SURYA PRAKASH MATCHA

HIGH PERFORMANCE ARCHITECTURES FOR ADAPTIVE EQUALIZERS USING DISTRIBUTED ARITHMETIC

HIGH PERFORMANCE ARCHITECTURES FOR ADAPTIVE EQUALIZERS USING DISTRIBUTED ARITHMETIC

A

Thesis submitted

for the award

of the degree of

Doctor of Philosophy

by

SURYA PRAKASH MATCHA

Under the

supervision

of

Dr. S. R. Ahamed

Department of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
Guwahati-781039, Assam, India
September 28, 2016

Dedicated
to
THE MOTHER
and
SRI AUROBINDO.

DECLARATION

This is to certify that the thesis entitled "*High Performance Architectures for Adaptive Equalizers using Distributed Arithmetic*", submitted by me to the *Indian Institute of Technology Guwahati*, for the award of the degree of *Doctor of Philosophy*, is an original work carried out by me under the supervision of *Dr. S. R. Ahamed*. The content of this thesis, in full or in parts, have not been submitted to any other University or Institute for the award of any degree or diploma.

Date:

Place:

Surya Prakash Matcha
Research Scholar
Dept. of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
Guwahati-781039, Assam, India

CERTIFICATE

This is to certify that the thesis entitled "High Performance Architectures for Adaptive Equalizers using Distributed Arithmetic ", submitted by Surya Prakash Matcha (09610218), a research scholar in the *Department of Electronics and Electrical Engineering*, *Indian Institute of Technology Guwahati*, for the award of the degree of *Doctor of Philosophy*, is a record of an original research work carried out by him under my supervision and guidance. The thesis has fulfilled all requirements as per the regulations of the institute and in my opinion has reached the standard needed for submission. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Date:
Place:

Dr. S. R. Ahamed
Associate Professor
Dept. of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
Guwahati-781039, Assam, India

True knowledge is not
attained by thinking.
It is what you are;
it is what you become.

— Sri Aurobindo.

ACKNOWLEDGMENTS

A sincere thanks and gratitude to God Almighty! I am deeply indebted to *The Mother, Mirra Alfassa* and *The Master, Sri Aurobindo*, whose constant guidance and grace has helped me to excel in many things and overcome many hardships in my life since my childhood.

Words cannot express how grateful I am to my mother and father for all of the sacrifices they have made on my behalf. The love, support and understanding that I received from them and from my brother were a continuing inspiration to me.

I take this opportunity to thank my supervisor Dr. Shaik Rafi Ahamed, whose constant encouragement during the course of this research has helped me in several ways. I am highly indebted to him for his insightful guidance at various stages of this dissertation and his mentorship during hard times.

I am grateful to my doctoral committee members Prof. S. R. M Prasanna, Prof. Rohit Sinha and Dr. A. Rajesh for their invaluable comments and suggestions for the betterment of this work.

I am grateful to Prof. Roy Paily, a senior faculty member in the VLSI specialization. The software tools required for VLSI design are available in the EEE department mainly because of the research projects going under his investigation. I was able to explore and gain some insights in to the ASIC design flow and FPGA implementation because of these tools. Without them, this work would not have been possible.

I should thank the course co-ordinators like Prof. P. K. Bora, Prof. Samarendra Dandapat, Prof. Ratnajit Bhattacharjee, Dr. Amit Kumar Mishra, Dr. A. K. Mitra and Dr. Amit Acharyya. The courses and their suggestions have definitely contributed for the completion of the thesis.

I am grateful to some of the then research scholars in the EEE department like Mr. Sumit Agarwal, Mr. Shashank Dwivedi, Mr. Samar Sailendra, Mr. Ramesh Mishra, Mr. Rajesh Mahle, Mr. Nagesh, Mr. Ashish Kumar Namdeo, Mr. Kannan, Mr. Sayantan Hazra, Mr. Sandeep Devasrii, Mr. Basudeb Behra, Mr. Raju Mandapati, Mr. Bhaskar Naik, Mr. Gaurav Saxena, Mr. Rahul Shrestha, Mr. Vinay, Mr. Pavan Kumar Manchi, Mr. Samdarshi, Mr. Sai Krishna Santosh, Mr. Saroj Mondal, Mr. Brajesh Rawat, Mr. Mridul Kanthi Malakar, Mr. Ujwal Deep, Mr. Gaurav Saini, Mr. Deepak Joshi, Mr. Satyabrata Das, Mr. Shivanshu Srivastava and few others. I have received a lot of suggestions and help from all of them. The research scholars under my supervisor, Mr. Ajay Kumar Maddirala, Mr. Hari Krishna Veldandi, Mr. Karam Singh, Ms. Babita Jajodia, Mr. Bhoopal Rao and Mr. Tasleem Khan have helped me in various aspects during my long stay at IIT Guwahati. I had a very good time with all of them.

I should not forget some of the research scholars from other departments too particularly, Mr. Padam Rajender, Mr. Partha Dey, Mr. Ruhit, Mr. Abhijit das, Mr. Santosh

Kumar, Mr. Munendar, Mr. Kiran Indukuri, Mr. Anil Kumar, Mr. Kishore, Mr. Ramesh Aadimoolam, Mr. Somsekhar Reddy, Mr. Lakshman Burugula, Mr. Suresh BM, Mr. Harsha, Mr. Sravan, Mr. Ravi, Mr. Enamullah khan, Mr. Prakash Saudugar, Mr. Agile Mathews, Mr. Rajender gone, Mr. Jitender Kumar and few others, along with the then postgraduate students like Mr. O. V. Prasad, Mr. Kakumanu Prasad, Mr. Musthyala Harish, Mr. Mehaboob Jani, Mr. M. Rajendra Kumar, Mr. Arun Chintha, Mr. Sriman Narayana, Mr. Srikanth Bhavana, Mr. Bharath Tippineni, Mr. V. Raju, Mr. Naren, Mr. Subba Reddy, Mr. Ravinder Reddy, Mr. Chandan Singh, Mr. Jagadeesh Gughalot, Mr. Suyog, Mr. Sunil Joshi, Mr. Ravi Gedela, Mr. Arvind Gunupura, Mr. Manoj B Rajashekara, Mr. Dhiraj Lote, Mr. Sudhanshu Baghel, Mr. Gaurav Jain, Mr. Vicky Chheda, Mr. Sachin Kashyap, Mr. Raju Jaitwar, Mr. Pradeep Suragala, Mr. Biki Teron, Mr. Eeswar Gurram, Mr. B. Santosh, Mr. Ramana Kasireddy, Mr. Nikhil, Mr. Kishore, Mr. Sai Krishna, Mr. Rakesh Reddy, Mr. Vijay Kumar, Mr. Arvind Polkam, Mr. Siva Namthoti, Mr. Raj Kumar, Mr. Gajendra Kumar, Mr. Vijay Raghav, Mr. Manikanteswar Reddy along with many others with whom I had lot of fun during my seven year stay at IIT Guwahati. These people have also helped me in various ways for the completion of this work. Among all these, a special mention to Mr. Sandeep Devasrii, Mr. Rajendra Kumar Maliga and Mr. Ajay Kumar Maddirala, the technical discussions with whom, have helped me to carry out simulations and make a faster progress in my work. A special mention also to Mr. Sai Krishna Santosh Gollapudi, Mr. Padam Rajender, Mr. Rajendra Kumar Maliga, Mr. Vinay, Mr. Pavan Kumar Manchi and Mr. Bhaskar Naik, from whom I received a lot of moral support and tremendous amount of help.

I have received some good suggestions and help from Mr. Sagar Koorapati of Ineda Systems Pvt. Ltd., who later turned out to be the co-author for a publication. I must also mention Mr. Chaitanya Kumar Matcha, my elder brother, who has shown me proper directions at various points of time during my pursuit.

I may be called ingrate if I do not mention here some of the people in the electronics engineering community who have shared their ideas, codes and creators of some open-source softwares and knowledge sharing websites like Wikipedia and Asic-World, discussion forums and creators of many internet resources which have helped me in various ways for the completion of the thesis.

A special thanks to all the faculty members, staff, alumni and laboratory assistants of IIT Guwahati who supported me in writing, and incited me to strive towards my goal. I must also thank many working staff of IIT Guwahati in the hostels, hostel messes, canteens and in specific the staff of Core-2 coffee shop who have helped me in serving the snacks during my presentations.

At the end I would like to express my thanks to all the friends, relatives and many others who have helped me in several ways for the completion of this work.

ABSTRACT

In communications, adaptive equalizers not only counter the intersymbol interference (ISI) introduced by the channel but also keep track on the changes in the channel characteristics out of which the adaptive decision feedback equalizer (ADFE) performs well because of its low-noise enhancement compared to linear equalizers (LEs). However high speed applications demand for a large number of taps in the feed forward filter (FFF) and feedback filter (FBF) of ADFE and hence the implementation and real-time operation of such an equalizer is a difficult task due to increased complexity and very small intersymbol period. Distributed Arithmetic (DA), which uses partial-products for the computation of the inner product of two vectors, can be a suitable technique since DA based realization of digital signal processing (DSP) algorithms can lead to low computational cost while achieving high system throughputs. Further, the modular and memory-based structure of DA can lead to the resultant ADFE in enjoying the advantages of ease of implementation making it more suitable for implementing on field-programmable-gate-arrays (FPGAs) and design of application specific integrated circuits (ASICs).

In this thesis, we first applied appropriate DA treatment for a least mean square (LMS) based adaptive filter. Although, the DA based implementation of systems with fixed coefficients such as the finite-impulse response (FIR) filters and transforms is easy, problem arises when realizing adaptive filters and recursive systems viz-a-viz the partial products of the coefficients need to be updated from time to time. For this purpose, we have taken an auxiliary look-up-table (LUT) storing the partial-products of most recent input samples along with the LUT that stores the partial-products of filter coefficients. For reducing the memory requirement of DA, we used the offset binary coding (OBC) scheme. The auxiliary LUT is updated from time to time by manipulation of its addressing bits and by the use of the entries in its consecutive address locations. Later, we explored the DA advantages by the realization of a non-adaptive decision feedback equalizer (DFE). One of the traditional ways of implementing the non-adaptive DFEs is by the use of one or more multiply-and-accumulate (MAC) units and such an implementation is performance limited by the presence of bulky multipliers. We have used the digit-serial nature of DA and the proposed structure has been shown to be suitable for real time implementations as they offer considerable savings over their MAC based counterparts. Next, we extended the DA philosophy for the realization of an LMS based ADFE. Such an ADFE is performance limited by the the presence of feedback loop and the weight-adaptation loop. The proposed DA based ADFE uses separate DA treatments for FFF and FBF where the idea of auxiliary LUT is used again for handling the weight-update operations. Lastly, the DA philosophy is applied for the realization of block ADFE. Block ADFEs suffer from the causality problem i.e., the block processing of FBF cannot be performed due to the presence of unknown decisions in every iteration which are to be sought out by the ADFE. We have computed these unknown decisions by using an efficient approach that uses two minimization criteria along with a bank of registers where all the symbol values used in the modulation scheme are stored. Once, the unknown decisions are computed, we have used the fast convolution techniques where the DA philosophy is applied for the reduction of computational complexities.

The initial attempt of optimizing the LMS adaptive filter using DA utilizes OBC scheme. The technique uses an auxiliary look-up-table which stores the partial-products of input samples to aid the weight-update operation. The proposed filter is mainly useful for the realization of higher order filters and is found to be utilizing between 20%-30% less chip-area compared to the best of existing architectures. The proposed non-adaptive DFE realizations can be configured to obtain three different architectures all of which use digit-serial input scheme. While for lower order FFF and FBF upto 40% of reduction in FPGA resources have been observed, the three architectures use more or less the same number of computational resources as that of multiplier-and-accumulator (MAC) based implementation for higher order FFF and FBF filters. In case of LMS-ADFE, the idea utilizes the OBC scheme and separate auxiliary look-up-tables for FFF and FBF filters in order to aid the weight-update operation. While the proposed DA based LMS-ADFE uses no hardware multipliers for its implementation, good computational savings have been achieved although it was found that there is an exponential increase in the size of memories because of the requirement for two auxiliary look-up-tables for FFF and FBF individually. Lastly, the proposed block-ADFE implementation uses two minimization criteria for the computation of unknown decisions in every block iteration. This idea is based on the fact that the unknown decisions take one value out of all symbol values used at the modulation scheme. The proposed DA based algorithm may be considered semi-optimal as there are some inconsistencies found in the FBF coefficients approaching the weiner solution while FFF coefficients do so.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 INTRODUCTION

Most of the physical channels, such as the telephone lines, are band-limited in nature and are responsible for the distortion of the transmitted signal. The distortion can be of two types namely amplitude distortion and phase distortion. Amplitude distortion occurs if the magnitude of the frequency response of the channel is not constant within the bandwidth of the signal. Phase distortion occurs if the phase of the frequency response of the channel is not a linear function of the frequency i.e., the delay introduced by the channel for all the frequency components of the signal is not the same. Typically, one or both the types of distortion can occur depending on the characteristics of the channel. Signal smearing can occur if the signal reaches the receiver via multiple paths which is more often in wireless communications. The causes for this include reflection & refraction of the signal with objects such as buildings, atmospheric effects such as atmospheric ducting & ionospheric reflection.

In case of digital communications, messages are transmitted in the form of symbols which are decoded at the receiving end. The distortion of the signals because of the fore-mentioned reasons can make the symbols overlap thus making the receiver difficult to distinguish a symbol from its neighboring symbols. This phenomenon of overlapping of successive symbols is known as *intersymbol interference* (ISI) which is depicted in Fig. 1.1. Measures have to be taken at the receiving end before the extraction of message to avoid the ISI as it may cause wrong interpretations of the message. Depending on the application, various methods are adopted to counter the ISI. One such method includes designing systems such that the impulse response is very short and only a small amount of energy smears from one symbol into the adjacent symbol. Usage of guard periods to separate the symbols in time domain can also help in elimination of the ISI. However, in most of the practical cases, the channel impulse response spans over many tens of signalling periods and the methods mentioned above may not be effective in equalizing such channels. Nyquist investigated the problem and put forth the conditions on the communication channel (impulse responses of transmit filter, channel and receive filter) such that no ISI can occur. He showed that, for zero ISI, the minimum theoretical system bandwidth is only half that of the data rate of transmission.

The most prominent method of approaching the ISI problem is to use a digital filter at the receiver end which approximates the inverse of the discrete time model of the channel impulse response. It can be noted that the nth received pulse in the absence of

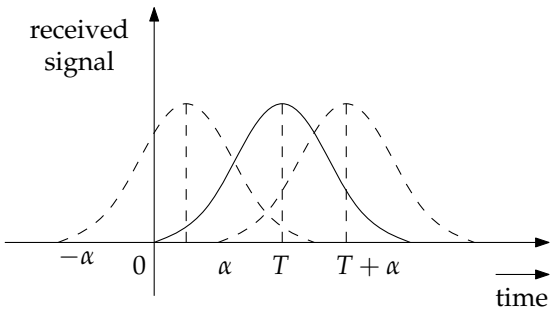

Figure 1.1: The Intersymbol Interference Phenomenon.

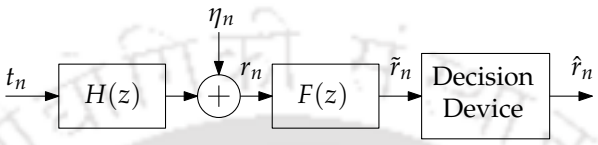

Figure 1.2: The Zero Forcing Equalizer system model.

noise can be given by the convolution sum as $r_n = t_n * h_n$ where, t_n is the nth transmitted pulse and h_n is the impulse response of the equivalent discrete time model of the channel. The impulse response can contain causal and anti-causal parts which contribute for the pre-cursor and post-cursor parts of the overall ISI. With $h_n \longleftrightarrow H(z)$, the ISI at the receiver can be completely eliminated by using a filter called *zero-forcing equalizer* whose transfer function is given by $F(z) = 1/H(z)$. Such a model of the transmitter, channel and equalizer system is shown in Fig. 1.2 where η_n represents the additive noise in the channel and $\tilde{r}_n$ denotes the pre-decision output. The error $e_n = t_n - \tilde{r}_n$ is a measure of the performance of the equalizer. With $e_n \longleftrightarrow E(z)$ and $\eta_n \longleftrightarrow N(z)$, $E(z)$ can be given as

$$\begin{aligned} E(z) &= T(z) - \tilde{R}(z) \\ &= T(z) - R(z) F(z) \\ &= T(z) - T(z) H(z) F(z) - N(z) F(z) \\ &= T(z) (1 - H(z) F(z)) - N(z) F(z) \end{aligned}$$

Now, the power spectrum $S_e(z)$ of the error signal can be given as

$$S_e(z) = S_t(z) |1 - H(z) F(z)|^2 + S_\eta(z) |F(z)|^2$$

where $S_t(z)$ and $S_\eta(z)$ are respectively the power spectra of the transmitted data symbols and the additive noise. The ISI contribution to the error vanishes completely if zero-forcing criteria i.e., $F(z) = 1/H(z)$ is imposed, however, the contribution of the noise power will be infinity at those frequencies where the channel transfer function becomes zero ($H(z) = 0$ for some z). Even without the presence of spectral nulls, the noise power will be greatly enhanced if there are any deep nulls in the spectrum of $H(z)$.

Decision Feedback Equalizers (DFEs) which use the regenerative effect of the decision device, are an effective means of equalizing the channels exhibiting deep spectral nulls. Unlike the linear equalizers, DFEs do not estimate the inverse of the channel directly,

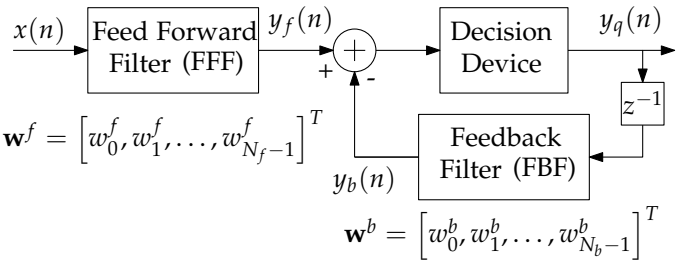

Figure 1.3: Block diagram of a decision feedback equalizer.

instead the filtered version of past decisions are subtracted out from the equalizer output. The basic block diagram of a DFE is shown in Fig. 1.3. A DFE basically consists of a feed forward filter (FFF), a feedback filter (FBF) and a decision device (quantizer). The FFF works directly on the received data and equalizes the anti-causal part of the channel transfer function. The residual ISI at the FFF output is then cancelled out by subtracting the FBF output from the FFF output. The output of FBF whose coefficients are carefully chosen, operates on the decisions made on the past symbols. The DFE works basically on the assumption that there are no errors at the output of the decision device. As long as the decisions are correct, the DFE can equalize the channel effectively with low noise enhancement. But if at-least one of the past decisions is incorrect it may lead to successive incorrect decisions and hence error propagation may occur. However, on typical channels the DFEs are typically self-corrective as the errors occur only in short bursts [38].

The DFE helps in equalizing the channel when the channel characteristics are constant. However, the channel coefficients of most of the practical channels vary with respect to time. This is more often in the case of wireless communications due to the relative movements in the physical objects present in the channel. In such cases, the DFE needs to adapt its filter coefficients based on the variations in the channel characteristics. Such an equalizer is known as *adaptive decision feedback equalizer* (ADFE). The general block diagram of a sample by sample ADFE is shown in Fig. 1.4. The FFF and FBF coefficients of ADFE are made to adapt based on a suitable adaptation algorithm and the ADFE is generally operated in two modes: (i) the initial training mode where a copy of the original transmitted sequence is used for training the filters i.e., the desired signal $d(n) = \hat{x}(n)$ and error $e(n) = \hat{x}(n) - y(n)$ (ii) the decision-directed mode where the output decisions are used for tracking any slow variations in the channel and in this mode, the desired signal $d(n) = y_q(n)$ and error $e(n) = y_q(n) - y(n)$. The error, along with the filter inputs are used to adjust the tap-weights of FFF and FBF in a time-recursive manner using the adaptive algorithms such as least-mean square (LMS), block LMS, recursive-least squares (RLS) etc.

1.2 PERFORMANCE ISSUES AND RELATED RESEARCH IN ADFE

The ADFEs play an important role in wireless communications and hence their implementation issues need to be addressed carefully by considering their design difficulty

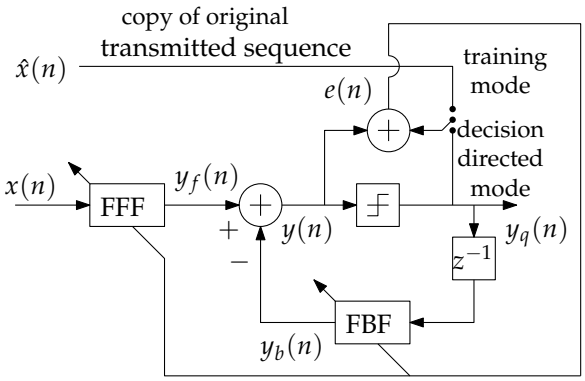

Figure 1.4: General block diagram of a sample by sample processing based adaptive decision feedback equalizer.

caused by the feedback loop and the non-linear device. A common problem faced with ADFE is that with increasing data transmission rate, more and more adjacent symbols get overlapped and thus the orders of both FFF and FBF increase. This is because, with higher data transmission rate, the signal bandwidth becomes large and when the signal bandwidth becomes more than the coherence bandwidth of the channel, the link suffers from multi-path fading which results in severe ISI. Hence, large number of filter taps are required for equalizing such channels. The resulting increase in complexity makes real time operation of the ADFE difficult due to very short symbol period, which means that lesser and lesser time will be available to carry out the computations while the volume of computation goes on increasing. The complexity can go up further if one employs fast converging equalizers such as those belonging to the RLS family, which require shorter training sequences and thus provides valuable savings in bandwidths. The power and chip area requirement also go up as the complexity increases.

Pipelining [46] of adaptive DFEs is difficult as the ADFE contains a non-linear device in the decision-feedback loop. The presence of weight-adaptation loop in the ADFE also makes it difficult to achieve pipelining. Hence, the initial works on high-speed ADFE architectures have almost exclusively adopted parallelization [17, 20, 39]. Algorithms in [20, 39] result in the loss of performance due to incorrect initialization of the FFF and a coding loss due to the incorrect initialization of FBF. Later, look-ahead based approaches were successfully applied to pipeline linear time-invariant recursive filters and feed-forward adaptive lattice filters. Look-ahead was also used in parallel implementation of linear recursive systems and in block implementation of recursive adaptive digital filters. However, algorithms that contain quantizer loops pose difficulty in pipelining the system using look-ahead computation. To overcome this, a method of pipelining algorithms with quantizer loops was proposed by Parhi in [34]. Here, using look-ahead computation, loops containing nonlinear devices are transformed to equivalent forms which contain no nonlinear operation. The drawback of this approach is that the hardware keeps growing with increase in the number of levels in the quantizer or the order of the predictor in algorithms with quantizer loops. Later Parhi *et. al* have successfully applied look-ahead and relaxed look-ahead [45] in order to pipeline adaptive DFEs. To achieve this, the functionality of the algorithm is maintained rather than the input-output behavior which resulted in good computational savings.

Later, there was an inclination in the field of research towards the use of block based and frequency domain based approaches which were inspired from the development of fast convolution techniques like the overlap-save and overlap-add methods. In [9, 7, 42], several block and frequency domain based techniques have been proposed. In [9], Berberidis et al. presented a new block ADFE that is mathematically equivalent to the conventional LMS-based sample-by-sample DFE but with considerably reduced computational load. A new block ADFE implemented in frequency domain was later proposed by the same author in [7]. In [42], Rontogiannis et al. subsequently presented a new efficient DFE appropriate for channels with long and sparse impulse response. It is shown that, in case of sparse channels, the FFF and FBF have a particular structure, which can be exploited to derive efficient implementation of DFE, provided that the time delays of the channel impulse response multipath components are known. In [55], a high speed pipelined ADFE is derived where a postcursor processing filter (PCF) is used to cancel the most significant postcursor Intersymbol Interference (ISI). For this purpose, a new updating ADFE scheme based on the Principle of Orthogonality is derived. It has been shown that the PCF-ADFE structure, while maintaining the similar hardware cost as that of PIPEADFE1 and PIPEADFE2, has a better convergence rate.

In [35], pipelining technique has been extended for the realization of a multiplexer loop based adaptive DFE. In this, a novel look-ahead approach is developed for parallel nested multiplexer loops, based on which pipelining or parallelism can be achieved. In [43], an efficient finite precision realization of the block adaptive decision feedback equalizer based on the block floating point (BFP) approach has been presented. The motivation behind going for this approach is that the direct floating point (FP) based approach would result in high processing cost and complexity due to additional steps in the floating point approach compared to the fixed point counterpart. Based on the lengths of FFTs and IIFTs and the exponents in the floating point representation of the data, the proposed approach is three to four times faster than the fixed point counterpart for moderately large values of the block lengths. A finite word length analysis of efficient block adaptive DFE (EBA-DFE) has been carried out in [27]. Here, both the truncation and round-off schemes have been studied and it has been shown that moderate word lengths are required for facilitating the hardware implementation of the EBA-DFE. A simplified approximate rounding scheme is also presented and it is shown that the proposed scheme is efficient in simplifying the complexity of rounding a product without having effect on the behavior of the algorithm. In [30], a new look-ahead method to break the feedback path for multi-gigabit DFE design is proposed. Here, parallelization has been adopted to increase the throughput rate of the system and is shown that the throughput rate can be proportional to the parallelization factor. The computational complexity here is lower than that of the multiplexer-based look-ahead provided that the tap number of the feedback filter is large.

A signed normalized block LMS adaptive decision feedback equalizer has been proposed in [28]. For this purpose, a normalized version with respect to the square euclidean norm of the tap input vector is used for the correction of the tap-weight vector. Here, once again the incoming data is partitioned into non-overlapping blocks and the filtering operations are performed in the frequency domain using the overlap and save FFT method. This facilitated for the easier choice of the step-size with which the proposed algorithm was convergent in the mean square sense. This is different from the time-domain based approach where the information of the largest eigen value of the correlation matrix of the input sequence is required. Simulation studies have shown that the proposed algo-

rithm has good convergence performance and better bit error rate (BER) performance compared to the existing algorithms. In the recent past [44], a block floating point (BFP) realization of ADFE has been taken up. Here, separate BFP treatments have been carried out for the FFF and FBF weights and separate weight-update relations have been developed for their respective mantissas and exponents. However, this technique suffers from co-efficient round-off noise errors.

In [25], an adaptive frequency-domain DFE (ADF-DFE) for single-carrier frequency division multiple access (SC-FDMA) systems has been developed. Here both the feed-forward and feedback filters operate in the frequency domain and are adapted using the block recursive least squares (RLS) algorithm. It is shown that the complexity of the ADF-DFE can be reduced when compared to its time-domain counterpart by exploiting a matrix structure in the frequency domain. The authors also extended the formulation for multiple-input-multiple-output (MIMO) SC-FDMA systems. It was observed that the proposed AFD-DFE is robust and enjoys significant reduction in computational complexity when compared with the frequency-domain non-adaptive DFE. In [31], a CORDIC based pipelined ADFE architecture using reformulated LMS algorithm has been proposed. The proposed ADFE is observed to have not only reduced number of multipliers in the feed forward path, but also easy to implement due to the flexibility and modularity in the structure. The resultant ADFE is suitable for mitigating severe channel distortion which arise due to ISI in high speed communications and found to be area efficient with good convergence. Based on the above literature, we conclude that performance enhancement of ADFE can be achieved by optimizations at both the algorithmic and architectural levels and so all these aforementioned works along with the demanding need for complexity reduction of ADFEs motivate us to take up the work of performance enhancement in ADFE. In this process, we found that distributed arithmetic is a suitable candidate for the performance enhancement in ADFE.

1.3 DISTRIBUTED ARITHMETIC AND ITS VARIANTS

Distributed Arithmetic (DA) is an efficient way of computing the inner product of two vectors where one of the vectors is known prior to implementation. The basic idea behind DA is that the pre-computed partial-products are stored in look-up-tables (LUTs) and through shift-and-accumulate operations, the inner product is computed. The computation of inner product of two vectors using DA is explained in the following paragraphs. Consider the inner product of two vectors $\mathbf{c}$ and $\mathbf{x}$ $(i = 0, 1, \ldots, N-1)$ which can be given as

$$y = \sum_{i=0}^{N-1} c_i x_i \tag{1.1}$$

If every sample of x_i is written in fixed point $Q1.B-1$ ([1]) signed two's complement representation, then

$$x_i = -b_{i,B-1} + \sum_{j=1}^{B-1} b_{i,B-1-j} 2^{-j} \tag{1.2}$$

where $b_{i,j} \in \{0, 1\}$.
Substituting (1.2) in (1.1) and re-arranging, we get

1 Appendix A

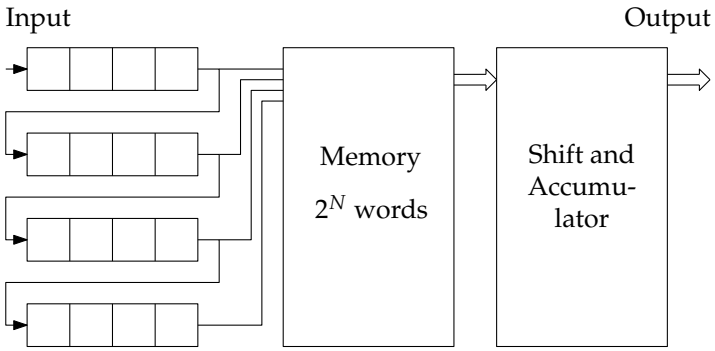

Figure 1.5: General Block Diagram of distributed arithmetic architecture.

$$y = -\sum_{i=0}^{N-1} c_i b_{i,B-1} + \sum_{j=1}^{B-1} \left\{ \sum_{i=0}^{N-1} c_i b_{i,B-1-j} \right\} 2^{-j} \tag{1.3}$$

Let

$$f_j = \sum_{i=0}^{N-1} c_i b_{i,B-1-j} \tag{1.4}$$

and

$$C_{B-1-j} = \begin{cases} -f_0 & j = 0 \\ f_j & j \neq 0 \end{cases} \tag{1.5}$$

Hence, (1.3) becomes

$$y = \sum_{j=0}^{B-1} C_{B-1-j} 2^{-j} \tag{1.6}$$

From (1.4) and (1.5), it can be observed that, taking j-th bit from each of x_i, the term C_{B-1-j} would take only one out of 2^N possible combinations which are nothing but the partial products of elements of c_i. Hence all these combinations can be stored in an LUT (usually a read-only memory (ROM)) whose address bits are formed by j-th bit of every x_i . Then the output y can be computed by shifting the partial products taken from the memory for every j-th $(j = 0, 1, \ldots, B-1)$ set of bits of x_i's and accumulating all of them using shifting operation. As there are B bits in all x_i's counting from LSB to MSB, the system would take a total of B clock cycles to compute the inner product. Fig. 1.5 shows the DA structure for computation of the inner product as per (1.6) with $B = 4$. Here the address bits are indexed by j and the memory word is indexed by i .

As there are 2^N partial products, the size of the LUT would be 2^N and so if N is large, the LUT size requirement would be high. DA provides the flexibility for the usage of multiple small sized LUTs. The idea can be understood if N is split as $N = M \times K$ in which case (1.4) becomes

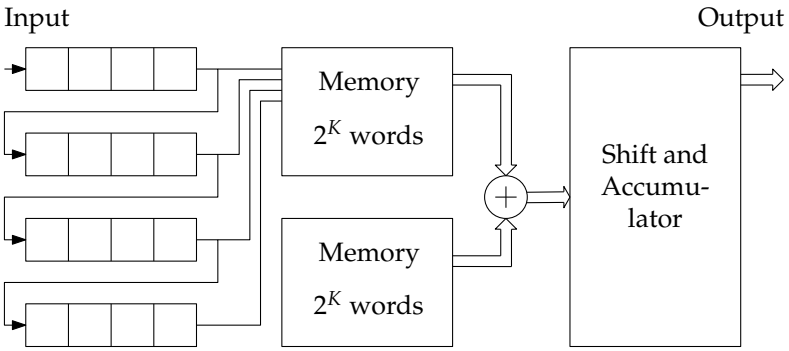

Figure 1.6: Distributed arithmetic architecture with ROM decomposition.

$$f_j = \sum_{i=0}^{MK-1} c_i b_{i,B-1-j} \tag{1.7}$$

$$= \sum_{i=0}^{K-1} c_i b_{i,B-1-j} + \sum_{i=K}^{2K-1} c_i b_{i,B-1-j} + \ldots + \sum_{i=(M-1)K}^{MK-1} c_i b_{i,B-1-j} \tag{1.8}$$

$$= \sum_{m=0}^{M-1} \sum_{i=mK}^{(m+1)K-1} c_i b_{i,B-1-j} \tag{1.9}$$

If $s = i - mK$ and if the dummy variable s is replaced by i, then

$$f_j = \sum_{m=0}^{M-1} \left\{ \sum_{i=0}^{K-1} c_{i+mK} b_{i+mK,B-1-j} \right\} \tag{1.10}$$

Hence, M number of LUTs can be used each of size 2^K, taking address lines from every K sets of x_i's. The DA structure corresponding to (1.10) with $K = 2$ and $B = 4$ is shown in Fig. 1.6 where the address bits are indexed by j, the LUT block is indexed by m and the LUT word is indexed by i. The number of clock cycles to compute the inner product remains the same but the number of arithmetic operations (adders) would increase. Specifically, the system requires a total of $M \times 2^K$ LUT locations and $M - 1$ adders and computes the inner product in B clock cycles. $K = N$ is same as (1.4) and $K = 1$ case will have N individual LUTs with two locations in each. In order to increase the speed of the system, multiple bits of x_i's can be used in parallel as the address lines to the LUT at the cost of increase in its size. This is obtained by splitting B as $B = P \times Q$ in which case (1.6) becomes

$$y = \sum_{j=0}^{PQ-1} C_{B-1-j} 2^{-j} \tag{1.11}$$

$$= \sum_{j=0}^{Q-1} C_{B-1-j} 2^{-j} + \sum_{j=Q}^{2Q-1} C_{B-1-j} 2^{-j} + \ldots + \sum_{j=(P-1)Q}^{PQ-1} C_{B-1-j} 2^{-j} \tag{1.12}$$

$$= \sum_{p=0}^{P-1} \sum_{j=pQ}^{(p+1)Q-1} C_{i,j} 2^{-j} \tag{1.13}$$

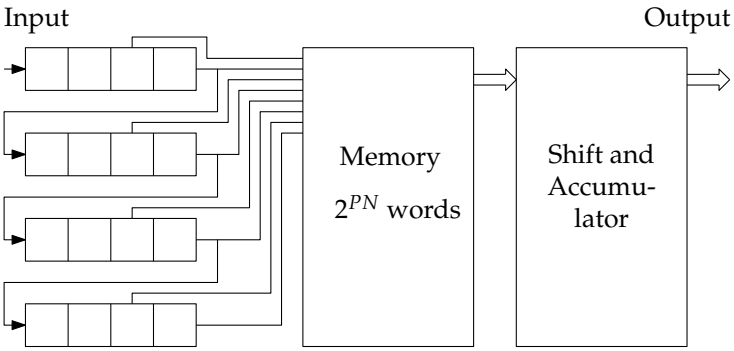

Figure 1.7: Distributed arithmetic architecture with digit-serial Scheme.

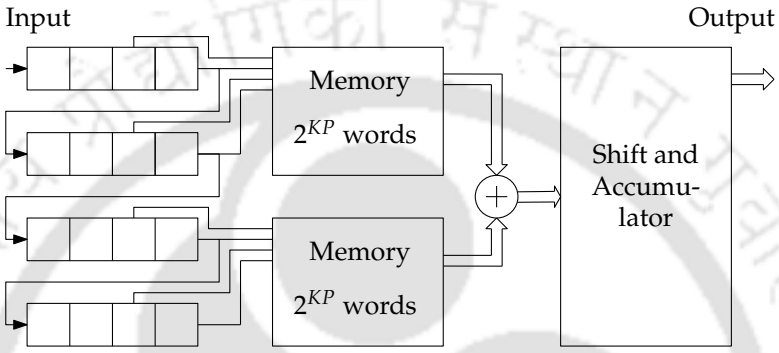

Figure 1.8: Distributed arithmetic architecture with combined ROM decomposition and digit-serial scheme.

If $s = j - pQ$ and if the dummy variable s is replaced by j, then

$$y = \sum_{p=0}^{P-1} \sum_{s=0}^{Q-1} C_{i,j+pQ} 2^{-(j+pQ)} \tag{1.14}$$

The DA structure (also called digit-serial DA) corresponding to (1.8) with $P = 2$ is shown in Fig. 1.7. In such case, the size of the memory would be 2^{PN} and the speed is increased by a factor of P and hence the inner product is computed in B/P clock cycles. Further it can be noted from (1.8) that the shifting is done for P units unlike the previous case where unit shifting is done on each of the partial product. If $P = B$ then all the B-bits of x_i's form the address lines and the inner product is computed in a single clock cycle while $P = 1$ is same as 1.6.

Hence, from (1.10) and (1.14), it can be noted that splitting N would result in the low memory size requirement at the expense of increased combinational logic and employing digit-serial would speed up the system at the cost of increased parallelism. Alternately, one can employ the ROM splitting as well as the digit-serial nature of DA as shown in Fig. 1.8 for the trade-offs between speed and hardware complexity based on the parameters M and P as explained above. These techniques apply equally well when the partial products of x_i's are stored and the bits of c_i's are used as the address bits to the LUT. Further, the techniques work well for the signed complement representation of the elements of the vectors [36].

1.4 DA BASED FIR FILTER DESIGN

One of the direct applications of DA would be the realization of a finite impulse response (FIR) filter as the output of an FIR filter at any instant is the weighted sum of present and past input samples. The state-of-the-art implementation of such filters typically involve DSP processors or custom logic design containing one or more hardware multiply-and-accumulate (MAC) units. One of the main advantages of DA is that the number of clock cycles it takes to compute the inner product depends on the bit-length of the input sequence unlike the case of MAC units where it depends on the filter order. By careful design using DA, one may reduce the total computational savings of the system upto 80% [53]. Further, the flexibility of configuring the DA architecture as per the speed/complexity requirements and its multiplier-free structure makes it one of the most effective technique for the realization of higher order filters.

1.4.1 *DA based FIR filter*

The output $y[n]$, $(n \in Z)$ of an N-tap FIR filter with the present input sample $x[n]$ is given as

$$y[n] = \sum_{i=0}^{N-1} w[i]x[n-i] \tag{1.15}$$

where $w[i]$ $(i = 0, 1, ...N-1)$ are the weights of the filter.

Representing each of the input sample $x[n-i]$ in the 2's-complement we have

$$x[n-i] = x_{n-i} = -b_{i,B-1} + \sum_{j=1}^{B-1} b_{i,B-1-j}2^{-j} \tag{1.16}$$

Substituting (1.16) in (1.15) and re-arranging we get

$$y[n] = \sum_{j=0}^{B-1} c_{B-1-j}2^{-j} \tag{1.17}$$

where

$$\begin{aligned} c_{B-1-j} &= \sum_{i=0}^{N-1} w_i b_{i,B-1-j} \quad (j \neq 0), \\ c_{B-1} &= -\sum_{i=0}^{N-1} w_i x_{i,B-1} \end{aligned}$$

For a given set of w_i $(i = 0, 1, ..., N-1)$, the terms c_{B-1-j}'s would take only one out of 2^N possible combinations which can be pre-computed and stored in a look-up table (LUT). The DA based implementation of a 4-tap FIR filter is shown in Fig. 1.9. The incoming bits of input samples are stored in the registers in the order that at any instant, the bits of most recent input sample are stored in the top most register while the bits of oldest sample are stored in the bottom most register. The least-significant bits (LSB) from each of the registers form the address lines to the LUT containing the partial products. The partial

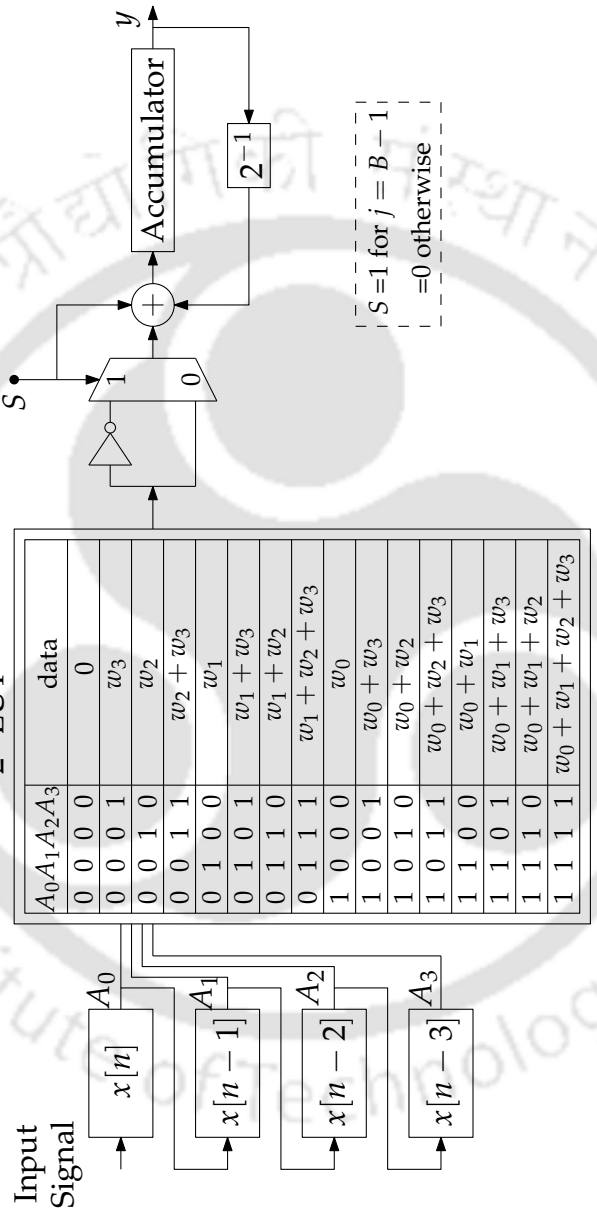

Figure 1.9: DA based implementation of a 4-tap FIR filter.

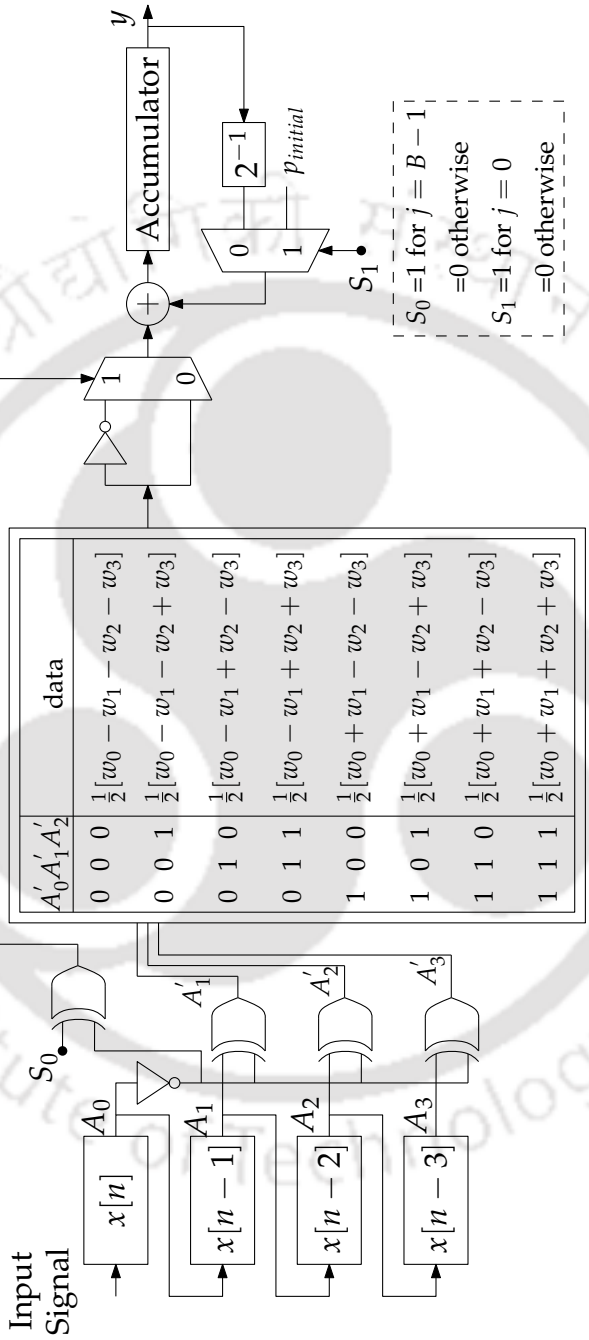

Figure 1.10: DA based implementation of 4-tap FIR filter with OBC scheme.

products are then shifted and accumulated for 'B' number of clock cycles to produce one sample of output.

1.4.2 *DA based FIR filter with offset binary coding scheme*

The ROM size in the above architecture can be further reduced using the offset-binary coding (OBC) technique [36] which can be derived as follows:
Re-writing (1.16) as $x_{n-i} = \frac{1}{2}[x_{n-i} - (-x_{n-i})]$ we have:

$$\begin{aligned} x[n-i] = & \frac{1}{2}\left[-\left(b_{i,B-1} - \bar{b}_{i,B-1}\right)\right] \\ & + \frac{1}{2}\left[+\sum_{j=1}^{B-1}\left(b_{i,B-1-j} - \bar{b}_{i,B-1-j}\right)2^{-j} - 2^{-(B-1)}\right] \end{aligned} \tag{1.18}$$

Choosing

$$d_{i,j} = \begin{cases} -(b_{i,j} - \overline{b_{i,j}}), & j \neq B-1 \\ -(b_{i,B-1} - \overline{b_{i,B-1}}), & j = B-1 \end{cases} \tag{1.19}$$

Substituting (1.18), (1.19) in (1.15) and re-arranging we get

$$\begin{aligned} y[n] = & \sum_{j=0}^{B-1}\left(\sum_{i=0}^{N-1}\frac{1}{2}w_i d_{i,B-1-j}\right)2^{-j} \\ & - \left(\frac{1}{2}\sum_{i=0}^{N-1}w_i\right)2^{-(B-1)} \end{aligned}$$

Defining

$$p_j = \sum_{i=0}^{N-1}\tfrac{1}{2}w_i d_{i,j}, \quad 0 \leq j \leq B-1$$

and

$$p_{initial} = -\frac{1}{2}\sum_{i=0}^{N-1} w_i \tag{1.20}$$

We arrive at

$$y[n] = \sum_{j=0}^{B-1} p_{B-1-j}2^{-j} + p_{initial}2^{-(B-1)} \tag{1.21}$$

Now, for a given set of $w_i\,(i = 0, 1, ..., N-1)$, the terms p_{B-1-j}'s would take one out of 2^N combinations, half of which would be the mirror image of other half [53]. Hence a 2^{N-1} sized ROM can be used the address of which can be obtained through the Ex-OR operation of all the LSB's with the LSB of the newest sample as shown in Fig. 1.10.

1.5 LITERATURE ON DISTRIBUTED ARITHMETIC BASED IMPLEMENTATIONS

The study on the development and the feasibility of DA technique in the realization of digital signal processing (DSP) algorithms takes us back into early 1950s. During 1950s, the first digital computers were introduced and the progress in their development has

influenced the progress in the field of digital signal processing. Since then, the development of DSP algorithms and their implementations have been influenced by demands of the applications, the progress in the integrated circuit (IC) technology and the available level of integration. DSP algorithms are typically characterized by computation intensive algorithms, primarily multiply-add operations. Some of the advanced algorithms, make use of division, square-root, rotation and trigonometric operations. The early computers were equipped with adders and as a result all the arithmetic operations including multiplication and division were implemented using multiply (controlled) and add (or add-subtract) operations. This motivated the emphasis to reduce the number of multiplication operations in DSP algorithms as they take up considerable amount of space.

Later in 1970s, the development of general-purpose microprocessors which operate at high speeds led to design of faster signal processing systems. During those decades, several efficient architectures such as co-ordinate rotation digital computer (CORDIC) and distributed arithmetic (DA) White [53] have been proposed for the implementation of DSP algorithms. While CORDIC is primarily is used for implementation of trigonometric functions and rotation operations, DA is used to implement the sum-of-products especially in custom implementations where the performance/cost ratio is critical. DA was first introduced by Croisier [16] in 1973 and further developed by Peled and Liu [37]. From then, DA was successfully applied in applications such as the telephone system at Bell Laboratories, air-to-ground missile digital autopilot, and few other communication system applications [53]. Since most of the DSP algorithms like convolution, correlation etc are principally the sum-of-product operations, the early works based on DA mostly included finite & infinite impulse-response filters [4, 57, 50, 51] and rotation operations rotation operations [49, 10] and few adaptive filters [15, 48, 52]. The disadvantage with the early DA architecture is that it requires single large memory in order to compute inner product of vectors whose lengths are very large. In [13], the authors proposed a technique for reduction of size of the ROM thereby making DA much useful in the implementation of large inner products. Later, DA has been successfully used in the implementation of other DSP algorithms such as cyclic convolution [11], discrete cosine transform (DCT) [56] and Fourier Transform [12, 40]. Recent works based DA once again include transformations and adaptive digital filters [3, 54, 21, 41, 32].

As explained earlier, a fixed coefficient filter can be easily realized using DA by storing the partial-products of filter coefficients in the LUT. But, the DA treatment to adaptive filters faces certain difficulties, not encountered in the fixed coefficient case, namely the two principle operations-filtering and weight updating are mutually coupled, thus, the partial-products of filter coefficients stored in the LUTs are to be recalculated before the filtering operation. The early attempts on DA based adaptive filters used approximations to standard adaptation algorithms which degrade the performance [15, 48, 52]. Allred et al., proposed a new method by maintaining an auxiliary LUT that serves the adaptation process apart from the LUT demanded by DA [2, 1, 3]. The authors showed that by proper choice of system parameters, the throughput is a constant regardless of the filter order. In the recent past, Guo et al., [22, 21] proposed two schemes in which only one LUT is maintained that stores the partial-products of input samples. Considerable hardware savings was achieved by using the bits of filter coefficients as the address bits to the LUT. However, in this the weight adaptation using the partial-products of input samples alone is a complicated process and account for much computational complexity since the partial products of filter coefficients are not stored. A block LMS based adaptive filter has been

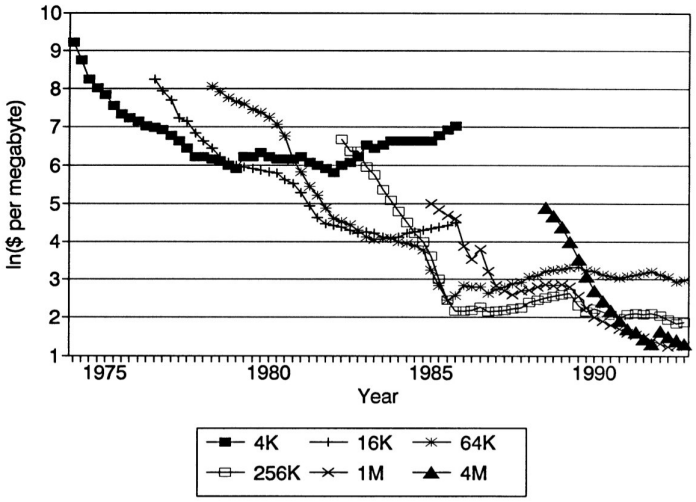

Figure 1.11: Timeline of the pricing of dynamic random-access memory (RAM). (Source: [26])

proposed in [32] where a novel LUT sharing technique has been proposed for handling the filtering and weight adaptation operations. The approach offers good computational savings as the number of computational units (adders) does not increase proportionately with respect to the block length. With semi-conductor memories becoming more cheaper and much smaller (Fig. 1.11 and 1.12), due to the feasibility of DA technique in realizing higher order filters along with the flexibility of configuring DA structure for speed and complexity requirements, we feel that DA is an appropriate technique in the realization of high speed structures for ADFEs.

1.6 PROBLEM FORMULATION

As described earlier, system complexity and difficulty in fast processing are the main problems in the implementation of equalizers. In case of high data rate applications, the complexity of the equalizers increase due to the requirement of large number of taps in the filters. As the complexity increases power and chip area requirement also go up. All these can make the real-time operation of equalizers difficult. Distributed arithmetic is a preferable method for the implementation of ADFEs since it eliminates the requirement of hardware multipliers and is capable of implementing large order filters with high throughput. Further, the recent trends in semi-conductor memories also make DA more suitable for realization of ADFE architectures on FPGA and ASIC platforms. However, direct implementation of DA adaptive filters on sample by sample basis may entirely eliminate the efficiency advantages of DA realization and complexity reduction of ADFEs based on DA realizations have not been studied well so far in the literature. This motivates us to exploit possible benefits of DA in the realization of equalizer structures.

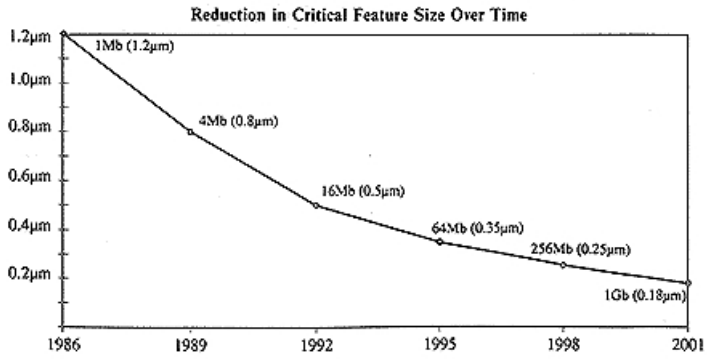

Figure 1.12: Graph showing the increments of gains in memory capacities over the years. (Source: Cymer Inc.)

1.7 ORGANIZATION OF THE THESIS

Based on the background above, the salient contributions made by this thesis are presented as follows:

Chapter 1

This introductory chapter provides a brief description of channel equalization used in the time-varying wireless environment. This chapter also provides the basic motivation behind the work, major research activities in complexity reduction of high speed ADFE over the last decades. The need for the DA treatment for the ADFE realization and the fundamental issues and challenges are also presented in Chapter 1. This chapter ends by summarizing the overview of its contents and the contributions of the thesis.

Chapter 2

Chapter 2 presents an efficient DA based implementation of the LMS adaptive filter. The purpose here is to develop a framework for use later for the realization of LMS ADFE and block ADFE. Although, an LMS adaptive filter based on DA implementation has already been studied well in the literature [15, 52, 3, 21], there was a possibility of further optimizing the architecture for area and speed. So, in this chapter, we exploited the use of DA-OBC scheme to store the partial-products of filter weights to gain efficiency advantages in area and speed. Further, the idea of maintaining an auxiliary-LUT for storing the partial-products of most recent input samples as given in Allred et al. [3] has been utilized. Although, the weight-update operation is not a difficult task when the similar partial-products of filter weights and recent input samples are available, the basic challenge lies in updating the auxiliary-LUT in every iteration i.e., eliminating the oldest input sample and maintaining the analogy of the OBC combinations of the recent input samples. The idea in this Chapter differs from the idea of Allred et al. [3] in that Allred et al. [3] uses the partial-products in direct DA where as the present work includes OBC-DA. Hence, the partial-products inside the look-up-tables contain more complex combinations of the filter

weights. This is achieved by manipulation of the addressing bits of the auxiliary-LUT and it is shown how the oldest input sample can be smartly eliminated in every iteration.

Chapter 3

In Chapter 3, the DA treatment for the efficient realization of non-adaptive DFE has been carried out. The purpose here is to explore the benefits of DA for complexity reduction and speed improvements of DFEs, the experience of which allows us to take up the realization of ADFE and block ADFE later. In order to compare the DA based DFE with that of traditional MAC based one, the throughput of both have been made equal. For this purpose, the digit-serial nature of DA has been utilized and the basic filtering equations of FFF and FBF have been recast to realize them using DA. A direct-memory architecture has been developed first where the lower-half entries of the memories storing partial-products are mirror image to the upper-half. Based on this, a reduced-memory architecture has been developed later where one half of the entries can be generated live using the other half. The approaches presented here are different from the traditional MAC based approach in that the MAC based approach uses the bulky multipliers which are responsible for high computational cost and processing delays, while the proposed approach replaces the multipliers with look-up-tables. This facilitates ease of implementing the design on FPGAs and ASICs too, taking into consideration the advances in the semi-conductor memory technologies in the last few decades. The proposed non-adaptive DFE with different configurations has been implemented on FPGA platform and it was observed that the DA based non-adaptive DFE outperforms the MAC based non-adaptive DFE both in terms of computational complexity and speed.

Chapter 4

In Chapter 4, using the framework of DA based LMS adaptive filter in Chapter 2 and non-adaptive DFE in Chapter 3, we take up the DA based realization of LMS ADFE. Majority of the works so far in the direction of complexity reduction of high speed ADFEs mainly adopted parallelism and pipelining. Parallelism can be a good choice if the filter orders in the ADFE are small and such a realization may become extremely complex for large filter orders. Piplening ADFE is not so easy task due the presence of the feedback loop and the non-linear device (quantizer) in the ADFE. However, using the look-ahead techniques, few attempts have been made in order to realize ADFEs. Most of these techniques either involve approximations to the standard algorithms involved in the ADFE which can result in performance loss or they may become more complex for filters of higher order. Hence, our first attempt in the DA treatment was to simplify the final set of equations governing the operation of ADFE into the form of sum of product of two vectors. However, this approach requires a very large memory even for small number of taps in FFF and FBF and the approach proved to be impractical. Hence, we used the idea presented in Chapter 2 and carry out DA treatment for the filtering and weight-update operations of FFF and FBF separately.

Chapter 5

Having considered the DA treatment of ADFE in Chapters 3 and extending the DA treatment to LMS algorithm provided in Chapter 2 to block LMS algorithm, we take up the realization of block ADFE in Chapter 5. Note that, the block processing of ADFE is not direct, since, the block processing of FFF is possible as the processing is done on received data, which are known a priori, the same is not the case with FBF as the processing is done on decision outputs, which are unknowns in every iteration and needs to be sought out by the ADFE. The most efficient approach of addressing this issue is to use tentative decisions in place of unknown ones in every iteration and then using an iterative procedure, replacing these tentative decisions with the correct set of decisions. Based on the fact that, the unknown decisions in every iteration takes one out of all the possible symbols used in the modulation scheme at the transmitter, we proposed an efficient approach to determine the unknown decisions. In this approach, the unknown decisions are computed by using two minimization criteria namely mean-absolute difference (MAD) and mean-square difference (MSD) and by storing all the symbol values (used at the transmitter) in a bank of registers.

Finally, the thesis is concluded by summing up the salient features of all the proposed schemes along with the suggestions for future work are presented in Chapter 6.

2

HIGH PERFORMANCE ARCHITECTURE FOR AN LMS BASED ADAPTIVE FILTER USING DISTRIBUTED ARITHMETIC

2.1 INTRODUCTION

Adaptive filters are widely used in many signal processing applications such as system identification and modelling, equalization, interference and echo cancellation etc [23]. Such filters are generally made of finite-impulse-response (FIR) filters whose coefficients are updated as per a minimization criteria. The output of an FIR filter is the weighted sum of present and past input samples and hence they can be realized using MAC units as is the case in DSP processors. If N is the number of filter taps, a single MAC unit would take $N-1$ clock cycles to compute one sample of output sequence. Multiple MAC units can be utilized in order to increase the speed of the system but the system cost goes up because the multipliers consume much area. Distributed Arithmetic (DA) discussed in Chapter 1, would be a suitable technique for the realization of higher order adaptive filters as it can achieve high throughputs without involving a hardware multiplier. Towards this objective, a high performance implementation scheme for least mean square adaptive filter is presented in this chapter. The purpose here is to develop a basic framework for use later (Chapter 4) for the realization of LMS based adaptive decision feedback equalizer.

A fixed coefficient filter can be easily realized using DA by storing the partial-products of filter coefficients in the LUT as discussed in Chapter 1. But, the DA treatment to adaptive filters faces certain difficulties, not encountered in the fixed coefficient case, namely the two principle operations-filtering and weight updating are mutually coupled, thus, the partial-products of filter coefficients stored in the LUTs are to be recalculated before the filtering operation. However, few attempts have been made to realize adaptive filters using DA by approximations to standard adaptation algorithms which degrades the performance [15, 52, 48]. Anderson et al., proposed a new method by maintaining an auxiliary LUT that serves the adaptation process apart from the LUT demanded by DA [2, 3]. They showed that by proper choice of system parameters, the throughput is a constant regardless of the filter order. In the recent past, Guo and DeBrunner [21, 22] proposed two schemes in which only one LUT is maintained that stores the partial-products of input samples. Considerable hardware savings was achieved by using the bits of filter coefficients as the address bits to the LUT. However, weight adaptation using the partial-products of input samples alone is a complicated process and account for much computational complexity since the partial products of filter coefficients are not stored. One approach to improve the speed of the system is to store the filter weights in registers and generating the partial products using AND gates and adder tree as shown in Fig. 2.1 and 2.2 where the partial-products can be generated alive instead of pre-computing and

storing them in memory. However, such a system would lack a mechanism for controlling the throughput and the system would be too complex for filters of very high order.

Hence, maintaining an auxiliary LUT is the best approach for the weight update mechanism in the realization of LMS adaptive filter using DA. However, the new challenge would be to update this auxiliary-LUT that stores the partial-products of most recent input samples. This is because, the oldest sample in one iteration of an LMS adaptive filter is not used in the successive iteration as shown in Fig. 2.3 and hence the auxiliary-LUT has to be updated from time to time for the elimination of the oldest sample and the incorporation of the newest sample. Although, Anderson et al., were successful in the realization of this, there is still some scope in further optimization of the DA-LMS adaptive filter. In the following sections, based on the OBC scheme, we propose a new weight adaptation strategy that accounts for low computational cost and high system throughput.

2.2 HIGH PERFORMANCE LMS ADAPTIVE FILTER ARCHITECTURE USING DA

Consider an LMS based adaptive filter as shown in Fig. 2.4 that processes an input sequence $x(n)$, $(n \in Z)$ and generates the output sequence $y(n)$ as per the following:

$$y(n) = \mathbf{w}^T \mathbf{x} \tag{2.1}$$

where $\mathbf{w}^T = [w_0(n), w_1(n), \ldots, w_{N-1}(n)]$ is the filter's tap-weight vector, $\mathbf{x}^T = [x(n), x(n-1), \ldots, x(n-N+1)]$ is the input sample vector and N being the number of filter coefficients. The filter weights are updated using:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x} \tag{2.2}$$

where $e(n) = d(n) - y(n)$ is the error signal and μ is an appropriate step size which is to be chosen as $0 < \mu < \frac{2}{[P\,tr\mathbf{R}]}$ for convergence of the algorithm ($\mathbf{R}$ is the auto-correlation matrix).

The block schematic of the distributed arithmetic based architecture for efficient implementation of the filtering and weight update operations of LMS adaptive filter as given by (2.1) and (2.2) is shown in Fig. 2.5. It consists of a register bank to store the incoming input samples, a primary-LUT (P-LUT) that stores the combinations of weights which is responsible for the DA filtering operation in every iteration and a secondary-LUT (S-LUT) that stores the combinations of input samples. It also consists of a register R_0 which along with S-LUT aids the weight-adaptation process, a shift and accumulate block of the DA-filtering operation and a combinational logic block that takes care of the weight adaptation process.

While the P-LUT stores the OBC combinations of the filter weights, the S-LUT stores the OBC combinations of input samples except with the term containing the most recent

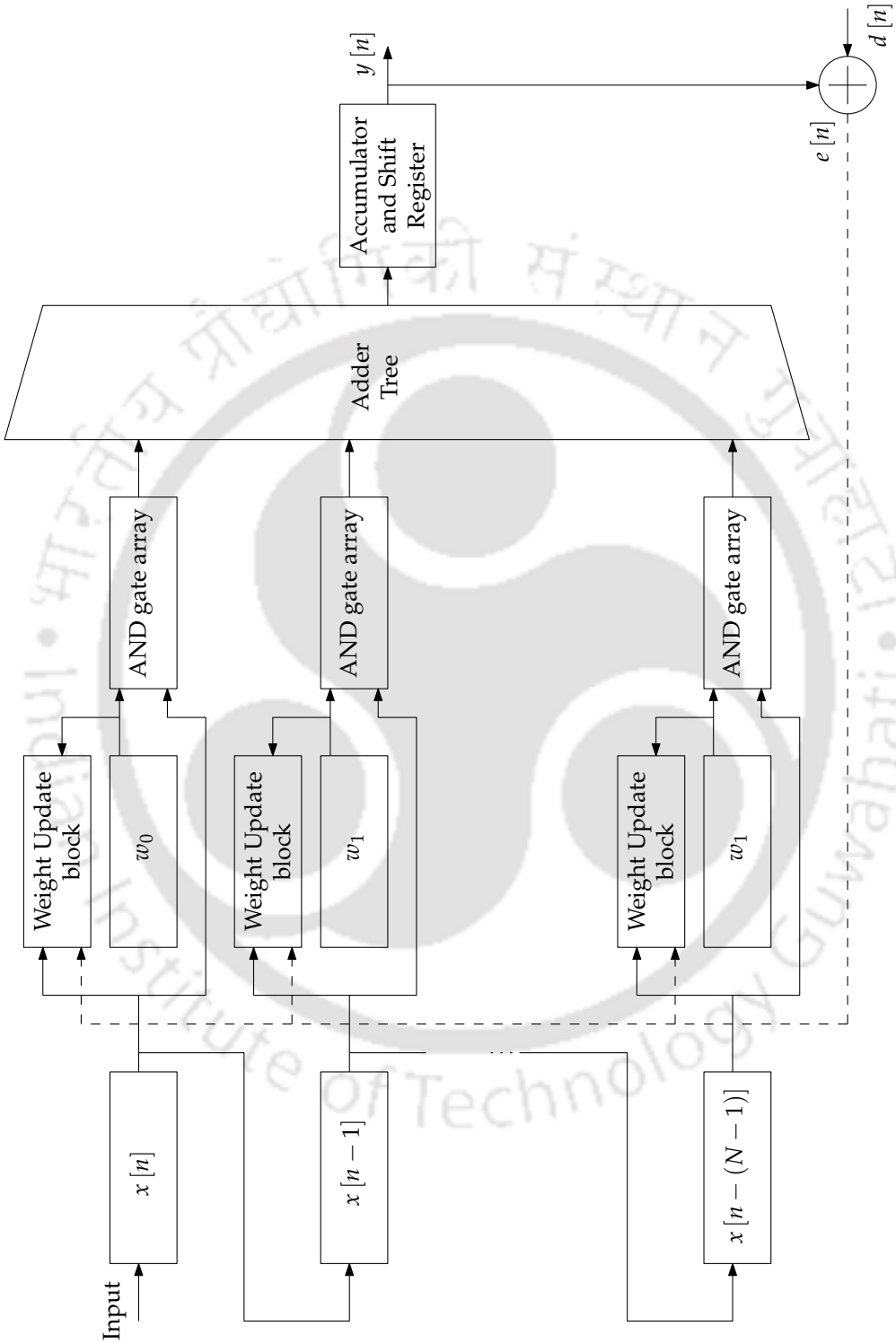

Figure 2.1: Block diagram of DA based LMS adaptive filter architecture that uses parallel weight updating blocks.

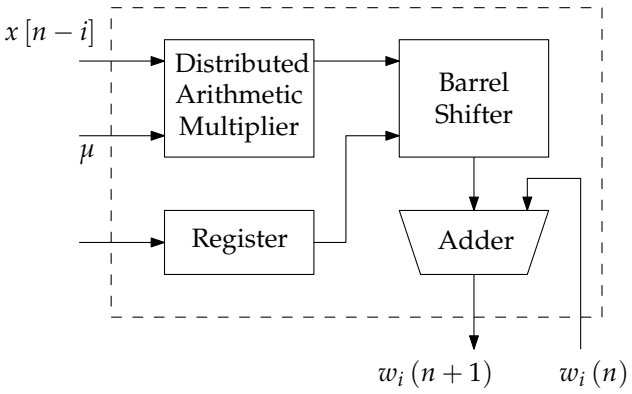

Figure 2.2: The internal structure of weight-update module.

input sample which is stored in register R_0. The entry of P-LUT at time n addressed by a can be mathematically expressed as

$$P_{(a)}(n) = \frac{1}{2}w_0(n) + \sum_{k=1}^{N-1}\frac{1}{2}w_k(n)(-1)^{q_{N-1-k}^{(a)}+1} = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{w} \qquad a = 0, 1, \ldots, 2^{N/2}-1 \tag{2.3}$$

where $\mathbf{w}^{\mathrm{T}} = [w_0(n), w_1(n), \ldots, w_{N-1}(n)]$, $\mathbf{a}^{\mathrm{T}} = [1, (-1)^{q_{N-2}^{(a)}+1}, \ldots, (-1)^{q_0^{(a)}+1}]$ and $q_l^{(a)}$ is the lth bit in the N-bit representation ($q_{N-1}^{(a)} = 0$) of address a. That is

$$a = \sum_{s=0}^{N-2} q_s^{(a)} 2^s \tag{2.4}$$

If the vector $\mathbf{w}$ in (2.3) is replaced by the vector $\mathbf{x}$ which is given as $\mathbf{x}^{\mathrm{T}} = [x(n), x(n-1), \ldots, x(n-N+1)]$, then a new look-up-table $T(n)$ is formed with its entry at address location a given as

$$T_{(a)}(n) = \frac{1}{2}x(n) + \sum_{k=1}^{N-1}\frac{1}{2}x(n-k)(-1)^{q_{N-1-k}^{(a)}+1} \qquad a = 0, 1, \ldots, 2^{N/2}-1 \tag{2.5}$$

$$T_{(a)}(n) = R_0(n) + S_{(a)}(n) \tag{2.6}$$

where $R_0(n)$ is the contents of register R_0 at time n and $S_{(a)}(n)$ is the entry of the S-LUT at time n addressed by a

$$R_0(n) = \frac{1}{2}x(n) \tag{2.7}$$

$$S_{(a)}(n) = \sum_{k=1}^{N-1}\frac{1}{2}x(n-k)(-1)^{q_{N-1-k}^{(a)}+1} \tag{2.8}$$

Fig. 2.6 shows the contents of P-LUT and S-LUT at time instant n for a 4-tap filter. It can be observed that S-LUT stores the OBC combinations (lower half) of input samples

Iteration 1	Iteration 2
$w_0(n+1) = w_0(n) + \mu e(n)x(n)$	$w_0(n+2) = w_0(n+1) + \mu e(n+1)x(n+1)$
$w_1(n+1) = w_1(n) + \mu e(n)x(n-1)$	$w_1(n+2) = w_1(n+1) + \mu e(n+1)x(n)$
$w_2(n+1) = w_2(n) + \mu e(n)x(n-2)$	$w_2(n+2) = w_2(n+1) + \mu e(n+1)x(n-1)$
$w_3(n+1) = w_3(n) + \mu e(n)\boxed{x(n-3)}$	$w_3(n+2) = w_3(n+1) + \mu e(n+1)\boxed{x(n-2)}$
Iteration 3	**Iteration 4**
$w_0(n+3) = w_0(n+2) + \mu e(n+2)x(n+2)$	$w_0(n+4) = w_0(n+3) + \mu e(n+3)x(n+3)$
$w_1(n+3) = w_1(n+2) + \mu e(n+2)x(n+1)$	$w_1(n+4) = w_1(n+3) + \mu e(n+3)x(n+2)$
$w_2(n+3) = w_2(n+2) + \mu e(n+2)x(n)$	$w_2(n+4) = w_2(n+3) + \mu e(n+3)x(n+1)$
$w_3(n+3) = w_3(n+2) + \mu e(n+2)\boxed{x(n-1)}$	$w_3(n+4) = w_3(n+3) + \mu e(n+3)\boxed{x(n)}$

Figure 2.3: Equations describing the weight-update operations of a 4-tap LMS adaptive filter in successive iterations.

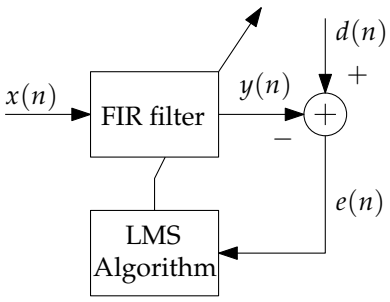

Figure 2.4: Block diagram of an LMS based adaptive filter.

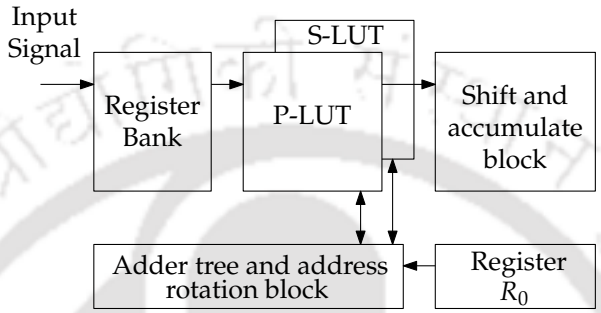

Figure 2.5: Architecture for LMS adaptive filter based on DA-OBC scheme.

except with the most recent input sample $x(n)$ which can be stored in a special register R_0. The benefit of storing the term containing the most recent input sample in register R_0 is explained in the following paragraphs.

The S-LUT update scheme in the DA based implementation from time n to $n+1$ is explained as follows. Averaging the S-LUT entry with its next consecutive entry would generate a term that is independent of the oldest input sample. The content of register R_0 is then subtracted and added with the result. For example, when $N = 4$, the average of the first and second entries of S-LUT would give $1/2[-x(n-1) - x(n-2)]$ which is independent of the term $x[n-3]$. Now subtraction and addition of the term $1/2[x(n)]$ with the result will generate the terms $1/2[-x(n) - x(n-1) - x(n-2)]$ and $1/2[x(n) - x(n-1) - x(n-2)]$ respectively which are then stored in the same consecutive locations of S-LUT. Similarly, the third and fourth locations of S-LUT are updated by subtracting and adding the term $1/2[x(n)]$ to their average and storing the difference and sum in exactly the same locations. In the similar manner, all the entries of S-LUT are updated using the sum and difference with the term $1/2[x(n)]$. Mathematically, the new entries $S_i(n+1)$ of S-LUT can be obtained from the old entries $S_i(n)$ with the index entry $i \in [0, 2^{N-1} - 1]$ as follows:

$$S_i(n+1) = (-1)^{i+1} R_0 + \frac{1}{2}\{S_{2\lfloor \frac{i}{2} \rfloor}(n) + S_{2\lfloor \frac{i}{2} \rfloor + 1}(n)\} \tag{2.9}$$

where $R_0(n)$ is the entry of register R_0 at time n.

When the new input sample $x(n+1)$ is arrived, the right shifted version of it that is the term $1/2[x(n+1)]$ is stored in the register R_0 which is useful for weight adaptation at time $n+1$. The S-LUT update scheme from time n to $n+1$ for a 4-tap filter is shown in Fig. 2.7 where the positive and negative terms of the input samples are represented by '1' and '0' respectively . It can be observed that at time $n+1$, there are enough contents in

Primary LUT

Address	LUT Contents
0 0 0	$1/2[w_0(n) - w_1(n) - w_2(n) - w_3(n)]$
0 0 1	$1/2[w_0(n) - w_1(n) - w_2(n) + w_3(n)]$
0 1 0	$1/2[w_0(n) - w_1(n) + w_2(n) - w_3(n)]$
0 1 1	$1/2[w_0(n) - w_1(n) + w_2(n) + w_3(n)]$
1 0 0	$1/2[w_0(n) + w_1(n) - w_2(n) - w_3(n)]$
1 0 1	$1/2[w_0(n) + w_1(n) - w_2(n) + w_3(n)]$
1 1 0	$1/2[w_0(n) + w_1(n) + w_2(n) - w_3(n)]$
1 1 1	$1/2[w_0(n) + w_1(n) + w_2(n) + w_3(n)]$

Secondary LUT

Address	LUT Contents
0 0 0	$1/2[-x(n-1) - x(n-2) - x(n-3)]$
0 0 1	$1/2[-x(n-1) - x(n-2) + x(n-3)]$
0 1 0	$1/2[-x(n-1) + x(n-2) - x(n-3)]$
0 1 1	$1/2[-x(n-1) + x(n-2) + x(n-3)]$
1 0 0	$1/2[+x(n-1) - x(n-2) - x(n-3)]$
1 0 1	$1/2[+x(n-1) - x(n-2) + x(n-3)]$
1 1 0	$1/2[+x(n-1) + x(n-2) - x(n-3)]$
1 1 1	$1/2[+x(n-1) + x(n-2) + x(n-3)]$

Figure 2.6: Contents of Primary-LUT and Secondary-LUT at time n for a 4-tap filter in the DA based implementation.

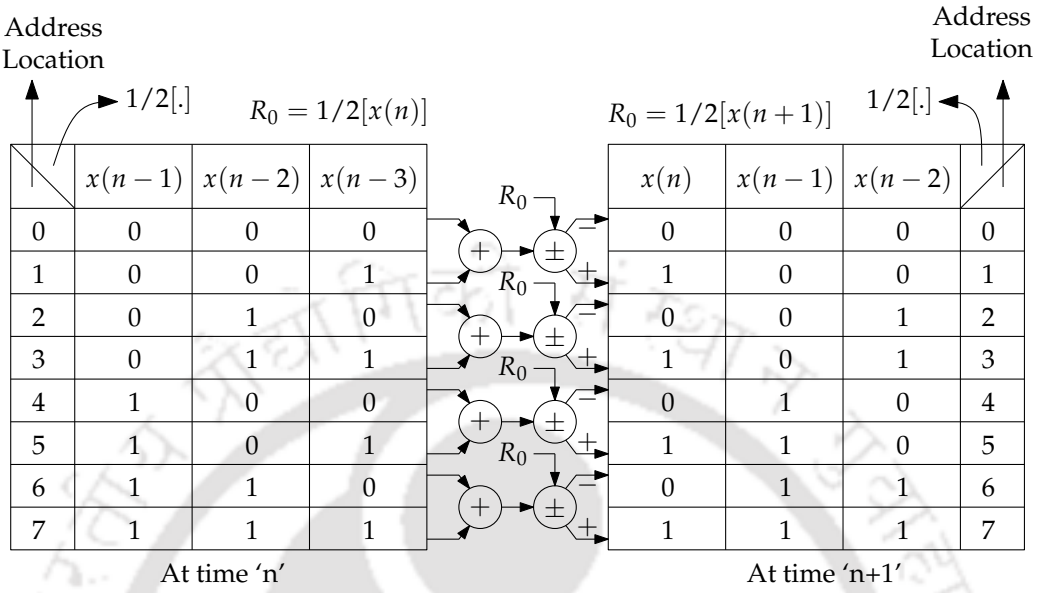

Figure 2.7: S-LUT update scheme from time n to $n+1$ for a 4-tap filter.

Address Location	At time 'n'			At time 'n+1'			At time 'n+2'			At time 'n+3'		
	x_{n-1}	x_{n-2}	x_{n-3}	x_n	x_{n-1}	x_{n-2}	x_{n+1}	x_n	x_{n-1}	x_{n+2}	x_{n+1}	x_n
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	1	0	0	0	1
2	0	1	0	0	0	1	1	0	0	0	1	0
3	0	1	1	1	0	1	1	1	0	0	1	1
4	1	0	0	0	1	0	0	0	1	1	0	0
5	1	0	1	1	1	0	0	1	1	1	0	1
6	1	1	0	0	1	1	1	0	1	1	1	0
7	1	1	1	1	1	1	1	1	1	1	1	1

Figure 2.8: Contents of S-LUT for a 4-tap filter at various time instants.

1: **initial** $a' = a$
2: **loop**
$y[n] \leftarrow \sum_{j=0}^{B-1} p_{B-1-j} 2^{-j} + p_{initial} 2^{-(B-1)}$
for $a = 0 : 2^{N-1} - 1$ **do**
$S_a(n) \leftarrow (-1)^{a+1} R_0(n-1) + \frac{1}{2}\{S_{2\lfloor \frac{a}{2} \rfloor}(n-1) + S_{2\lfloor \frac{a}{2} \rfloor + 1}(n-1)\}$
end for
3: $R_0(n) \leftarrow \frac{1}{2}x[n]; e[n] \leftarrow d[n] - y[n]$
4: $a' = cirleftshift(a, N)$
5: **for** $a = 0 : 2^{N-1} - 1$ **do**
$P_a(n+1) \leftarrow P_a(n) + ue[n]\{R_0(n) + S_{a'}(n)\}$
end for
$p_{initial} = -P_{2^{(N/2)}-1}(n+1)$
6: $n \leftarrow n + 1$
7: **end loop**

Figure 2.9: Algorithm explaining the operation of the DA based LMS adaptive filter.

the S-LUT but their locations are not in proper order for the weight adaptation. By close observation it can be seen that the contents of S-LUT are placed in the bit-circularly right shifted address locations. For example, consider the data (at time $n+1$) at the address location one (001) given as $1/2[x(n) - x(n-1) - x(n-2)]$ which is supposed to be in the address location four (100) - circularly right shifted version of address bits of location one (001). Similarly, consider the data at address location five (101) given as $1/2[x(n) + x(n-1) - x(n-2)]$ which is supposed to be in the address location six (110) nothing but the circularly right shifted version of address bits of location five (101).

If we represent '1' and '0' as the positive and negative terms in the OBC combinations of input samples, at time n, the contents of the S-LUT would look like the binary sequence of the input samples as shown in the Fig. 2.8. At time $n+1$, the contents would be circularly right shifted version of that binary sequence. At time $n+2$, it would be circularly right shifted version of the sequence at time $n+1$ and so on. Hence for accessing the entries of S-LUT in each iteration, instead of physically moving the contents, the address bits to the S-LUT are circularly left-shifted. At time $n+3$, the sequence once again would be a normal binary sequence as shown in Fig. 2.8. Accessing time can be reduced by maintaining two smaller S-LUTs namely ODD-LUT and EVEN-LUT as shown in Fig. 2.11 where the even location entries and odd location entries are stored respectively. $\mu e[n]$ can be quantized to the powers of 2 and hence the multiplication of $\mu e[n]$ with the term $x[n-i]$ can be performed using the shift operation as in [3]. $p_{initial}$ can be stored in a register and can be updated in every iteration easily since the term $p_{initial}$ (refer (Equation 1.20)) is nothing but the additive inverse of the entry of P-LUT at its last address location. The algorithm explaining the overall operation of the DA based LMS adaptive filter is shown in Fig. 2.9. The error curves for the DA based and MAC based implementations for a 4-tap filter are shown in Fig. 2.10.

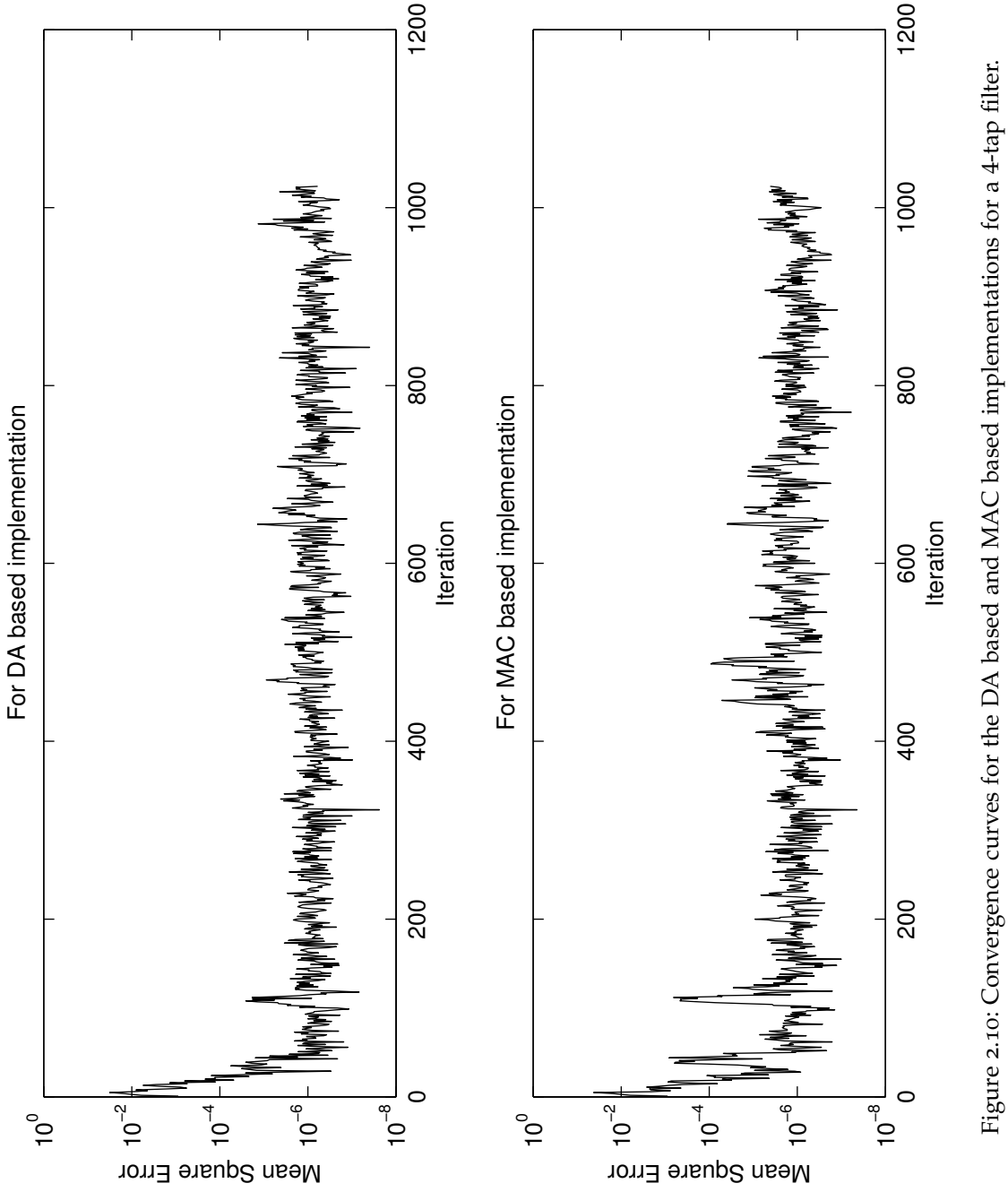

Figure 2.10: Convergence curves for the DA based and MAC based implementations for a 4-tap filter.

2.3 PERFORMANCE ANALYSIS

2.3.1 *Area*

In order to verify the validity of the DA based adaptive filter implementation scheme, a 4-tap FIR filter has been implemented on Xilinx® Spartan3E[1] XC3S500E FPGA. We refer to the scheme in [3] as DA_0 scheme and the first and second proposed schemes in [21] as $DA_1$and DA_2 schemes respectively. Assuming N to be composite, the filter can be split into m filters each having k-tap DA base units $(N = m \times k)$. The memory requirements for DA_0, DA_1, DA_2 are $(2^{m+1} - 2)(k/m)$, $(2^m - 1)(k/m)$ and $(2^{m-1})(k/m)$ respectively [21] and for the presented scheme, it is same as DA_1 scheme. Synthesis results using $180nm$ standard cell library of the DA based adaptive filter implementation scheme along with the existing schemes is shown in Fig. 2.12. The schemes DA_1 and DA_2 although perform the filtering operation using DA, the weight adaptation scheme is similar to that of the conventional adaptive filter implementation. So, despite the fact that they have a low memory requirement, the weight adaptation blocks account for much hardware complexity. Apart from the weight adaptation blocks, extra storage registers and adder units are required for the pre-computation of certain terms in case of DA_2 although it has the lowest memory requirement of all the schemes. The presented DA based architecture utilizes very less chip area compared to DA_0 scheme which stems from the fact that it demands half the memory size than that of DA_0 scheme and utilizes less combinational logic. The presented scheme outperforms all the other schemes and is more advantageous for large filters as can be seen in Fig. 2.12.

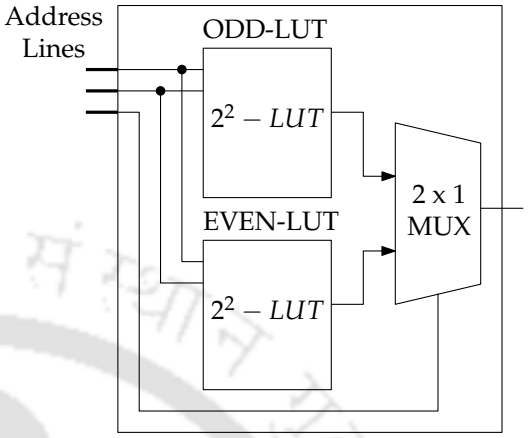

Figure 2.11: Two LUTs storing the odd and even addressed location entries of S-LUT.

2.3.2 *Throughput*

The throughput[2] comparison curves for the DA based implementation scheme and existing schemes are shown in Fig. 2.13. The curves are obtained by choosing k=2, 4, 8 and setting the system clock at 100MHz. It is obvious that the throughput of all these schemes depend on the size of LUTs since updating them takes the longest time in the entire system processing time. In the presented scheme, the filtering operation is done in parallel with updating the S-LUT which takes $max(B, 2^{\frac{k}{2}-1})$ clock cycles. Error $e[n]$ is calculated alongside updating the register R_0 which takes one clock cycle. The update of P-LUT takes $2^{\frac{k}{2}}$ clock cycles during which $p_{initial}$ is also updated. The adder tree would take $\lceil log_2(m) \rceil$ clock cycles and hence the presented filter would take a total of $max(B, 2^{\frac{k}{2}-1}) + 2^{\frac{k}{2}} + \lceil log_2(m) \rceil + 1$ clock cycles for one complete filtering and weight update operation. While the schemes DA_1 and DA_2 suffer from severe throughput bottle-

1 The details can be found at www.xilinx.com/support/documentation/data_sheets/ds312.pdf
2 Appendix A

neck, the DA based architecture has the same throughput as that of DA_o for $k = 2$. However, it is observed that for large values of k, the DA based implementation is more advantageous. For instance, when $k = 4$, the presented scheme produces two times more throughput than that of DA_o and almost twelve times more when $k = 8$. Further, it is observed that the throughput is constant regardless of the base filter size k as is the case in DA_o scheme.

2.4 CONCLUSION

In this Chapter, an FIR adaptive filter implementation based on distributed arithmetic is presented. OBC scheme has been employed in order to reduce the memory requirement. Unlike the recent existing schemes, the presented DA based implementation scheme uses a secondary LUT that aids in the weight adaptation process. The OBC scheme stores complex combinations of input samples and the oldest input sample is smartly eliminated in order to update the S-LUT. The design utilizes very less chip area and operates at higher throughput rate compared to all the existing architectures. Further, the design can be adopted to filters of any order and is in particular useful for filters of higher order and large base units.

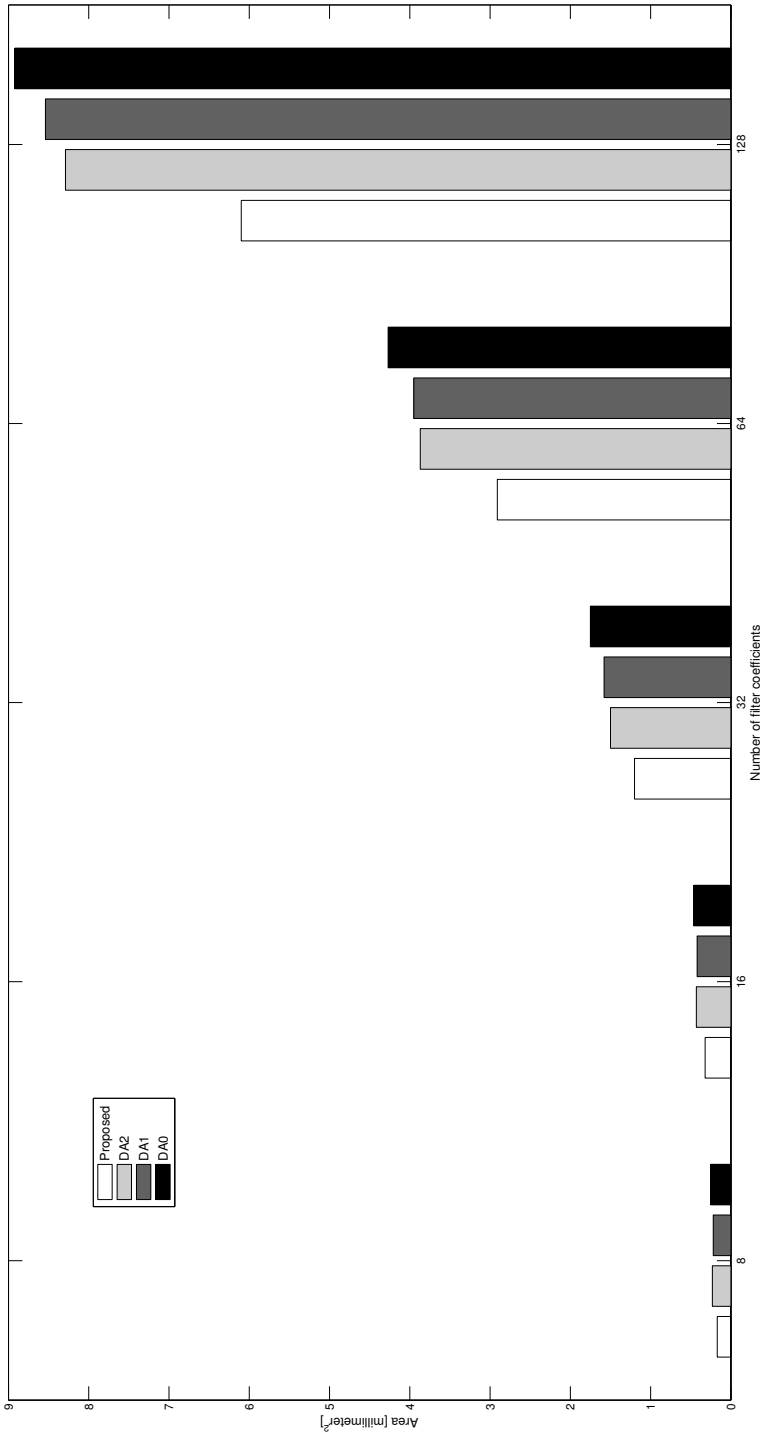

Figure 2.12: Area comparison chart for the presented and existing DA schemes.

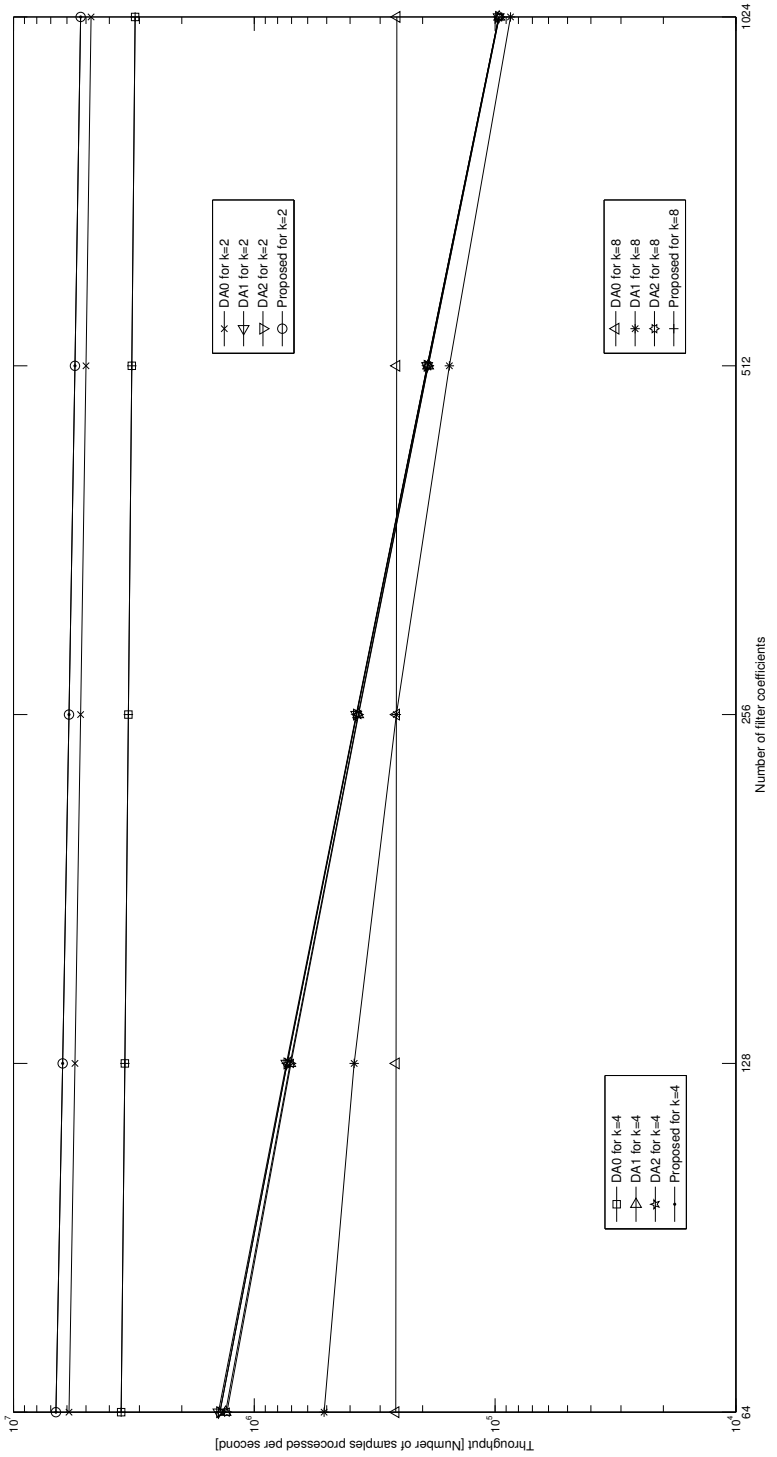

Figure 2.13: Throughput curves for the presented and existing DA schemes for k=2, 4 and 8.

3

DIGIT-SERIAL DA BASED REALIZATION OF THE DECISION FEEDBACK EQUALIZER

3.1 INTRODUCTION

Decision feedback equalizers (DFE) are widely used for effectively equalizing the channels that exhibit nulls in their frequency spectrum . Unlike the linear equalizers, DFEs do not estimate the inverse of the channel directly. A DFE basically consists of a feed forward filter (FFF), a feedback filter (FBF) and a decision device (quantizer). The FFF works directly on the received data and equalizes the pre-cursor part of the intersymbol interference (ISI). The residual post-cursor part of ISI is then cancelled out by subtracting the FBF output from the FFF output. The output of FBF whose coefficients are carefully chosen, operates on the decisions made on the past symbols. The DFE works basically on the assumption that there are no errors at the output of the decision device. As long as the decisions are correct, the DFE can equalize the channel effectively with low noise enhancement. The problem with DFEs is that the sizes of FFF and FBF increase as the transmission data rate increases. This is because of the fact that as the transmission data rate increases, more and more symbols get overlapped which demands for large number of taps for FFF and FBF. Hence, when implementing the DFEs more number of multiply-and-accumulate (MAC) units are to be employed and operated in parallel in order to cope up with the transmission speed. But, due to the presence of large number of multipliers, the system would become complex and the real-time implementation becomes difficult. Distributed Arithmetic as discussed in Chapter 1, can be employed for the realization of DFE since it can realize vectors of any size without the presence of a hardware multiplier. In this Chapter, the DA treatment for the basic non-adaptive decision feedback equalizer has been taken up. The purpose here is to explore the possible benefits of DA in realizing the DFE so as to gain an experience for the realization of adaptive DFE (ADFE) in the later chapters.

3.2 THE DFE ARCHITECTURE BASED ON DIGIT-SERIAL DA

Consider a DFE shown in Fig. 3.1 with 'N_f' number of FFF coefficients and 'N_b' number of FBF coefficients which process the input signal $u(n)$, $(n \in Z)$ and the previous output decisions $s(n)$ respectively. The equations describing the operation of the DFE are given as

$$r_q(n) = Q[r(n)] \tag{3.1}$$

where $Q[.]$ is the quantization operation.

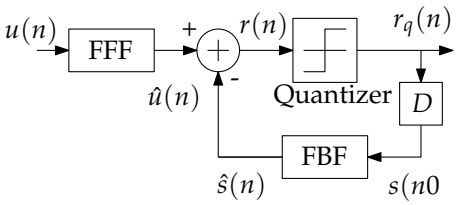

Figure 3.1: General block diagram of a decision feedback equalizer.

$$r(n) = \hat{u}(n) - \hat{s}(n) \tag{3.2}$$

$$s(n) = r_q(n-1) \tag{3.3}$$

and the output of FFF and FBF filters respectively are given as follows:

$$\hat{u}(n) = \sum_{i=0}^{N_f-1} w_i u(n-i) = \mathbf{w}^T\mathbf{u} \tag{3.4}$$

$$\hat{s}(n) = \sum_{i=0}^{N_b-1} v_i s(n-i) = \mathbf{v}^T\mathbf{s} \tag{3.5}$$

where $\mathbf{w}^T = \left[w_0, w_1, \ldots, w_{N_f-1}\right]$, $\mathbf{v}^T = [v_0, v_1, \ldots, v_{N_b-1}]$ are the coefficients of FFF and FBF respectively.

If each of $u(n)$ and $s(n)$ is represented by their signed 2's-complement representation, we have:

$$u(n-i) = -u_{i,B-1} + \sum_{j=1}^{B-1} u_{i,B-1-j}2^{-j} \tag{3.6}$$

$$s(n-i) = -s_{i,B-1} + \sum_{j=1}^{B-1} s_{i,B-1-j}2^{-j} \tag{3.7}$$

Employing the digit-serial arithmetic, using Equation 1.14 we have

$$\begin{aligned}\hat{u}(n) = &\left[-f(w_i, u_{i,0})\,2^0 + \sum_{j=1}^{P_f-1} f\left(w_i, u_{i,P_f-1-j}\right)2^{-j}\right]\left(2^{P_f}\right)^{Q_f-1} \\ &+ \sum_{k=0}^{Q_f-2}\left\{\sum_{j=0}^{P_f-1} f\left(w_i, u_{i,P_f-1-j}\right)2^{-j}\right\}\left(2^{P_f}\right)^{k}\end{aligned} \tag{3.8}$$

where $f\left(w_i, u_{i,P_f-1-j}\right)$ is given by

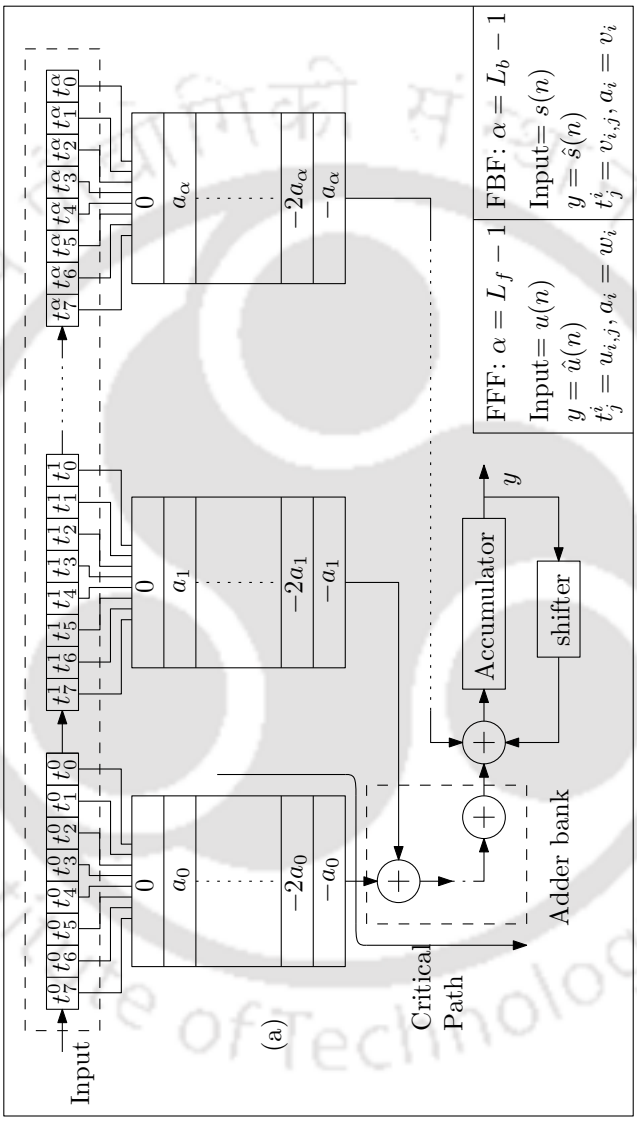

Figure 3.2: The internal structure of direct memory architecture.

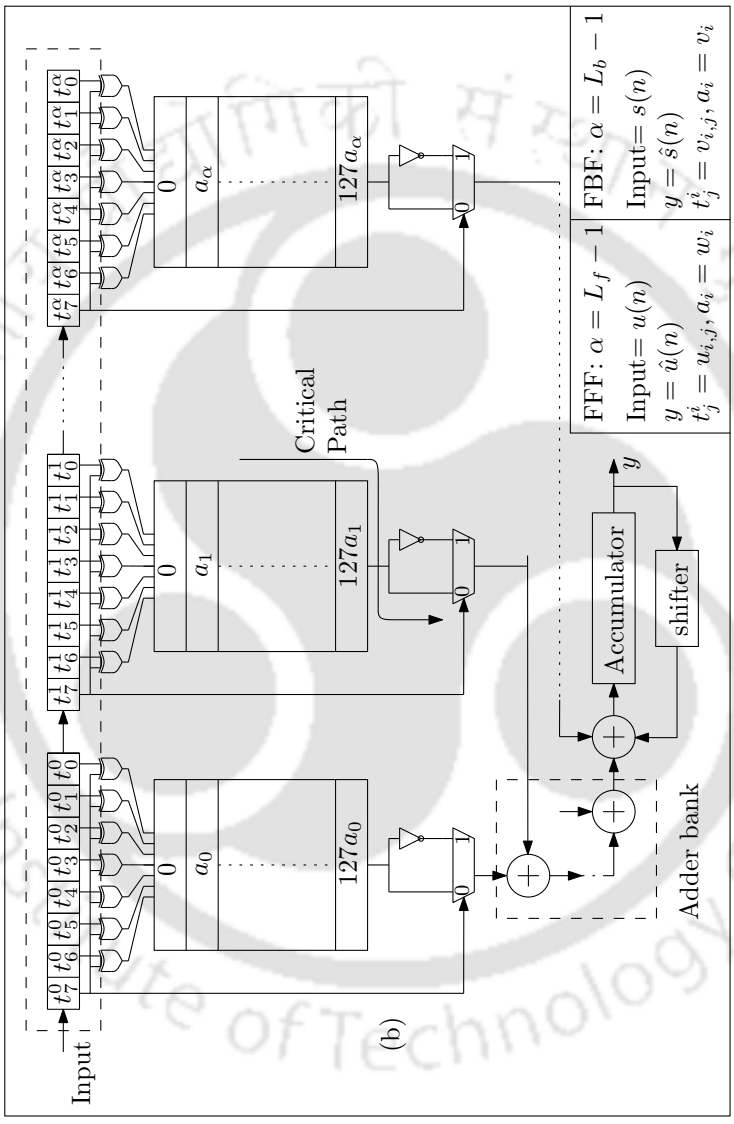

Figure 3.3: The internal structure of reduced memory architecture.

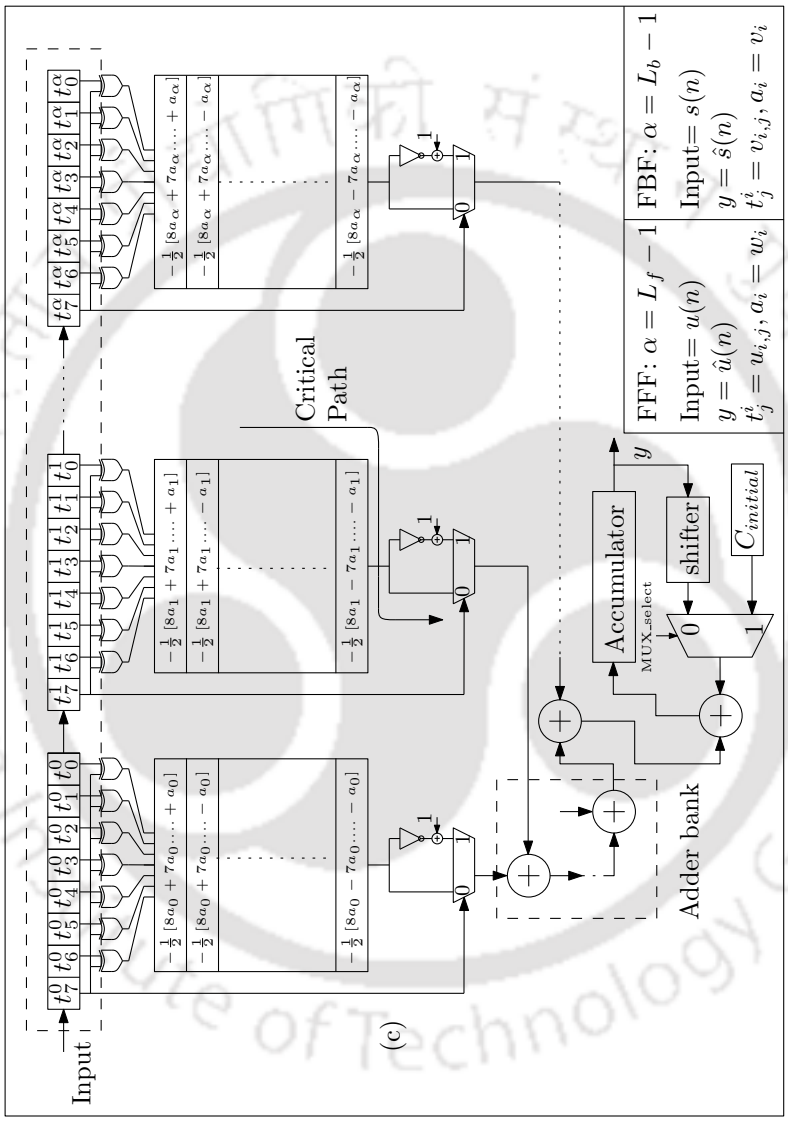

Figure 3.4: The internal structure of OBC memory architecture.

$$f\left(w_i, u_{i,P_f-1-j}\right) = \sum_{i=0}^{N_f-1} w_i u_{i,P_f-1-j} \tag{3.9}$$

Now, if the ROM decomposition technique is also employed, then using Equation 1.10 of chapter 1, we have

$$f\left(w_i, u_{i,P_f-1-j}\right) = \sum_{l=0}^{L_f-1} \sum_{i=lM_f}^{lM_f+M_f-1} w_i u_{i,P_f-1-j} \tag{3.10}$$

Hence, the equations (3.8) & (3.10) describe the realization of FFF using both digit-serial input and ROM decomposition techniques. Similarly, for FBF we can have

$$\begin{aligned}\hat{s}(n) = &\left\{-f\left(v_i, s_{i,0}\right)2^0 + \sum_{j=1}^{P_b-1} f\left(v_i, s_{i,P_b-1-j}\right)2^{-j}\right\}\left(2^{P_b}\right)^{Q_b-1} \\ &+ \sum_{k=0}^{Q_b-2}\left\{\sum_{j=0}^{P_b-1} f\left(v_i, s_{i,P_b-1-j}\right)2^{-j}\right\}\left(2^{P_b}\right)^{k}\end{aligned} \tag{3.11}$$

$$f\left(v_i, s_{i,P_b-1-j}\right) = \sum_{l=0}^{L_b-1} \sum_{i=lM_b}^{lM_b+M_b-1} v_i s_{i,P_b-1-j} \tag{3.12}$$

The internal structure for all the three (direct-memory, reduced-memory and OBC memory) DA based architectures are shown in Fig. 3.2, 3.3 and 3.4 respectively. It can be seen that each of the filters has their own DA block that work on both digit-serial and ROM decomposition techniques. The parameters P_f and P_b decide the number of digits used from each of input register for FFF and FBF respectively. Similarly, the parameters L_f and L_b decide the number of ROMs operated in parallel for FFF and FBF respectively. The filters FFF and FBF can be synchronized by choosing same number of digits from the each of the input register i.e., by choosing $P_f = P_b$. Hence, the output of FFF & FBF can be computed using just the shift and add operations which makes the DFE free of hardware multipliers. The detailed internal structure of the FFF and/or FBF block for all the three schemes are shown in Fig. 3.2, 3.3 and 3.4.

3.2.1 *Direct-memory architecture*

The internal structure of direct-memory architecture consists of a register bank, a set of memory blocks, bank of adders and a shift-accumulator block. Input bits enter the register bank serially which form the address bits to the memories. The adder bank consists of a set of adders where the outputs from the memories are summed up. The result is then shift-accumulated to compute the output. Figs. 3.2, 3.3 and 3.4 show the case where $P = B$, however the parameters P and L can be chosen based on the speed and complexity requirements. It can be seen from Fig. 3.2 that the contents of upper-half of the memories look like the mirror-image of the lower-half and hence a memory storing only one half of the contents can be used to perform DA filtering. The idea is particularly useful when the requirement for the memory size is large.

3.2.2 *Reduced-memory architecture*

The reduced memory architecture consists of the same internal blocks as that of the direct-memory architecture but requires only half the memory size as required by the direct memory scheme. In this scheme, when the 'upper-half' contents are stored, the lower-half can be generated by using a multiplexer (MUX) and a NOT gate as shown in Fig. 3.3. The idea here is that, since one half of the contents are nothing but the additive inverse version of the other half, these contents can be generated by just accessing the corresponding locations of the memory and by negating them using the NOT gate. Now, the most significant bit (MSB) is neglected in forming the address bits to the memory, instead it is used to find the effective address using the Ex-OR operation with each of the remaining address bits.

3.2.3 *OBC based-memory architecture*

The OBC based memory architecture also consists of a register bank, a set of memory blocks, bank of adders and a shift-accumulator block. The memory size requirement for these blocks is same as that of the reduced-memory architecture, however the contents (partial-products of filter coefficients) are more complex as can be observed from Fig. 3.4. The other half of the partial-products can be obtained by performing the two's complement operation on the contents of the memories which can be obtained by the usage of adder and NOT gates. The system also uses an extra adder and a register to store the $C_{initial}$ term as described by (Equation 1.20).

The decision device takes an input signal that lies between a set of signal levels and quantizes it to a set of pre-defined levels that depends on the modulation scheme employed. The control circuitry sends the required timing signals for the operation of FFF and FBF. The internal architecture for DA-OBC scheme that employs the ROM decomposition as well as the digit-serial scheme is shown in Fig. 3.4.

3.3 PERFORMANCE ANALYSIS

In order to test the DA based architecture, a DFE based channel-equalizer system has been created like the one shown in Fig. 3.5. For the ease of analysis, we chose $N_b = N_f - 1$. The input to the DFE and the weights of FFF and FBF are chosen to be fixed-point Q2.6 signed 2's - complement representation and the system is implemented using verilog HDL. To attain the maximum speed, P_f and P_b are chosen to be eight and the outputs of each memory are summed up by the adder bank and hence one sample of output of both FFF and FBF is computed in one clock cycle. The decision device quantizes its input to either of the two symbols (as BPSK[1] scheme is used). A rough estimations and comparisons of maximum usable frequency (F_{max}) and number of logic elements (LEs)[2] for the MAC based and DA based implementations for various FFF taps implemented on Altera® Cyclone III EP3C55F484C6 at different operating conditions (0°C, 85°C) are summarized in Table 3.1. From the table, it can be observed that the DA based implementation outperforms

1 Appendix A

2 Logic elements (LEs) are the smallest units of logic in the Cyclone III device family architecture. Each logic element consists of four input look-up-table, a programmable register and many other features. (www.altera.com/literature/hb/cyc3/cyc3_ciii51002.pdf)

the MAC based implementation in all the cases. For the case of 4-tap FFF, the difference in the number of logic elements for DA based and MAC based implementations is very less however the maximum usable frequency (F_{max}) is a bit more in case of DA based implementation. Further, it can be seen that as the filter order increases, the difference in the number of logic elements between the MAC based and DA based implementation is getting increased while the maximum usable frequency of DA based implementation is approaching to that of MAC based implementation. This can be explained as follows. For lower order filters, the number of MAC units in the MAC based implementation is quite low and therefore the number of memories replacing the multipliers in case of DA based implementation will also be less. But as the filter order increases, the number of memories replacing the multipliers in DA based implementation will be high and hence the difference in the number of logic elements utilized will be high. The critical path[3] (that decides the maximum usable frequency) of the FFF filter for the MAC based and DA based direct-memory implementation are given as $CP_{MAC} = T_m + NT_A + T_{quantizer}$ and $CP_{DA-direct-memory} = T_{memory} + (N-1)\,T_A$ where T_{memory}, T_m, T_A and $T_{quantizer}$ are the computation times of memory, multiplier, adder and quantizer units respectively. For lower order filters, because of the presence of less number of adder units, the critical path of MAC based and DA based implementation differs. As the filter order increases, the number of adders increase and hence the critical path of the DA based implementation approaches that of MAC based implementation due to the fact that the terms T_{memory} and T_M become negligible compared to the term $(N-1)\,T_A$. In case of reduced-memory architecture, even though the memory size is halved, the additional hardware overhead is added by the Ex-OR gates, multiplexer and adder units. The number of Ex-OR gates is fixed for a fixed P (which depends on B) while the number of adder units and MUXs' depend upon the value of M and P. The critical path in this case would be $CP_{DA-reduced-memory} = T_{reduced-memory} + T_{MUX} + NT_A$. Since the computation time of a memory does not vary much with respect to size of it, the additional computation time of reduced-memory architecture is just $T_{MUX} + T_A$ units.

3.4 CONCLUSION

In this Chapter, a distributed arithmetic based realization of the decision feedback equalizer has been presented. In order to attain the maximum speed, digit serial input has been employed, apart from that ROM decomposition technique is also employed to eliminate the exponential increase in the size requirement of memory as the FFF and FBF filter orders increase. Two DA-based architectures have been presented and results show that both are efficient when compared with MAC based implementation. A third architecture which uses DA-OBC scheme has also been presented which uses slightly more number of hardware resources compared to reduced memory architecture. For lower order filters, the maximum usable frequency is a bit high compared to that of MAC based implementation while the number of logic elements is only few units less. For higher order filters, the number of logic elements is almost half that of the MAC based implementation. The DA based architecture can be used in high data-rate modems such as the case of IEEE

3 Appendix A

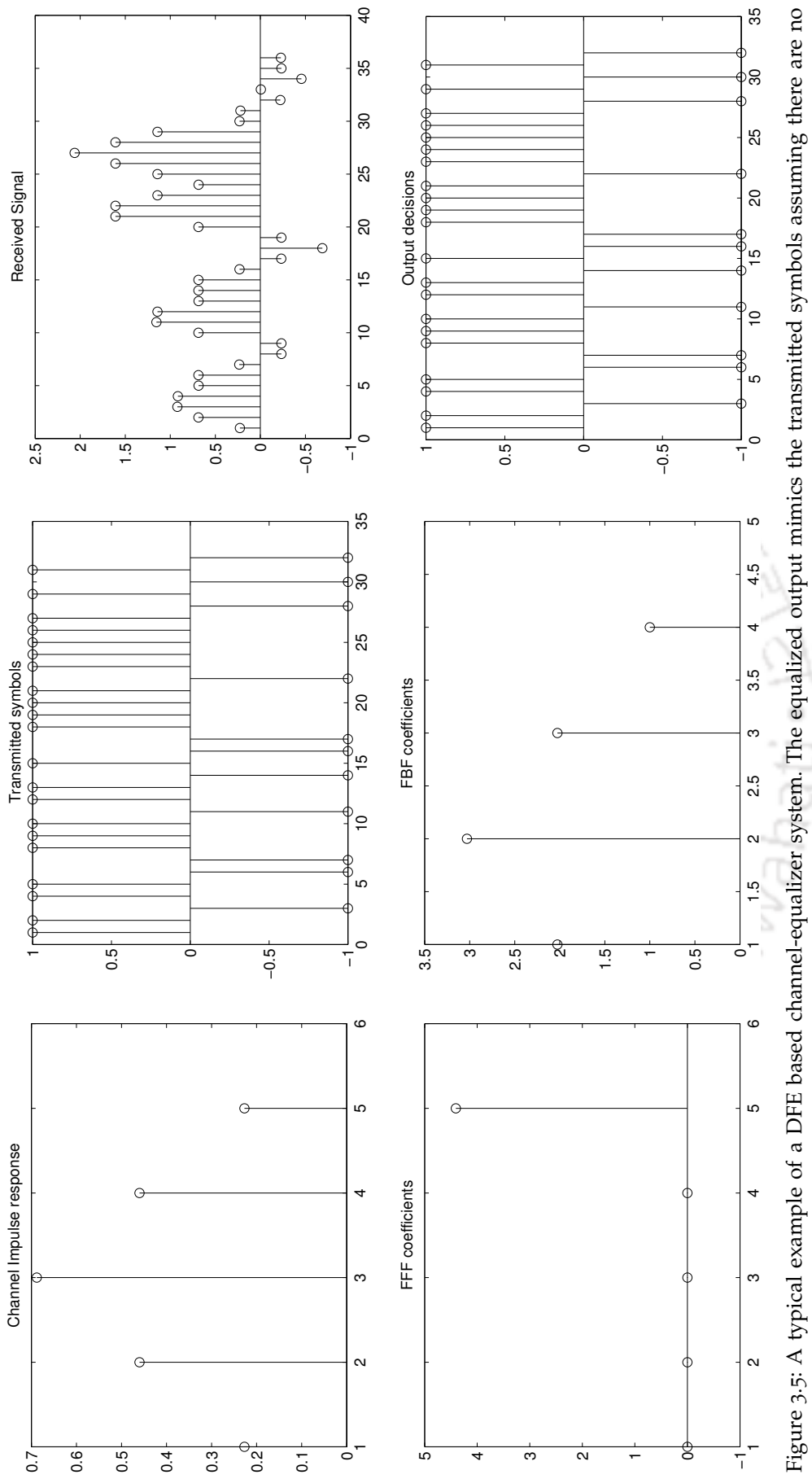

Figure 3.5: A typical example of a DFE based channel-equalizer system. The equalized output mimics the transmitted symbols assuming there are no errors in the past decisions.

Table 3.1: Comparison of F_{max} and number of LEs for MAC based and DA based implementations on Altera® Cyclone III EP3C55F484C6.

Size of FFF	MAC based Implementation		DA based implementation					
			Direct Memory		Reduced Memory		OBC Memory	
	$F_{max}(MHz)$ (0°C, 85°C)	Number of LEs	$F_{max}(MHz)$ (0°C, 85°C)	Number of LEs	$F_{max}(MHz)$ (0°C, 85°C)	Number of LEs	$F_{max}(MHz)$ (0°C, 85°C)	Number of LEs
4-tap	60.35, 56.47	420	67.94, 60.87	249	62.93, 55.92	431	51.86, 46.12	481
8-tap	30.6, 28.13	1007	38.2, 34.15	495	35.73, 31.89	853	31.0, 27.71	891
16-tap	15.36, 14.38	1775	19.56, 17.51	984	19.68, 17.57	1623	18.6, 16.64	1708
32-tap	7.76, 7.21	3312	10.24, 9.14	1918	10.17, 9.07	3296	9.73, 8.7	3376

802.11b scenarios. The speed of the DA based implementation can be further increased by choosing less number of adder units which results in the reduction of critical path.

4

A DA BASED REALIZATION OF THE ADAPTIVE DECISION FEEDBACK EQUALIZER

4.1 INTRODUCTION

In tele-communications, the transmitted symbols are often prone to distortion due to the band-limited nature of the transmission medium. The distortion can be of two types as the frequency components of the transmitted signal are non-equally attenuated (amplitude distortion) and non-equally delayed (delay distortion). This distortion is more severe in case of high-speed and long-distance wireless communications. Further, the distortion cannot be same at every time-instant due to the time-varying nature of the channels. The distortion causes overlapping of adjacent symbols known as Intersymbol Interference (ISI) which is an undesired phenomenon as it may cause erroneous symbols at the receiver thus making the communications less reliable. Equalizers are widely used at the receiver end in order to eliminate the ISI. Linear Equalizers are a class of equalizers which provide a good means of channel equalization by estimating the inverse of the channel transfer function. The ISI can be completely removed, however, the noise at the equalizer output may be enhanced at those frequencies where the channel has deep nulls in its frequency spectrum. Adaptive decision feedback equalizers (ADFEs) are an effective means of combating the ISI with low noise enhancement. An ADFE basically consists of a feed forward filter (FFF), a feedback filter (FBF) and a decision device. The feed forward filter (FFF) filters out the pre-cursor part of the ISI which occurs due to the anti-causal part of the channel transfer function. The remaining post-cursor portion of the ISI is cancelled out by the FBF using the decisions made by the decision device on the past symbols. The low noise enhancement of the ADFE is because, the decision device removes (quantizes) the noise present in the signal, assuming there are no decision errors. When there are erroneous decisions, error propagation occurs which may effect the reliability, however, in practice ADFEs are typically self-corrective as the errors occur only in short bursts. Distributed arithmetic can be used for the reduction of the computational complexities that arise due to the requirement of large number of taps in FFF and FBF. An interesting approach would be to unite the filtering equations of FFF and FBF into a single equation which would be in the form of sum of product of vectors. In such case, the resultant weight vector would contain the tap weights of both FFF and FBF and the resultant input sample vector would contain the terms of input samples of both FFF and FBF. However, such an approach would require a very large memory even for small filter orders and the idea would be impractical. Hence, the idea presented in chapter 2, i.e., maintaining an auxiliary look-up-table for the DA realization which could be useful in reducing the complexity involved in the implementation of an LMS based ADFE. In this Chapter, the DA based realization

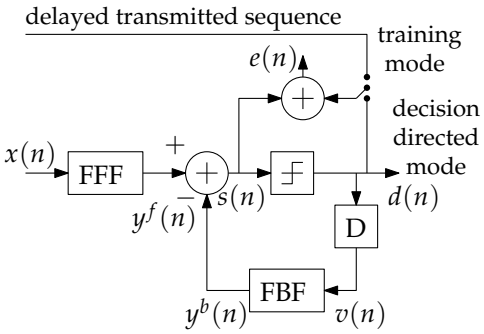

Figure 4.1: Block diagram of an adaptive decision feedback equalizer.

of LMS based ADFE has been taken up where separate DA treatments are carried out for FFF and FBF.

4.2 MATHEMATICAL FORMULATION OF DA BASED ADFE

Consider an adaptive decision feedback equalizer as shown in Fig. 4.1, with $\mathbf{w}^{(f)} = \left[w^{(f)}(0), w^{(f)}(1), \ldots, w^{(f)}(N_f-1)\right]^T$ and $\mathbf{w}^{(b)} = \left[w^{(b)}(0), w^{(b)}(1), \ldots, w^{(b)}(N_b-1)\right]^T$, the tap-weight vectors of FFF and FBF respectively are to be updated using the LMS algorithm. The equations governing the operation of the ADFE are:

$$y^f(n) = \sum_{i=0}^{N_f-1} w^{(f)}(i)\, x(n-i) = \mathbf{w}^{(f)}\mathbf{x} \tag{4.1}$$

$$y^b(n) = \sum_{i=0}^{N_b-1} w^{(b)}(i)\, v(n-i) = \mathbf{w}^{(b)}\mathbf{v} \tag{4.2}$$

$$s(n) = y^f(n) - y^b(n) \tag{4.3}$$

$$d(n) = Q[s(n)] \tag{4.4}$$

$$v(n) = d(n-1) \tag{4.5}$$

$$e(n) = \hat{d}(n) - s(n) \tag{4.6}$$

where $\mathbf{x}$ and $\mathbf{v}$ are the vectors containing the ADFE input samples and delayed decision output samples respectively and $Q[.]$ is the quantization operation. The sequence $\hat{d}(n)$ represents the delayed version of the original transmitted sequence in case of training mode and the output decisions in case of decision-directed mode.
For DA based realization, each of $x(n-i)$ terms may be represented in $Q_{1,B-1}$ fixed-point signed 2's-complement representation as,

$$x\left(n-i\right)=-b_{i,B-1}^{(f)}+\sum_{j=1}^{B-1}b_{i,B-1-j}^{(f)}2^{-j} \tag{4.7}$$

Using the framework of offset-binary coding technique discussed in subsection 1.4.2, $x\left(n-i\right)$ can be represented as $x\left(n-i\right)=\left(1/2\right)\left[x\left(n-i\right)-\left(-x\left(n-i\right)\right)\right]$ which gives,

$$x\left(n-i\right)=-\frac{1}{2}\left[d_{i,B-1}^{(f)}\right]+\frac{1}{2}\sum_{j=1}^{B-1}\left[d_{i,B-1-j}^{(f)}\right]2^{-j}-2^{-(B-1)} \tag{4.8}$$

where $d_{i,B-1-j}^{(f)}=\left(b_{i,B-1-j}^{(f)}-\overline{b}_{i,B-1-j}^{(f)}\right)$ and $\overline{b}_{i,B-1-j}^{(f)}$ is the complement of $b_{i,B-1-j}^{(f)}\in[0,1]$. By substituting (4.7) in (4.1) and re-arranging, we get

$$y^{f}\left(n\right)=-\frac{1}{2}\left[\sum_{i=0}^{N_f-1}w^{(f)}\left(i\right)d_{i,B-1}^{(f)}\right]+\frac{1}{2}\sum_{j=1}^{B-1}\left[\sum_{i=0}^{N_f-1}w^{(f)}\left(i\right)d_{i,B-1-j}^{(f)}\right]2^{-j}-W_{initial}^{(f)}2^{-(B-1)} \tag{4.9}$$

where,

$$W_{initial}^{(f)}=\frac{1}{2}\sum_{i=0}^{N_f-1}w^{(f)}\left(i\right) \tag{4.10}$$

It can be observed that the terms in the square braces of (4.9) can take one out of 2^{N_f} possible combinations depending upon $b_{i,B-1-j}^{(f)}$ which are nothing but the partial-products of coefficients of FFF. These partial-products can be stored in memory which we call it as FFF-LUT1. The content of FFF-LUT1 at the address location a can be mathematically expressed as

$$FFF-LUT1_{(a)}\left(n\right)=\frac{1}{2}w^{(f)}\left(0\right)+\sum_{k=1}^{N_f-1}\frac{1}{2}w^{(f)}\left(k\right)\left(-1\right)^{m_{N_f-1-k}^{(a)}+1}=\frac{1}{2}\mathbf{a}^{T}\mathbf{w}^{(f)} \tag{4.11}$$

where $\mathbf{w}^{(f)}=[w^{(f)}(0),w^{(f)}(1),\ldots,w^{(f)}(N_f-1)]^{T}$, $\mathbf{a}^{\mathrm{T}}=[1,(-1)^{m_{N_f-2}^{(a)}+1},\ldots,(-1)^{m_{0}^{(a)}+1}]$ and $m_l^{(a)}$ is the lth bit in the N-bit representation ($m_{N_f-1}^{(a)}=0$) of address a. That is

$$a=\sum_{s=0}^{N_f-2}m_s^{(a)}2^{s} \tag{4.12}$$

Let the weight vector $\mathbf{w}^{(f)}$ in (4.11) be replaced with the vector $\mathbf{x}$ to form a new equation, which can be given as
Clearly,

$$R_0^{(f)}\left(n\right)=\frac{1}{2}x\left(n\right) \tag{4.13}$$

and

$$FFF-LUT2_{(a)}\left(n\right)=\sum_{k=1}^{N_f-1}\frac{1}{2}x\left(n-k\right)\left(-1\right)^{m_{N_f-1-k}^{(a)}+1} \tag{4.14}$$

Exclusive of the most recent sample i.e., $\frac{1}{2}x(n)$, (4.14) represents the partial products of the input samples. These partial-products (OBC combinations) can be stored in a memory and we call it as FFF-LUT2. The partial products of FFF-LUT2 follow the same analogy as the partial-products of FFF filter weights described by (4.10).

Similarly, if each of the terms $v(n-i)$ can be represented in $Q_{1,B-1}$ fixed-point signed 2's-complement representation and by using the relation $v(n-i) = (1/2)[v(n-i) - (-v(n-i))]$,

$$v(n-i) = -\frac{1}{2}\left[d_{i,B-1}^{(b)}\right] + \frac{1}{2}\sum_{j=1}^{B-1}\left[d_{i,B-1-j}^{(b)}\right]2^{-j} - 2^{-(B-1)} \tag{4.15}$$

By substitution of (4.15) in (4.2),

$$y^b(n) = -\frac{1}{2}\left[\sum_{i=0}^{N_b-1} w^{(b)}(i)\, d_{i,B-1}^{(b)}\right] + \frac{1}{2}\sum_{j=1}^{B-1}\left[\sum_{i=0}^{N_b-1} w^{(b)}(i)\, d_{i,B-1-j}^{(b)}\right]2^{-j} - W_{initial}^{(b)}2^{-(B-1)} \tag{4.16}$$

$$W_{initial}^{(b)} = \frac{1}{2}\sum_{i=0}^{N_b-1} w^{(b)}(i) \tag{4.17}$$

$$FBF - LUT1_{(a)}(n) = \frac{1}{2}w^{(b)}(0) + \sum_{k=1}^{N_b-1}\frac{1}{2}w^{(b)}(k)(-1)^{m_{N_b-1-k}^{(a)}+1} = \frac{1}{2}\mathbf{a}^T\mathbf{w}^{(b)} \tag{4.18}$$

$$R_0^{(b)}(n) = \frac{1}{2}x(n) \tag{4.19}$$

and

$$FBF - LUT2_{(a)}(n) = \sum_{k=1}^{N_b-1}\frac{1}{2}x(n-k)(-1)^{m_{N_b-1-k}^{(a)}+1} \tag{4.20}$$

The equations (4.9) and (4.16) correspond to the filtering operations of the FFF and FBF filters respectively. With partial products following the same analogy in both the look-up-tables (memories) of FFF and FBF, the weight-update operation of FFF can be performed as follows. Access the first location of the $FFF - LUT2_{(a)}(n)$ and add it with the term $R_0^{(f)}(n)$. Multiply the result with $\mu e(n)$ (can be performed using shift operation if $\mu e(n)$ is quantized to powers of 2). Add the product to the same location of the $FFF - LUT1_{(a)}^{old}(a)$ and store the result back in the same location of $FFF - LUT1_{(a)}^{old}(a)$. Repeat the steps for all the address locations of $FFF - LUT1_{(a)}^{old}(a)$ and $FFF - LUT2_{(a)}(n)$.

The same procedure can be used for the weight-update operation of FBF and the equations describing the weight-update operations for FFF and FBF respectively are given as follows:

$$FFF - LUT1_{(a)}^{new}(n) = FFF - LUT1_{(a)}^{old}(a) + \mu e(n)\left[R_0^{(f)}(n) + FFF - LUT2_{(a)}(n)\right] \tag{4.21}$$

$$FBF - LUT1_{(a)}^{new}(n) = FBF - LUT1_{(a)}^{old}(n) + \mu e(n)\left[R_0^{(b)}(n) + FBF - LUT2_{(a)}(n)\right] \tag{4.22}$$

where $FFF-LUT1_{(a)}^{old}(n)$, $FBF-LUT1_{(a)}^{old}(n)$ and $FFF-LUT1_{(a)}^{new}(n)$, $FBF-LUT1_{(a)}^{new}(n)$ are respectively the partial-products of old and new set of weights of FFF and FBF respectively.

It can be observed that the terms $R_0^{(f)}$, $R_0^{(b)}$ may take special attention as they are not included in the partial products described by (4.13) and (4.19). These terms, which are useful in the weight-update operation (refer (4.20) and (4.21)) may be stored in registers. The purpose and benefit of storing them in the separate registers is explained in the following section. Based on the above equations, in the following section, we derive an efficient architecture for the adaptive decision feedback equalizer using distributed arithmetic.

4.3 OBC SCHEME FOR THE DA BASED ARCHITECTURE

The micro-architecture for the proposed ADFE using distributed arithmetic OBC scheme is shown in Fig. 4.2. As described by the equations (4.11) & (4.14), $FFF-LUT1$, $FFF-LUT2$ store the partial-products of the tap-weights of FFF and FBF respectively. The memories $FFF-LUT2$ and $FBF-LUT2$ store the partial products of corresponding samples useful for the weight-update operation. The operation of the FFF portion of ADFE for one complete filtering and weight-update operations is as follows: Bits of arriving input samples are stored in the registers one bit per clock. The least-significant bits of all the registers form the address to the FFF-LUT1. Assuming the length of the registers to be B-bits, the computation of the FFF filtering output based on the latest input sample$x(n)$ along with the past samples can be done in B clock cycles. The FFF-LUT2 which stores the partial-products of the input samples have to be updated along with the register $R_0^{(f)}$ so as to incorporate the newest sample $x(n)$ which will be useful for the weight-update operation in the current iteration. For this, the average of the consecutive contents of the $FFF-LUT2$ are utilized along with the addition and subtraction of the term present in the $R_0^{(f)}$ register.

For example, let us consider the case of $N_f=4$. The corresponding contents of $FFF-LUT2$ and $R_0^{(f)}$ before updating is shown in the left side of Fig. 4.3. The average of the even addressed and the next consecutive odd addressed location entries of the $FFF-LUT2$ would generate terms which are independent of the most recent input sample $x(n-4)$. The addition and subtraction of the generated terms with the entry of $R_0^{(f)}$ ($\frac{1}{2}x(n-1)$ in this example) would generate terms that are useful for the weight-update operation. These new terms can be stored consecutively in the same address locations of $FFF-LUT2$ as that of the terms used for averaging. The updated $FFF-LUT2$ however, will not be in the proper order or do not follow the same analogy as that of the contents of $FFF-LUT1$ and hence weight-update operation cannot be continued. However, the updated $FFF-LUT2$ can be properly accessed by circular shifting of its address lines. To understand this, let us consider the term $(1/2)[-x(n-1)-x(n-2)+x(n-3)]$ which is actually supposed to be at the address location 1 (i.e., 001 in binary form) of the updated $FFF-LUT2$. However, using the above update scheme, this term is actually placed in the address location 2 (i.e., 010 in binary form) as shown in the . Similarly, consider the term

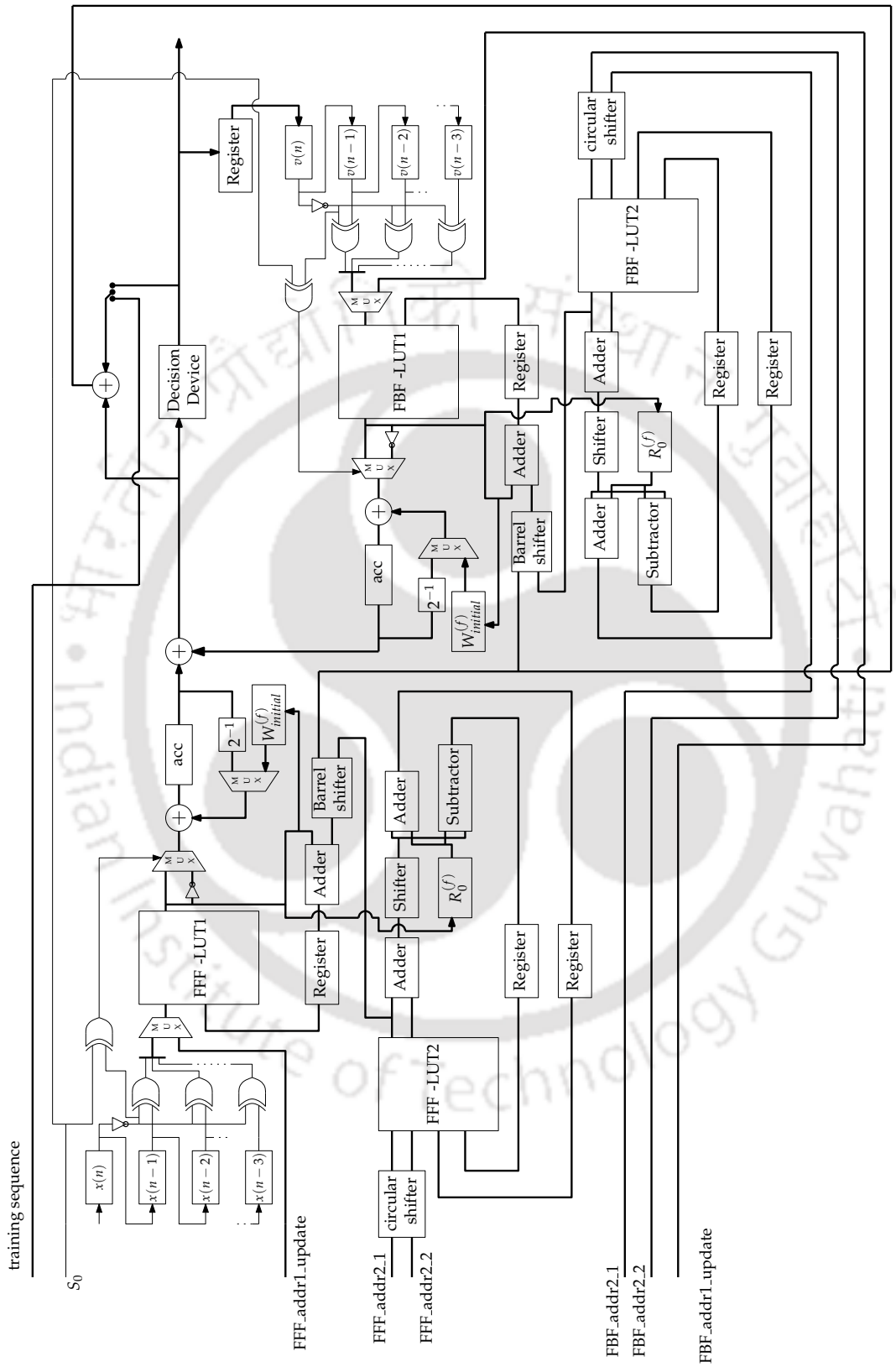

Figure 4.2: Micro-architecture for DA based ADFE.

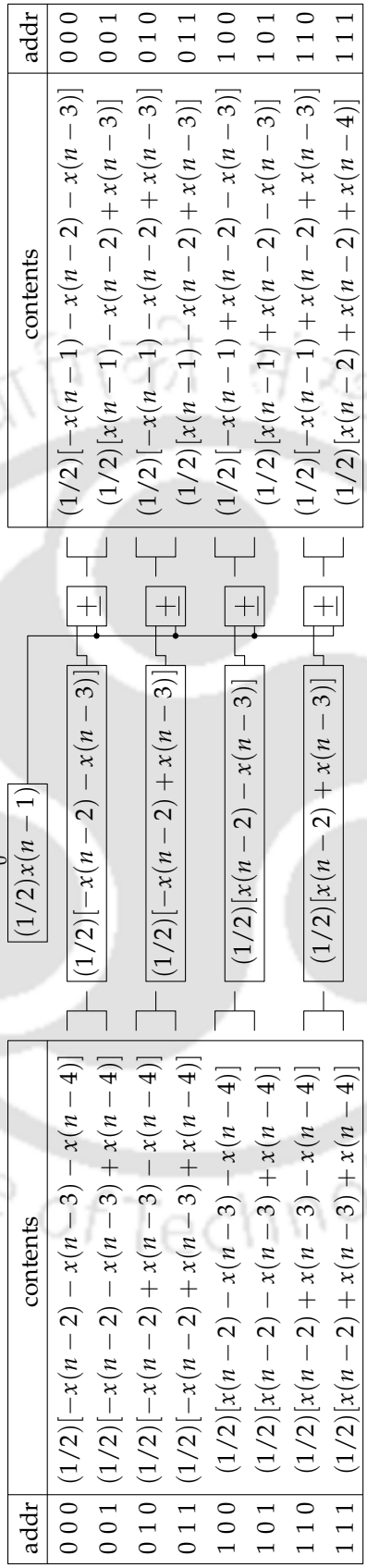

Figure 4.3: The look-up-table update scheme for FFF-LUT2.

$(1/2)\left[x\left(n-1\right)-x\left(n-2\right)+x\left(n-3\right)\right]$ which is supposed to be in the address location 5 (i.e., 101 in the binary form) of the updated $FFF-LUT2$. However, it is placed in the address location 3 (i.e., 011 in the binary form). Hence, it can be observed that, using the above $FFF-LUT2$ update scheme, the terms in the updated $FFF-LUT2$ are the terms that are supposed to be present at the address location (to be in the same pattern of OBC combinations) formed by circular left-shift of the addressing bits of $FFF-LUT2$. Hence, instead of re-arranging the terms, the $FFF-LUT2$ can be accessed by circular-left shifting the addressing bits during the weight-update operation. The term $R_0^{(f)}$ which stores the shifted version of the newest input sample can be updated (to store the term $\frac{1}{2}x\left(n\right)$) along-side updating the $FFF-LUT2$. Updating the $FFF-LUT2$ can in turn be done in parallel with the filtering operation. Further, a dual-port memory can be used for $FFF-LUT2$ which is more convenient for the update operation.

Similar filtering and LUT update scheme can be used for FBF and once one sample of the FFF and FBF outputs are available, the error $e\left(n\right)$ can be computed. In order to avoid the presence of hardware multipliers, the term $\mu e\left(n\right)$ can be quantized to fall in the powers of 2 and hence the multiplication of the term $\mu e\left(n\right)$ with the input sample during the gradient estimation can be done using the shift operation. Now, the weight-update operations of FFF and FBF as described by (4.21) & (4.22) can be achieved by the following sequence of operations: (i) Access the term at the first address location of $FFF-LUT2/FBF-LUT2$ and add it to the entry of $R_0^{(f)}/R_0^{(b)}$. (ii) Multiply the result with the term $\mu e\left(n\right)$ using a barrel shifter. (iii) Add the resultant term to the entry at the first address location of $FFF-LUT1/FBF-LUT1$ and store the result back in the same location of $FFF-LUT1/FBF-LUT1$. (iv) Repeat the steps (i)-(iii) for all the other address locations. The algorithm describing the operation of ADFE for one complete filtering and weight-update operations has been summarized in Fig. 4.4.

4.4 EXTENSION TO SIGN-LMS AND SIGNED-REGRESSOR LMS ADFE

Consider the filtering equation of the feedforward filter (FFF) of the adaptive decision feedback equalizer like the one shown in Fig. 4.1 whose weights are updated using Sign-LMS algorithm:

$$\begin{aligned} y\left(n\right) &= \mathbf{w}^T\mathbf{x} \\ &= \sum_{i=0}^{N-1} w_i^{(f)} x\left(n-i\right) \end{aligned}$$

where $w_i^{(f)}$, $i=0,1,2,\ldots,N_f-1$ are the filter taps of the adaptive filter.
If each of $x\left(n-i\right)$ is represented in 2's-complement form and using OBC scheme discussed in subsection 1.4.2, $y\left(n\right)$ can be expressed as:

$$y\left(n\right)=\sum_{j=0}^{B-1} W_{B-1-j}^{(f)} 2^{-j} - W_{extra}^{(f)} 2^{-(B-1)} \tag{4.23}$$

The above filtering operation can be performed using DA processing unit by taking the LSB of each of x_{n-i}. The terms $W_{B-1-j}^{(f)}$ and $W_{extra}^{(f)}$ in (4.23) are given as

Step-1:
a) Computation of FFF output:
$\hat{x}(n) \leftarrow -\frac{1}{2}\left[\sum_{i=0}^{N_f-1} w^{(f)}(i)\, d_{i,B-1}^{(f)}\right] + \frac{1}{2}\sum_{j=1}^{B-1}\left[\sum_{i=0}^{N_f-1} w^{(f)}(i)\, d_{i,B-1-j}^{(f)}\right] 2^{-j} - W_{initial}^{(f)} 2^{-(B-1)};$
b) Computation of FBF output:
$\hat{v}(n) \leftarrow -\frac{1}{2}\left[\sum_{i=0}^{N_b-1} w^{(b)}(i)\, d_{i,B-1}^{(b)}\right] + \frac{1}{2}\sum_{j=1}^{B-1}\left[\sum_{i=0}^{N_b-1} w^{(b)}(i)\, d_{i,B-1-j}^{(b)}\right] 2^{-j} - W_{initial}^{(b)} 2^{-(B-1)};$
c) Updating $FFF-LUT2_{(a')}(n)$**:**
for $a' = 0 : 2^{N_f-1} - 1$ *do*
$FFF-LUT2_{(a')}(n) \leftarrow (-1)^{a'+1} R_0^{(f)}(n-1)$
$+\frac{1}{2}\left\{FFF-LUT2_{\left(2\left\lfloor\frac{a'}{2}\right\rfloor\right)}(n-1) + FFF-LUT2_{\left(2\left\lfloor\frac{a'}{2}\right\rfloor+1\right)}(n-1)\right\};$
end for
d) Updating $FBF-LUT2_{(a')}(n)$**:**
for $a' = 0 : 2^{N_b-1} - 1$ *do*
$FBF-LUT2_{(a')}(n) \leftarrow (-1)^{a'+1} R_0^{(b)}(n-1)$
$+\frac{1}{2}\left\{FBF-LUT2_{\left(2\left\lfloor\frac{a'}{2}\right\rfloor\right)}(n-1) + FBF-LUT2_{\left(2\left\lfloor\frac{a'}{2}\right\rfloor+1\right)}(n-1)\right\};$
end for
Step-2:
a) Updating $R_0^{(f)}$**:**
$R_0^{(f)} \leftarrow \frac{1}{2}x(n);$
b) Updating $R_0^{(b)}$**:**
$R_0^{(b)} \leftarrow \frac{1}{2}v(n);$
c) Computation of error $e(n)$**:**
$e(n) = \hat{d}(n) - s(n);$
Step-3:
$a' = circleftshift(a');$
Step-4:
a) Updating $FFF-LUT1_{(a)}(n)$**:**
for $a = 0 : 2^{N_f-1} - 1$ *do*
$FFF-LUT1_{(a)}^{new}(n) = FFF-LUT1_{(a)}^{old}(a) + \mu e(n)\left[R_0^{(f)}(n) + FFF-LUT2_{(a')}(n)\right];$
end for
b) Updating $FBF-LUT1_{(a)}(n)$**:**
for $a = 0 : 2^{N_b-1} - 1$ *do*
$FBF-LUT1_{(a)}^{new}(n) = FBF-LUT1_{(a)}^{old}(n) + \mu e(n)\left[R_0^{(b)}(n) + FBF-LUT2_{(a')}(n)\right];$
end for
Step-5:
a) $W_{initial}^{(f)} = FFF-LUT1_{\left(2^{(N_f/2)}-1\right)}^{new}(n);$
b) $W_{initial}^{(b)} = FBF-LUT1_{\left(2^{(N_b/2)}-1\right)}^{new}(n);$
Step-6:
$n \leftarrow n+1;$
repeat;

Figure 4.4: Algorithm showing the operation of the DA based ADFE.

$$W_{B-1-j}^{(f)} = \begin{cases} -F_0^{(f)} & j = 0 \\ F_j^{(f)} & j \neq 0 \end{cases} \tag{4.24}$$

$$W_{extra}^{(f)} = \frac{1}{2}\sum_{i=0}^{N-1} w_i^{(f)} \tag{4.25}$$

where

$$F_j^{(f)} = \sum_{i=0}^{N-1} w_i^{(f)} d_{i,B-1-j}^{(f)} \tag{4.26}$$

The terms in (4.26) are the partial-products which can be stored in a memory. If only the upper half of the partial-products are stored in the memory, then the entry of the memory at address location a can be given by

$$F_j^{(f)(a)} = \frac{1}{2}w_0^{(f)} + \sum_{i=1}^{N-1} w_i^{(f)} (-1)^{p_{N-1-i}^{(a)}+1} \tag{4.27}$$

where $p_i^{(a)}$ is the ith bit in the binary representation of a.
To derive the weight-update equation for the DA based implementation, let us consider the weight-update equation of sign-LMS adaptive and re-write it on a sample basis which is given by

$$w_i^{(f)}(n+1) = w_i^{(f)}(n) + \mu sign(e(n))\, x(n-i) \tag{4.28}$$

If both the LHS and RHS of the above equation are multiplied by the term $\sum_{i=0}^{N-1}(-1)^{p_{N-1-i}^{(a)}+1}$, then by re-arranging

$$\begin{aligned} &\frac{1}{2}w_0^{(f)}(n+1) + \sum_{i=1}^{N-1}(-1)^{p_{N-1-i}^{(a)}+1} w_i^{(f)}(n+1) \\ &= \frac{1}{2}w_0^{(f)}(n) + \sum_{i=1}^{N-1}(-1)^{p_{N-1-i}^{(a)}+1} w_i^{(f)}(n) \\ &+\mu sign(e(n))\left[\frac{1}{2}x(n) + \sum_{i=1}^{N-1}(-1)^{p_{N-1-i}^{(a)}+1} x(n-i)\right] \end{aligned} \tag{4.29}$$

In (4.29), the summation related to the term $x(n-i)$ are nothing but the partial-products of the input samples and hence it may be convenient to store the partial-products of input samples in a memory so that the weight-update operation can be performed using the partial-products of filter weights and input samples.
Letting,

$$G_j^{(f)}(n) = \sum_{i=1}^{N-1}(-1)^{p_{N-1-i}^{(a)}+1} x(n-i) \tag{4.30}$$

$$R^{(f)}(n) = \frac{1}{2}x(n) \tag{4.31}$$

and by substitution of (4.30) & (4.31) in (4.29)

$$F_j^{(f)}(n+1) = F_j^{(f)}(n) + \mu sign(e(n)) \left[R^{(f)}(n) + G_j^{(f)}(n) \right] \tag{4.32}$$

Since, in every iteration, a new input sample is used for the filtering while the oldest sample is not considered, the memory that stores $G_j^{(f)}(n)$ has to be updated from time to time. This may be obtained by taking the average of the pairs of consecutive locations of the memory (which eliminates the oldest input sample) and then by addition and subtraction of the term $R(n)$ with the result and storing them back in the same corresponding locations of the memory. Mathematically, this can be written as

$$\begin{aligned} G_j^{(f)(a)}(n) &= (-1)^{a+1} R^{(f)}(n-1) \\ &\quad + \frac{1}{2} \left[G_j^{(f)\left(2\left\lfloor \frac{a}{2} \right\rfloor\right)}(n-1) + G_j^{(f)\left(2\left\lfloor \frac{a}{2} \right\rfloor+1\right)}(n-1) \right] \end{aligned} \tag{4.33}$$

Using the above memory update scheme, although, the memory has been updated, the new partial-products containing the newest set of samples are not in proper order. This can be corrected by accessing the memory just by circularly left shifting its address bits as described in section 4.3. Considering the memory storing the partial-products of weights and input samples of FFF as $MEM_w^{(f)}$ and $MEM_X^{(f)}$ respectively, the step-by-step processing of the DA based realization for one complete iteration is as follows:

- Bits of input samples arriving are stored in the buffers and are shifted one bit per clock cycle. The LSBs of each of the buffers are used as the address bits to $MEM_w^{(f)}$ and the filtering takes places for B clock cycles (B is the bit-length of the buffers) as per (4.23) for the computation of one sample of output.
- Using the term $R_0^{(f)}(n-1)$ which is stored in a register, the memory $MEM_x^{(f)}$ is updated using the scheme described by (4.33).
- The term $R_0^{(f)}(n-1)$ is updated so as to incorporate the term related to the newest input sample which becomes $R_0^{(f)}(n)$.
- The error $e(n)$ and thereby $sign(e(n))$ is computed using the filtered output and the desired sequence.
- The term $\mu sign(e(n))$ is quantized to fall into levels which are in powers of 2 so that the multiplication becomes a shifting operation.
- The partial-products of $MEM_w^{(f)}$ are then updated using (4.32) which makes it ready for filtering operation in the next iteration. For this purpose, the memory $MEM_x^{(f)}$ is accessed by circularly left shifting its address bits.
- The term $W_{extra}^{(f)}$ is updated for the filtering operation in the next iteration and this is done just by using the content in the last address location of the updated memory $MEM_w^{(f)}$.

Similar algorithm can be applied for the feedback filter (FBF) of the ADFE and the same algorithm can be extended to signed-regressor LMS algorithm in which case the $sign(.)$ function in (4.28) is applied to the input sample in place of the error. The convergence curves for the sign-LMS and signed-regressor LMS based ADFEs are shown in 4.7.

4.5 PERFORMANCE ANALYSIS

In order to verify the performance of the proposed technique, simulations were carried out using an ADFE channel-equalizer model where set of 1000 message signal samples, each modulated using binary phase shift keying (BPSK) scheme have been transmitted into an Additive White Gaussian Noise (AWGN) channel for the initial training of the equalizer. The channel induces ISI among the transmitted symbols which are also corrupted by the white noise present in the channel. The length of the FFF and FBF are 3 and 6 respectively. Out of the 1000 transmitted symbols, the first set of 200 symbols are used as the pilot sequence that are used for training the ADFE (training mode) and the remaining set of symbols are used for the decision directed mode of operation of ADFE. The convergence plots for the proposed and MAC based implementations is shown in Fig. 4.5 and the convergence curves for DA based implementation for different bit lengths of the input vector are shown in Fig. 4.6. These plots have been obtained by taking the average of 50 independent MATLAB simulations. Once the ADFE has been trained to estimate the changes in the characteristics of the channel as well as to track any slow variations in it, a set of 500 message signals were transmitted each once again modulated using BPSK scheme for communication purpose. It can be observed that the approximation of the error signal to powers of 2 has not affected the convergence performance.

4.6 COMPUTATIONAL COMPLEXITY

The computational complexity of the proposed architecture along with the existing architectures are given in Tab 4.1. The computational complexity of the proposed architecture can be computed as follows assuming DA ROM decomposition can be employed where N_f and N_b can be split as $N_f = m_f \times 2^{k_f}$ and $N_b = m_b \times 2^{k_b}$. The DA processing unit in the FFF uses $m_f + 1$ number of adder units. The FFF-LUT2 update scheme for FFF uses a total of 5 adder units. Similarly the DA unit and the FBF-LUT2 update scheme uses $m_b + 6$ number of adder units. The subtraction of FBF output from the FFF output and the computation of error takes 4 adder units. This gives a total of $m_f + m_b + 16$ number of adder units. The proposed structure is free of hardware multipliers because of the DA realization and also due to the quantization of the term $\mu e(n)$ to the nearest powers of 2. Assuming the bit-length of the data at every stage is rounded-off to B-bits, the register bank that stores the input samples to the FFF takes $N_f B$ number of flip-flop units. The look-up-table FFF-LUT1 takes $m_f 2^{k_f} B$ number of flip-flop units assuming the bit-width of each of the memory location is B bits. While the accumulator in the DA processing unit takes B flip-flops, the FFF-LUT2 look-up-table update scheme would require $5B$ number of flip-flops. Similarly the FBF unit takes $\left(N_b + m_b 2^{k_b} + 6\right) B$ number of flip-flop units along with B-flops for storing the output decision in every iteration and hence the total number of flops for the proposed architecture would be $\left(N_f + N_b + m_f 2^{k_f} + m_b 2^{k_b} + 13\right) B$. Apart from these, the architecture uses few combinatorial units such as 2x1-multiplexers, shifter units and barrel shifters. The size of memories for $FFF - LUT1$ and $FBF - LUT1$ would be 2^{k_f} and 2^{k_b} respectively. The auxiliary LUTs useful for the weight-update operation also need same sized memories and hence the total memory requirement would be $2^{(k_f+1)} + 2^{(k_b+1)}$. Assuming the scenario where the post-cursor components of ISI are more which is often the case, let $N_b = 2N_f$. Hence, $m_b \times 2^{k_b} = 2 \times m_f \times 2^{k_f}$. Assuming,

same number of adder units are used for FFF and FBF, we have $k_b = k_f + 1$ which means that the total memory size requirement would be 6.2^{k_f}.

4.7 CONCLUSION

In this Chapter, an LMS based ADFE has been realized using distributed arithmetic. The ADFE equations have been recast in such a way that the structure can be realized using distributed arithmetic. The partial-products of filter coefficients of FFF and FBF which are useful for the filtering operation have been stored in the look-up-tables named $FFF - LUT1$ and $FBF - LUT1$ respectively. Two more look-up-tables namely $FFF - LUT2$ and $FBF - LUT2$ which store the partial-products of the recent filter input samples have been used which aid in the weight-update operation. These look-up-tables have been updated from time to time using registers namely $R_0^{(f)}$ and $R_0^{(b)}$ and using the manipulation of their address bits.

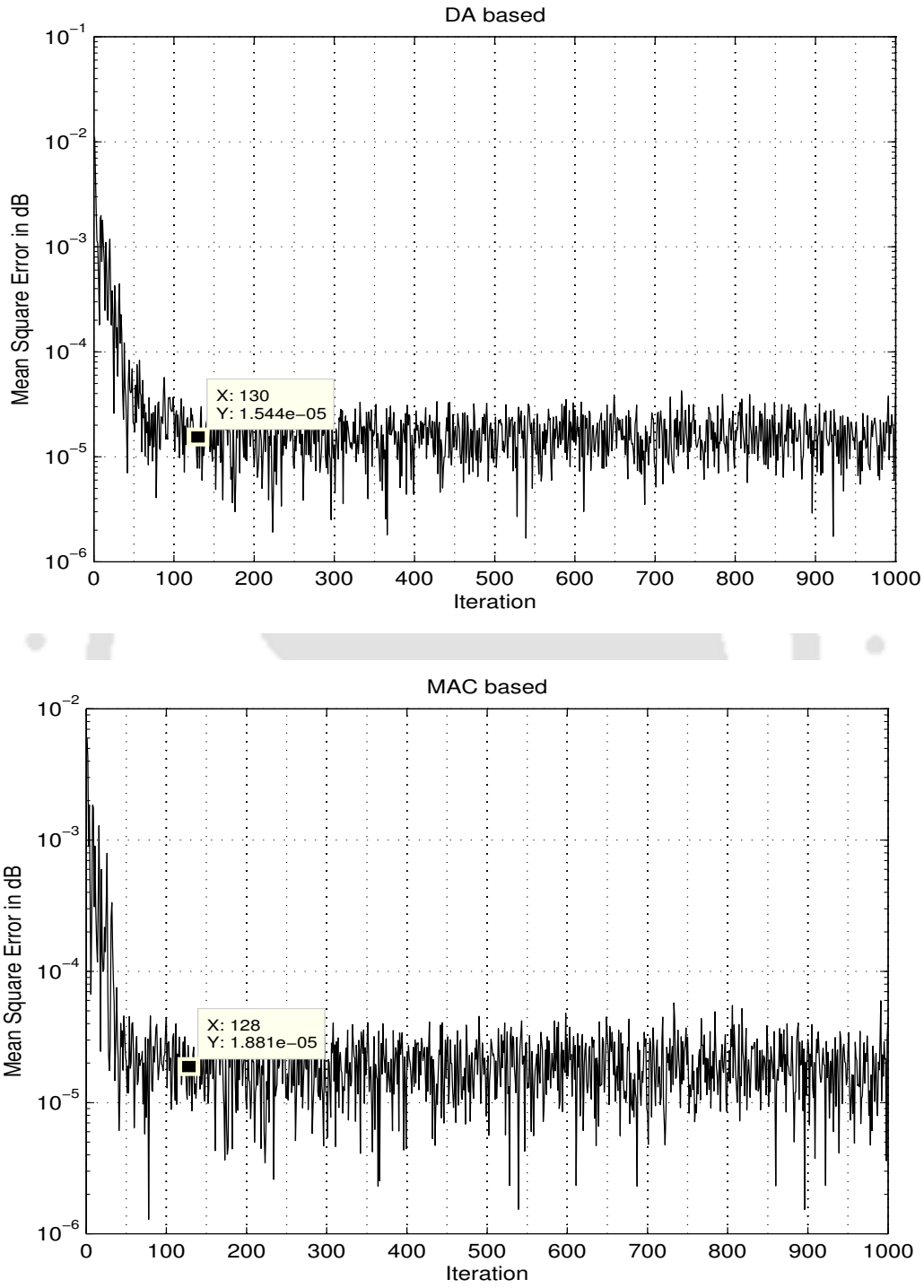

Figure 4.5: Convergence curves for the DA based and MAC based ADFEs.

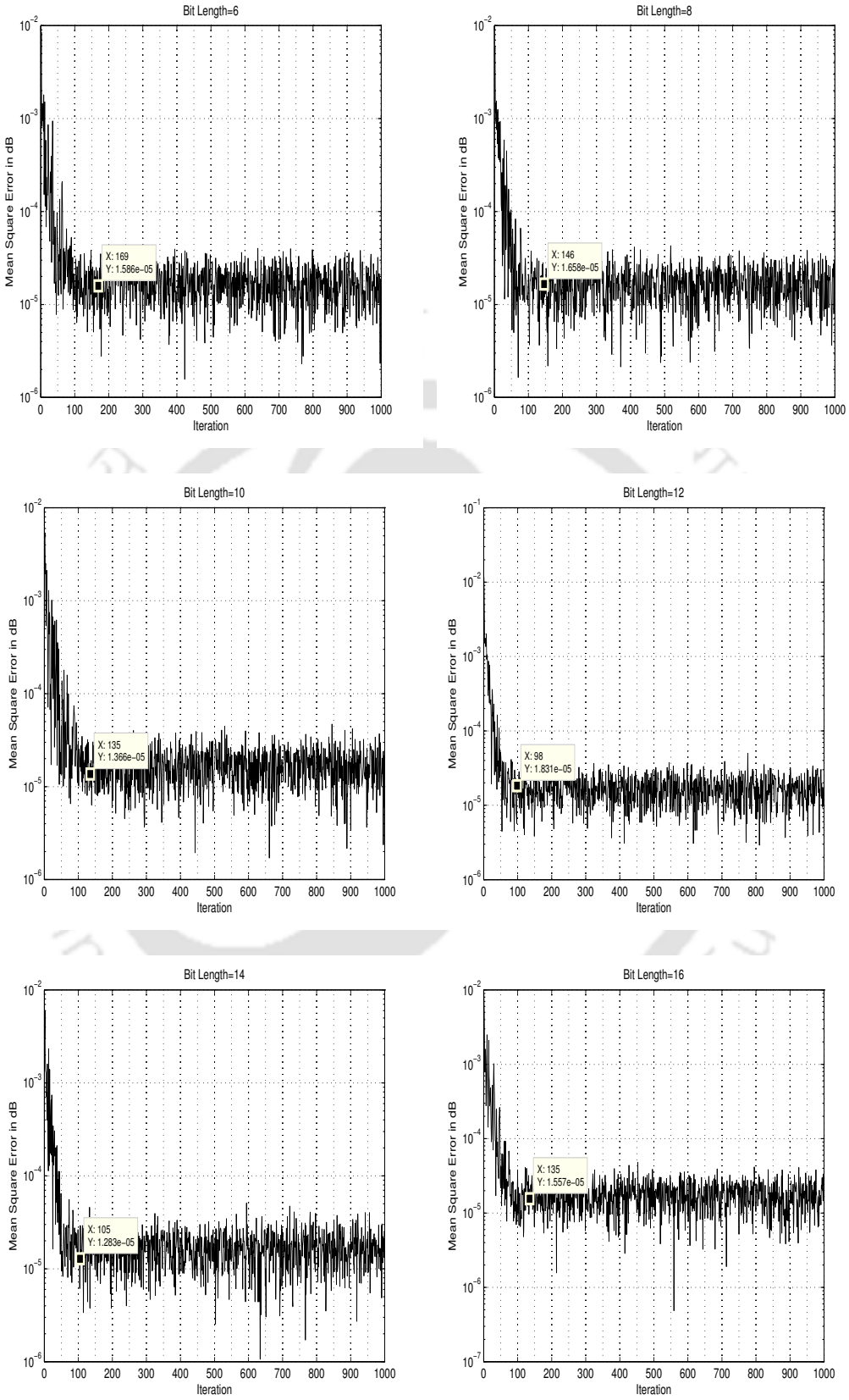

Figure 4.6: Convergence curves for LMS ADFE for different bit lengths.

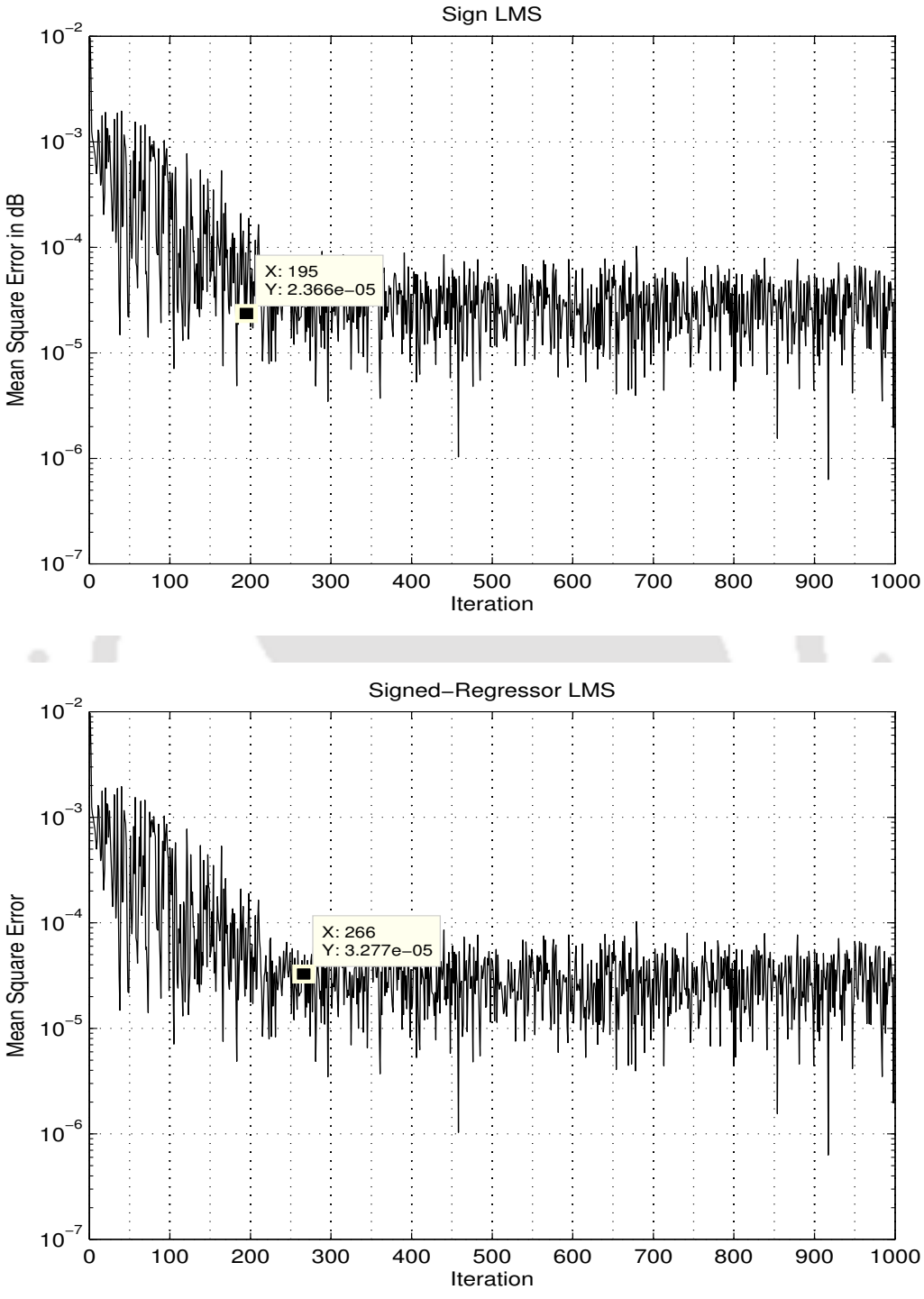

Figure 4.7: Convergence curves for Sign-LMS and Signed Regressor LMS ADFEs realized using DA.

	No. of adder units	No. of multiplier units	No. of shifter units	No. of flip-flops
Proposed DA based ADFE	$16 + m_f + m_b - 2$	0	2	$\left(N_f + N_b\right) B + \left(m_f 2^{k_f} + m_b 2^{k_b}\right) B + 13B$ ($N_f = m_f \times k_f$, $N_b = m_b \times k_b$ and B is the bit-length used at every stage of the system.)
Existing scheme 1 [33]	$10N_f + 1$	$10N_f$	-	$9N_f - 1$
Existing scheme 2 [44]	$N_f + N_b + 3$	$N_f + N_b + 2$	$2N_f + 2N_b + 4$	-
Existing scheme 3 [29]	$2^{B+2} + N_b - 1$	$N_b - 1$	-	$N_b/2$
Existing scheme 4 [29]	$2(B)^{N_b/2}$	0	-	$N_b/2$

Table 4.1: Comparison of computational complexity of the DA based ADFE with existing schemes.

5

DA BASED APPROACH FOR THE REALIZATION OF THE BLOCK ADAPTIVE DECISION FEEDBACK EQUALIZER

5.1 INTRODUCTION

As discussed in Chapter 1, a common problem faced by ADFE is that with increasing data transmission rate, the ISI components become more and hence the order of both the FFF and FBF would increase. The resulting increase in complexity would make real time operation of the ADFE difficult. Block processing is one of the approaches in reducing the complexities in digital filters [14, 47], as the block based computations such as the convolution and correlation operations can be implemented using fast fourier transform (FFT) techniques. However, block processing of ADFE poses difficulty in that the block processing of FFF input, i.e., the received data is possible as they are known a priori, but the same cannot be achieved with FBF input, i.e., the decision outputs which are to be evaluated by the ADFE. Few attempts [8, 18, 7] were made in the literature in order order to realize block ADFE. In [8], the technique developed enjoys the advantages of frequency domain schemes i.e., ease of implementation and low complexity, however the technique imposes few restrictions on choosing the block length with respect to the lengths of FFF and FBF. In [18], a hybrid scheme has been implemented where only the feedforward part is implemented in the frequency domain, whereas the feedback part is implemented in the time domain. Hence, the scheme does not enjoy all the advantages of a full frequency-domain implementation. The scheme in [7] uses two iterative schemes where the unknown decisions are replaced with tentative decisions and using an iterative procedure, the unknown decisions are finally made to converge to optimal decisions. However, the idea requires computation of certain terms (like the matrix inversion) which makes the hardware very complex. In this Chapter, the DA treatment has been taken up for the realization of the block ADFE. Throughout the Chapter, the following conventions are used: (i) $x(n)$ represents a sample at time instant n. (ii) $\mathbf{x}(n)$ represents a vector of appropriate dimension. (iii) $\mathbf{X}(n)$ represents a matrix of appropriate dimensions. (iv) $\mathbf{X}_{\mathcal{F}}(n)$ represents a matrix containing samples in frequency domain. (v) $\mathbf{X}_c(n)$ represents a circular square matrix. (vii) $\nabla\mathbf{X}(n)$ and $\triangle\mathbf{X}(n)$ represent the upper and lower triangular matrices and finally (vii) $X_{m,n}(n)$ is a special matrix and has been explained as per the context. In the Chapter, the subscripts/superscripts f and b represent the terms related to FFF and FBF respectively.

5.2 THE BLOCK LMS (BLMS) ALGORITHM

Consider a block LMS based adaptive filter depicted in Fig. 5.1, that takes an input sequence $x(n)$, $(n \in Z)$, which is partitioned into non-overlapping blocks of length L each by means of a serial-to-parallel converter. The blocks of data so produced are applied to a block FIR filter of length N, one block at a time. The tap weights of the filter are updated after the collection of each block of data samples, so that the adaptation of the filter proceeds on a block by block basis rather than on a sample-by-sample basis as in conventional LMS algorithm.

With the j-th block, $(j \in Z)$ consisting of $x(jL+s)$, $s \in Z_p = 0, 1, \ldots, L-1$, the filter coefficients are updated from block to block as,

$$\mathbf{w}(j+1) = \mathbf{w}(j) + \mu \sum_{s=0}^{N-1} \mathbf{x}(jL+s)\, e(jL+s) \tag{5.1}$$

where $\mathbf{w}(j) = [w_0(j), w_1(j) \ldots w_{N-1}(j)]^T$ is a tap weight vector corresponding to the j-th block, $\mathbf{x}(jL+s) = [x(jL+s), x(jL+s-1) \ldots x(jL+s-L+1)]^T$ and $e(jL+s)$ is the output error at $n = jL+s$, given by,

$$e(jL+s) = d(jL+s) - y(jL+s) \tag{5.2}$$

The sequence $d(jL+s)$ is the so-called desired response available during the initial training period and $y(jL+s)$ is the filter output at $n = jL+s$, given as,

$$y(jL+s) = \mathbf{w}^T(j)\, \mathbf{x}(jL+s) \tag{5.3}$$

The parameter μ, popularly called step size parameter is to be chosen as $0 < \mu < \frac{2}{[P\, tr\mathbf{R}]}$ for convergence of the algorithm ($\mathbf{R}$ is the auto-correlation matrix).

5.3 THE CAUSALITY PROBLEM IN BLOCK ADFE

Consider a block adaptive decision feedback equalizer shown in Fig. 5.2, whose feed forward filter (FFF) and feedback filter (FBF) weights are to be updated using the block LMS algorithm with a block length L. With k being the block index, the equations governing the operation of the ADFE are given as

$$\hat{\mathbf{x}}(k) = \mathbf{X}(k).\mathbf{w}^f(k) \tag{5.4}$$
$$\hat{\mathbf{v}}(k) = \mathbf{V}(k).\mathbf{w}^b(k) \tag{5.5}$$
$$\mathbf{s}(k) = \hat{\mathbf{x}}(k) - \hat{\mathbf{v}}(k) \tag{5.6}$$
$$\mathbf{v}(k) = \mathbf{d}(k-(1/L)) \tag{5.7}$$
$$\mathbf{d}(k) = Q[s(k)] \tag{5.8}$$
$$\mathbf{e}(k) = \hat{\mathbf{d}}(k) - \mathbf{s}(k) \tag{5.9}$$

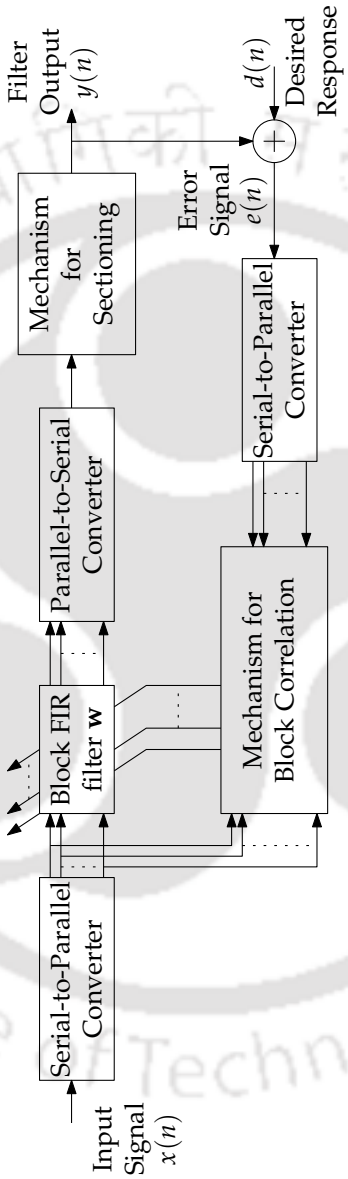

Figure 5.1: Block diagram of a Block LMS based adaptive filter.

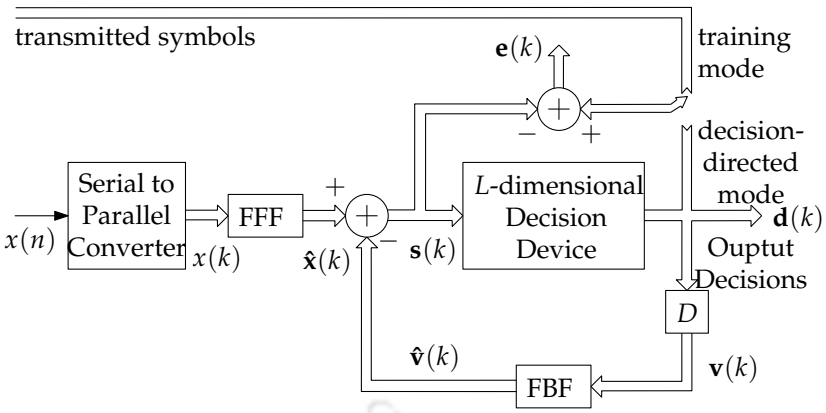

Figure 5.2: A block adaptive decision feedback equalizer.

$$\hat{\mathbf{d}}(k) = \begin{cases} \tilde{\mathbf{x}}(k) & \text{training mode} \\ \mathbf{d}(k) & \text{decision} - \text{directed mode} \end{cases} \tag{5.10}$$

where $\mathbf{w}_f(k) = \left[w_{f,0}, w_{f,1}, \ldots, w_{f,N_f-1}\right]^T$, $\mathbf{w}_b(k) = \left[w_{b,0}, w_{b,1}, \ldots, w_{b,N_b-1}\right]^T$ are the weights of FFF and FBF respectively and $Q[.]$ is the quantization operator of the L-dimensional decision device. Further, $\hat{\mathbf{d}}(k)$ is the desired vector which is the vector containing the delayed original transmitted sequence vector $\tilde{\mathbf{x}}(k)$ in case of training mode and output decision vector $\mathbf{d}(k)$ in case of decision-directed mode. The vector $\mathbf{e}(k)$ is the error vector used for updating the tap-weights of FFF and FBF.
Here $\hat{\mathbf{x}}(k)$ and $\hat{\mathbf{v}}(k)$ are row vectors containing L samples of FFF and FBF outputs respectively. $\mathbf{s}(k)$ and $\mathbf{d}(k)$ represent the row vectors containing the L samples of adder output and their corresponding decisions respectively. The matrices $\mathbf{X}(k)$ and $\mathbf{V}(k)$ are respectively given as

$$\mathbf{X}(k) = \left[\begin{array}{c|c} \mathbf{X}_1(k) & \mathbf{X}_2(k) \end{array}\right] \tag{5.11}$$

$$\mathbf{V}(k) = \left[\begin{array}{c|c} \mathbf{V}_1(k) & \mathbf{V}_2(k) \end{array}\right] \tag{5.12}$$

The matrices $\mathbf{X}_1(k)$ and $\mathbf{X}_2(k)$ in (5.11) are given as

$$\mathbf{X}_1(k) = \begin{bmatrix} x(Lk+L-1) & \ldots & x(Lk) \\ x(Lk+L-2) & \ldots & x(Lk-1) \\ \vdots & \ddots & \vdots \\ x(Lk+1) & \ldots & x(Lk+N_f-L) \\ x(Lk) & \ldots & x(Lk+N_f-L-1) \end{bmatrix} \tag{5.13}$$

$$\mathbf{X}_2(k) = \begin{bmatrix} x(Lk-1) & \ldots & x(Lk+N_f-L) \\ x(Lk-2) & \ldots & x(Lk+N_f-L-1) \\ \vdots & \ddots & \vdots \\ x(Lk-N_f-L-1) & \ldots & x(Lk-N_f+2) \\ x(Lk-N_f-2) & \ldots & x(Lk-N_f+1) \end{bmatrix} \tag{5.14}$$

The matrix $\mathbf{V}_1(k)$ in (5.12) is given as,

$$\mathbf{V}_1(k) = \nabla\mathbf{V}_1(k) + \triangle\mathbf{V}_1(k) \tag{5.15}$$

where $\nabla\mathbf{V}_1(k)$, $\triangle\mathbf{V}_1(k)$ are the upper and lower triangular matrices, which are given as,

$$\nabla\mathbf{V}_1(k) = \begin{bmatrix} v(Lk+L-1) & \dots & v(Lk+1) & 0 \\ v(Lk+L-2) & \dots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ v(Lk+1) & \dots & 0 & 0 \\ 0 & \dots & 0 & 0 \end{bmatrix} \tag{5.16}$$

$$\triangle\mathbf{V}_1(k) = \begin{bmatrix} 0 & 0 & \dots & v(Lk) \\ 0 & 0 & \dots & v(Lk-1) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & v(Lk) & \dots & v(Lk+N_b-L) \\ v(Lk) & v(Lk-1) & \dots & v(Lk+N_b-L-1) \end{bmatrix} \tag{5.17}$$

and

$$\mathbf{V}_2(k) = \begin{bmatrix} v(Lk-1) & \dots & v(Lk+N_b-L) \\ v(Lk-2) & \dots & v(Lk+N_b-L-1) \\ \vdots & \ddots & \vdots \\ v(Lk-N_b-L-1) & \dots & v(Lk-N_b+2) \\ v(Lk-N_b-2) & \dots & v(Lk-N_b+1) \end{bmatrix} \tag{5.18}$$

From the equations (5.6)-(5.8) and (5.16), it can be observed that, for the computation of a block of decisions, decisions within the same block are required. In other words, the decisions corresponding to the terms of the upper triangular matrix $\nabla\mathbf{V}_1(k)$ are unknowns for the current block of inputs to the FBF. Hence, the block evaluation of $\hat{\mathbf{v}}(k)$ becomes a "non-causal" operation. To avoid this causality problem, an iterative procedure was proposed in [7] where the unknown decisions are properly initialized with tentative decisions using two initialization schemes namely, initialization scheme 1 (IS1) and initialization scheme 2 (IS2). In IS1, the initial tentative decisions are set to zero vector to compute the current block of decisions. The unknown decisions so obtained are used to compute the next block of decisions. This procedure is continued until the block of decisions in two successive iterations are matched. However, if convergence in a reduced number of steps is desirable, an alternative approach IS2 was used by exploiting all the available information. In IS2, the initial tentative decisions are chosen by removing the quantizer operation $Q[.]$ in (5.8), i.e., by setting $\mathbf{d}(k) = s(k)$. This involves computation of inverse of an $L \times L$ matrix. To avoid this complexity, we propose an approach that uses a cost function which is minimized based on either of the two criteria namely absolute difference (MAD) and mean-square difference (MSD). The approach is based on the fact that each of the unknown decisions can assume one of the symbol values out of the set of symbol values used in the modulation scheme employed at the transmitter. The algorithm for the computation of the unknown decisions based on this approach is shown in 5.3. This approach

requires a bank of registers as shown in Fig.Fig. 5.5 and 5.6 where all the symbols used in the modulation scheme are stored. In case of MAD, the cost function is chosen to the absolute mean of the difference between the desired signal value and the output of the adder that sums up FFF's and FBF's outputs. Similarly, in case of MSD, the cost function is chosen to be the absolute mean of the squares of the difference between the desired value and the adder output. Here, the desired sequence in case of training mode is the delayed version of the transmitted symbol sequence and ADFE output sequence in case of decision directed mode.

By minimization of either of these cost functions, each of the $L-1$ unknown decisions are initialized by successively using all the combinations of '$L-1$' symbol values. The set of '$L-1$' unknown decision are directly assigned to those values of the symbols to which the cost function is minimum. The processing elements (PE) for MAD and MSD are shown in Fig. 5.5 and 5.6 respectively. Apart from the bank of registers that store all the symbol values, the PE in case of MAD as shown in Fig. 5.5 consists of a multi-input NOR gate and register with chip enable input. The operation of the processing element is as follows. The subtractor performs the difference of the training input (sample from previous symbol cycle) and the output of memory the address of which is generated from a counter (typically from control circuitry). The multi-input NOR gate checks whether the difference (cost function) is zero or not. The counter provides the address values starting from zero to the last location and accordingly the register passes that value of the symbol value for which the difference is zero. In case of MSD, the PE requires an extra adder, multiplier and a register for performing the squaring and accumulation operations which is shown in Fig. 5.6. The presence of decisions computing block which works on either MAD/MSD criterion in the decision feedback loop is shown in Fig. 5.4.

5.4 FORMULATION OF THE DA BASED BLOCK ADFE

The frequency domain equations describing the operation of the block ADFE of Fig. 5.2 are given as

$$\hat{\mathbf{x}}(k) = \mathbf{R}_{0,L}\mathcal{F}^{-1}\left\{\mathbf{X}_{\mathcal{F}}(k)\,\mathbf{w}^{f}_{\mathcal{F}}(k)\right\} \tag{5.19}$$

$$\hat{\mathbf{v}}(k) = \mathbf{R}_{0,L}\mathcal{F}^{-1}\left\{\mathbf{V}_{\mathcal{F}}(k)\,\mathbf{w}^{b}_{\mathcal{F}}(k)\right\} \tag{5.20}$$

$$\mathbf{s}(k) = \hat{\mathbf{x}}(k) - \hat{\mathbf{v}}(k) \tag{5.21}$$

$$\mathbf{v}(k) = \mathbf{d}(k-(1/L)) \tag{5.22}$$

$$\mathbf{d}(k) = Q[\mathbf{s}(k)] \tag{5.23}$$

$$\mathbf{e}(k) = \hat{\mathbf{d}}(k) - \mathbf{s}(k) \tag{5.24}$$

```
1: loop
     for k = 1 : (N/L) do % Block index
      for i = 1 : number_of_all_symbol_combinations do
       decisions(((k − 1) ∗ L + 2) : ((k − 1) ∗ L + L))) = all_comb_decisions(i);
       fbf_output = filter((decisions(((k − 1) ∗ L + 2) : ((k − 1) ∗ L + L)));
       adder_output = fff_output + fbf_output;
       for j = 1 : L do
        slicer_output(j) = bpsk_decision_device(adder_output(j));
       end for
       decisions((k − 1) ∗ L + 1 : k ∗ L) = slicer_output;
       e_vec = desired_signal_train((k − 1) ∗ L + 1 : k ∗ L) − adder_output;
       % For MAD case
       count = 0;
       for iii = 1 : L do
        if(abs(e_vec(iii)) == 0)
         count = count + 1;
        end
       end for
       if (count == L)
        break;
       end if
       % For MSD case
       mse = 0;
       for iii = 1 : L do
        mse = mse + (e_vec(iii))^2;
       end for
       if (count == L);
        break;
       end if
      end for
     end for
2: end loop
```

Figure 5.3: Algorithm for the computation of unknown decisions.

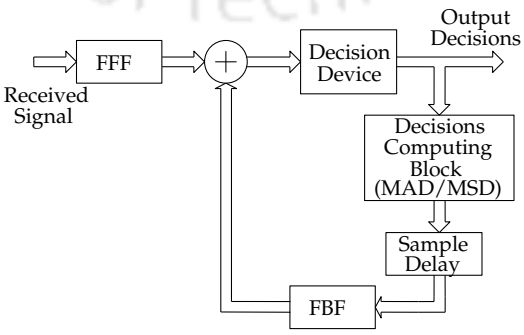

Figure 5.4: Block ADFE with the decisions computing block in the feedback loop.

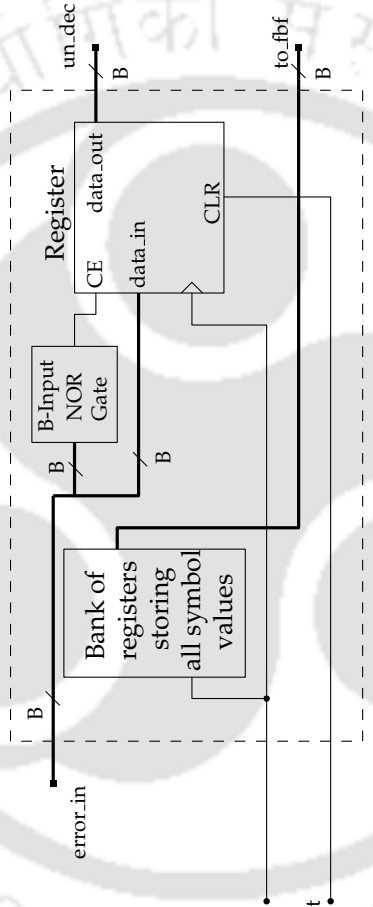

Figure 5.5: Processing Element in case of MAD.

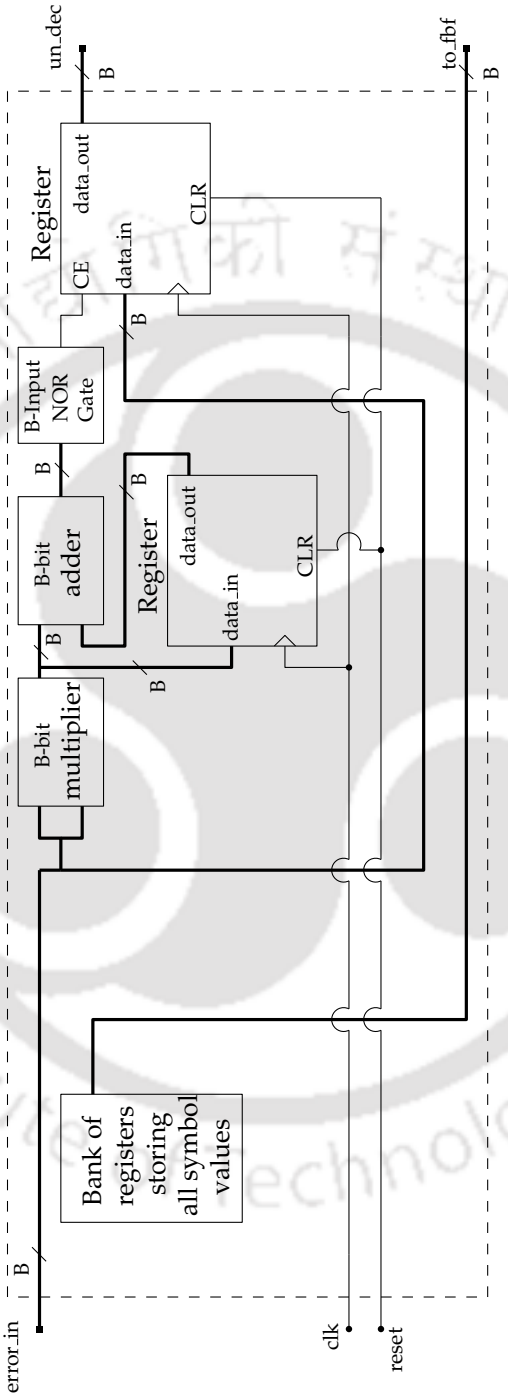

Figure 5.6: Processing Element in case of MSD.

where, $Q[.]$ is the quantization operation of the L-dimensional decision device and $\hat{\mathbf{d}}(k)$ is the vector containing the corresponding samples from the original transmitted sequence in case of training mode and the decisions vector $\mathbf{d}(k)$ in case of decision-directed mode. Further, $\hat{\mathbf{x}}(k)$, $\hat{\mathbf{v}}(k)$ are the output vectors of FFF and FBF respectively. The matrix $\mathbf{R}_{0,L}$ is used for the selection of the last L valid samples of the filter output vector (as obtained from the overlap-save method) and is given as

$$\mathbf{R}_{0,L} = \left[\begin{array}{c|c} \mathbf{0} & \mathbf{I}_L \end{array} \right] \tag{5.25}$$

where $\mathbf{0}$ is the $L \times (N-1)$-dimensional ($N = N_f$ for FFF and $N = N_b$ for FBF) all-zero matrix and $\mathbf{I}_L$ is the L-dimensional identity matrix.

The matrices $\mathcal{F}$ and $\mathcal{F}^{-1}$ are respectively the $M \times M$ ($M = N_f + L - 1$ in case of FFF and $M = N_b + L - 1$ in case of FBF)-dimensional Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) matrices which may be given as,

$$\mathcal{F} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & e^{-j2\pi/M} & \dots & e^{-j2\pi(M-1)/M} \\ 1 & e^{-j4\pi/M} & \dots & e^{-j4\pi(M-1)/M} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{-j2\pi(M-1)/M} & \dots & e^{-j2\pi(M-1)^2/M} \end{bmatrix} \tag{5.26}$$

$$\mathcal{F}^{-1} = \frac{1}{M} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & e^{j2\pi/M} & \dots & e^{j2\pi(M-1)/M} \\ 1 & e^{j4\pi/M} & \dots & e^{j4\pi(M-1)/M} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{j2\pi(M-1)/M} & \dots & e^{j2\pi(M-1)^2/M} \end{bmatrix} \tag{5.27}$$

The matrices $\mathbf{X}_{\mathcal{F}}(k)$, $\mathbf{V}_{\mathcal{F}}(k)$ are respectively given as

$$\mathbf{X}_{\mathcal{F}}(k) = \mathcal{F}\mathbf{X}_c(k)\mathcal{F}^{-1} \tag{5.28}$$

$$\mathbf{V}_{\mathcal{F}}(k) = \mathcal{F}\mathbf{V}_c(k)\mathcal{F}^{-1} \tag{5.29}$$

where, $\mathbf{X}_c(k)$ and $\mathbf{V}_c(k)$ are the $(N_f + L - 1) \times (N_f + L - 1)$ and $(N_b + L - 1) \times (N_b + L - 1)$-dimensional circular matrices respectively, which are given as,

$$\mathbf{X}_c(k) = \begin{bmatrix} x(kL - N_f + 1) & \dots & x(kL + N_f - 2) \\ x(kL - N_f + 2) & \dots & x(kL - N_f + 3) \\ \vdots & \ddots & \vdots \\ x(kL + L - 1) & \dots & x(kL - N_f + 1) \end{bmatrix} \tag{5.30}$$

$$\mathbf{V}_c(k) = \begin{bmatrix} d(kL - N_b + 1) & \dots & d(kL + N_b - 2) \\ d(kL - N_b + 2) & \dots & d(kL - N_b + 3) \\ \vdots & \ddots & \vdots \\ d(kL + L - 1) & \dots & d(kL - N_b + 1) \end{bmatrix} \tag{5.31}$$

In (15) and (16), $\mathbf{w}_{\mathcal{F}}^{f}(k)$ and $\mathbf{w}_{\mathcal{F}}^{b}(k)$ are the vectors containing the frequency domain samples of the zero-padded tap-weight vectors of FFF and FBF respectively and are given as

$$\mathbf{w}_{\mathcal{F}}^{f}(k) = \mathcal{F}\tilde{\mathbf{w}}^{f} \tag{5.32}$$

$$\mathbf{w}_{\mathcal{F}}^{b}(k) = \mathcal{F}\tilde{\mathbf{w}}^{b} \tag{5.33}$$

and

$$\tilde{\mathbf{w}}^{f}(k) = \begin{bmatrix} \mathbf{w}^{f}(k) \\ \mathbf{0} \end{bmatrix} \tag{5.34}$$

$$\tilde{\mathbf{w}}^{b}(k) = \begin{bmatrix} \mathbf{w}^{b}(k) \\ \mathbf{0} \end{bmatrix} \tag{5.35}$$

where, $\mathbf{w}^{f}(k)$ and $\mathbf{w}^{b}(k)$ are the tap-weight vectors of FFF and FBF respectively.

From the properties of circular matrices, the matrices $\mathbf{X}_{\mathcal{F}}(k)$ and $\mathbf{V}_{\mathcal{F}}(k)$ will be the diagonal matrices and the diagonal elements correspond to the FFT of the first column of $\mathbf{X}_{c}(k)$ and $\mathbf{V}_{c}(k)$ respectively. In matrix notation, they may be written as

$$\mathbf{X}_{\mathcal{F}}(k) = diag\left[\mathbf{x}_{\mathcal{F}}(k)\right] \tag{5.36}$$

$$\mathbf{V}_{\mathcal{F}}(k) = diag\left[\mathbf{v}_{\mathcal{F}}(k)\right] \tag{5.37}$$

and

$$\mathbf{x}_{\mathcal{F}}(k) = \mathcal{F}\left\{\mathbf{x}(k)\right\} \tag{5.38}$$

$$\mathbf{v}_{\mathcal{F}}(k) = \mathcal{F}\left\{\mathbf{v}(k)\right\} \tag{5.39}$$

where,

$$\mathbf{x}(k) = \left[x\left(kL - N_f + 1\right), x\left(kL - N_f + 2\right), \ldots, x\left(kL + L - 1\right)\right]^{T} \tag{5.40}$$

$$\mathbf{v}(k) = \left[v\left(kL - N_f + 1\right), v\left(kL - N_f + 2\right), \ldots, v\left(kL + L - 1\right)\right]^{T} \tag{5.41}$$

are the first columns of $\mathbf{X}_{c}(k)$ and $\mathbf{V}_{c}(k)$ respectively.
Further, the weight-update recursion for FFF and FBF are respectively given by the equations,

$$\mathbf{w}_{\mathcal{F}}^{f}(k+1) = \mathbf{w}_{\mathcal{F}}^{f}(k) + \mu\mathbf{P}_{N_f,0}\mathbf{X}_{\mathcal{F}}^{*}(k)\,\mathbf{e}_{\mathcal{F}}(k) \tag{5.42}$$

$$\mathbf{w}_{\mathcal{F}}^{b}(k+1) = \mathbf{w}_{\mathcal{F}}^{b}(k) + \mu\mathbf{P}_{N_b,0}\mathbf{V}_{\mathcal{F}}^{*}(k)\,\mathbf{e}_{\mathcal{F}}(k) \tag{5.43}$$

where,

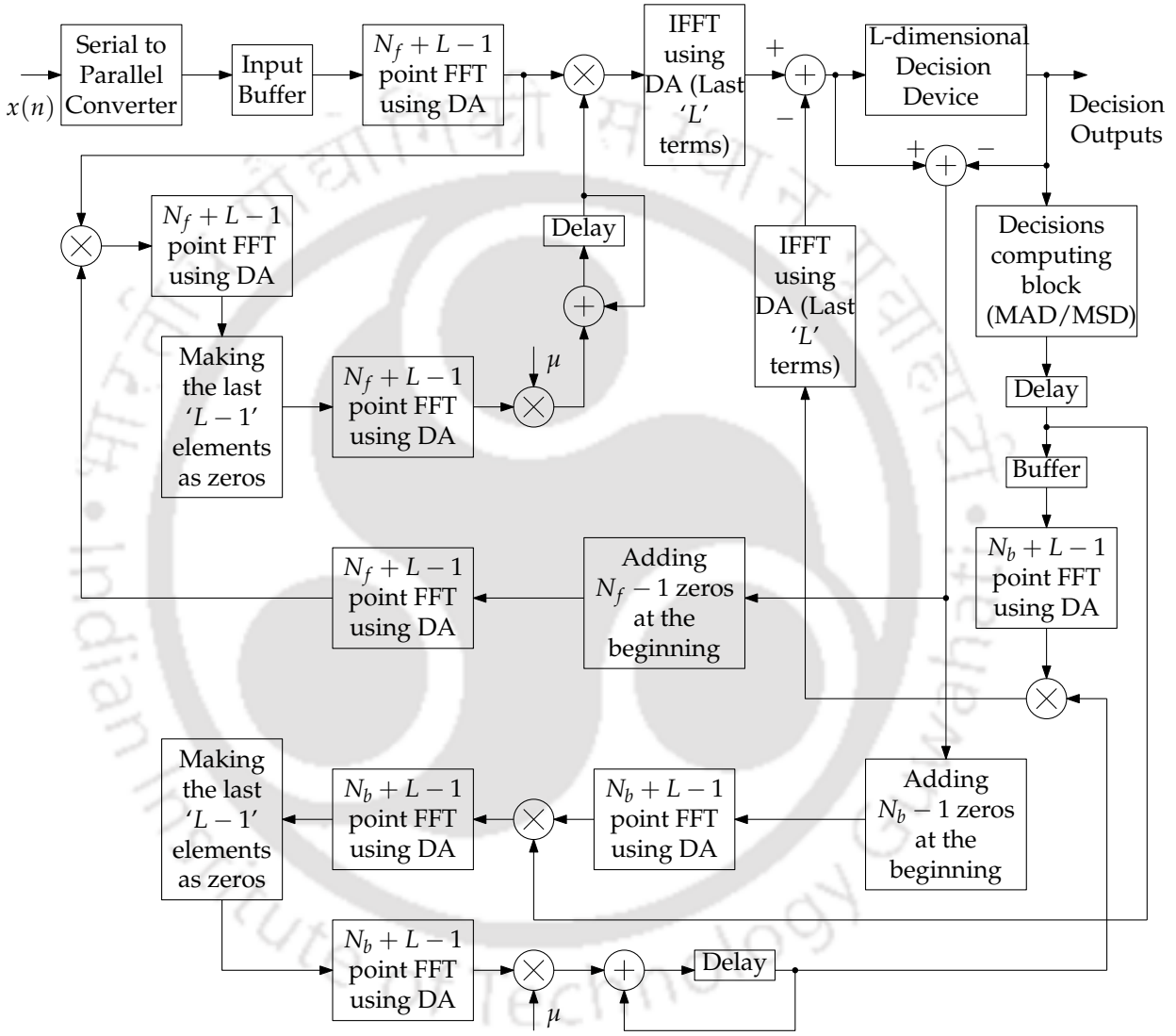

Figure 5.7: The block diagram of block ADFE implemented in the frequency domain.

$$\mathbf{e}_{\mathcal{F}}(k) = \mathcal{F}\tilde{\mathbf{e}}(k) \tag{5.44}$$

Here $\mathbf{X}^{*}_{\mathcal{F}}(k)$ and $\mathbf{V}^{*}_{\mathcal{F}}(k)$ represent the complex conjugates of $\mathbf{X}_{\mathcal{F}}(k)$ and $\mathbf{V}_{\mathcal{F}}(k)$ respectively. Further, $\tilde{\mathbf{e}}(k) = \begin{bmatrix} \mathbf{0} & \mathbf{e}(k) \end{bmatrix}^{T}$ and the matrices $\mathbf{P}_{N_f,0}$, $\mathbf{P}_{N_b,0}$ are required to ensure that the last $L-1$ samples of the IFFT of $\mathbf{w}^{f}_{\mathcal{F}}(k)$ and $\mathbf{w}^{b}_{\mathcal{F}}(k)$ are constrained to zeros.

Although, the derivations of frequency-domain block LMS based adaptive filters involve extending the vectors to a length of $L+N-1$ (N being the length of filter under consideration), in practice, the vectors are chosen to be of length $L+N$. Further, $L=N$ may be chosen for maximum efficiency where N is typically in the powers of 2. Hence, assuming N_f, N_b and L are all in powers of 2, the FFT/IFFT operations in (5.19), (5.20), (5.28), (5.29), (5.32), (5.33), (5.38) and (5.39) may be given as

$$\mathbf{a}_{\mathcal{F}} = \mathcal{F}\mathbf{a}_n = \frac{1}{\sqrt{M}} \sum_{n=0}^{M-1} a_n e^{-j\frac{2\pi}{M}kn} \tag{5.45}$$

where a_n is the nth element of vector a_n and $M = L+N$ and $N = N_f$ and $N = N_b$ in case of FFF and FBF respectively. Using the procedure described above, each of the FFT and IFFT operations may be realized using the distributed arithmetic technique for the efficient realization of block ADFE and this can be obtained as follows.
If each of a_n is represented in signed 2's-complement representation, as given by

$$a_n = -b_{n,B-1} + \sum_{j=1}^{B-1} b_{n,B-1-j} 2^{-j} \tag{5.46}$$

where $b_{n,B-1-j}$ is the $(B-1-j)$th-bit in the B-bit binary representation of a_n, then

$$\begin{aligned} a_n e^{-j\frac{2\pi}{M}kn} = &- \left[b_{n,B-1} e^{-j\frac{2\pi}{M}kn} \right] \\ &+ \sum_{j=1}^{B-1} \left[b_{n,B-1-j} e^{-j\frac{2\pi}{M}kn} \right] 2^{-j} \end{aligned} \tag{5.47}$$

Now, since $b_{n,B-1-j} \in [0,1]$, the expressions inside the square braces of above equations may take one out of 2 possible combinations (partial-products of twiddle factors) which may be stored in a memory as the twiddle factors are known constants prior to the implementation. Hence, (5.47) may be computed by right-shift (due to the term 2^{-j}) and accumulate (due to the summation) operations. This is known as the distributed arithmetic (DA) based realization and requires no hardware multiplier for its implementation. Hence, all the multipliers present in the FFT/IFFT units can be realized using DA and the IFFTs can also be realized using the same structure of FFT. When the filter lengths are not in the powers of 2, other FFT algorithms (such as the Prime-Factor FFT algorithm, Rader's FFT algorithm etc) may be chosen and the hardware complexity depends on the type of algorithm chosen. Such an implementation for block LMS based adaptive filter can be found in [5, 6].

The detailed block diagram of the block ADFE implemented in the frequency domain is shown in Fig. 5.7. The operation of FFF is as follows: The received samples arrive serially which are stored for parallel processing using a serial-to-parallel converter. These samples are buffered taking the newest set of L samples along with N_f-1 old samples for conversion into frequency domain using an FFT block as described by (5.38). The set of

FFF co-efficients which are appended with $L-1$ zeros are also converted into frequency domain using an FFT block as described by (5.32) and (5.34). These set of frequency domain samples of received signal along with the FFF coefficients are multiplied sample by sample using a multiplier and the multiplied set of samples are converted back into time domain using an IFFT block after which only the significant set of L samples are taken for further processing. The corresponding equation describing the above operations are given by (5.19). All these operations are also carried for FBF where the input samples are nothing but the decision outputs which are generated by the decisions computing block that operates on either the MAD or MSD criterion. Here the initial decision sample is taken to be any symbol value out of all the symbol values used and the FBF coefficients can be initialized to zeros or random values as is the case with FFF coefficients. A block of error samples are generated using outputs of FFF and FBF. These error samples are zero-padded at the beginning for conversion into frequency domain. These frequency domain error samples are used for the weight-update operations for FFF and FBF as described by (5.42) and (5.43).

5.5 PERFORMANCE ANALYSIS

To verify the validity of the proposed approach, we have simulated a channel-equalizer model in MATLAB. The channel is taken to be a raised-cosine AWGN channel which has the property of distorting the transmitted symbols. For simplicity of the decision device, the modulation scheme is chosen to be binary phase shift keying (BPSK) and the lengths of FFF and FBF are chosen to be 9 and 6 respectively. These choices are based on the linear prediction interpretation of the DFE as given in [24]. A total of 500 symbols are transmitted, out of which 60 are used for training mode and remaining are used for the decision-directed mode of operation. For better convergence of the FBLMS algorithm, step-normalization has been employed [19]. The mean square error in decibels (dB) for the proposed approach under different block lengths for MAD and MSD cases are shown in Fig. 5.8 and Fig. 5.9 respectively. These curves have been plotted by taking the average of 100 independent MATLAB simulations. The computational complexity of the proposed ADFE is as follows.

A single N-point FFT/IFFT block takes approximately $Nlog_2(N)$ multiplications and $Nlog_2(N)$ additions. Each B-bit multiplier unit when realized using DA requires $4B$ flip-flops (B is the bit-width of registers) and an adder unit. The feed forward filter contains 5 FFT/IFFT blocks and hence requires $10(N_f+L)log_2(N_f+L)$ adder and $20B(N_f+L)log_2(N_f+L)$ flip-flops. The filtering and weight-update operations for FFF takes (N_f+L) additions and $3(N_f+L)$ multiplications respectively. This gives a total of $10(N_f+L)log_2(N_f+L)+(N_f+L)$ adders, $3(N_f+L)$ multipliers and $20B(N_f+L)log_2(N_f+L)$ flip-flops for FFF alone. Similarly, FBF requires $10(N_b+L)log_2(N_b+L)+(N_b+L)$ adders, $3(N_b+L)$ multipliers and $20B(N_b+L)log_2(N_b+L)$ flops. The computation of the difference between the FFF output and FBF output and the calculation of the error (training or decision-directed mode) will take $2L$ additions each. Hence, the computational complexity of the block ADFE alone would be approximately $3(N_f+N_b+2L)$ multiplications,

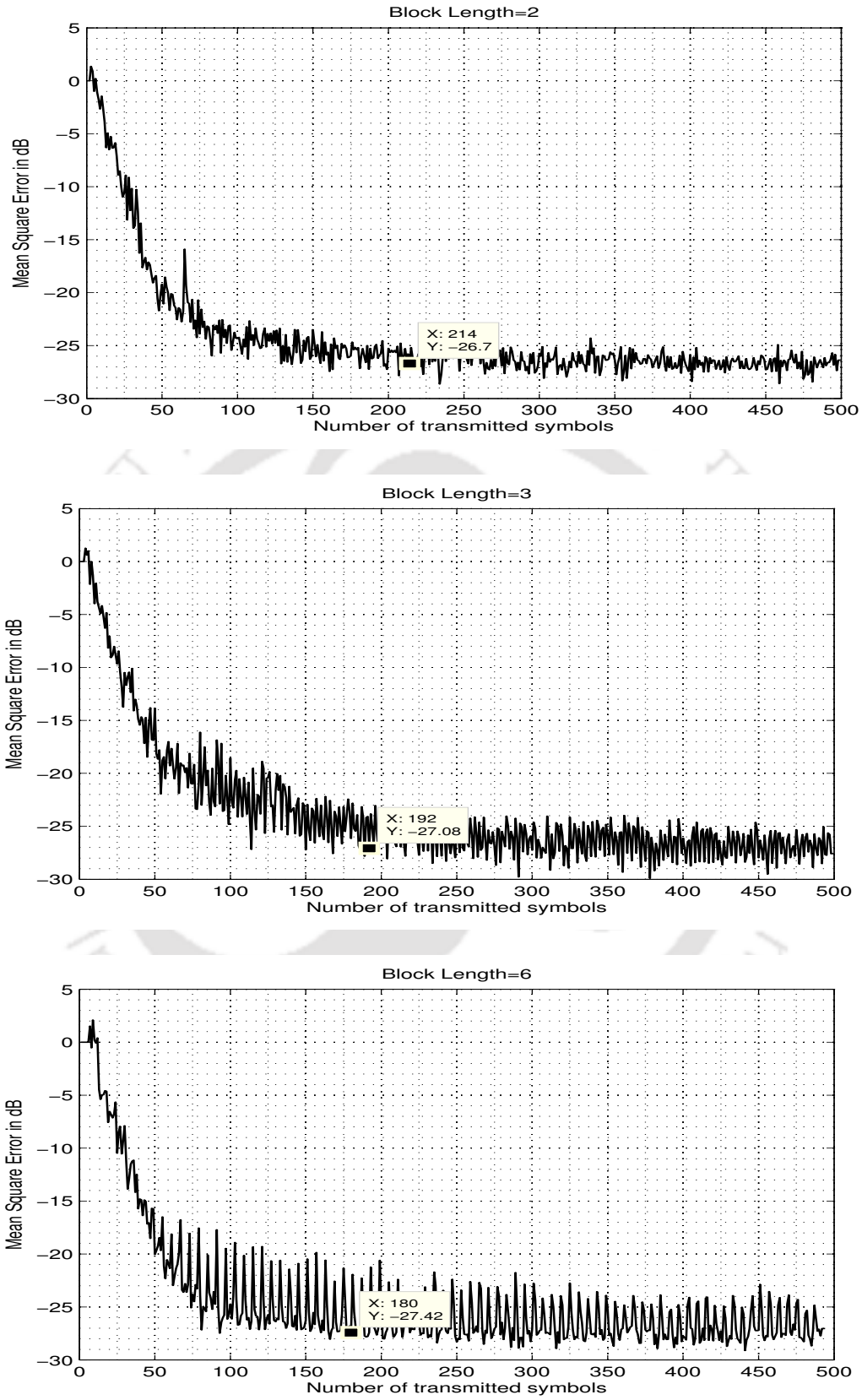

Figure 5.8: Convergence curves in case of MAD criterion.

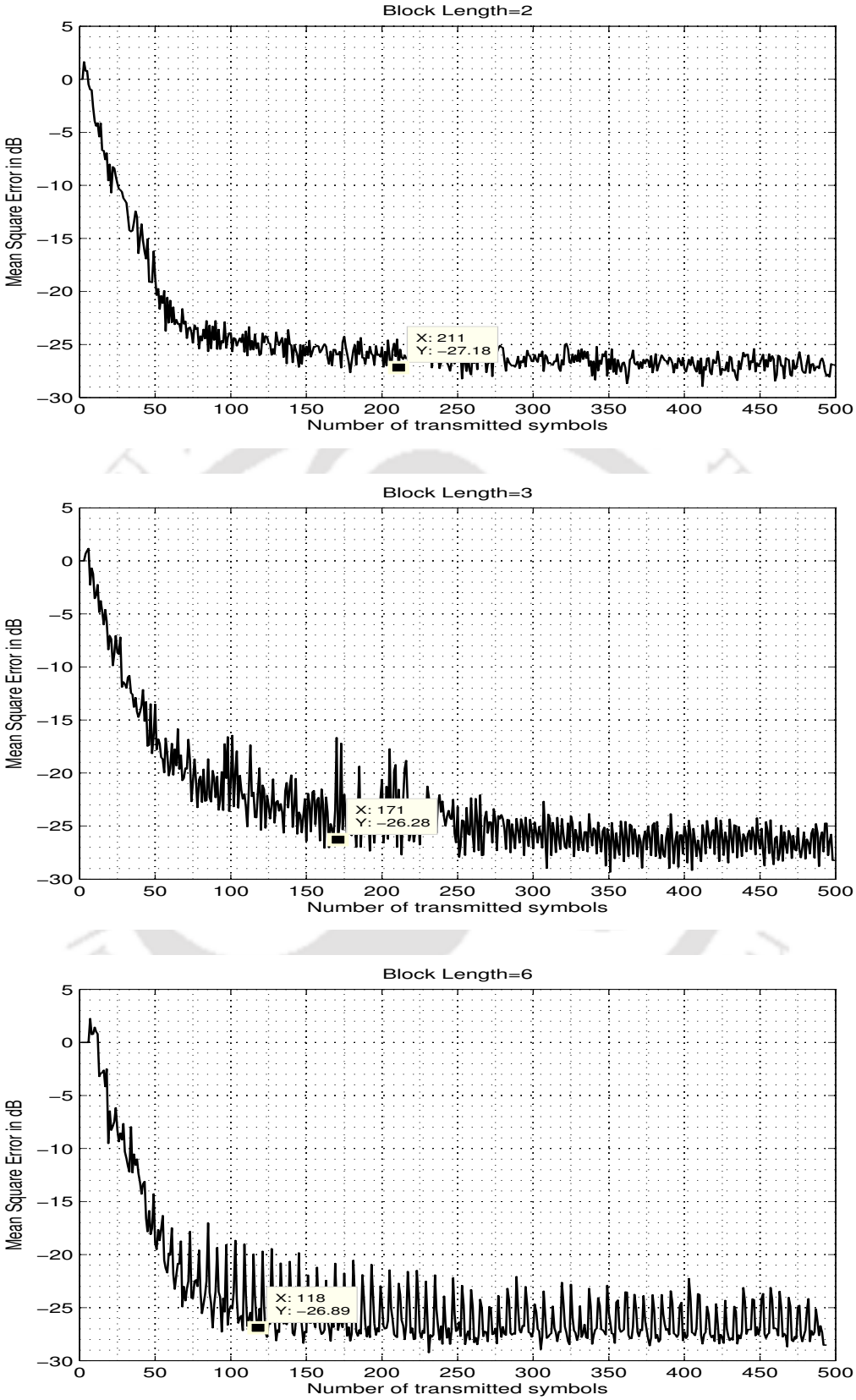

Figure 5.9: Convergence curves in case of MSD criterion.

$10(N_f + L)log_2(N_f + L)+10(N_b + L)log_2(N_b + L)+(N_f + N_b + 2L)$ addition operations and $20B(N_f + L)log_2(N_f + L)+20B(N_b + L)log_2(N_b + L)$ flip-flops.

Apart from this, the proposed approach in case of MAD requires the basic processing element as described previously. If we assume that at every stage of the system, the data is rounded-off to B bits and if N_t symbols are transmitted, a flip-flop based registers would take approximately N_tB flip-flops. The subtractor and the register in the basic processing element would take $B + 2$ flip-flops along with $(B - 1)$ NOR gates required for the multi-input NOR gate used in the PE. Hence, for the computation of the $L - 1$ unknown decisions, when PEs are used in parallel, this would take $N_tB\,(L - 1) + (B + 2)\,(L - 1)$ flip-flops and $(B - 1)\,(L - 1)$ NOR gates apart from the computations required for the basic ADFE structure as described in the previous paragraph.

In case of MSD, apart from the hardware required for the PE of MAD, a register, a multiplier and an adder are required. Hence, with parallelism, the system would take $(L - 1)$ extra adders, $N_tB\,(L - 1) + 2\,(L - 1)\,(B + 2)$ flip-flops, $(L - 1)$ multipliers and $(B - 1)\,(L - 1)$ NOR gates for the computation of unknown decisions. The total computational complexity of block ADFE for both MAD and MSD cases have been given in Tab. 5.1 and Tab. 5.2 respectively.

For practical channels with long channel impulse response such as the case of HDTV broadcasting, the lengths of tap-weights of FFF and FBF are very large. Let us consider the typical case where $N_f = 64$, $N_b = 128$ and $L = 128$ and assuming the modulation scheme to be 64-ary QAM, we have $N_t = 64$. In such case, a rough estimation of gate count in case of MAD and MSD are 7.36 million gates and 7.47 million gates respectively. Thus the proposed implementation, in comparison with the existing scheme [7] which takes 15.04 million gates has a computational savings of around 50%. However, the final set of FBF filter weights are not found to be exactly matching with the weiner solution although the error signal is getting converged. Hence, a correction factor is required which adjusts the final set of filters' weights to approach the weiner solution. As the proposed approach has a computational savings of 50% compared to the existing schemes, if the hardware required for the correction factor does not exceed a certain limit, then the proposed approach would be an efficient technique for the implementation of the block ADFE.

5.6 CONCLUSION

In this Chapter, an approach for the computation of the unknown decisions which appear in the block formulation of ADFE is presented. The unknown decisions are computed by the minimization of a cost function which can be chosen based on either of the two criteria namely mean absolute difference and mean square difference. For this, samples from the previous symbol cycle along with a memory containing all the symbols used in the modulation scheme are used. Using the presented approach for eliminating the causality problem and using DA, we implemented block ADFE. The DA based approach has good convergence characteristics and is computationally efficient.

Table 5.1: Total computational complexity for MAD case.

Multipliers	$3\left(N_f + N_b + 2L\right)$
Adders	$10\left(N_f + L\right) log_2\left(N_f + L\right) + 10\left(N_b + L\right) log_2\left(N_b + L\right) + \left(N_f + N_b + 2L\right)$
Flip-Flops	$20B\left(N_f + L\right) log_2\left(N_f + L\right) + 20B\left(N_b + L\right) log_2\left(N_b + L\right) + N_t B\left(L - 1\right) + \left(B + 2\right)\left(L - 1\right)$
2-Input NOR gates	$\left(B - 1\right)\left(L - 1\right)$

Table 5.2: Total computational complexity for MSD case.

Multipliers	$3\left(N_f + N_b + 2L\right) + (L-1)$
Adders	$10\left(N_f + L\right) log_2\left(N_f + L\right) + 10\left(N_b + L\right) log_2\left(N_b + L\right) + \left(N_f + N_b + 2L\right) + (L-1)$
Flip-Flops	$20B\left(N_f + L\right) log_2\left(N_f + L\right) + 20B\left(N_b + L\right) log_2\left(N_b + L\right) + N_t B\left(L-1\right) + 2\left(B+2\right)\left(L-1\right)$
2-Input NOR gates	$\left(B-1\right)\left(L-1\right)$

	Multipliers	Adders	2-input NOR gates
Proposed (MAD case)	$21N$	$30Nlog_2(3N) + 40Nlog_2(4N) + 7N$	$(B-1)(2N-1)$
Proposed (MSD case)	$23N-1$	$30Nlog_2(3N) + 40Nlog_2(4N) + 9N - 1$	$(B-1)(2N-1)$
Existing scheme 1 [7]	$29Nlog_2(N)$	$58Nlog_2(N) + 140N$	$24302Nlog_2(N) + 89210N$
Existing scheme 2 [7]	$45Nlog_2(N) + 161N$	$90Nlog_2(N) + 204N$	$37710Nlog_2(N) + 123826N$

Table 5.3: Total computational complexity for the proposed and existing schemes for the case $N_f = N$ and $N_b = L = 2N$.

6

CONCLUSION

6.1 SUMMARY OF THE PRESENT WORK

In this thesis, we have made an attempt to develop suitable DA treatment for efficient realization of adaptive decision feedback equalizer.

In Chapter 2, an efficient DA based realization of the LMS adaptive filter is presented with an objective of optimizing the adaptive filter for use later for the realization of LMS based ADFE. The proposed scheme uses the philosophy of [3] to store the partial products of recent input samples to aid the weight-update operation. Here, OBC scheme has been employed for the reducing the size of the look-up-tables. The error in each iteration is quantized to the powers of 2 which accounts for only a negligible loss in convergence performance. It is shown that the LUT storing the partial-products of input samples can be updated in every iteration using the most recent input sample along with circular shift of addressing bits of LUT. The performance analysis for different base filter sizes have been carried out through simulations. The proposed scheme enjoys considerable speed up over the existing schemes and also consumes less chip area.

The DA treatment for the realization of basic DFE has been considered in Chapter 3. To maximize the speed, the digit-serial nature of DA has been taken up. For this purpose, first, the filtering operations of FFF and FBF have been recast for DA treatment independently. Based on these equations, a direct-memory architecture has been developed where the lower half contents of the LUTs are mirror image to the upper half. Later, for reduced complexity, a new architecture namely reduced-memory architecture has been developed where one-half of the contents of the LUTs are generated alive using a combinational logic. A channel-equalizer model has been created for testing the DFE. The proposed DFE has been implemented on Altera® Cyclone III EP3C55F484C6 FPGA platform and synthesis on FPGA platform has been carried out for extensive range of FFF filter lengths. Both the proposed structures can operate at a higher frequency and consume less computational resources than the MAC based counterparts.

In Chapter 4, the DA treatment for the LMS based adaptive decision feedback equalizer has been carried out. For this purpose, the idea in Chapter 2 i.e., updating the look-up-table that stores the OBC combinations of filters' input samples is utilized. The architecture of the DA based LMS ADFE utilizes separate look-up-tables for filtering and weight-update operations in both FFF and FBF and the entire structure works on simple arithmetic operations such as addition and shifting operations. Further, the architecture is free of hardware multipliers and the convergence performance is observed to be no less than the traditional MAC based implementations.

Finally, in Chapter 5, a large objective of realizing the block ADFE using DA has been taken up. The inherent causality problem i.e., the presence of unknown decisions in every

iteration has been shown with detailed mathematics. These unknown decisions are computed using two minimization criteria namely mean absolute difference (MAD) and mean square difference (MSD). For this purpose, a memory storing all possible symbol values used in the modulation scheme at the transmitter has been used. Once, the unknown decisions are computed, it is shown that the entire structure can be implemented in the frequency domain where the DA philosophy has been applied to the multipliers present in the FFTs/IFFTs. The convergence analysis has been carried out through extensive simulations. It has been shown that the proposed structure uses upto 50% less computational resources but with some inconsistencies observed in the FBF coefficients approaching the weiner solution.

6.2 SCOPE OF THE FUTURE WORK

In this thesis, we have tried to demonstrate the effectiveness of DA scheme in a meaningful realization of the adaptive decision feedback equalizer. However, it is felt that there may exist considerable scope for extending the work reported here along various directions for future investigations. Some of them are outline below.

- This thesis has concentrated primarily on DA treatment for Least Mean Square (LMS) based ADFE. On the other hand, recursive least square (RLS) based ADFE which has superior performance over LMS ADFE has not been considered. It will therefore be pertinent to take up a similar exercise and develop means for DA based realization of RLS-based ADFE.
- In this thesis, for eliminating the causality problem in block ADFE, the set of unknown decisions in every iteration are checked with all the symbol values and here only the binary phase shift keying (BPSK) scheme has been used. Although, the FFF co-efficients are found to be converging to weiner solution, there are inconsistencies found in the FBF co-efficients approaching to weiner solution. Hence, a correction factor may be introduced which can be used to make all the co-efficients of FBF to approach the weiner solution.
- It would be interesting to take up the challenge of using complex modulation schemes such as M-ary PSK in which case the time complexity of block ADFE would be high.
- The primary focus of this thesis is the reduction of computational complexity and improving the speed of operation of ADFEs. On the other hand, application of low power techniques for the realization of ADFEs would be a good topic of research.
- The proposed DA based schemes may be extended to multi-input-multi-output (MIMO) equalization techniques.

A

APPENDIX A

A.1 AWGN

Additive White Gaussian Noise (AWGN) is an additive noise generally used to model communication channels. The spectral density of AWGN is a constant over all frequencies and the amplitude has a normal distribution with an average value of zero.

A.2 BINARY PHASE SHIFT KEYING

Binary Phase Shift Keying (BPSK) is the simplest form of phase-shift keying digital modulation scheme where two phases of carrier signal separated by 180 degrees are used one for transmitting a '0' and the other for transmitting a '1'.

A.3 CRITICAL PATH IN A DFG

The critical path in a data flow graph (DFG) of a DSP system is defined as the longest path between any two storage elements. It determines the minimum clock feasible for that DSP system.

A.4 FIXED-POINT $Qn.m$ FORMAT

$Qn.m$ is a fixed-point number format for the representation of digital data where n bits are used for integer part and m bis are used for fractional part.

A.5 THROUGHPUT OF A SYSTEM

The throughput of a system is defined as the number of samples processed per second. It is one of the measures of performance of a system. If t is the number of clock cycles taken by the system to process one sample, then the throughput can be given as

$$\text{Throughput} = \frac{\text{clock rate}}{t}$$

BIBLIOGRAPHY

[1] D. J. Allred, W. Huang, V. Krishnan, H. Yoo, and D. V. Anderson. An FPGA implementation for a high throughput adaptive filter using distributed arithmetic. In *Proc. 12th Annual IEEE Symp. Field-Programmable Custom Computing Machines FCCM 2004*, pages 324–325, 2004. doi: 10.1109/FCCM.2004.15. (Cited on page 14.)

[2] D. J. Allred, Heejong Yoo, V. Krishnan, W. Huang, and D. V. Anderson. A novel high performance distributed arithmetic adaptive filter implementation on an FPGA. In *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP '04)*, volume 5, 2004. doi: 10.1109/ICASSP.2004.1327072. (Cited on pages 14 and 19.)

[3] D. J. Allred, Heejong Yoo, V. Krishnan, W. Huang, and D. V. Anderson. LMS adaptive filters using distributed arithmetic for high throughput. *IEEE Trans. Circuits Syst. I, Reg. Papers*, 52(7):1327–1337, 2005. doi: 10.1109/TCSI.2005.851731. (Cited on pages 14, 16, 19, 27, 29, and 81.)

[4] M. Arjmand and A. Roberts, R. On Comparing Hardware Implementations of Fixed-Point Digital Filters. *IEEE Circuits Syst. Mag.*, 3(2):2–8, 1981. (Cited on page 14.)

[5] S. Baghel and R. Shaik. FPGA implementation of Fast Block LMS adaptive filter using Distributed Arithmetic for high throughput. In *Proc. Int Communications and Signal Processing (ICCSP) Conf*, pages 443–447, 2011. doi: 10.1109/ICCSP.2011.5739356. (Cited on page 73.)

[6] S. Baghel and R. Shaik. Low power and less complex implementation of fast block LMS adaptive filter using distributed arithmetic. In *2011 IEEE Students' Technology Symposium (TechSym)*, pages 214–219, Jan 2011. (Cited on page 73.)

[7] K. Berberidis and P. Karaivazoglou. An efficient block adaptive decision feedback equalizer implemented in the frequency domain. *IEEE Trans. Signal Process.*, 50(9): 2273–2285, 2002. doi: 10.1109/TSP.2002.801884. (Cited on pages 5, 61, 65, 77, and 80.)

[8] K. Berberidis and Jacques Palicot. A frequency-domain decision feedback equalizer for multipath echo cancellation. In *Global Telecommunications Conference, 1995. GLOBECOM '95., IEEE*, volume 1, Nov 1995. (Cited on page 61.)

[9] K. Berberidis, T. A. Rontogiannis, and S. Theodoridis. Efficient block implementation of the decision feedback equalizer. *IEEE Signal Process. Lett.*, 5(6):129–131, 1998. doi: 10.1109/97.681426. (Cited on page 5.)

[10] W.P. Burleson and L.L. Scharf. A VLSI implementation of a cellular rotator array. In *IEEE Custom Integrated Circuits Conf.*, pages 8.1/1–8.1/4, May 1988. (Cited on page 14.)

[11] H.-C. Chen, J.-I. Guo, C.-W. Jen, and T.-S. Chang. Distributed arithmetic realisation of cyclic convolution and its DFT application. *IEE Proceedings -Circuits, Devices and Systems*, pages 615–629, 2005. doi: doi:10.1049/ip-cds:20041173. (Cited on page 14.)

[12] Hun-Chen Chen, Jiun-In Guo, and Chien-Wei Jen. A new group distributed arithmetic design for the one dimensional discrete Fourier transform. In *Proc. IEEE Int. Symp. Circuits and Systems ISCAS 2002*, volume 1, 2002. doi: 10.1109/ISCAS.2002.1009867. (Cited on page 14.)

[13] Jung-Pil Choi, Seung-Cheol Shin, and Jin-Gyun Chung. Efficient ROM size reduction for distributed arithmetic. In *Proc. ISCAS 2000 Geneva Circuits and Systems The 2000 IEEE Int. Symp*, volume 2, pages 61–64, 2000. doi: 10.1109/ISCAS.2000.856258. (Cited on page 14.)

[14] G. Clark, S. Mitra, and S. Parker. Block implementation of adaptive digital filters. *IEEE Trans. Acoust., Speech, Signal Process.*, 28(6):584–592, 1981. doi: 10.1109/TCS.1981.1085018. (Cited on page 61.)

[15] C. F. N. Cowan and J. Mavor. New digital adaptive-filter implementation using distributed-arithmetic techniques. *IEE Proc. F Communications, Radar and Signal Processing*, 128(4):225–230, 1981. doi: 10.1049/ip-f-1:19810040. (Cited on pages 14, 16, and 19.)

[16] A. Croisier, D. J. Esteban, M. E. Levilion, and V. Rizo. Digital Filter for PCM encoded signals, April 1973. (Cited on page 14.)

[17] G. W. Davidson, D. D. Falconer, and A. U. H. Sheikh. An investigation of block-adaptive decision feedback equalization for frequency selective fading channels. In *Proc. IEEE Int Communications ICC '88. Digital Technology - Spanning the Universe. Conf. Record. Conf*, pages 360–365, 1988. doi: 10.1109/ICC.1988.13591. (Cited on page 4.)

[18] D. Falconer and S. L. Ariyavisitakul. Modulation and equalization criteria for 2-11 ghz fixed broadband wireless systems. In *Contribution to IEEE 802.16 Broadband Wireless Access Working Group, Document IEEE 802.16.3c-00/13*, Aug 2000. (Cited on page 61.)

[19] B. Farhang-Boroujeny. *Adaptive Filters Theory and Applications*. John Wiley & Sons, 1999. (Cited on page 74.)

[20] A. Gatherer and T. H.-Y. Meng. High sampling rate adaptive decision feedback equalizers. In *Proc. Int Acoustics, Speech, and Signal Processing ICASSP-90. Conf*, pages 909–912, 1990. doi: 10.1109/ICASSP.1990.115995. (Cited on page 4.)

[21] R. Guo and L. S. DeBrunner. Two High-Performance Adaptive Filter Implementation Schemes Using Distributed Arithmetic. *IEEE Trans. Circuits Syst. II, Exp. Briefs*, 58(9):600–604, 2011. doi: 10.1109/TCSII.2011.2161168. (Cited on pages 14, 16, 19, and 29.)

[22] Rui Guo and L. S. DeBrunner. A novel adaptive filter implementation scheme using distributed arithmetic. In *2011 Conf. Record of the Forty Fifth Asilomar Conf. on Signals, Systems and Computers (ASILOMAR)*, pages 160 –164, Nov. 2011. doi: 10.1109/ACSSC.2011.6189976. (Cited on pages 14 and 19.)

[23] Simon Haykin. *Adaptive Filter Theory*. Prentice-Hall, 1996. (Cited on page 19.)

[24] L. Honig and D.G. Messerschmitt. *Adaptive Filters: Structures, Algorithms and Applications*. Springer, 1984. (Cited on page 74.)

[25] N. Iqbal, N. Al-Dhahir, A. Zerguine, and A. Zidouri. Adaptive Frequency-Domain RLS DFE for Uplink MIMO SC-FDMA. *IEEE Trans. on Vehicular Tech.*, 64(7):2819–2833, July 2015. (Cited on page 6.)

[26] Douglas A. Irwin and Peter J. Klenow. Learning-by-Doing Spillovers in the Semiconductor Industry. *Journal of Political Economy*, 102(6):pp. 1200–1227, 1994. (Cited on pages xiii and 15.)

[27] P. Karaivazoglou, V. Paliouras, K. Karagianni, and K. Berberidis. Finite word length analysis of the EBA-DFE. In *Int. Symp. on Commun., Control and Sig. Process.*, pages 1067–1071, March 2008. (Cited on page 5.)

[28] C. S. Kumar, D. Madhavi, R. A. Shaik, and K. V. V. S. Reddy. A new sign normalized block based adaptive decision feedback equalizer for wireless communication systems. In *Proc. IEEE Int Computational Intelligence and Computing Research (ICCIC) Conf*, pages 1–6, 2010. doi: 10.1109/ICCIC.2010.5705722. (Cited on page 5.)

[29] Chih-Hsiu Lin, An-Yeu Wu, and Fan-Min Li. High-Performance VLSI Architecture of Decision Feedback Equalizer for Gigabit Systems. *IEEE Trans. Circuits Syst. II, Exp. Briefs*, 53(9):911–915, 2006. doi: 10.1109/TCSII.2006.881165. (Cited on page 60.)

[30] Y. C. Lin, M. T. Shiue, and S. J. Jou. 10Gbps decision feedback equalizer with dynamic lookahead decision loop. In *IEEE Int. Symp. on Circ. and Sys.*, pages 1839–1842, May 2009. (Cited on page 5.)

[31] A. Mandal and R. Mishra. Design of Complex Non-Linear Adaptive Equalizer in Mitigating Severe Intersymbol Interferences. *Sig. Process. Sys.*, 84:225–236, 2016. (Cited on page 6.)

[32] B.K. Mohanty and P.K. Meher. A High-Performance Energy-Efficient Architecture for FIR Adaptive Filter Based on New Distributed Arithmetic Formulation of Block LMS Algorithm. *IEEE Trans. Signal Process.*, 61(4):921–932, Feb 2013. (Cited on pages 14 and 15.)

[33] Mrityunjoy Chakraborty and Suraiya Pervin. Pipelining the adaptive decision feedback equalizer with zero latency. *Signal Processing*, 83(12):2675 – 2681, 2003. (Cited on page 60.)

[34] K. K. Parhi. Pipelining in algorithms with quantizer loops. *IEEE Trans. Circuits Syst.*, 38(7):745–754, 1991. doi: 10.1109/31.135746. (Cited on page 4.)

[35] K. K. Parhi. Design of multigigabit multiplexer-loop-based decision feedback equalizers. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 13(4):489–493, 2005. doi: 10.1109/TVLSI.2004.842935. (Cited on page 5.)

[36] K.K. Parhi. *VLSI Digital Signal Processing Systems: Design And Implementation*. Wiley India Pvt. Limited, 2007. ISBN 9788126510986. (Cited on pages 9 and 13.)

[37] A. Peled and Bede Liu. A new approach to the realization of nonrecursive digital filters. *IEEE Trans. Audio Electroacoust.*, 21(6):477–484, 1973. doi: 10.1109/TAU.1973.1162521. (Cited on page 14.)

[38] S. U. H. Qureshi. Adaptive equalization. *IEEE Proc.*, 73(9):1349–1387, 1985. doi: 10.1109/PROC.1985.13298. (Cited on page 3.)

[39] K. J. Raghunath and K. K. Parhi. Parallel adaptive decision feedback equalizers. *IEEE Trans. Signal Process.*, 41(5):1956–1961, 1993. doi: 10.1109/78.215315. (Cited on page 4.)

[40] M. Rawski, M. Wojtynski, T. Wojciechowski, and P. Majkowski. Distributed Arithmetic Based Implementation of Fourier Transform Targeted at FPGA Architectures. In *Proc. 14th Int. Conf. Mixed Design of Integrated Circuits and Systems MIXDES '07*, pages 152–156, 2007. doi: 10.1109/MIXDES.2007.4286139. (Cited on page 14.)

[41] AS Remya Ajai and Nithin Nagaraj. A Novel Methodology for Memory Reduction in Distributed Arithmetic Based Discrete Wavelet Transform. *Procedia Engineering*, 30: 226–233, 2012. (Cited on page 14.)

[42] A. A. Rontogiannis and K. Berberidis. Efficient decision feedback equalization for sparse wireless channels. *IEEE Trans. Wireless Commun.*, 2(3):570–581, 2003. doi: 10.1109/TWC.2003.811189. (Cited on page 5.)

[43] R. Shaik, M. Chakraborty, and S. Chattopadhyay. An efficient finite precision realization of the block adaptive decision feedback equalizer. In *2008. ISCAS 2008. IEEE Int. Symp. on Circuits and Systems*, pages 1910–1913, May 2008. (Cited on page 5.)

[44] Rafi Ahamed Shaik and Mrityunjoy Chakraborty. A block floating point treatment to finite precision realization of the adaptive decision feedback equalizer. *Signal Processing*, 93(5):1162 – 1171, 2013. (Cited on pages 6 and 60.)

[45] N. R. Shanbhag and K. K. Parhi. Pipelined adaptive DFE architectures using relaxed look-ahead. *IEEE Trans. Signal Process.*, 43(6):1368–1385, 1995. doi: 10.1109/78.388851. (Cited on page 4.)

[46] Naresh R Shanbhag and Keshab K Parhi. *Pipelined Adaptive Digital Filters*. Kluwer Academic Publishers, 1994. (Cited on page 4.)

[47] J.J. Shynk. Frequency-domain and multirate adaptive filtering. *IEEE Signal Processing Mag.*, 9(1):14–37, Jan 1992. (Cited on page 61.)

[48] G. Sicuranza and G. Ramponi. Adaptive nonlinear digital filters using distributed arithmetic. 34(3):518–526, 1986. doi: 10.1109/TASSP.1986.1164852. (Cited on pages 14 and 19.)

[49] S.G. Smith and S. A. White. Hardware approaches to vector plane rotation. In *Int. Conf. on Acoust., Speech, Signal Process. (ICASSP)*, pages 2128–2131 vol.4, April 1988. (Cited on page 14.)

[50] B.S. Tan and G.J. Hawkins. Speed-optimised microprocessor implementation of a digital filter. *IEEE Proc. Computers and Digital Techniques*, 128(3):85–93, May 1981. (Cited on page 14.)

[51] Lars Wanhammar. Implementation of wave digital filters using distributed arithmetic. *Signal Processing*, 2(3):253 – 260, July 1980. (Cited on page 14.)

[52] C.-H. Wei and J.-J. Lou. Multimemory block structure for implementing a digital adaptive filter using distributed arithmetic. *IEE Proc. G Electronic Circuits and Systems*, 133(1):19–26, 1986. doi: 10.1049/ip-g-1:19860003. (Cited on pages 14, 16, and 19.)

[53] S. A. White. Applications of distributed arithmetic to digital signal processing: a tutorial review. *IEEE ASSP Mag.*, 6(3):4–19, 1989. doi: 10.1109/53.29648. (Cited on pages 10, 13, and 14.)

[54] Jiafeng Xie, Jianjun He, and Guanzheng Tan. FPGA realization of FIR filters for high-speed and medium-speed by using modified distributed arithmetic architectures. *Microelectronics journal*, 41(6):365–370, 2010. (Cited on page 14.)

[55] Meng-Da Yang, An-Yeu Wu, and Jyh-Ting Lai. Fast convergent pipelined adaptive DFE architecture using post-cursor processing filter technique. *IEEE Trans. on Circ. and Sys. II: Exp. Briefs*, 51(2):57–60, Feb 2004. (Cited on page 5.)

[56] Sungwook Yu and Jr. Swartziander, E. E. DCT implementation with distributed arithmetic. *IEEE Trans. Comput.*, 50(9):985–991, 2001. doi: 10.1109/12.954513. (Cited on page 14.)

[57] J. Zeman and H.Troy Nagle. A High-Speed Microprogrammable Digital Signal Processor Employing Distributed Arithmetic. *IEEE J. Solid-State Circuits*, 15(1):70–80, Febraury 1980. (Cited on page 14.)

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

International Peer Reviewed Journals:

1. M. Prakash and R. Shaik, "Low-area and high-throughput architecture for an adaptive filter using distributed arithmetic," *IEEE Transactions on Circuits and Systems II: Express Briefs.*, vol. 60, no. 11, pp. 781–785, Nov 2013.
2. M. Prakash, R. Shaik and K. Sagar, "An Efficient Distributed Arithmetic based realization of the Decision Feedback Equalizer," *Circuits, Systems and Signal Processing, Springer*, vol. 35, issue. 2, pp. 603-618, Feb 2016.
3. M. S. Prakash and R. Shaik, "DA based approach for the implementation of block adaptive decision feedback equalizer," *IET Signal Processing*, vol. 10(6), pp. 676-684, Aug 2016.

National and International Peer Reviewed Conference Proceedings:

1. M. Surya Prakash and R. Shaik, "High performance architecture for LMS based adaptive filter using distributed arithmetic," in *International Conference on Information and Computer Applications (ICICA 2012)*, Hong Kong, vol. 24, pp. 18–22, Mar. 2012.
2. M. Prakash and R. Shaik, "Low complexity hardware architectural design for adaptive decision feedback equalizer using distributed arithmetic," in *IEEE International Conference on Computer Systems and Industrial Informatics (ICCSII)*, Dubai, pp. 1–5, Dec 2012.
3. M. Surya Prakash and R. Shaik, "A Distributed Arithmetic based Approach for the Implementation of the Sign-LMS Adaptive Filter," in *IEEE International Conference on Signal Processing and Communication Engineering Systems (SPACES 2015)*, India, pp. 326-330, Jan. 2015.

Manuscripts under preparation:

1. M. Prakash and R. Shaik, "Realization of Adaptive Decision Feedback Equalizer using Distributed Arithmetic," to be submitted after revision to IEEE Transactions on Signal Processing/IEEE Transactions on VLSI systems/IET Signal Processing/Elsevier Signal Processing.

COLOPHON

This thesis was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Final Version as of September 28, 2016 (`classicthesis` version 4.1).

MATCHA SURYA PRAKASH

AIM: To gain expertise in the areas of interest and to achieve perfection at work

PERSONAL DETAILS

Born	19 May, 1988	Vijayawada, India
Contact Info.	surya@iitg.ac.in, msp.iitg@gmail.com	+91-8500651905, +91-9085171359

EDUCATION

2009-2016	Doctor of Philosophy (Ph.D) from Indian Institute of Technology Guwahati	Research area on DSP Architectures	Course work Cummulative Performance Index (CPI): 7.5/10
2005-2009	Bachelor of Technology (B.Tech) from Narasaraopet Engineering College affiliated to Jawaharlal Nehru Technological University Kakinada	Specialization in Electronics and Communication Engineering (ECE)	Overall Percentage: 75.10

TECHNICAL SKILLS

Languages	C, Verilog HDL, MATLAB Programming
Tools	MATLAB®, Xilinx® ISE, Synopsys® Design Compiler
Areas of Interest	Signal Processing, Architectures for DSP
Operating Systems	Linux (Ubuntu), Microsoft® Windows XP/Vista/7
Other Tools and Packages	LaTeX, Microsoft® Office, Altera® Quartus, Synopsys® VCS, Synopsys® IC Compiler, Bluespec Compiler

PROFESSIONAL EXPERIENCE

2009-2016	Teaching Assistantship at IIT Guwahati Worked as a teaching assistant for the following courses of B.Tech and M.Tech: Digital Circuits Laboratory, VLSI Digital Signal Processing Laboratory, VLSI Design Laboratory, Embedded Systems Laboratory, Signals, Systems and Networks, Digital Signal Processors Laboratory, Basic Electronics Laboratory
Sept. 2011	Workshop at IIT Guwahati Worked as an organizer under Dr. Shaik Rafi Ahamed for the Quality Improvement Programme Short Term course on "VLSI Architectures for Image and Video Processing Systems"

PERSONAL ACHIEVEMENTS

Scored 95.51 and 97.26 percentile in Graduate Aptitude Test in Engineering (GATE) conducted in the years 2008 and 2009 respectively.

INTERESTS

Programming in MATLAB & Verilog HDL, Spiritual Science, Astronomy, playing Chess, Origami, Android Customization