty filters of sixteen taps.

### 3.4.5 Effects of coefficient storage register precision

Figure 23 shows the original design of the WUC along with the step-size multiply and update of the tap weight. During the performance analysis phase of the project, discrepancies were found between the RTL results and the predicted Matlab results. In an effort to characterize where the differences were being introduced, a cycle accurate Matlab model was written.

The model accounted for the effects of round-off, truncation, and limited precision mathematical operations that were occurring in the RTL. The investigation showed that the difference was not being introduced by the limited precision of the coefficient used in the FSE filter output calculation. Rather, the error was being introduced by the truncation of the weight update calculation prior to multiplication by the step-size. When the truncation was removed and the coefficient storage register changed to 25 bits, the bit error rate improved significantly.

There are two possible explanations for this behavior. The first is that when the mean of the quantization error exceeds the mean of the tap update, the SNR of the update falls

**Table 3. The table shows the indices of the sample data values that are in each filter and tap after the first 50 samples have entered the adaptive equalizer. To calculate the output of a given filter, the coefficient values stored in the taps across the top row are multiplied by the data sample numbers listed at the intersection of the tap and filter number in the body of the table. The 20 results are summed in a row to generate the filter result. To find the sample data values that should be used to calculate the tap update value for a given tap, the table is rotated 90 degrees so that the tap row becomes the column entry on the left side of the table (return the page to its normal reading orientation). For each item in the filter row, use the error that was calculated for that filter.**

| Tap | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Filter | | | | | | | | | | | | | | | | | | | | |
| 0 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| 2 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 |
| 3 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 |
| 4 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 5 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |
| 6 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 |
| 7 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 |
| 8 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| 9 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 |
| 10 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| 11 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 |
| 12 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
| 13 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 |
| 14 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 |
| 15 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 |

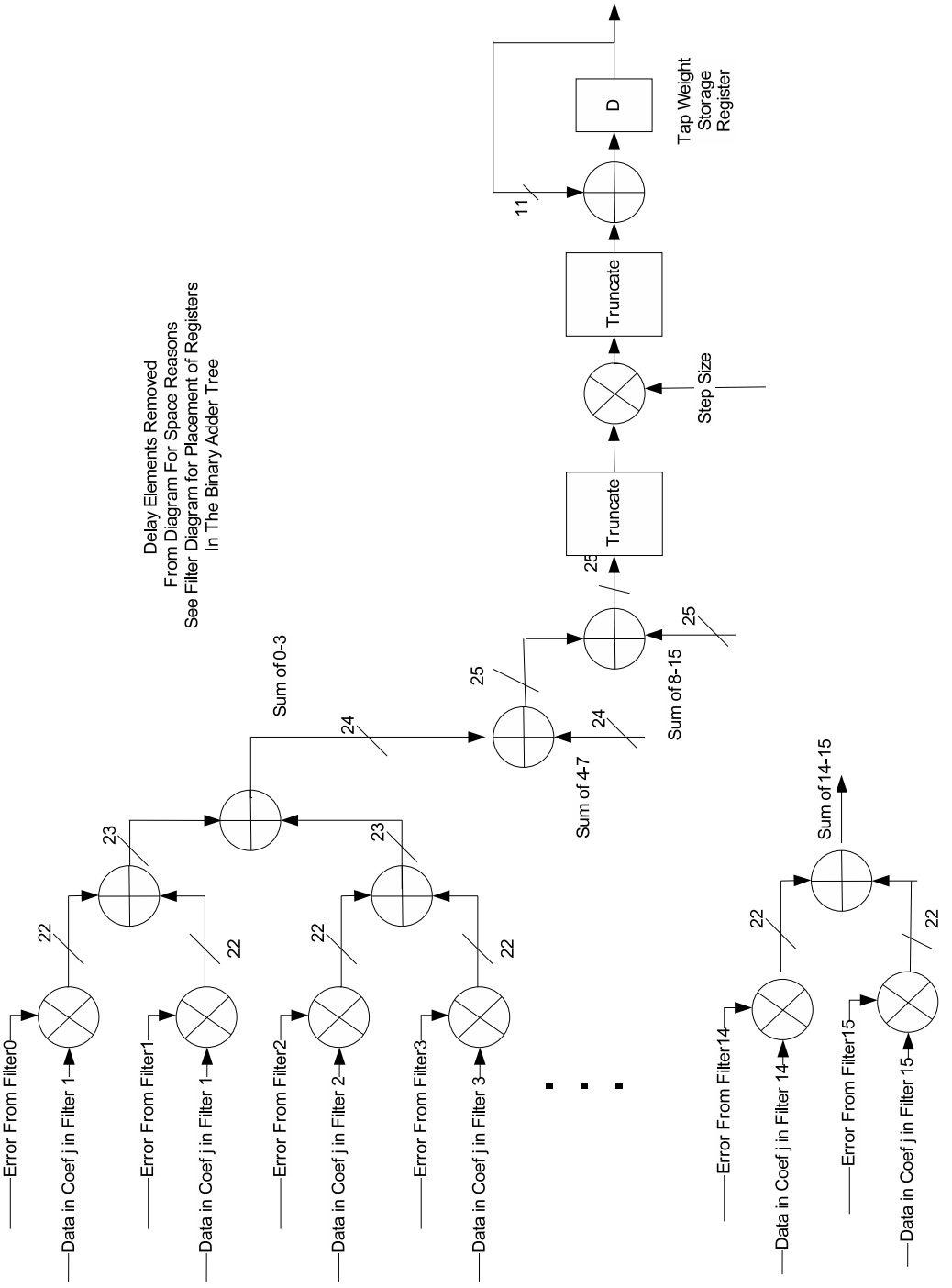


**Figure 23. This diagram shows the original design of the weight update calculation circuit. The circuit is a 16 tap filter, one tap for each of the FSE filters. Each tap coefficient represents the error of the corresponding FSE filter output. The 25 bit result of the FIR filter is truncated to 11 bits so that the update value can be scaled by the step size. Once the weighted error is calculated, it is added to the current tap weight for use in the next FSE calculation cycle.**

below one and the system cannot converge. The other explanation is that when the average tap update is less than one LSB, the tap update gets truncated to zero, and the system stops converging. Essentially, the tap weights start oscillating around the optimum value.

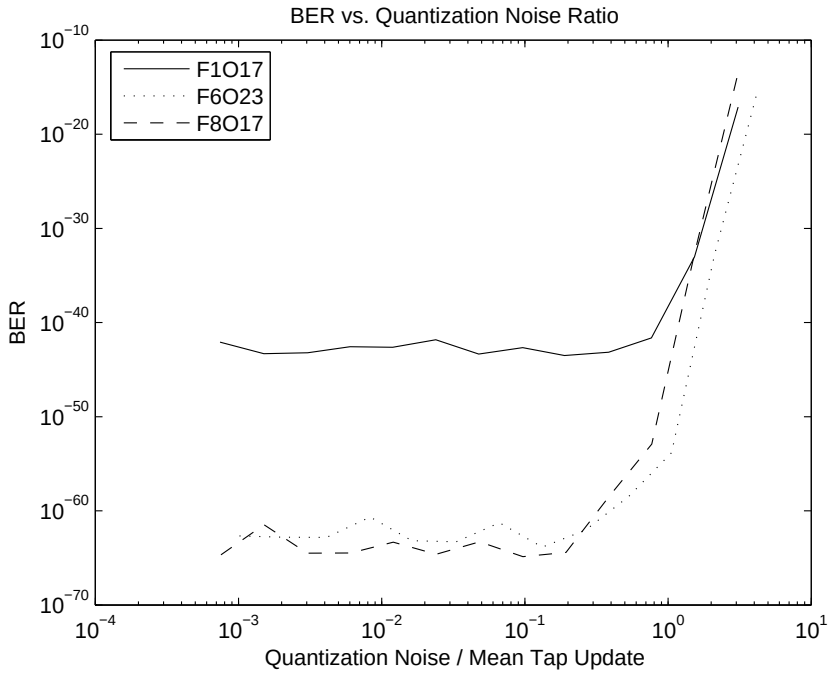


**Figure 24. A demonstration of how the circuit BER is affected by the ratio $\frac{mean(quantization\ noise)}{mean(coefficient\ update)}$. Three fibers demonstrate that when the quantization noise exceeds the average tap update value, the quantization noise becomes the noise floor in the system and sets the system performance.**

The analysis of the experimental results showed that when the ratio of the mean of the quantization error to the mean update of the center tap exceeded one, the BER was adversely affected. When $\frac{mean(Quantization\ Error)}{mean(Center\ Tap\ Update\ Value)} > 1$, the quantization error becomes the dominant noise source. Figure 24 shows how the LSB of the tap weight storage register affects the circuit BER for a selection of three fibers. The abscissa of the graph is the ratio between the mean of the quantization error and the mean of the coefficient update for the most significant tap. Table 4 shows the correspondence between the predicted mean [34] of the error and the measured mean error. To measure the mean error, the cycle accurate Matlab model calculated the tap weight update value twice: once as an infinite precision Matlab variable and once as a fixed point RTL calculation. The

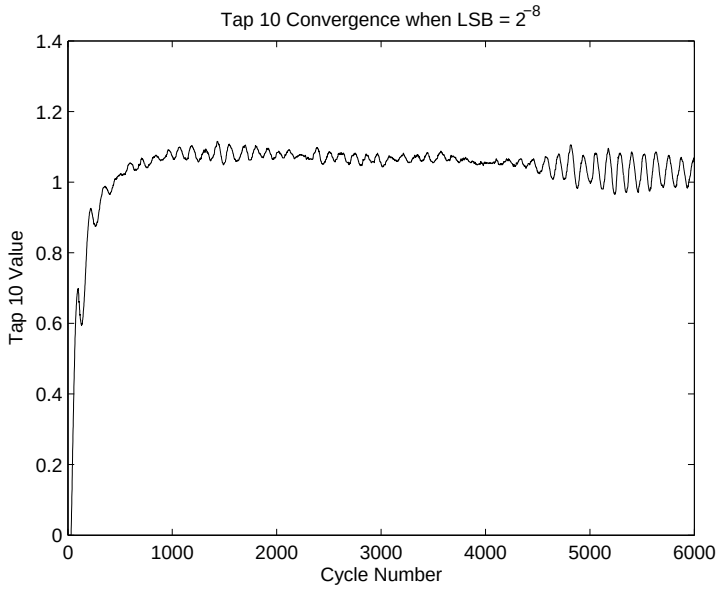


**Figure 25. The behavior over time of the center tap for fiber 10, offset 17, when the LSB of the tap storage register is $2^{-8}$. Rather than converging to a steady-state behavior, the tap weight begins to oscillate.**

difference between the two calculations was stored and the mean value calculated at the end of the simulation.

**Table 4. This table demonstrates that the error caused by truncating the error calculation closely matches the predicted values.**

| Fiber | LSB | Predicted Mean Quant. Error | Measured Mean Quant. Error |
|---|---|---|---|
| F1O17 | 11 | -0.00024467 | -0.00024414 |
| F1O17 | 14 | -3.0518e-005 | -3.0266e-005 |
| F6O23 | 11 | -0.00024467 | -0.00024545 |
| F6O23 | 14 | -3.0518e-005 | -3.0836e-005 |
| F8O17 | 11 | -0.00024467 | -0.00024541 |
| F8O17 | 14 | -3.0518e-005 | -3.0722e-005 |

Figures 25 and 26 show how the center tap for F1O17 converges with time for the cases where the LSB is $2^{-8}$ and $2^{-14}$. In the $2^{-8}$ case, tap oscillation is observed. In Figures 27 and 28, the tap update value for the center tap is plotted against time. In the LSB = $2^{-8}$ graph, over 50% of the tap updates over time are exactly zero. In the case where the LSB is $2^{-14}$, only 2% of the coefficient tap updates over time were zero.

Changing the coefficient storage from 11 bits to 25 bits allows the full precision of the

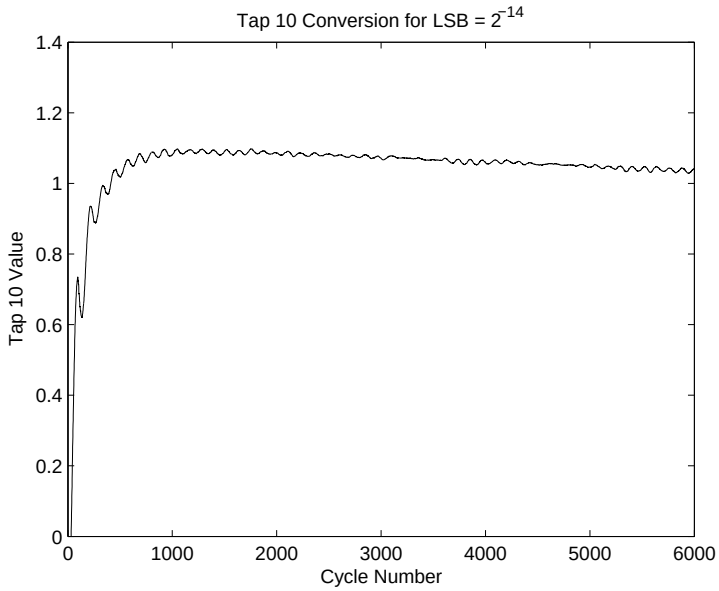


**Figure 26. The behavior over time of the center tap for fiber 10, offset 17, when the LSB of the tap storage register is $2^{-14}$. The tap weight is approaching its steady-state value, without oscillation.**

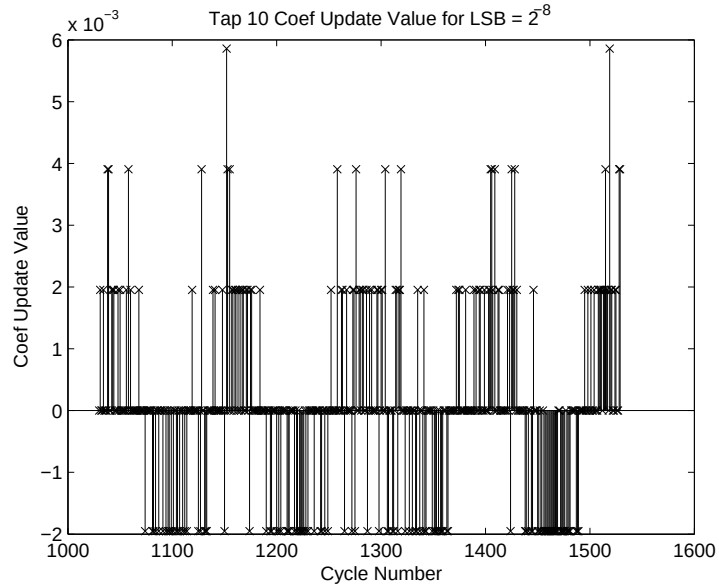


**Figure 27. The tap weight update values over time of the center tap for fiber 10, offset 17, when the LSB of the tap storage register is $2^{-8}$. Over 50 % of the tap updates in this case are zero, indicating that the calculated error was too small to be represented and was truncated to zero.**

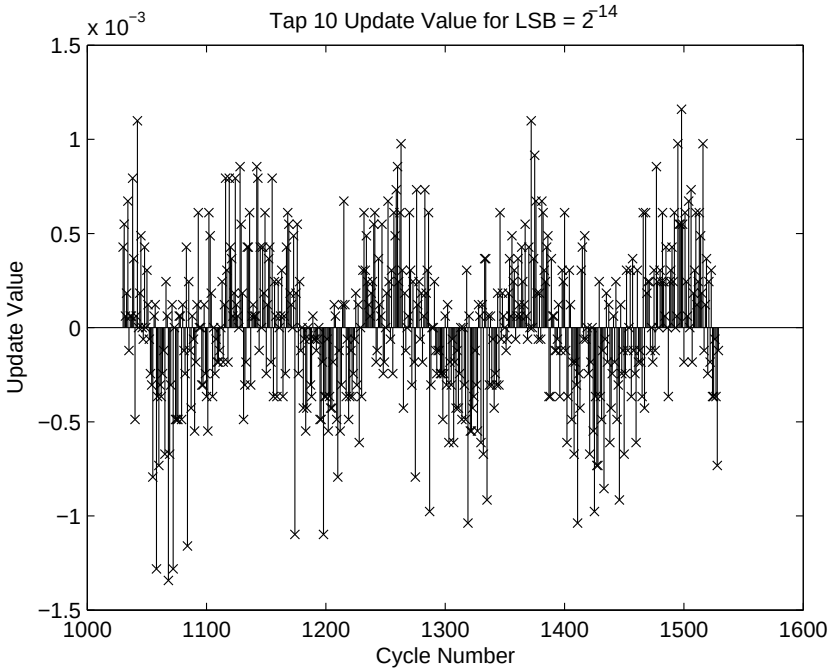


**Figure 28. The tap weight update values over time of the center tap for fiber 10, offset 17, when the LSB of the tap storage register is $2^{-14}$. Only 2% of the tap updates in this case are zero, indicating that the calculated error was too small to be represented and was truncated to zero.**

weight update circuit to be used but, then, introduces a timing closure problem. In the original design, the results of the weight update circuit were truncated to 11 bits so that the truncated result could be multiplied by an 11-bit step size before being added to the current 11 bit tap weight. (See Figure 20.) If the full 25 bit result of the weight update circuit is to be added to the existing tap weight, the step size multiply will be a 25 bit number multiplied by an 11 bit number, which results in a 36 bit number. Truncating the 36 bit result to 25 bits does not affect the convergence of the filter. The problem is closing timing. Previously (Section 3.4.1 on page 53), it was reported that the largest multiply that could be performed at 625 MHz was two 11-bit operands.

There are several potential ways to solve the timing problem caused by expanding one of the inputs to the multiplier. The first method examined was to pipeline the multiply operation and trade latency for speed. This solution is conceptually simple, but considerably more difficult to implement:

1. The synthesis library does not allow asymmetric inputs into the multiplier. Therefore, both inputs had to be changed to 25 bits, and the 11 bit step size was sign extended to 25 bits.

2. The output vector expanded to 50 bits wide, and the multiplier grew to four times the size of the original 11 bit multiplier.

3. When the 25-bit x 25-bit multiplier was instantiated with the deepest pipeline available in the synthesis library, timing was not passed at a clock rate of 625 MHz.

In order to implement a full 25-bit multiplier at 625 MHz, a different multiplier core must be used. The synthesis library used did not have a large selection of pipelined multiplier cores, although a different library might have an alternative architecture that would pass timing. The other solution would be to hand code a pipelined multiplier core using VHDL. This solution was rejected in favor of a more elegant solution.

Originally, the adaptive equalizer was conceptualized as having a variable step size that could be modified from the external general purpose processor. However, when the adaptive equalizer design was simulated in Matlab, the step size was never modified from the default of 1E-3. Further investigation showed that this step size was sufficient for all fibers and that, for those fibers that did not converge, changing the step size did not substantially change the performance.

Therefore, the implemented solution for the step size multiplication problem was to modify the design to use a fixed step size of 0.0009765, which is $2^{-10}$ and is very close to 0.001. Because the multiplicand is a power of two, the multiply operation can be replaced with a binary digit shift. In this case, by shifting the binary point of the weight update error calculation by ten bits to the right, the multiplication operation can be replaced. This modification results in a substantial area and power savings. If later a step size other than a power-of-two is required, the first attempted solution should be to approximate the step size as the sum of a small number of powers-of-two and to use single cycle 25-bit adders

to add shifted versions of the weight update calculation.

The errors that caused the redesign of the circuit demonstrated that greater precision was required in the tap storage register. Therefore, the bit definition of the storage register was modified from what is used in other places of the circuit. In the 25 bit output of the weight update calculation circuit, the binary point is between bits 14 and 13. To multiply the result by $2^{-10}$, the data values could be shifted ten bits to the right, but sign extending the data value and shifting the binary point ten spaces to the left is much more area efficient. By doing so, the binary point is now between bits 24 and 23, with bit 24 representing $2^0$. During all the simulation runs, the maximum magnitude of any coefficient was less than two. In order to ensure the circuit has enough operating margin, the maximum coefficient storage value is set to plus 3, which allows a margin of almost 50% and places the sign bit of the new vector at bit 26. The weight update calculation result is sign extended from bit 24 to 26 and bits 26 down to 2 are relabeled as 24 down to 0. This procedure truncates the least significant two bits from the weight update calculation and assigns the LSB of the storage register to be $2^{-21}$.

Figure 29 shows the implemented version of the weight update circuit. The stored coefficient value is truncated to 11 bits before the multiply operation in the FSE with no adverse affects on the system performance.

### 3.4.6 Implementation Figures of merit

After the code was implemented it was synthesized and passed timing at a rate of 669 MHz, which is a 7% margin over the desired rate of 625 MHz. This margin compensates for any routing path estimation errors that the first pass synthesis engine makes and is considered standard design practice in industry when using this synthesis methodology.

The power consumption, while high, can be reduced by taking several straightforward steps. This estimate assumes that all of the circuit is functioning all of the time. This assumption is not necessary as, once the taps have converged, the error calculation and weight update circuit could be turned off or run infrequently. Cycling this circuitry on a

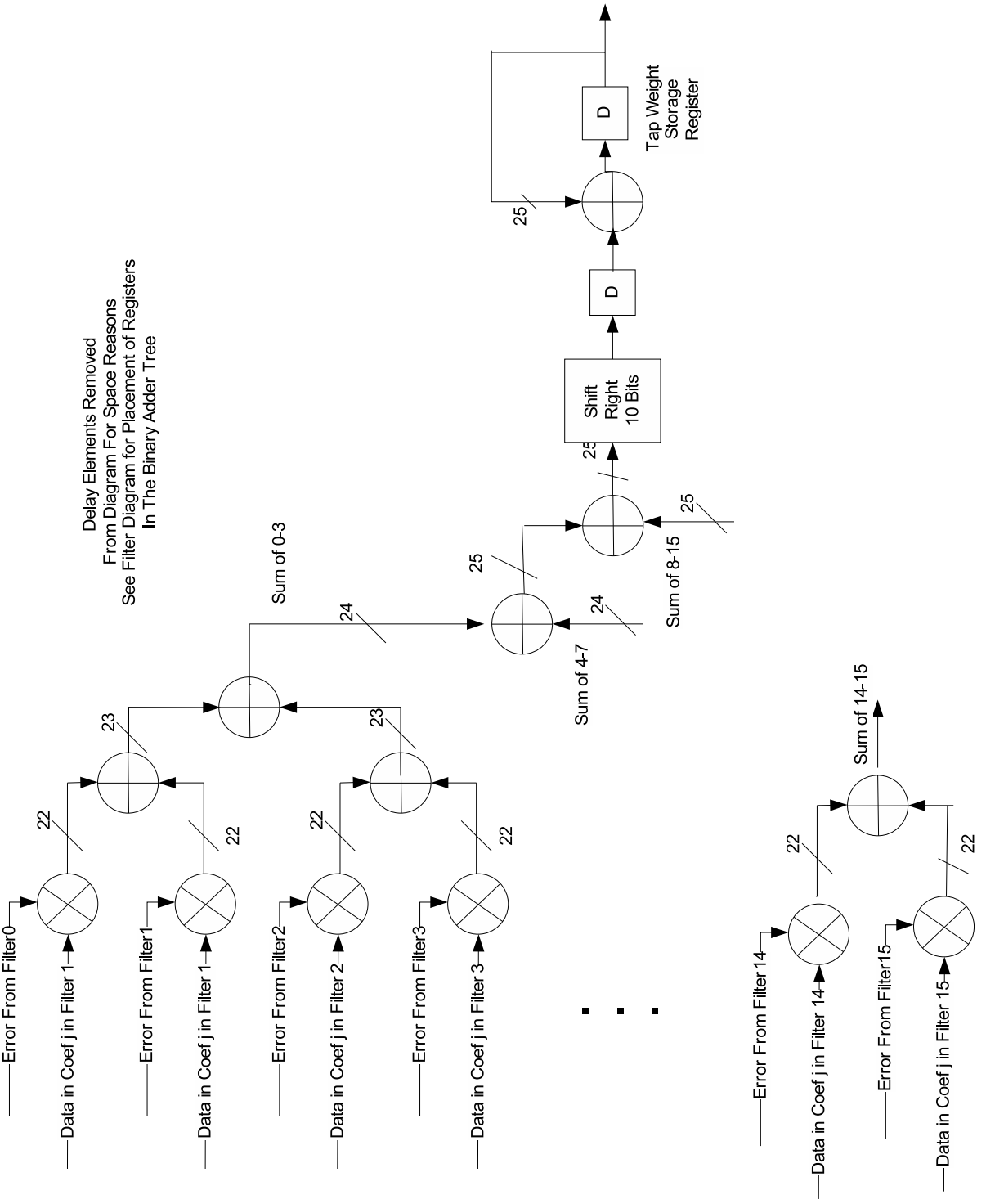


**Figure 29. The data flow diagram of the weight update calculation filter circuit as implemented. The truncation of the error so that it could be multiplied by the step-size has been replaced with a binary shift to the right by ten binary digits. This binary shift is almost equivalent to the multiplication by a step size of 1E-3 and allows the full 25-bit precision of the output to be preserved. The tap storage register has also been modified to store a 25-bit value. The tap value is truncated to 11 bits before being used in the FSE multiplication.**

**Table 5. Linear equalizer implementation figures of merit.**

| Parameter | Value | Comment |
|---|---|---|
| Synthesized Clock Speed | 669 MHz | 7% routing margin above 625 MHz |
| Number of Multipliers | 656 | 11 bit full fixed point multipliers |
| Number of Adders | 640 | 25 bit full fixed point adders |
| Number of 90nm instances | 582E3 | instance = 90nm cell |
| Power Estimate | 16.5 W | Worst case switching frequency |

low duty cycle would cut the power consumption almost in half. This approach, while feasible, would still be far from competitive with an analog implementation.

## 3.5 Linear equalizer results

All 324 channel models were simulated with a test length of $100,000$ random bits. The default cursor position was set to zero. When the results were tabulated, 265 of the 324 fibers converged with a bit error rate of less than $1E-12$. Investigation of the non-converging channels found several fibers that did not have their largest tap weight near the center of the filter once the filter had converged.

Those fibers that failed to converge were tested with a reference shifting program that walked the cursor over a range of negative ten to positive ten, and an additional twenty-three fibers converged under a different cursor alignment. This cursor position search would normally be controlled by an external processor running a software search algorithm. Hardware supports this search by providing the ability to shift the reference tap on software command. Fiber F34O17 is one of the fibers that converged once a shift had been applied. Figure 30 shows how shifting the main tap to the center position affects the performance of the filter.

Figures 31 and 32 show the difference in the input to the slicer and the resulting BER when the cursor location is shifted in the RTL. The Figures show the input to the quantizer over time. The abscissa axis represents the sample number and the ordinate axis is the value produced by the forward filter to the slicer. The graphs shows how the system converges around the [+1,-1] decision points. The standard deviation and mean of the data points

around each tail set the bit error rate. Figure 30 demonstrates that shifting the reference value by three sample times improves the BER from 1E-9 to 1E-20.

Once the fibers that required a cursor shift to converge are accounted for, the overall performance of the linear equalizer implementation of the BDLMS algorithm can be examined. A plot of the results sorted by BER are shown in Figure 33. Overall, the linear equalizer can equalize 88.1% of the fiber channels.

### 3.5.1 Fibers that do not converge with the LE

That there would be some fibers that would not converge with the linear equalizer was expected. After all, the DFE is considered a required solution in this type of application. Using a Matlab model of a one-tap DFE, fiber 87, offset 20 was examined. The best result that can be obtained with fiber 87, offset 20, is $3E-11$. The impulse response (Figure 34)

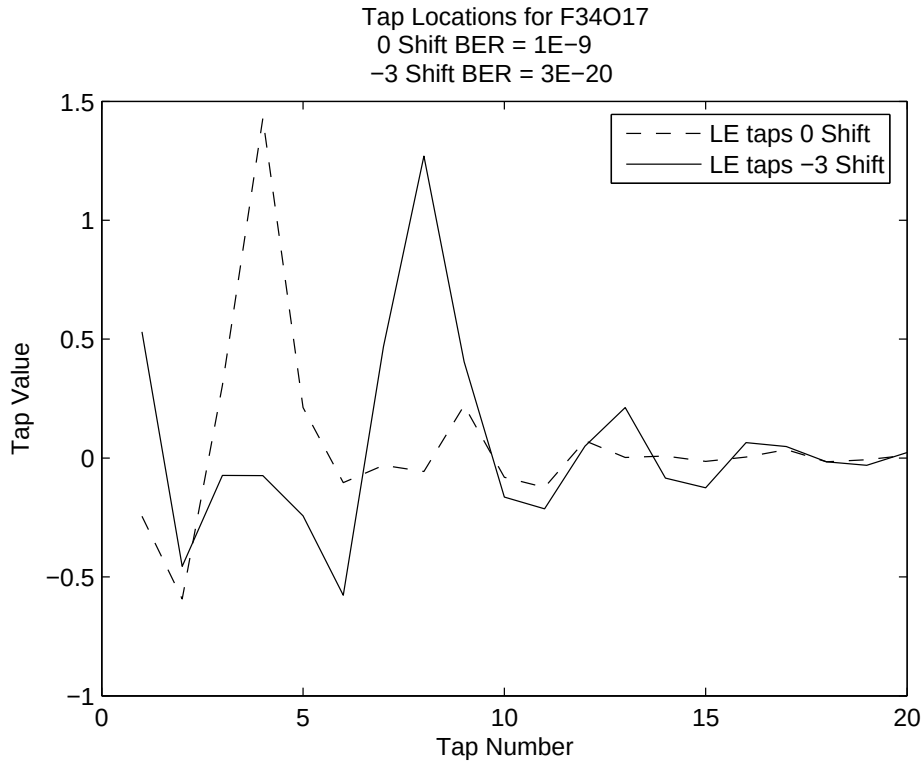


**Figure 30. A demonstration of how performance is improved when the main lobe of the filter is near the center tap. The Figure shows the tap values for fiber 34, offset 17, with a shift of zero and negative 3. In the unshifted, default alignment, the exponent of the BER is half that of the shifted alignment.**

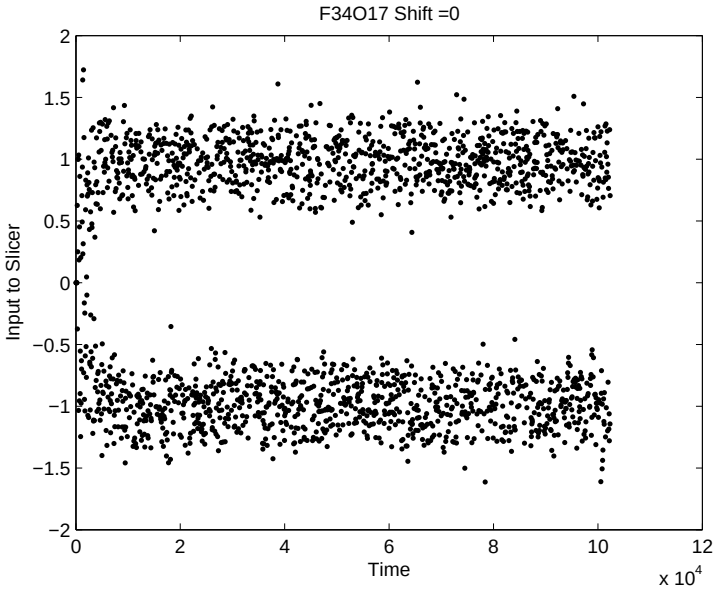


**Figure 31. Convergence of fiber 34, offset 17, with cursor shift =0. With shift =0, the BER of the circuit is** $1E-9$**.**

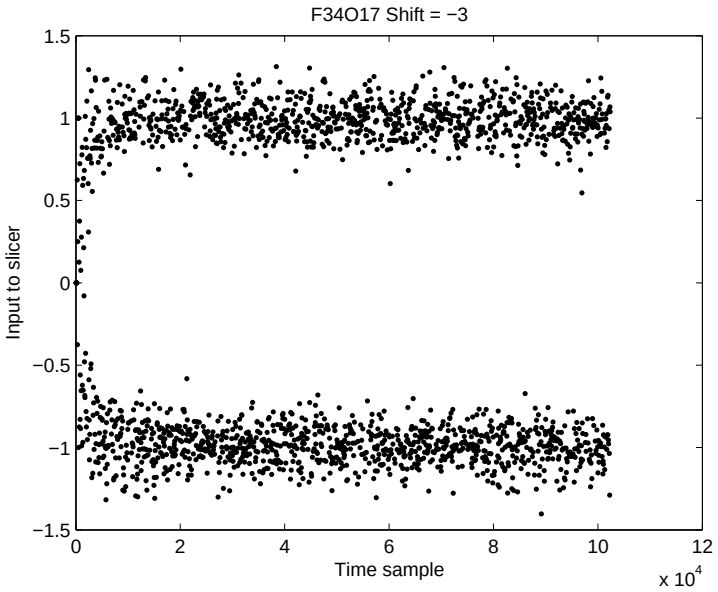


**Figure 32. Convergence of fiber 34, offset 17, with cursor shift = -3. With shift = -3, the BER of the circuit is** $1E-20$**. Notice that the legs of the graph taper significantly more than the shift =0 graph.**

is almost a delta function. The impulse response graph is calculated in time steps of 160 GHz, so the taps are very closely spaced.

In Figure 35, the post convergence linear equalizer taps are plotted. The spacing between these taps is 20 GHz. By comparing the taps with the impulse response, it may be observed that the implemented linear equalizer taps are not spaced close enough to limit the peak to the center position and the side-band taps are forced into alternating positive and negative values by trying to approximate the energy in the delta response. In other words, the impulse response falls off too quickly and cannot be approximated by a linear equalizer with taps spaced as far apart as this implementation. The best BER the LE can accomplish with this channel is 3E-11, but with a single tap DFE, the BER improves to 3E-20.

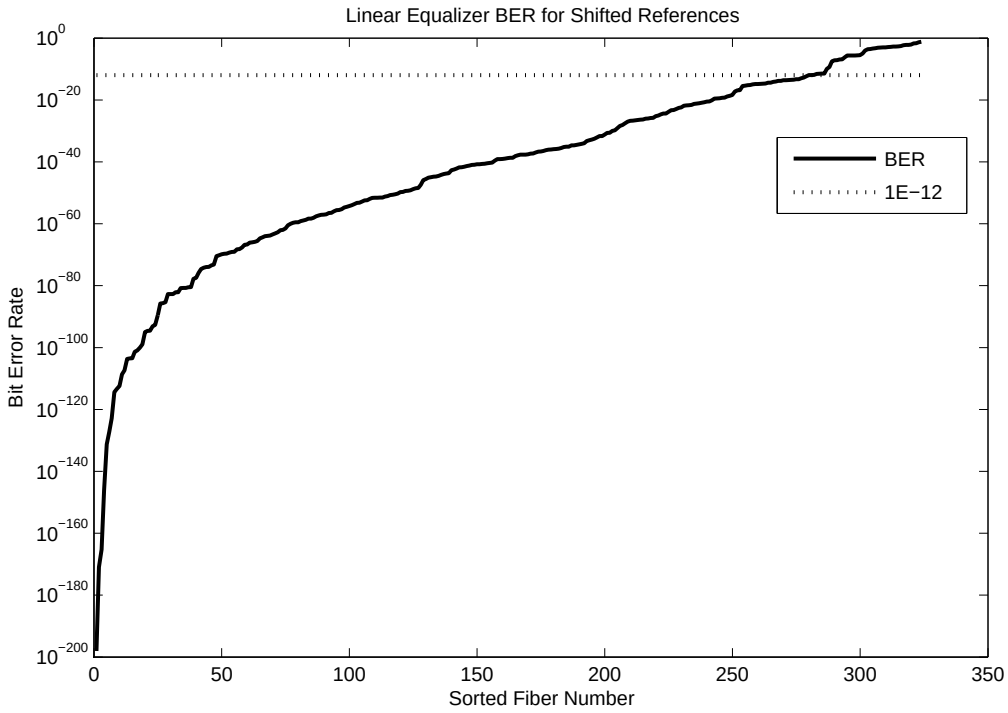


**Figure 33. The 20 tap, T/2 linear equalizer results. 88.1% of the test fibers were equalized by the linear equalizer.**

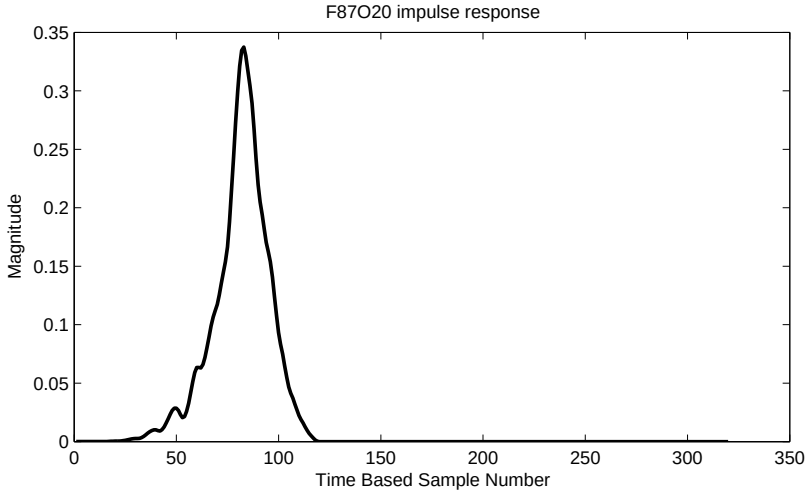


**Figure 34. The impulse response of fiber 87, offset 20, which was not able to be equalized by the linear equalizer implementation. Notice that there are over 300 samples in the impulse response, and that the samples were obtained by sampling at 160 GHz.**

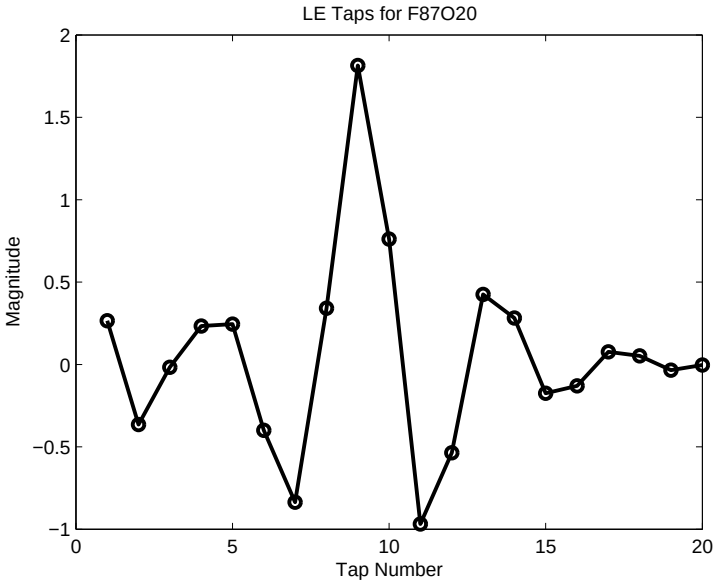


**Figure 35. The post-simulation tap weights for fiber 87, offset 20. The spacing between these taps is 20 GHz. Notice that the tap spacing is too far apart and the filter cannot approximate the very narrow temporal response of the fiber.**

## 3.6 Analysis of LE results, comparison with IEEE 802.3aq committee

In the analysis performed by the IEEE 802.3aq standards body [1], a pure FSE was determined to require over 30 taps in order to equalize 95% of the fibers and still imposed a larger optical power penalty than what was budgeted.

This research has demonstrated a method to equalize 88% of the subject fibers with only 20 FSE taps. The implementation has been shown to pass traffic with a minimum BER of 1E-12, even when the additive noise figures used to generate the data are derived from the worst case noise figures measured from an existing PHY.

# CHAPTER 4

# DECISION FEEDBACK EQUALIZER RESULTS

## 4.1 Introduction

The ideal solution to an inter-symbol interference that can be modeled as an FIR system is a decision feedback equalizer. As shown in Section 3.5.1 on page 74, there are channels in the data set that would benefit from having a DFE architecture. However, the DFE adds a feedback loop to the system. This feedback loop is the primary implementation impediment of the high-speed DFE circuit. As discussed in section 1.3.2 starting on page 13, the removal of the feedback loop has been the focus of many research papers. Figure 36 shows an example of an implementation of a serial DFE.

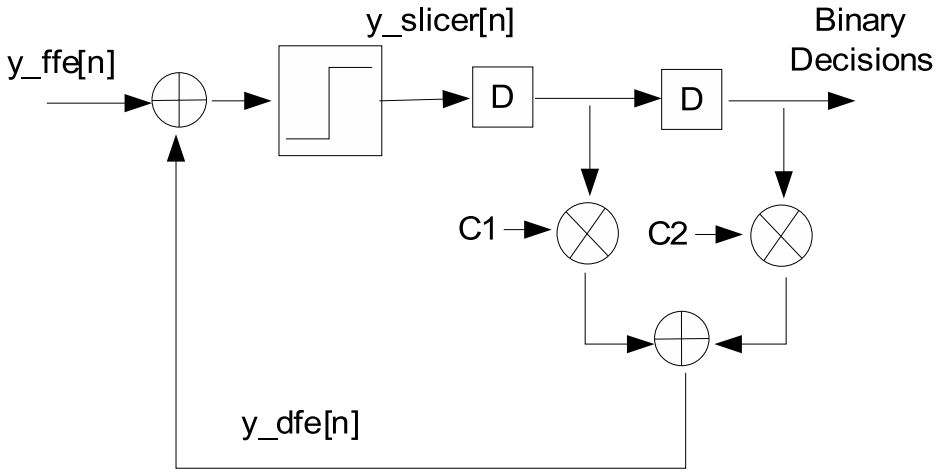


**Figure 36. The feedback portion of a DFE filter. The two-tap DFE filter is implemented in a "standard" serial format.**

The feedback loop begins at the output of the $2^{nd}$ delay element and proceeds through the DFE filter, the addition of the FSE output, the slicer, and back to the first delay element. Unlike the FSE, where the $n^{th}$ output does not affect the value of the $(n + 1)^{st}$ output, the previous DFE outputs must be resolved to calculate the current DFE output. To process

data at line rate, the feedback loop must execute at 10 GHz.

Although the standard cell process used is very fast and can handle large cones of logic at high clock rates, this process is not fast enough for the outputs of a full adder to resolve in a single clock period, much less the path through the feedback filter. In order to implement a DFE in a digital process, the feedback loop must either be removed or modified to operate in our process at the symbol rate.

## 4.2 Addition of a DFE section to the BDLMS algorithm

In addition to the feedback loop, the DFE solution must be a scalable addition to the current linear equalizer architecture. Designing a DFE solution that prevents the forward filter from operating at speed is not a workable solution. Therefore, the anticipated system diagram can be drawn by taking into account these restrictions.

Extending the parallel linear equalizer architecture (Figure 19, page 55) to add a DFE core results in the DFE system implementation shown in Figure 37. The same timing requirements that were present in the linear equalizer design must still be satisfied for the DFE design. Therefore, the system design must allow 16 outputs to be calculated in a single clock cycle.

Additionally, the weight update circuit must calculate the DFE update equation in addition to the FSE taps. Once these constraints on the system are understood, potential solutions can be evaluated.

### 4.2.1 Unrolling the DFE feedback loop

Parhi's [21] CDMA loop unrolling paper provides a starting point for the methodology of unrolling a digital feedback loop. Substituting the actual values (±1) for the older slicer output allows a further simplification to be made as shown in (15).

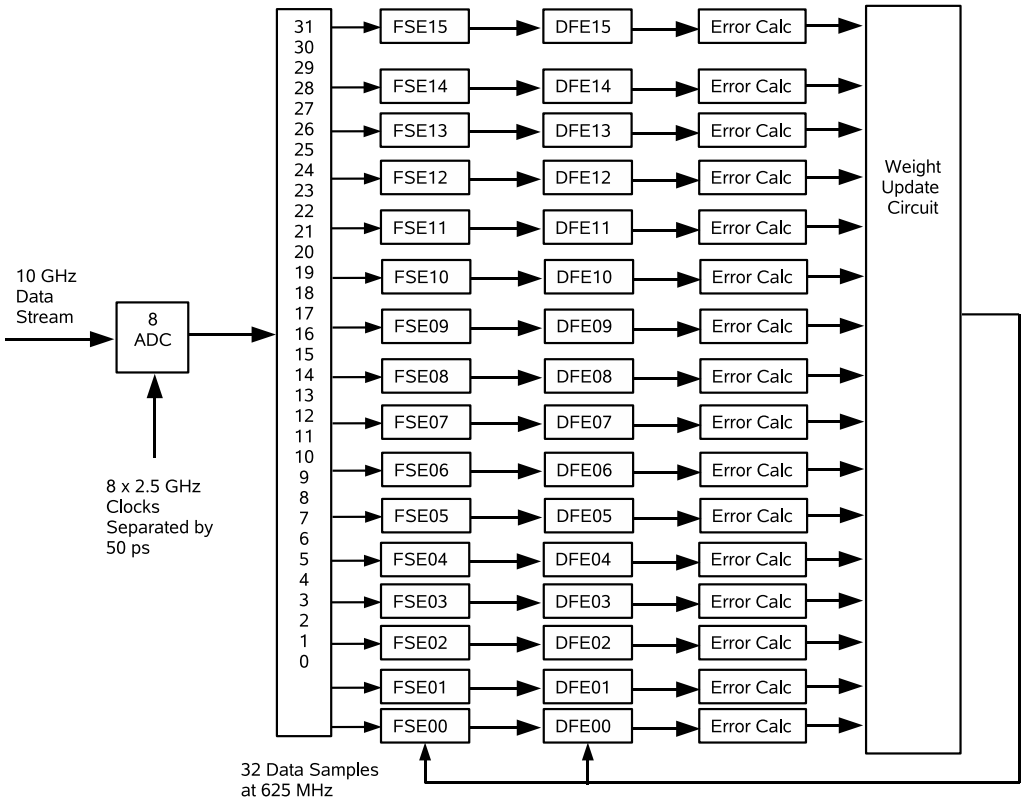


**Figure 37. The anticipated data flow block diagram for the Block Delayed LMS algorithm with DFE filters attached. This diagram takes into account the constraints imposed by the linear equalizer and DFE solution spaces.**

$$
\begin{aligned}
y\_slicer[n] &= Q\{FeedBackFilter[n] + y\_fffe[n]\} \\
FBF[n] &= \sum_{i=1}^{NumTaps} Coef_i[n] * y\_slicer[n-i] \\
FBF[n] &= C_1[n] * y\_slicer[n-1] + C_2[n] * y\_slicer[n-2] \\
&\text{but} \quad y\_slicer[n-i] \in \pm 1 \\
&\text{so} \quad FBF[n] = \pm C_1 \pm C_2 \\
y\_slicer[n] &= \begin{cases}
Q(y\_fffe[n] + C_1[n] + C_2[n]) & (y\_slicer[n-1:n-2] == [00]) \\
Q(y\_fffe[n] + C_1[n] - C_2[n]) & (y\_slicer[n-1:n-2] == [01]) \\
Q(y\_fffe[n] - C_1[n] + C_2[n]) & (y\_slicer[n-1:n-2] == [10]) \\
Q(y\_fffe[n] - C_1[n] - C_2[n]) & (y\_slicer[n-1:n-2] == [11])
\end{cases}
\end{aligned}
\tag{15}
$$

Once the FBF output is shown to be an additive combination of the filter coefficients, all possible combinations of the coefficients can be calculated in advance as long as the number of taps are not large enough that area begins to become a concern. The implementation of Equation 15 is illustrated in Figure 38.

The primary benefit of this formulation is that the FBF can be pipelined between the output of the FFF and the output of the quantizer as shown in Figure 38. The math in the DFE can be pipelined as long as the number of cycles added are not egregious. Because of the small step size ($2^{-10}$), the maximum update is $Step\ Size\ *\ Average\ (Data\ *\ Error)$. Assuming a worse case average value for the data of $\pm 1$ and a worst case error condition of $\pm 2$, the difference between any two updates is $2^{-9}$. Over a delay of $N$ cycles, the worst case error in the coefficients is $N * 2^{-9}$ or roughly $N\ *\ 2E-3$. Simulations showed that this worst case update does occasionally occur, as the largest update between iterations of the design was recorded to be $2E-3$. When the largest tap weight update for each converged fiber was recorded, the mean maximum tap update was found to be on the order of $5E-4$. This amount of error is on the same order of magnitude as the quantization error introduced if the LSB of the tap storage register represents $2^{-11}$. The difference between the quantization

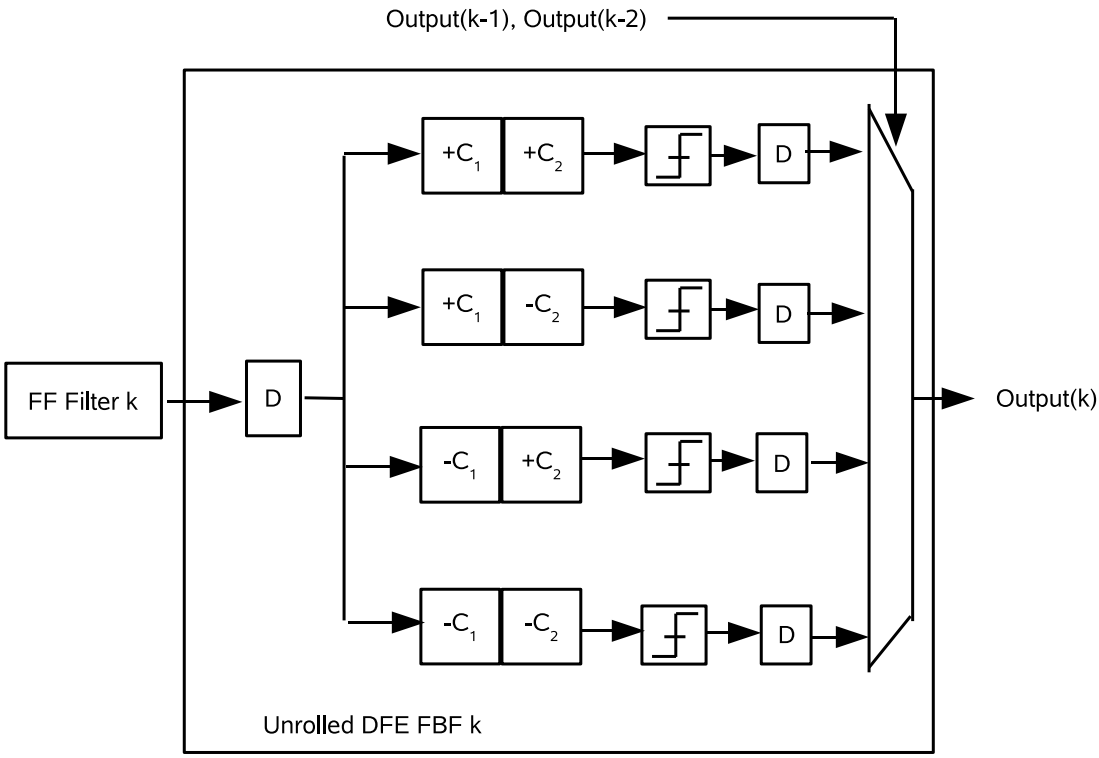


**Figure 38. The data flow diagram for the unrolled DFE core. All possible combinations of the FSE filter output and DFE coefficients are pre-calculated. The outputs from prior filters are used to select the correct output of the current filter.**

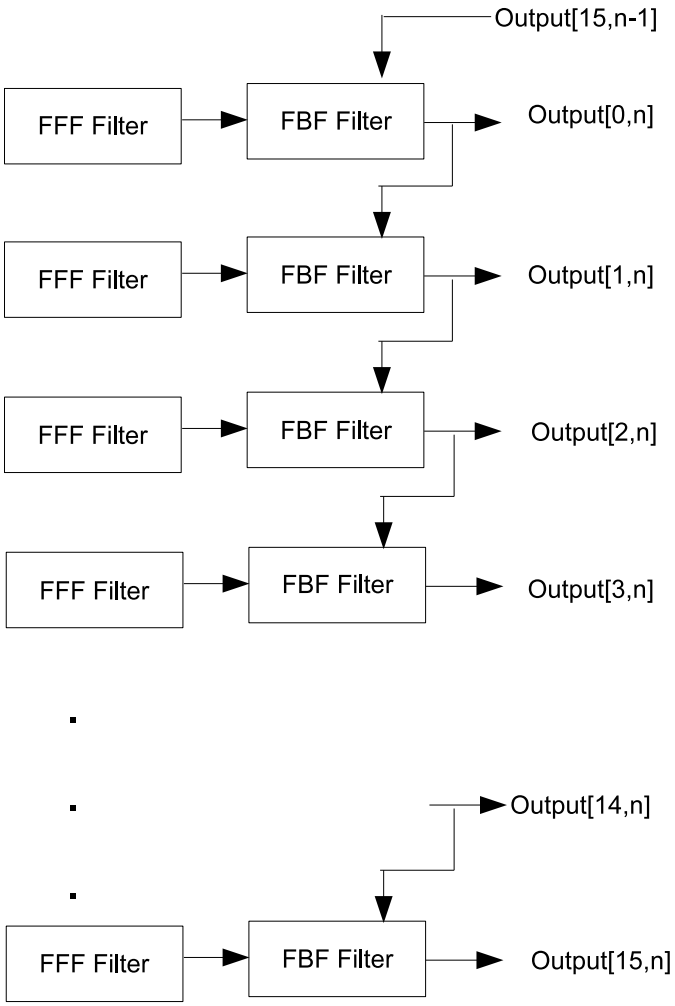


**Figure 39. An example of the critical path in a single tap DFE circuit. The DFE critical path is the series of 16 muxs connected in series. Output zero is the oldest value, and is calculated first. Every other output from one to fifteen is calculated in numerical order. All 16 outputs must be resolved within a 625 MHz clock cycle.**

induced noise and this delayed source is that quantization induced noise is permanently lost. The delayed coefficients eventually receive their update. As shown in the original DLMS papers [4], if the output decision is based on an out of date set of coefficients, there is little impact to the overall convergence characteristics, as long as the updates occur faster that the channel impulse response drifts.

For a two tap DFE implementation, the critical path has been reduced from sixteen, two tap FIR filters, to sixteen, four-input muxes. Figure 39 demonstrates the critical path through the feedback filter blocks. Figure 39 shows that output 15 depends on the resolution

of output 14, which depends on output 13, etc. For simplicity, the figure is drawn showing the critical path for a single tap DFE system. In a two-tap DFE, output 2 would depend on outputs 1 and 0.

## 4.3 DFE core implementation

Once the DFE core is unrolled, the DFE cell can be inserted between the FSE filter and the error calculation block. Figure 40 shows how the paths inside the DFE core contribute to the critical path. The diagram shows that the unrolled algorithm's critical path starts with the storage register for the "sliced" variable, propagates through 16, four-input muxes, and ends back at the storage register. Although the DFE blocks are attached to parallel forward filters, the DFE blocks are connected together in series.

Previous diagrams have been oriented with output 15 at the top of the diagram. Output zero is the oldest output, and must be calculated first, with outputs one through fifteen following in order.

### 4.3.1 DFE core synthesis results

Once the system block diagram was finalized, both Matlab and RTL models were developed and iterated until a reasonable implementation was defined and analysis demonstrated improved results when compared with the linear equalizer. Once the Matlab and RTL simulations produced equivalent results, the final step in the process was to determine what logic/algorithm changes were necessary in order to close timing.

The full DFE module, consisting of 16 two-tap DFE cores was test synthesized, and failed timing by approximately 1.3 ns. A second test synthesis was performed using 16 one-tap DFE modules, and the circuit failed again, this time by 257 ps. As a final attempt to pass timing, the critical timing path of the DFE core was implemented as a stand-alone test circuit (Figure 41) and synthesized. The test circuit consisted of two, one bit inputs that are registered inside the block, and a one bit mux.

When synthesized, the block contains just the critical timing path of the full DFE core,

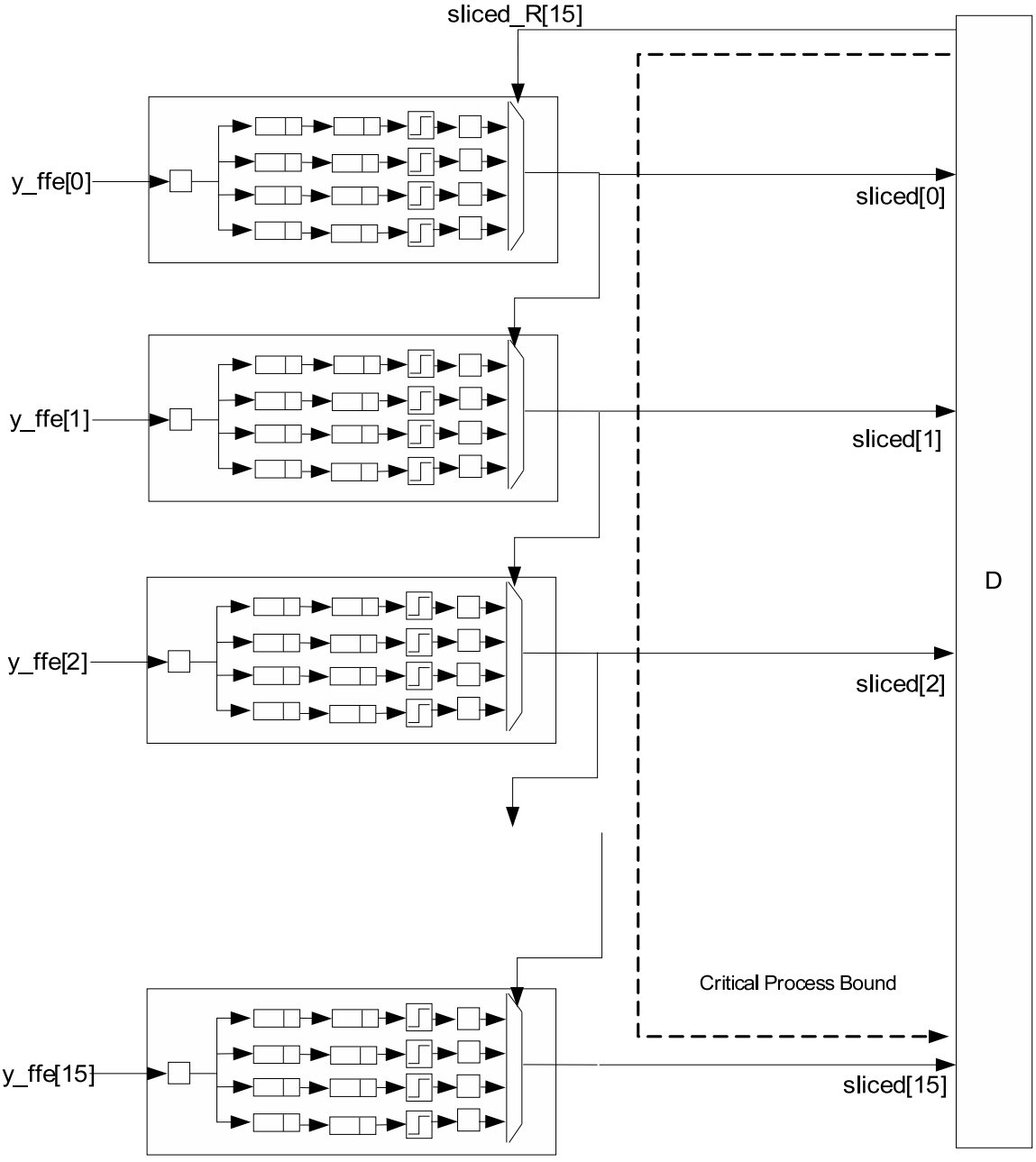


**Figure 40. A block diagram of the DFE critical path showing the components inside the DFE core that contribute to the critical path.**

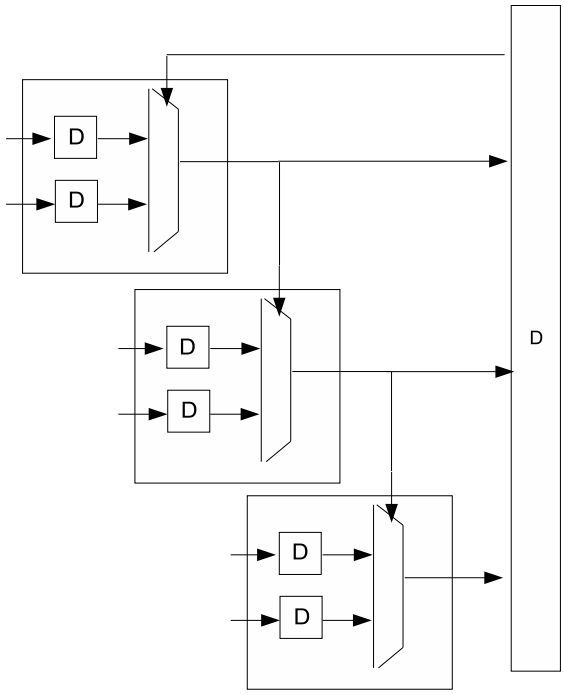


**Figure 41. Three instances of the DFE test core and how the output from the mux is routed to the selection port of the next mux in the critical path. The test core for each FBF contains only two, one-bit registers and a one bit mux. The resulting synthesized block is much more narrow than the full DFE core. As a result, when all 16 cores are instanced, the critical path through the muxes passes timing.**

the clock-to-output path of a register, and the delay path through a two-input mux. The size of the synthesized block is just large enough to contain the registers and the mux. The test core was then connected in the same fashion as the full DFE core, and synthesized. This test circuit of 16 serially connected DFE cores passed timing with a critical delay path of 1.56 ns, or a maximum execution speed of 640 MHz. This is a margin of only 2.56%, less than the 7% margin that is normal design practice.

The critical timing path in the test circuit and full DFE circuit were compared and found to consist of the same logic cone. The critical path consists of the chain of logic that starts with output number 15 from the previous calculation, runs through the select line and output port of 15 multiplexers, and ends at the input port of the output data register. The only differences between the test and full DFE circuit were in the amount of path delay between the register to mux and mux output to the selection port of the next mux in the chain. Figure 41 shows the layout of the test circuit that was synthesized. In this case, the

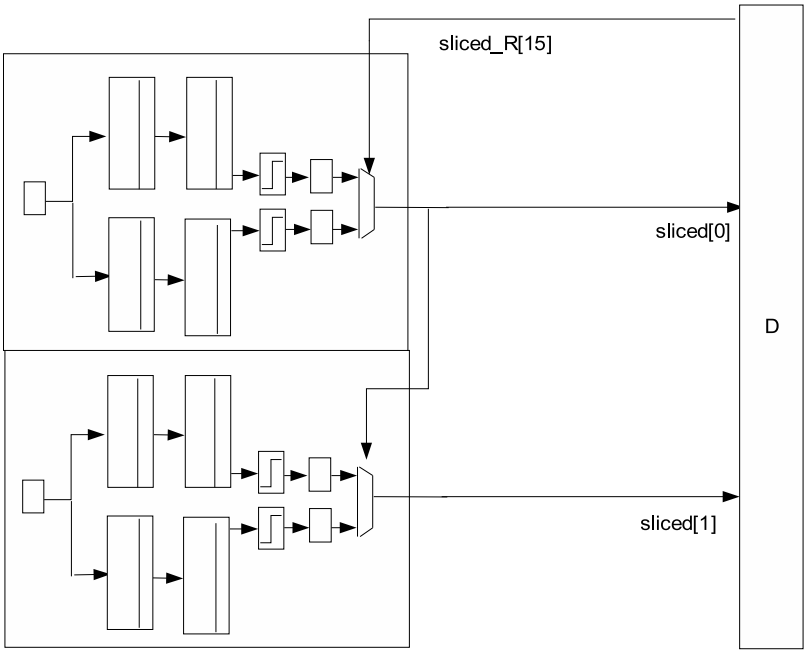


**Figure 42. The synthesized size of the full DFE core. Two instances of the full DFE core are shown connected together. The critical path in this case is much longer than the equivalent path in the test DFE core synthesis.**

outer blocks are drawn to the physical scale that they synthesized to. Observe that the outer boundary of the DFE core is close to the edge of the mux, which exists in the critical path.

Figure 42 shows the synthesized size of a small portion of the full 16 parallel one tap DFE cores. In this case, the adders inside the core have made the DFE core much taller than the test circuit. The boundary of the DFE core is very far away from the edge of the mux. Therefore, the DFE filter result signal must travel significantly further in the full implementation when compared to the test circuit.

The ideal solution would be to keep the logic inside the DFE cores the same, but stretch the containers so that the output of a mux is very close to the input of the next mux in line. Figure 43 demonstrates what the ideal synthesis result would look like for a one-tap DFE core.

There are several solutions to this problem, but unfortunately, all of them involve finding additional resources. The compute server that was available for the synthesis was an

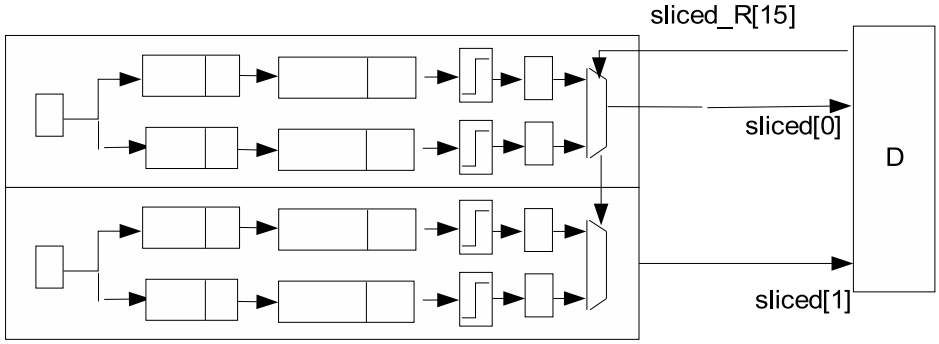


**Figure 43. The ideal aspect ratio for the synthesized full DFE core is shown. Notice that the height of the synthesized container is roughly that of the mux, and that the path between the two muxes is very short.**

older machine with only 4 GB of RAM[1]. There were computers with more RAM available for use, but those machines had an operating system conflict with the synthesis tool. Therefore, the only choice of machine was the older machine with limited amounts of RAM.

When the full parallel DFE core was synthesized with the 16 sub-cores in a top-down fashion, the synthesis machine ran out of memory. Next, the synthesis was performed in a bottom-up order. In this method, the DFE core was synthesized, routed, and saved as a macro. Next, the top-level synthesis was performed, instancing the previously saved DFE core macro.

The DFE core was labeled with a 'set_dont_touch' attribute to prevent it from being loaded into memory 16 times. When just the DFE core was synthesized, the timing constraints placed on the inputs did not result in the long/narrow implementation, instead the synthesis tool claimed that the circuit was un-achievable, and gave up. A cell boundary for the synthesis tool could be defined, but that functionality requires a software license that was not available for this research.

Another solution would be to individually synthesize the component parts of the DFE core and then try to instantiate them into a long/narrow block. This approach would take a

[1]When compared with the amount of RAM a "normal" PC contains, 4 GB of RAM would appear to be a very large amount. The average ASIC place and route platform today is a 64 bit server containing 16 to 32 GB of RAM.

very long time and require more skill with the synthesis tool that was not available.

A tool-based solution to this problem was searched for extensively, but eventually all the possible methods were rejected for one reason: lack of resources. Once this result became obvious, the only solution was to validate the test synthesis as a reasonable simulation of what could be accomplished with enough resources. A design review was held with several synthesis tool experts[2], and the experimental results discussed. The result of the design review was that the critical path synthesis test results of the single tap test core were sufficient to show that timing closure could be achieved on the mux output net if the routing of the net was performed with sufficient care and a realistic set of resources.

### 4.3.2 BDLMS architecture with DFE

Once the synthesis constraints for the DFE cores were found to preclude a DFE core larger than one tap, the complete system diagram for the DFE implementation of the BDLMS algorithm can be drawn (Figure 44). The addition of the DFE cores modifies the LMS system design slightly:

1. The data that is used in the error calculation step must be saved for the additional latency caused by the DFE calculation time, adding an additional 480 registers to the power and circuit area.

2. The weight update circuit must now calculate an additional tap update for the DFE tap, adding an additional 16-tap FIR filter, with the associated area and power costs.

## 4.4 Decision feedback equalizer circuit results

Using the one tap, pipelined DFE system design previously discussed, all 324 channel models were simulated with the default cursor position set at "0." Of the 324 channels, 287 converged with a bit error rate less than $1E-12$. Upon inspection there were several

[2]For help with this problem, I consulted with R. Suffridge and J. Mulrooney. R. Suffridge is an ASIC Architect with Intel and has successfully designed the architecture for over 20 commercial telecom ASICs for Nortel and Intel. J. Mulrooney is a synthesis tool lead with Intel and has over 15 years of ASIC synthesis experience, the last five years of which has been spent working with the library in question.

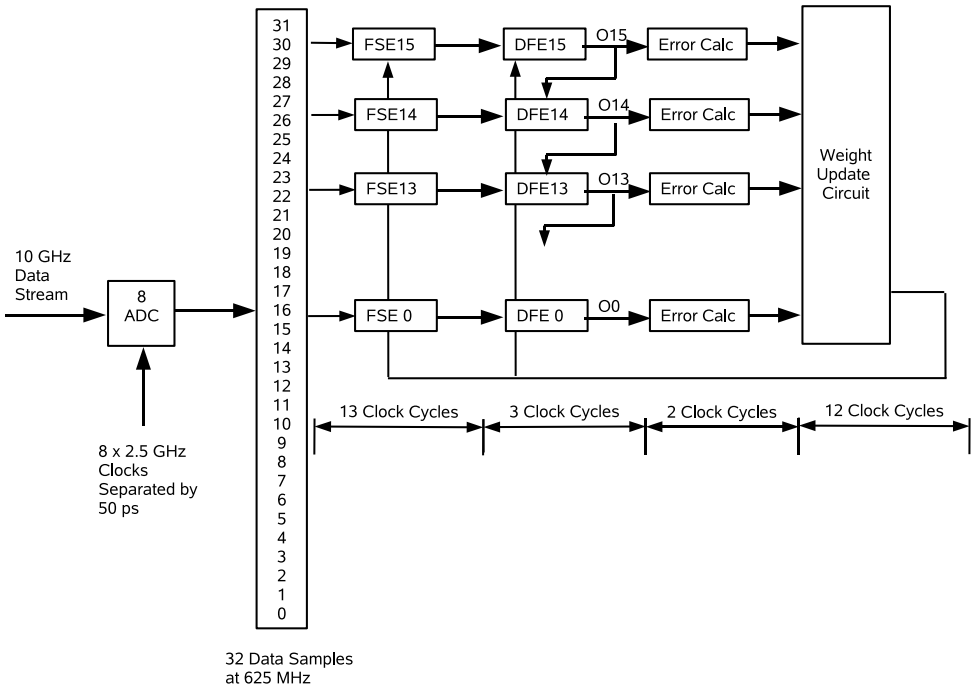


**Figure 44. The block delayed LMS algorithm with DFE core block diagram. The diagram shows how the DFEs are instanced in parallel along with the corresponding FSE circuit but at the same time, are connected in series between the DFE instances. The overall latency for the DFE BDLMS is thirty, 625 MHz clock cycles.**

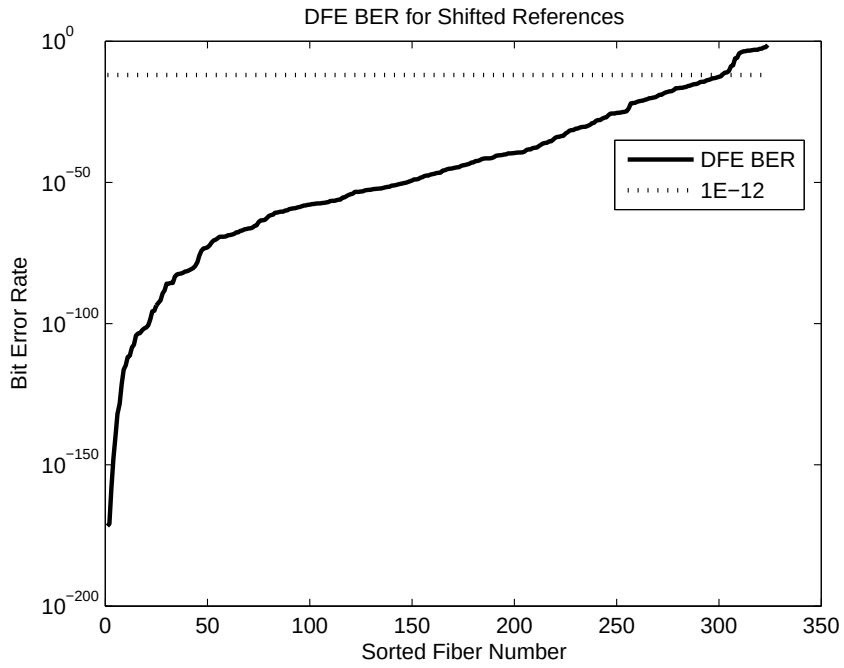


**Figure 45. The performance of the one-tap DFE, 20 tap T/2 FSE BDLMS circuit. The circuit is able to equalize 93.8% (or 304 out of 324) of the IEEE fiber set. Of the fibers that do not converge with a single DFE tap, 15 were found to converge with one to five additional DFE taps, with 11 converging when two DFE taps are used.**

fibers that were found to not have their main lobe near the center of the filter. Those fibers that failed to converge were tested with a Matlab program that shifted the cursor over a range of negative ten to positive ten. Any improvements were recorded and the equivalent RTL simulation performed. As a result, an additional 17 fibers converged with new alignments. After the cursor alignment experiment was complete, a total of 304 out of 324 fibers (93.8%) converged to a BER less than 1E-12 (Figure 45.)

### 4.4.1 Fibers that do not converge with 1 DFE tap.

While being able to equalize almost 94% of the fibers is a vast improvement from the linear equalizer performance, it still does not meet the IEEE standard's goal of 95% equalization. The number of DFE taps required to converge the remaining channels was investigated and is shown in Table 6.

The results indicate that adding a second DFE tap to the implementation will result in

**Table 6. For each fiber that was not equalized by the one-tap DFE, a test was performed to find the number of DFE taps required in order to reach the performance target. This table shows that five of the fibers could not be equalized with six taps. The largest gain is achieved by adding a second DFE tap.**

| | |
|---|---|
| Number of Fibers that equalize with 2 DFE Taps | 11 |
| Number of Fibers that equalize with 3 DFE Taps | 1 |
| Number of Fibers that equalize with 4 DFE taps | 1 |
| Number of Fibers that equalize with 5 DFE Taps | 1 |
| Number of Fibers that equalize with 6 DFE Taps | 1 |
| Number of Fibers that do not equalize with more than 6 DFE Taps | 5 |

a solution that meets the IEEE goals of equalizing 95% of the worst case, legacy fiber. Repeating this research with a 60 or 45 nm process would likely achieve the goal of implementing a two tap DFE system. Meeting the IEEE 802.3aq standard's suggested implementation of 20 FSE taps and 4 DFE taps is unlikely even with a faster process. However, the $2^{nd}$ DFE tap equalizes an additional 11 channels, bringing the total to 315, or 97.2% of the test set. With this implementation, it may not be necessary to implement the complete IEEE standard.

### 4.4.2 Comparison with serial DFE algorithm

An interesting question is how these results compare with a 10 GHz, serial DFE algorithm, using Matlab's double precision floating-point mathematical library. Figure 46 shows how the RTL results compare against the Matlab theoretical results. This figure shows that the double precision, non-delayed serial results are better than those obtained by the physical implementation. This result is expected, as truncation effects alone reduce the precision and BER of the algorithm. For most of the channels, the differences are minor and unimportant, but for those channels that are close to the 1E-12 BER performance metric, Figure 47 shows that there are eight channels where the performance penalty imposed by the parallel implementation is enough to cause the circuit to have an excess error rate. For those eight channels, the serial BER can be as much as 80 dB better than the RTL implementation. This was the penalty for implementing the design in a fixed point, digital process.

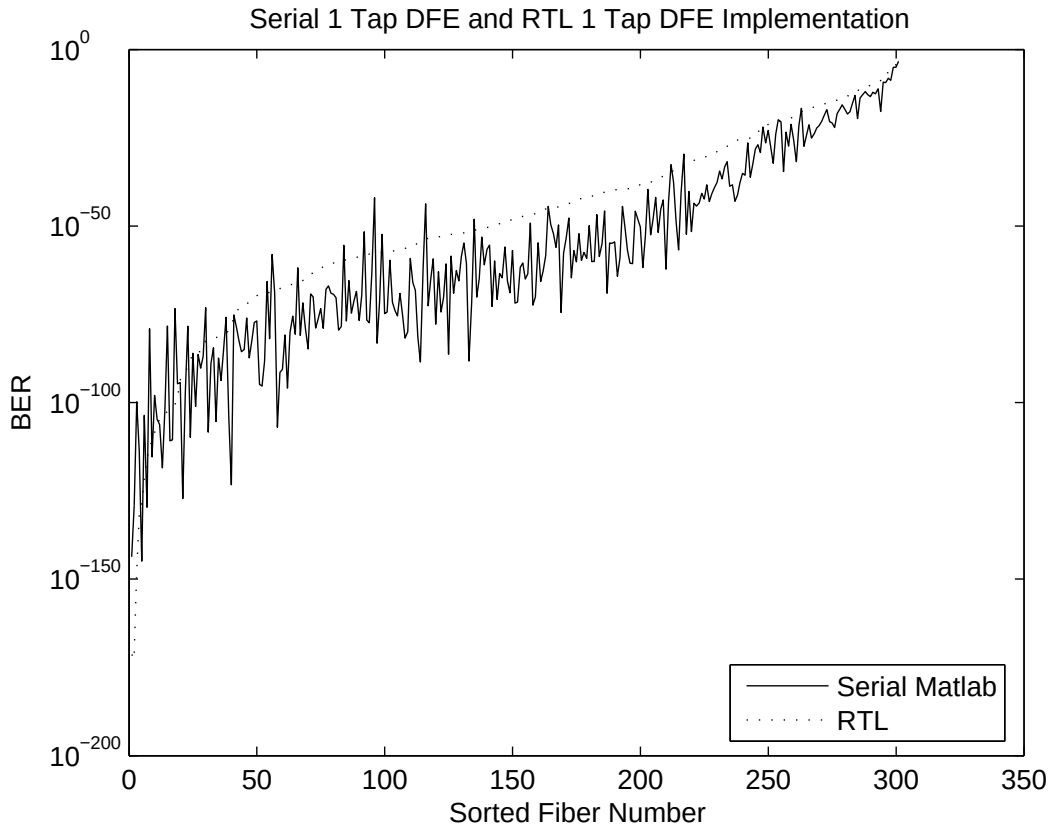


**Figure 46. The performance of the RTL DFE circuit is compared against the perfect Matlab serial implementation. This Figure is sorted by the Matlab BER. There are some channels that appear to perform better in the RTL implementation, but those results all occur at the very low BER rates where estimates of BER are less reliable. At realistic BER rates, the serial implementation performs better than the parallel, RTL implementation.**

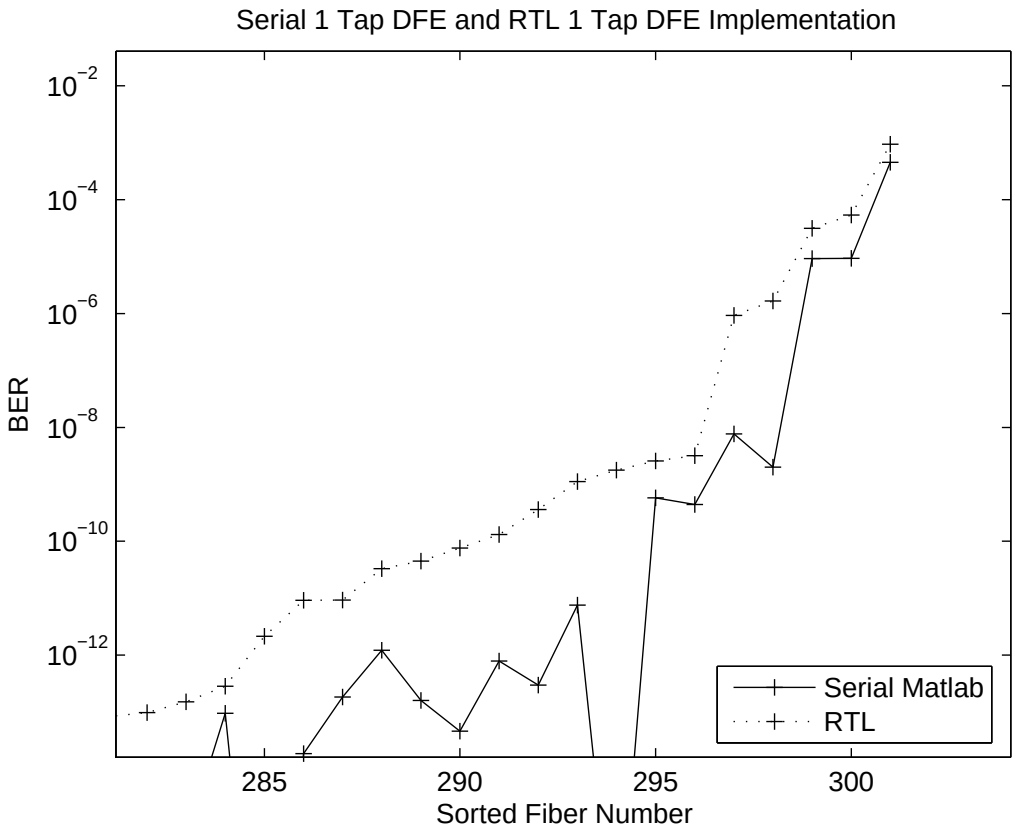


**Figure 47. The high BER region of the DFE performance graph is examined in greater detail. There are eight channels whose performance was sufficient in the serial Matlab model but when implemented in RTL was deficient.**

## 4.5 Summary of DFE Results

Using VLSI loop unrolling techniques, the iteration process bound that has limited the performance of digitally implemented DFE designs has been pipelined and shown to have substantially similar performance to the standard serial algorithm. While a combined synthesis of the full DFE adaptive equalizer circuit was not possible because of CPU memory limitations, the performance of the individual components has been characterized and been judged to be achievable by those experienced in synthesis scripting and control.

The resulting design of 20 FSE taps and 1 DFE tap is able to equalize 93.8% of the fibers and an additional DFE tap will enable the circuit to equalize 97.2% of the fibers modeled, exceeding the IEEE standard's requirement of 95% while achieving it using two fewer DFE taps than was recommended.

# CHAPTER 5

# CONCLUSIONS AND FUTURE WORK

## 5.1 Research Conclusions

Using a novel method, a serial adaptive equalizer has been converted into a parallel implementation. This work has resulted in an RTL implementation of an adaptive equalizer for 10Gb Ethernet. This implementation significantly reduces the complexity of the design and layout tasks because it uses existing lower rate ADCs, and the remaining implementation is performed in a purely digital process. The linear equalizer design is able to equalize 88% of the worst case fibers.

The iteration process bound that served as the primary impediment to the implementation of a parallel DFE algorithm has been unrolled, leading to the first RTL implementation of a 10 Gbps adaptive equalizer. The presented one-tap DFE equalizer has been shown to equalize 93.8 % of the fiber data sets. Process improvements have been identified that have the potential to improve the equalization rate to 97%, exceeding the original IEEE study group's goal of equalizing 95% of the subject fibers.

## 5.2 Research Contributions

The described research makes several novel contributions to the general knowledge of adaptive filters.

1. **Digital control of adaptive filters.** The advantages of analog adaptive filters have been discussed earlier in this thesis. These advantages include reduced power consumption and the use of monolithic ADCs. One of the major drawbacks with an analog design is the tap convergence algorithm must be controlled by analog logic, imposing a limit on the complexity of the convergence algorithm. In addition, the taps may not be pre-set based on a'priori knowledge, as there is not a digital interface from which to control the taps. By implementing a digital control loop, the

proposed design is the first 10 GHz adaptive filter that allows a general purpose processor to interact with the adaptive filter, and allow the pre-loading of coefficients. In addition, because the gradient descent algorithm is designed in digital logic, the LMS algorithm may be replaced by another method with little impact on the high speed portion of the circuit. LMS is used because it is simple to implement, fast, and has low overhead. The proposed design allows the LMS logic to be replaced with a more robust or complicated algorithm without impacting the performance.

2. **10 GHz linear equalizer.** The linear equalizer design demonstrates a parallel, scalable filter that is speed independent. By providing a method to break the adaptive equalizer into parallel blocks, the adaptive equalizer may now be implemented in a digital as opposed to analog process. The analog process is much more subject to process, temperature, and signal integrity effects. The design process is essentially a manual one, any changes to the design may require a completely new layout. A digital design process may be re-targeted to a new foundry or process using only CPU cycles, and the post-layout design checking is much more tool driven, requiring substantially fewer man hours, thus becoming much less error prone. The presented linear equalizer design could easily be expanded to equalize a 20 GHz system by keeping the same clock rate and doubling the number of filter instances. To double the operating speed of an analog system would require a complete re-design of the entire circuit. The proposed digital design could be converted from a 10 GHz system to a 20 GHz system very quickly.

3. **Parallel ADCs for an adaptive filter.** Analog adaptive filters use monolithic ADCs running at very high rates of speed. The main obstacle to the design of these converters is the conversion speed of the analog signal to a digital representation, not the sample window. By demonstrating that the digital adaptive filter is tolerant of parallel ADCs, this project has eliminated the dependence on the ADC conversion speed and

moved the dependency to the sampling window. This allows project teams to either greatly increase their sample rate, or, use the current design for several generations of projects, rather than having to design a new ADC for every rate increase.

4. **Digital 10 GHz DFE implementation.** By unrolling the DFE feedback loop, a one tap DFE has been demonstrated and shown to be feasible. Until now, if a problem required a DFE, an analog implementation was required. There are many problems that would benefit from having a DFE solution, but the implementation cost was too great. Now, simple DFEs may be designed into solutions that were previously off-limits because of the cost of the analog implementation.

## 5.3 Future Work

There are several areas of work that are candidates for future research.

- Up-sample and interpolate the measured impulse responses to allow more precise clock jitter simulation. The goal of the research would be to characterize how much clock jitter on the ADCs can be tolerated before significant performance degradation occurs. During this research, experiments were performed where the delay between the LE taps and the DFE taps was the primary variable, but no conclusive results were found. These experiments were performed before a serious error in the simulation was resolved, so the error may have been masking the effects of the delay. In addition, the range of the delay was constrained to under 100 cycles. In order to prove that the predicted delay boundary of 625e3 clock cycles is correct, the upper range of the delay time needs to be extended. In addition, a new data generation step needs to be created to simulate a changing channel at 1 KHz. This would allow a boundary to be found on how much the DFE can be pipelined in an effort to implement 2 DFE taps.

- Can the selective tap update methods suggested in [16, 17] be utilized to reduce the power consumption, and if so, by how much?

- The proposal [27] to implement the EDC inside the ADC might be modified to work with a post-resolution ADC value. Initial investigation into the recasting of this algorithm suggests that the critical path is the summation of three operands. This critical path was test synthesized and found to meet timing for summing three, five-bit operands within an 833 ps clock period. Thus, if the algorithm can be successfully recast, then implementation should be feasible.

- M-ary phase shift keying (M-PSK) can be considered a super set of the pulse amplitude modulation method used in the 10 Gbps Ethernet standard. Like 10 GbE, M-PSK uses a blind equalization algorithm to remove multi-path and ISI effects. The methods used in M-PSK tend to be computationally complex and not of the type that can be performed in an analog implementation. An investigation of M-PSK blind equalization algorithms could be performed with an eye towards conversion into a parallel algorithm. The blind equalization algorithm could then be applied in place of the LMS methodology proposed here, removing the need for a training sequence or eye-opening monitor.

# REFERENCES


[1] S Bhoja. Equalizer simulation results for 10Gb/s MMF channels. Presentation to the IEEE 802.3aq committee, January 2004.

[2] S. U. H. Qureshi. Adaptive equalization. *Proceedings of the IEEE*, 73(9):1349–1387, 1985. 0018-9219.

[3] J.R Barry, E.A Lee, and D. G Messerschmitt. *Digital Communications*. Kluwer Academic Press, third edition, 2004.

[4] G. Long, F. Ling, and J. G. Proakis. The LMS algorithm with delayed coefficient adaptation. *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on*, 37(9):1397–1405, 1989. 0096-3518.

[5] G. Clark, S. Mitra, and S. Parker. Block implementation of adaptive digital filters. *Circuits and Systems, IEEE Transactions on*, 28(6):584–592, 1981. 0098-4094.

[6] G. Long, F. Ling, and J. G. Proakis. Corrections to 'the LMS algorithm with delayed coefficient adaptation'. *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on]*, 40(1):230–232, 1992. 1053-587X.

[7] C. A. Belfiore and Jr. Park, J. H. Decision feedback equalization. *Proceedings of the IEEE*, 67(8):1143–1156, 1979. 0018-9219.

[8] M. Rupp and A. H. Sayed. Robust FxLMS algorithms with improved convergence performance. *Speech and Audio Processing, IEEE Transactions on*, 6(1):78–85, 1998. 1063-6676.

[9] K. Berberidis and S. Theodoridis. A new fast block adaptive algorithm. *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on]*, 47(1):75–87, 1999. 1053-587X.

[10] K. R. Santha and V. Vaidehi. Design of synchronous and asynchronous architectures for DFT based adaptive equalizer. pages 383–389, 2004.

[11] F. Laichi, T. Aboulnasr, and W. Steenaart. Effect of delay on the performance of the leaky LMS adaptive algorithm. *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on]*, 45(3):811–813, 1997. 1053-587X.

[12] S. Karkada, C. Chakrabarti, and A. Spanias. High sample rate architectures for block adaptive filters. volume 4, pages 131–134 vol.4, 1994.

[13] S. C. Douglas, Zhu Quanhong, and K. F. Smith. A pipelined LMS adaptive fir filter architecture without adaptation delay. *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on]*, 46(3):775–779, 1998. 1053-587X.

[14] Y. Yi, R. Woods, L. K. Ting, and C. F. N. Cowan. High speed FPGA-based implementations of delayed-LMS filters. *The Journal of VLSI Signal Processing*, 39(1 - 2): 113–131, 2005.

[15] Suraiya Chakraborty, Mrityunjoy; Pervin. Pipelining the adaptive decision feedback equalizer with zero latency. *Signal Processing*, 83:2675–2681, 2003.

[16] Xuejing Wang, Fan Ye, and Junyan Ren. An optimization of VLSI architecture for DFE used in ethernet. volume 1, pages 24–32, 2005.

[17] K. Dogancay and O. Tanrikulu. Adaptive filtering algorithms with selective partial updates. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on]*, 48(8):762–769, 2001. 1057-7130.

[18] N. R. Shanbhag and K. K. Parhi. Pipelined adaptive DFE architectures using relaxed look-ahead. *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on]*, 43(6):1368–1385, 1995. 1053-587X.

[19] A. Gatherer and T. H. Y. Meng. High sampling rate adaptive decision feedback equalizers. In *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90, 1990 International Conference on*, pages 909–912 vol.2, 1990.

[20] R. D. Poltmann. Conversion of the delayed LMS algorithm into the LMS algorithm. *Signal Processing Letters, IEEE*, 2(12):223, 1995. 1070-9908.

[21] K. K. Parhi. Pipelining in algorithms with quantizer loops. *Circuits and Systems, IEEE Transactions on*, 38(7):745–754, 1991. 0098-4094.

[22] Yang Meng-Da, Wu An-Yeu, and Lai Jyh-Ting. High-performance VLSI architecture of adaptive decision feedback equalizer based on predictive parallel branch slicer (ppbs) scheme. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(2):218–226, 2004. 1063-8210.

[23] W.C.Jr Black and D.A. Hodges. Time-interleaved converter arrays. *Solid-State Circuits, IEEE Journal of*, SC-15(6):1022–1029, 1980.

[24] S. Milijevic and T. Kwasniewski. 4 Gbit/s receiver with adaptive blind DFE. *Electronics Letters*, 41(25):1373–1374, 2005. 0013-5194.

[25] M. Li, S. Wang, and T. Kwasniewski. DFE architectures for high-speed backplane applications. *Electronics Letters*, 41(20):1115–1116, 2005. 0013-5194.

[26] Li Miao, Wang Shoujun, Chen Jing, and T. Kwasniewski. Design and optimization of multi-tap DFE for high-speed backplane data communications. pages 601–604, 2005.

[27] A. Varzaghani and Yang Chih-Kong Ken. A 6-GSamples/s multi-level decision feedback equalizer embedded in a 4-bit time-interleaved pipeline A/D converter. *Solid-State Circuits, IEEE Journal of*, 41(4):935–944, 2006. 0018-9200.

[28] C. Xia, M. Ajgaonkar, and W. Rosenkranz. On the performance of the electrical equalization technique in MMF links for 10-gigabit ethernet. *Lightwave Technology, Journal of*, 23(6):2001–2011, 2005. 0733-8724.

[29] D. Boerstler, K. Milki, E. Hailu, H. Kihara, E. Lukes, J. Peter, S. Pettengill, J. Qi, J. Strom, and M. Yoshida. A 10+ GHz low jitter wide band PLL in 90 nm PD SOI CMOS technology. pages 228–231, 2004.

[30] Jonathan Ingham, Richard Penty, and Ian White. University of Cambridge multimode-fiber model results, release 1.2. Release Notes for version 1.2 of the model, October 2004.

[31] Jan Peeters Weem. Equalizer simulation results for 10Gb/s MMF channels. Internal status report regarding program activities, January 2005.

[32] S Bhoja, P. Voois, and A. Shanbhag. An overview of electronic dispersion compensation techniques for 10-Gbit/s FDDI grade MMF. Presentation to the IEEE 802.3aq committee, January 2004.

[33] G Agrawal. *Fiber-Optic Communications Systems*. John Wiley And Sons (ASIA), third edition, 2003.

[34] G.A. Constantinides, P.Y.K Cheung, and W. Luk. Truncation noise in fixed-point SFGs. *Electronics Letters*, 35(23):2012–2014, 1999. 0098-4094.