LOW COMPLEXITY DISTRIBUTED ARITHMETIC BASED PIPELINED VLSI ARCHITECTURES FOR LMS ADAPTIVE FILTERS

MOHD. TASLEEM KHAN

Low Complexity Distributed Arithmetic based Pipelined VLSI Architectures for LMS Adaptive Filters

A

Thesis submitted

for the award of the degree of

DOCTOR OF PHILOSOPHY

By

MOHD. TASLEEM KHAN

DEPARTMENT OF ELECTRONICS AND ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, ASSAM, INDIA

JULY 2019

Certificate

This is to certify that the thesis entitled "**Low Complexity Distributed Arithmetic based Pipelined VLSI Architectures for LMS Adaptive Filters**", submitted by **Mohd. Tasleem Khan** (136102022), 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.

Dated:

Guwahati.

Prof. Shaik Rafi Ahamed
Professor
Dept. of Electronics and Electrical Engg.
Indian Institute of Technology Guwahati
Guwahati - 781 039, Assam, India.

To

My dear parents

Mohd. Waseem Khan and Shamim Rabbani

for their love and support

&

My guide

Prof. Shaik Rafi Ahamed

for his guidance and inspiration

Acknowledgements

First and foremost, sincere thanks and gratitude to God Almighty! I would like to express my sincere thanks to the many people who have contributed to this effort. I am deeply indebted to *The Father, Mohd. Waseem Khan* and *The Mother, Shamim Rabbani* whose constant support and grace have helped me to excel in different stages of life from my childhood. Words are not enough to express my gratefulness for the sacrifices they have made on my behalf. They always asked every other week, for five years, with the same enthusiasm, how far along I was in research. I would like to thank my brothers and sisters for their loving support and help during the course of my research. A special thanks to my brother *Mohd. Faheem Khan* for his kind support during the hardships in my research life. The opportunities that they have given me and their unlimited sacrifices are the reasons where I am and what I have accomplished so far.

I feel great privilege in expressing my deepest and most sincere gratitude to my supervisor *Prof. Shaik Rafi Ahamed*, for his excellent guidance throughout my study which sparked various ideas during the course of my research. His kindness, dedication, hard work and attention to detail have been a great inspiration to me. My heartfelt thanks to you sir for the unlimited support and patience shown to me. I would particularly like to thank him for all his help in patiently and carefully correcting all my manuscripts. I have no doubts that finishing my degree in a proper and timely manner was impossible without his help, advice and suggestions.

I am also very thankful to my doctoral committee members *Prof. Roy Paily Palathinkal*, *Dr. Ramesh Kumar Sonkar* and *Prof. Girish Sampath Setlur* for sparing their precious time to evaluate the progress of my work. This research was supported by the Special Manpower Development Programme for Chip to System Design (C2SD-III) through the Ministry of Electronics and Information Technology, Govt. of India. Special gratitude goes to Prof. Roy Paily Palathinkal because software tools required for VLSI design are available through the aforementioned project under his investigation.

I would also like to thank the Head of the Department and the other faculty members for their kind help in carrying out this work. I am also grateful to all the members of the research and technical staff of the department without whose help I could not have completed this thesis. My special thanks to Josephine S madam for providing an excellent computing facility and various resources useful for the research work.

Thanks go out to all my friends at the VLSI-I, II Laboratories. They have always been around to provide useful suggestions, companionship and created a peaceful research environment. They all contributed directly or indirectly to this thesis, be it academic help and proofreading.

The support of many fellow students has helped me get through tough times during the course of my research life. My work in this remote place definitely would not be possible without their love and care that helped me to enjoy my new life in this IITG. Special thanks to some of these names include Mohd. Faheem Khan, Abshar Hasan, Suraj Kumar Mandal, Jinti Hazarika, Shalini Kumari, Himakshi Misra, Rituparna Choudhary, Prerna Pushpa, Mohammad Imran, Aqib Jawed, Debashree Kundu, Sudipta Bhattacharya, Gagan Deep Singh, Indrajit Das, Kamlesh Badiyari, Prateek Kumar Sharma, Pralay Chakrabarty, Satyajit Das, Srinu VD M, Rahul Sharma, Bhoopal Rao, Harikrishna Veldandi, Ajay Kumar Maddirala, Babita Jajodia, Rajan Singh, Vimal Kumar Singh, Ranendra Kumar Sarma, Raju Agarwal, Ravi Prakash, Anik Batabyal, Atanu Purkayastha, Abdul Basit, Himangshu Gogoi.

I may be called ungrateful person if I do not mention the electronics engineering community forums who have shared their ideas and websites Wikipedia and discussion groups which have helped me in various ways for the completion of thesis.

I have received some good suggestions and help from my senior Dr. Surya Prakash Matcha who later turned out to be the co-author for a publication.

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 my research work. Finally, I believe this research experience will greatly benefit my career in the future.

Mohd. Tasleem Khan

Abstract

Least-mean-square (LMS) algorithm is widely used in system identification, channel equalization, noise cancellation, and several other areas of digital signal processing due to its simplicity and ease of implementation. In most of the cases, LMS algorithm is not employed directly, rather a combination of LMS units in parallel or in series or in block is used to obtain the desired performance. For instance, the parallel combination of two LMS adaptive filters with different step-size has fast convergence and low steady-state error, the series combination of two LMS adaptive filters in the feedforward and the feedback topology has better performance against noise and interference, and the block implementation of LMS adaptive filters is used to realize higher order filters. Due to its ubiquitous applications, LMS adaptive filters have several implementation complexity issues which need to be addressed. One of them is that its order primarily determines the number of multipliers. For high-throughput applications, the number of multipliers required for the implementation would be very high, while the critical path has to be made shorter, thus the real-time implementation of such filters for higher order is a challenging task. Pipelining is, therefore, necessary for LMS adaptive filter to achieve the throughput requirements, but it has an adverse effect on the convergence rate and steady-state error. Thus, obtaining low-complexity LMS adaptive filters with good convergence performance without compromising the throughput is an interesting area of research. Distributed arithmetic (DA) is an efficient multiplierless approach for implementation of LMS adaptive filter. Such implementations enable to realize LMS adaptive filters with low-complexity and high throughput. DA based implementations basically consist of a look-up table (LUT) followed by a shift-accumulate (SA) unit. But, the direct usage of DA for complexity reduction of the adaptive filter, especially in high-throughput applications would be challenging, since time required to access LUT and to compute SA unit is significant. The modularity feature of DA makes it amenable for implementing on field-programmable gate arrays (FPGA) and

the design of application specific integrated circuit (ASIC).

In this thesis, we first derived three optimal complexity pipelined architectures for LMS adaptive filter using offset-binary-coding (OBC) DA. Although it is straightforward to pre-compute and store the filter partial products in LUT for the realization of non-adaptive filter such as finite impulse response (FIR) filter using DA, problem arises while generating them using hardware elements to overcome the access time of LUT. This is because the number of hardware elements required to generate the filter partial products for LUT-less design grows exponentially with filter order. To address this issue, we implemented partial products of input samples serially by representing the filter coefficients in OBC-form. But, this produces non-OBC terms at the output during some initial clock cycles due to pipelined nature of filter, which are subsequently corrected in the error computation unit. The reason for choosing the OBC scheme in the implementation of pipelined LMS adaptive filter is to exploit the redundancies between the partial products for higher radices. This also has an advantage of less computation time required by the SA unit. All the proposed architectures are extended for large order filter design with modification in the correction of non-OBC terms. Next, we applied two's complement (TC) DA to the pipelined realization of convex combination of two parallel LMS adaptive filters which greatly improves the convergence performance. Unlike conventional LMS algorithm, the combination of two LMS filter in parallel can provide fast convergence and low steady-state error by transferring the filter coefficients from one LMS unit to other. However, the computational requirements are significantly higher due to two LMS units and coefficient transfer scheme. To alleviate this problem, we employed a single DA based LMS adaptive filter with a new coefficient transfer scheme. Further reduction in computational complexity is achieved by sharing the partial products and employing bit-level coefficient update accumulators. The reduction in hardware elements is utilized to transfer the filter coefficients by switching the step-size. As a result, it enhances the filter convergence properties with an efficient criterion based on the correlation between adjacent delayed error samples. The duration of correlation between adjacent errors is then compared with a pre-defined time window in every iteration. In the sequel, an analytical expression for pre-defined time window is derived in terms of filter parameters such as filter order, coefficients wordlength

and step-size. Later, we considered low-complexity DA based VLSI implementation of the adaptive filter for channel equalization problem in 5G communication system. In this work, we employed two LMS adaptive filters in series with one in the feedforward path and other in the feedback path with a decision device. The overall system is designed to achieve the throughput requirement of 5G while maintaining computational complexity relatively lower than the best existing design. The design first utilizes the pipelined implementation of non-adaptive feedback filter using OBC. It is based on the fact that when radix-size becomes equal to the wordlength of decisions, the implementation of DA based LMS adaptive filter can be made SA-less. In this design, decisions are OBC coded to derive low-complexity SA-less architecture for adaptive feedback filter. The proposed architecture pre-computes and stores the coefficients in two LUTs separated by pipelined registers for complexity reduction. Further reduction in complexity is achieved by exploiting the symmetries between the stored contents of both the LUTs. The proposed architecture is pre-speed up by two, retimed and unfolded to meet the throughput requirements of 5G. To adapt the feedback coefficients, a novel strategy is presented to update the LUT contents of both the stages by adding them with the contents of external LUT storing the decisions. The contents of decision LUT are updated before the filtering operation in every iteration. Parallel error multiplexer is employed so as to remove the effect of non-OBC terms in order to improve the convergence performance. Lastly, a low-complexity design of pipelined block LMS adaptive filter is proposed for noise cancellation in in-ear headphones applications. Here, both physical LUT and SA unit are employed, and their effects on throughput performance are compensated by block processing. It utilizes OBC scheme for complexity reduction which stores the partial products of input samples in a LUT. The symmetries in LUT contents allowed them to split into two smaller LUTs, but requires the contents of the external register to be updated along with the error computation. The splitted LUTs are shared to compute the filter output and coefficient-increment terms in the same block iteration. A novel strategy is developed to update the LUT contents in fewer number of clock cycles as compared to the best existing design. To validate the performance of the proposed architectures, ASIC and FPGA implementations are carried out to estimate area, power, throughput, number of flip-flops and slice LUTs.

Contents

List of Figures

List of Tables

List of Acronyms

AD	Adaptation Delay
ADF	Adaptive Filter
ADFE	Adaptive Decision Feedback Equalizer
ADP	Area Delay Product
ASIC	Application Specific Integrated Circuit
BER	Bit Error Rate
BLMS	Block Least Mean Square
BS	Barrel Shifter
BiCom	Bit Complexity
ByCom	Byte Complexity
CSFA	Carry Save Full Adder
CCU	Convex Combination Unit
CLMS	Convex Combination Least Mean Square
CMOS	Complementary Metal Oxide-Semiconductor
CPD	Critical Path Delay
CUU	Coefficient Update Unit
DA	Distributed Arithmetic
DAT	Data Arrival Time
DFE	Decision Feedback Equalizer
DFL	Decision Feedback Loop
DLMS	Delayed Least Mean Square
DLUT	Decision LUT
ECU	Error Computation Unit
EPS	Energy Per Sample

EPO	Energy Per Output
FB	Feedback
FF	Feedforward
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FU	Filtering Unit
ISI	Inter Symbol Interference
ISR	Initial State Remover
LAD	Location of Adaptation Delay
LSB	Least Significant Bit
LMS	Least Mean Square
LUT	Look Up Table
LOD	LUT Output Delay
MAC	Multiply Accumulate
MMP	Minus Minus Plus
MSB	Most Significant Bit
MSE	Mean Square Error
MSP	Minimum Sampling Period
NCC	Number of Clock Cycles
NOC	Number of Clocks
OBC	Offset Binary Coding
OC	Offset Circuitry
PDP	Power Delay Product
PSAR	Pipelined Shift-Accumulate Based on Redundant Adder
PLUT	Parallel Look-up Table
PPG	Partial Product Generator
PR& SI	Pipelined Register and Sign Inversion
QAM	Quadrature Amplitude Modulation
RTL	Register Transfer Level
SA	Shift Accumulate

S-PLUT	Shared Parallel Look-up Table
SM	Sign Magnitude
SNR	Signal to Noise Ratio
TC	Two's Complement
TSMC	Taiwan Semiconductor Manufacturing Company
TSP	Two Stage Pre-computation
VMA	Vector Merge Adder
WPBS	Word Parallel Bit Serial

List of Symbols

α	Switching activity
β	Reduction in dynamic power consumption
B	Wordlength of input samples
c	Index to denote the component filters in CLMS ADF
C	Total capacitance
C_{chrg}	Charging or discharging capacitance in critical path
$d(n)$	Desired signal
$d_k(n)$	LUT contents
$d_{ot}(n)$	Offset term in OBC-DA based FIR filter
D	Delay operator
D_k	Delay operator whose contents are accessed at rate k
D_P	P number of delay operators
$\delta(n)$	Recent tentative decision
$\eta(n)$	Additive background noise
$e(n)$	Error signal
$\mathbb{E}[\bullet]$	Expectation operator
γ	Number of bits to represent the radix-size
i	Index to denote the inputs (or filter coefficients)
j	Index to denote the DA base unit
k	Index to denote a bit in filter coefficients
L	Order of DA base unit
M	Number of DA Base Unit
μ	Step-size of ADF
μ_0	Step-size of ADF_0

μ_1	Step-size of ADF_1
μ_f	Step-size of FF-ADF
μ_b	Step-size of FB-ADF
μ_B	Step-size of BLMS ADF
n	Discrete time signal time index
N	Order of FIR Filter
N_f	Order of FF-ADF
N_b	Order of FB-ADF
P_{total}	Total dynamic power consumption
$Q[\bullet]$	Quantizer operator
r	Radix size
$[\bullet]^T$	Transpose operator
T_A	Computational delay of an adder
T_{ACC}	LUT access delay
T_{FA}	Computational delay of a full adder
T_M	Computational delay of a 2-to-1 multiplexer
T_{MUL}	Computational delay of a multiplier
T_X	Computational delay of an XOR gate
$\mathbf{w}(n)$	Coefficient vector of LMS ADF
$\mathbf{w}^0(n)$	Coefficient vector of ADF_0
$\mathbf{w}^1(n)$	Coefficient vector of ADF_1
$\mathbf{w}^f(n)$	Coefficient vector of FF-ADF
$\mathbf{w}^b(n)$	Coefficient vector of FB-ADF
$w_i(n)$	Coefficients of LMS ADF
$w_i^0(n)$	Coefficients of LMS ADF_0
$w_i^1(n)$	Coefficients of LMS ADF_1
$w_i^f(n)$	Coefficients of FF-ADF
$w_i^b(n)$	Coefficients of FB-ADF
W	Wordlength of filter coefficients
$\mathbf{x}(n)$	Input vector

$\mathbf{X}(n)$	Input matrix
$y(n)$	Output of FIR filter
$y_0(n)$	Output of ADF_0
$y_1(n)$	Output of ADF_1
$y_f(n)$	Output of FF-ADF
$y_b(n)$	Output of FB-ADF
ζ	Pre-defined time window
$z(n)$	Output of ADFE

1

Introduction

Contents

1.1 Introduction

With the advent of the integrated circuit (IC) technology, the electronic components to be integrated become more and more complex. The complexity is further increased by the amount of functionality required for a given application. In the last five decades, the development in technology scaling has provided several million times higher level of integration. This trend is popularly known as "Moores Law" named after scientist Gordon Moore who made an observation that the number of transistors in an IC doubled every year [1]. However, as technology scales deep into the sub-micron regime, the impact of leakage power and process parameters on the performance of digital signal processing (DSP) algorithm starts unveiling. Thus, technology in a way has been facing challenges while the complexity of design keeps on increasing. Later, the advancement in device technologies such as multi-threshold devices have lifted up the effect of leakage power and process variation on the performance of DSP algorithms [2]. But, the increase in complexity of design has remained a problem due to the increased requirement of functionality on the ICs. The complexity of a digital design is measured in terms of hardware and time complexities. Basically, the hardware complexity depicts the number of computational elements involved in a particular design. While the time complexity indicates how long the design would take to compute the output for a given input. The analysis of design complexity is helpful when comparing with existing designs or seeking improvements. The study of design complexity falls within a branch of theoretical computer science known as computational complexity theory [3]. To reduce the complexity of a design, disruptive innovations and drastic improvements are being made at both algorithmic and architectural levels. The fundamental challenge is to map data processing algorithms onto the underlying hardware while meeting application constraints for area, power and throughput. Further, the emphasis is on very large scale integration (VLSI) implementation of these designs in reducing the complexity (or area and power) for a specified speed, and in increasing the speed of these circuits to make these designs suitable for high-throughput applications [4]. For instance, the reduction of area or power is important in time-multiplexed applications such as speech and mobile radio. On the other hand, video and radar applications require higher speed. This thesis addresses a similar problem in principle with an aim of reducing the complexity of pipelined least-mean-square (LMS) adaptive filter (ADF) using distributed arithmetic (DA).

The development in DSP systems particular to adaptive signal processing has been increasing enormously in the past few decades. This is mainly due to the demand of ADFs in applications such

as system identification, channel equalization, noise cancellation, signal prediction, adaptive arrays as well as many others [5, 6]. Certainly, this would not have been possible without VLSI technology to provide computational requirements for these applications [2]. A filter is said to be adaptive when its coefficients are modified with the arrival of every new sample based on a given algorithm. In other words, the adaptation algorithm to be chosen should follow the changes in the input signal. The most important aspect in the design of ADF is that it should be capable of operating in real-time environment. This requires the appropriate choice for the selection of some parameters [7] prior to its implementation such as:

- *Filter structure:* The relationship between input and output of the ADF is determined with its transfer function. The most widely used structure is finite impulse response (FIR) filter as it can implement almost any sort of frequency response. The other possible choices are FIR lattice and infinite impulse response (IIR) filters. However, the primary goal of this aspect is to determine the computational requirement for a given adaptive algorithm and the overall speed of the adaptation process.

- *Convergence rate and steady-state error:* Depending on the requirements, the coefficients of an ADF can be made to converge slow or fast to the optimum solution. In most of the scenarios, the coefficients of ADF do not attain the optimum values, rather reaches close to the optimum leading to a steady-state error. Hence, it always leads to an error in the steady-state due to the difference in estimated and actual values of filter coefficients. As a thumb rule, for a given algorithm, a faster convergence yields a higher steady-state error and vice-versa.

- *Complexity considerations:* The desired real-time characteristic of an ADF is determined through the computational requirements. Efforts are being made to obtain fast versions of more complex algorithms in view of simultaneously reducing the computational and memory requirements. Regardless of the hardware complexity that results from a particular implementation, the computational complexity of an ADF is determined by the requirements of the adaptation algorithm. When designing an ADF, it seems reasonable to seek an adaptation algorithm whose order of complexity is no greater than the order of complexity of the basic filter structure itself.

1.1.1 Applications of ADF

The basic block diagram of an ADF is shown in Fig. 1.1. It processes the input signal $x(n)$ and produces the output signal $y(n)$ which is subsequently used to compute the error signal $e(n)$ by subtracting from the desired signal $d(n)$, as per $e(n) = d(n) - y(n)$. The error signal is used by the adaptation algorithm to update the coefficient vector $\mathbf{w}(n)$ based on some error minimizing criteria. In particular, the adaptation process aims to minimize some metric of the error signal in order to make the output of ADF as close as to the desired signal in a statistical sense. The ADF as shown in Fig. 1.1 can be perfectly fit in several practical applications such as system identification, channel equalization, noise cancellation etc [5,6]. Each application requires the knowledge of underlying basic adaptive filtering framework. Precisely, the distinctive feature of each application is the primary factor on the basis of which the input signal and the desired signal of the ADF are chosen. Once these signals are determined, any known properties of them can be used to understand the expected behavior of the ADF when attempting to minimize the error function. In this thesis, DSP applications such as system identification, channel equalization and noise cancellation are considered to validate the proposed low-complexity architectures of ADF. Therefore, it is important to discuss them from ADFs point-of-view.

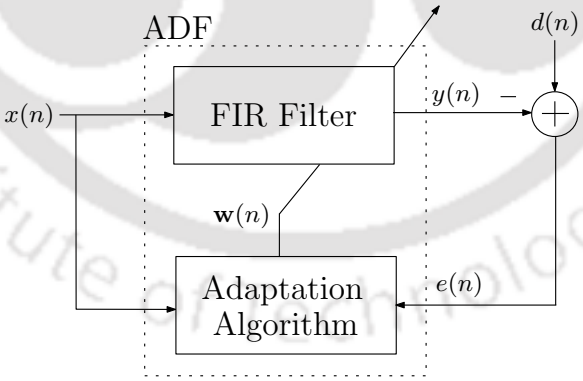


Fig. 1.1: Block diagram of an ADF.

1.1.1.1 System Identification

The typical set up of the system identification application is depicted in Fig. 1.2, where $\mathbf{w}^u(n)$ is the impulse response of the unknown system, $\eta(n)$ is the noise associated at the output of unknown system, $\mathbf{w}(n)$ is the coefficient vector of ADF, $y_u(n)$ is the output of the unknown system and $x(n)$ is the input to both the systems. Observably, the desired signal in this case is given by $d(n) = y_u(n) + \eta(n)$. Here

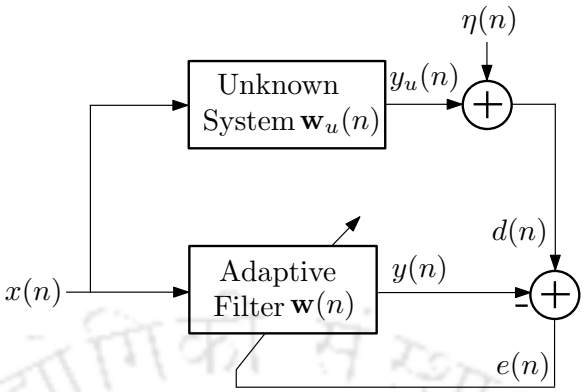


Fig. 1.2: Block diagram of system identification configuration of an ADF.

the ADF estimates the signal $y_u(n)$ at its output. If the output of ADF $y(n)$ approaches to $y_u(n)$, then the ADF is able to model a portion of the unknown system. The primary aim is to identify the best possible model of the unknown system for some set of system parameters. As the filter reaches to convergence, the desired response $d(n)$ is expressed as

$$d(n) = \mathbf{x}^T(n)\mathbf{w}_{opt}(n) + \eta(n) \tag{1.1}$$

where $\mathbf{w}_{opt}(n)$ is the optimum coefficient vector of the unknown system at time instant n. While identifying an unknown system, it is important for an ADF to have fast convergence and low steady-state error in order to provide close approximate at its output. The ideal adaptation procedure adjust $\mathbf{w}(n)$ such that $\mathbf{w}_{opt}(n) = \mathbf{w}(n)$ when n approaches to infinity. Some real-world applications of the system identification scheme include modeling of multi-path communication channels, control systems, seismic exploration etc [8, 9].

1.1.1.2 Channel Equalization

Most of the channels in communication system are band-limited in nature due to which distortions in the amplitude and phase of the transmitted signal are bound to occur. This may even lead to overlapping of symbols with nearby symbols over several signaling periods of time. The phenomenon of overlapping of successive symbols is commonly known as inter-symbol interference (ISI). If the transmission data rate is high, more and more symbols get overlapped, thereby inhibiting the transmission of data at higher rates. Channel equalization is therefore of prime importance for reliable detection of symbols. ADF is widely used in channel equalization to counter the ISI to decipher the received information. This can be explained by considering a system model of adaptive channel equalization

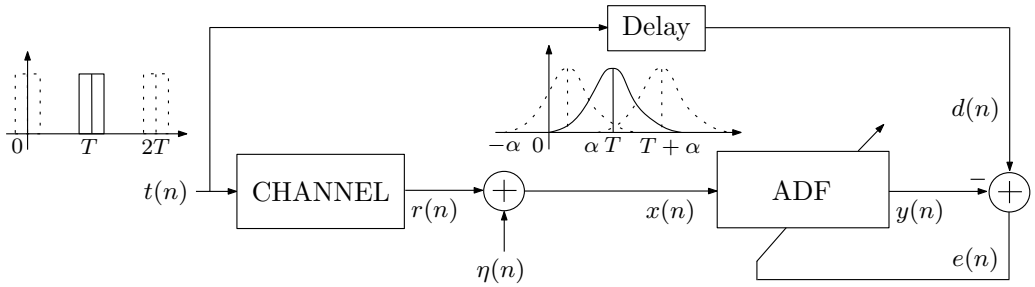


Fig. 1.3: Block diagram of channel equalization configuration of an ADF.

shown in Fig. 1.3 where $h(n)$ is the channel impulse response and $\eta(n)$ is the additive white noise in the channel. Note that the channel impulse response $h(n)$ contains causal, anti-causal component or both which results in pre-cursor ISI, post-cursor ISI component or both in the received symbol $x(n)$ respectively. However, in the presence of noise, the actual received symbol becomes $x(n) = r(n)+\eta(n)$. The ISI at the receiver can be eliminated by using an ADF whose transfer function is reciprocal of the channel transfer function. The direct application of ADF in channel equalization leads to noise amplification, especially when the channel contains the spectral (or deep) nulls. This can be well understood by calculating the error between the transmitted signal and the equalized signal $e(n) = t(n) - y(n)$. Assuming that the z-transforms of $e(n)$, $t(n)$ and $y(n)$ are respectively $E(z)$, $T(z)$ and $Y(z)$, we get

$$E(z) = T(z) - Y(z) \tag{1.2}$$

Substituting $Y(z) = X(z)W(z)$, $X(z) = R(z) + N(z)$ and $R(z) = T(z)H(z)$ in (1.2), we have

$$E(z) = T(z)(1 - H(z)W(z)) - N(z)W(z) \tag{1.3}$$

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

$$S_e(z) = S_t(z)|1 - H(z)W(z)|^2 + S_\eta(z)|W(z)|^2 \tag{1.4}$$

where $S_t(z)$ and $S_\eta(z)$ are the power spectra of the transmitted data symbols and the additive noise. It is clear from (1.4) that the ADF eliminates the ISI component from the power spectrum, if and only if $W(z) = 1/H(z)$. However, it leads to noise amplification at spectral nulls ($|H(z)| = 0$) or deep nulls ($|H(z)| \approx 0$) due to the second term in (1.4), where $|\bullet|$ denotes the magnitude operator. Moreover, such a configuration only eliminates the pre-cursor component of ISI. This has been addressed by a more complex system which is commonly known as adaptive decision feedback equalizer (ADFE) [10].

1.1.1.2.1 ADFE: It consists of two ADFs in series with one in feedforward (FF) path and other in feedback (FB) path with a decision device. It utilizes the regenerative effect of a decision device in the feedback loop to overcome the noise amplification. Hence, it can perform well even when channel contains spectral and/or deep nulls. The typical configuration of such filter is shown in Fig. 1.4. The purpose of FF-ADF is to remove the pre-cursor ISI component from the received data, while FB-ADF acts on the residual data to remove the post-cursor ISI component by subtracting from the output of FF-ADF, according to

$$z(n) = y_f(n) - y_b(n) \tag{1.5}$$

where $z(n)$ is the output of ADFE, $y_f(n) = \mathbf{x}^T(n)\mathbf{w}^f(n)$, $y_b(n) = \mathbf{d}^T(n)\mathbf{w}^b(n)$, $\mathbf{x}(n) = [x(n), x(n-1), ..., x(n-N_f+1)]^T$, $\mathbf{w}^f(n) = [w_0^f(n), w_1^f(n), ..., w_{N_f-1}^f(n)]^T$, $\mathbf{d}(n) = [\delta(n-1), \delta(n-2), ..., \delta(n-N_b)]^T$, $\mathbf{w}^b(n) = [w_0^b(n), w_1^b(n), ..., w_{N_b-1}^b(n)]^T$ and, N_f and N_b are the orders of FF and FB filters respectively.

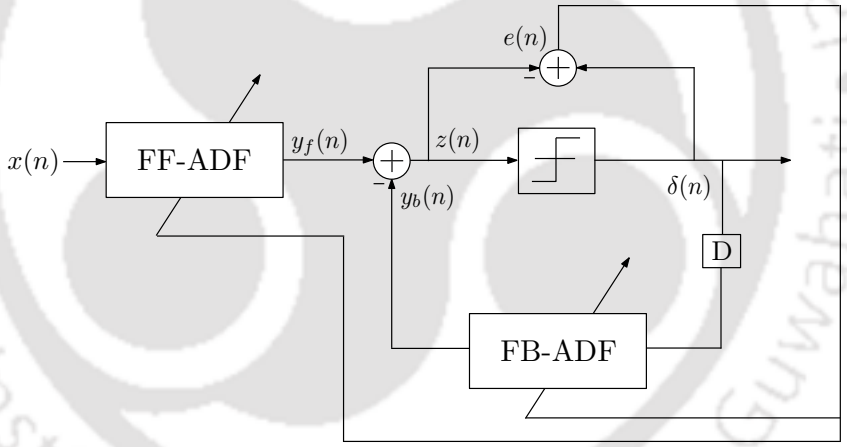


Fig. 1.4: Block diagram of conventional ADFE.

The most recent tentative decision $\delta(n)$ is obtained by quantizing the output of ADFE, according to

$$\delta(n) = Q[z(n)] \tag{1.6}$$

where $Q[\bullet]$ denotes the quantizer operation. An error signal $e(n)$ is computed, as per

$$e(n) = \delta(n) - z(n) \tag{1.7}$$

The coefficients of FF and FB filters are usually updated based on LMS criterion, according to

$$\mathbf{w}^f(n+1) = \mathbf{w}^f(n) + \mu_f e(n)\mathbf{x}(n) \tag{1.8}$$

$$\mathbf{w}^b(n+1) = \mathbf{w}^b(n) + \mu_b e(n)\mathbf{d}(n) \tag{1.9}$$

where μ_f and μ_b are the step-sizes of FF and FB filters respectively. It is clear that every iteration involves $2N_f + 2N_b + 2$ multiplications ($N_f + N_b$ for filtering, $N_f + N_b + 2$ for coefficient adaptation), $2N_f + 2N_b$ additions ($N_f + N_b - 2$ for filtering, $N_f + N_b$ for coefficient adaptation, 1 for filter output, 1 for error computation) and 1 quantizer (or decision device).

1.1.1.3 Noise Cancellation

The aim of noise cancellation is to eliminate the background noise by recreating its replica. In this, ADF plays a key role in estimating the noise over the time in order to cancel with the actual signal. A typical configuration of adaptive noise cancellation (ANC) of an ADF is shown in Fig. 1.5. The desired primary input signal comprises of the original loudspeaker $\hat{d}(n)$ and a component of interfered noise signal $\eta(n)$. The ADF is used to estimate the noise signal from the desired signal. Subtracting the estimated noise signal by ADF from the desired signal ideally leads to the actual signal. An error signal $e(n)$, is therefore calculated as

$$e(n) = \hat{d}(n) + \eta(n) - y(n) \tag{1.10}$$

This error signal $e(n)$ is used to update the filter coefficients until the mean square error is minimized.

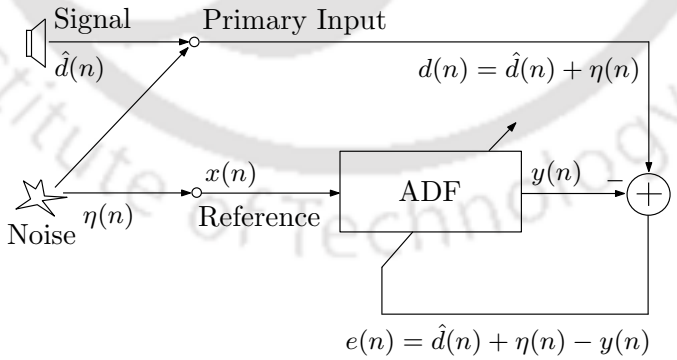


Fig. 1.5: Block diagram of adaptive noise cancellation configuration of an ADF [6].

1.2 LMS Algorithm

As stated, ADFs have numerous practical applications in many DSP and communication systems. The filter coefficients of ADFs can be updated using either LMS or recursive-least-square (RLS)

algorithm. The computational simplicity and ease of hardware implementation make the LMS based ADF more popular than RLS based ADF. In most of the cases, the direct application of standalone LMS ADF could not achieve the desired level of performance. For instance, it either provides fast convergence rate or low steady-state error for a given step-size in system identification; it leads to noise amplification in channel equalization, especially when the channel contains the spectral (or deep) nulls; it is computationally expensive to implement higher order filters in noise cancellation scenarios. To address these issues, either combination of ADFs in parallel or in series or in block based on LMS algorithm are usually employed depending upon the application. For instance, the combination of two ADFs in parallel has better convergence performance in terms of convergence rate and steady-state error. The combination of two ADFs in series has better performance against interference. Block approach is suitable for the implementation of higher order ADFs. However, the performance improvement is achieved at the cost of increased hardware complexities. In the following subsections, the LMS algorithm and its variants are presented along with their hardware complexities.

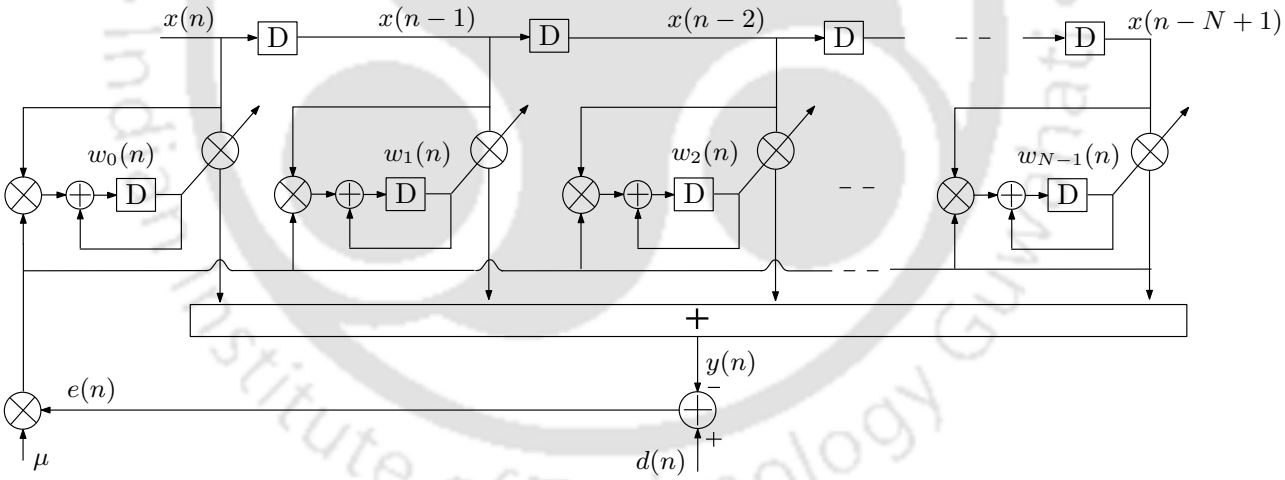


Fig. 1.6: Circuit schematic of an ADF based on conventional LMS algorithm.

The circuit schematic for the implementation of conventional N^{th} order LMS algorithm is shown in Fig. 1.6. It takes the input sequence $\mathbf{x}(n) = [x(n), x(n-1), ..., x(n-N+1)]^T$ and produces the output $y(n)$, as per

$$y(n) = \mathbf{x}^T(n)\mathbf{w}(n) \tag{1.11}$$

where $\mathbf{w}(n) = [w_0(n), w_1(n), ..., w_{N-1}(n)]^T$ is the coefficient vector. An error signal $e(n)$ is computed

by subtracting the output $y(n)$ from the desired signal $d(n)$, according to

$$e(n) = d(n) - y(n) \tag{1.12}$$

Using (1.12), the filter coefficients are updated for the next iteration according to LMS criterion as

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n) \tag{1.13}$$

where μ is the step-size. The coefficient update accumulators as shown in Fig. 1.6, memorizes the coefficients values and updates the current value by adding with coefficient increment terms $\mu e(n)\mathbf{x}(n)$. It is clear that every iteration involves $2N+1$ multiplications (N for filtering and $N+1$ for coefficient adaptation) and $2N$ additions ($N-1$ for filtering, N for coefficient adaptation and 1 for error computation). Depending upon the value of step-size, the LMS algorithm either yields faster convergence rate or lower steady-state error. Specifically, a large value of step-size leads to faster convergence rate while yielding a higher steady-state error, and vice-versa.

1.2.1 CLMS Algorithm

It is clear from the above discussion that the LMS algorithm either provides fast convergence rate or low steady-state error based on the selection of step-size. In order to provide better trade-off in convergence rate and steady-state error, Garcia et al. in [11] proposed a convex combination LMS (CLMS) algorithm which consists of two LMS ADFs with different step-sizes arranged in parallel. The block diagram of ADF based on CLMS algorithm is shown in Fig. 1.7. The outputs $y_c(n)$ of ADF_c in CLMS algorithm with input vector $\mathbf{x}(n)$ can be computed as

$$y_c(n) = \mathbf{x}^T(n)\mathbf{w}^c(n) \tag{1.14}$$

where $\mathbf{w}^c(n) = [w_0^c(n), w_1^c(n), ..., w_{N-1}^c(n)]^T$ is the coefficient vector of ADF_0 and ADF_1 for $c = 0$ and $c = 1$ respectively. The coefficients of ADF_0 and ADF_1 are updated based on the LMS criterion, as per

$$\mathbf{w}^c(n+1) = \mathbf{w}^c(n) + \mu_c e_c(n)\mathbf{x}(n) \tag{1.15}$$

where μ_c is the step-size of ADF_0 and ADF_1 for $c = 0$ and $c = 1$ respectively. Assuming $\mu_0 > \mu_1$ which implies ADF_0 is a fast filter and ADF_1 is a slow filter. Note the error $e_c(n)$ of component filters are obtained by subtracting the respective output signal $y_c(n)$ from the desired signal $d(n)$, as per

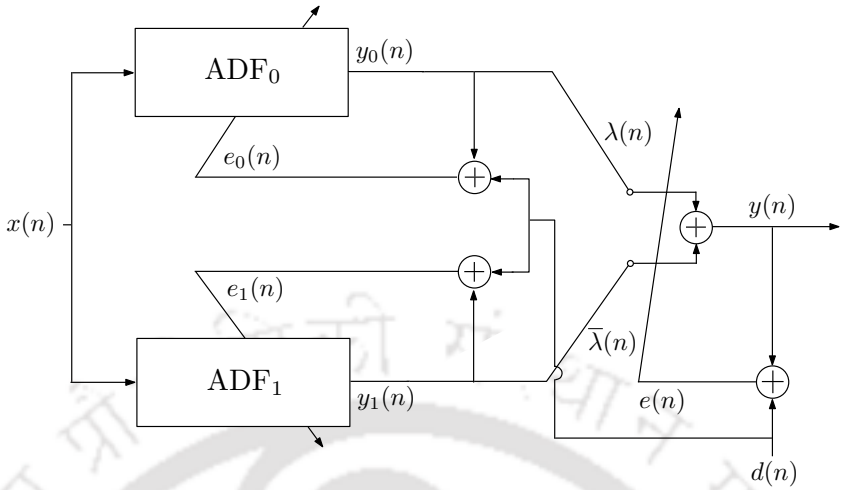


Fig. 1.7: Block diagram of ADF based on CLMS algorithm.

$e_c(n) = d(n) - y_c(n)$. According to [11], the output $y(n)$ of CLMS ADF is given by

$$y(n) = \lambda(n)y_0(n) + \overline{\lambda}(n)y_1(n) \tag{1.16}$$

where $\lambda(n)$ is mixing parameter which lies in the interval of $[0, 1]$ and its complement $\overline{\lambda}(n) = 1 - \lambda(n)$. As the name suggests the convergence rate and steady-state error of combined filter are adjusted by $\lambda(n)$, as per

$$\lambda(n+1) = \frac{1}{1 + \mathrm{e}^{-a(n)}} \tag{1.17}$$

where $\mathrm{e}^{(\bullet)}$ is an exponential function and $a(n)$ is an auxiliary variable. In every iteration, $a(n)$ is updated based on the combined error $e(n) = d(n) - y(n)$, according to

$$a(n+1) = a(n) + \mu_a e(n)[y_0(n) - y_1(n)]\lambda(n)\overline{\lambda}(n) \tag{1.18}$$

where μ_a is the step-size to control the adaptation rate of auxiliary variable $a(n)$. It is interesting to note from (1.18) that when mixing parameter $\lambda(n)$ becomes 0 or 1, the adaptation of auxiliary variable $a(n)$ eventually stops. It is clear that every iteration involves $4N + 10$ multiplications ($2N$ for filtering, $2N + 2$ for coefficient adaptation, 2 for output mixing, 6 for auxiliary variable adaptation), $4N + 3$ additions ($2N - 2$ for filtering, $2N$ for coefficient adaptation, 2 for error computation, 1 for output mixing, 2 for auxiliary variable adaptation) and 1 comparator.

1.2.2 BLMS Algorithm

There are some DSP applications where the ADF orders of a few hundreds and thousands are needed. For instance, active noise control in the headphones, the return of speaker echo to the far-end side of the telephone line etc. In such applications, one can find that the conventional LMS algorithm, as discussed in Section 1.2, is computationally expensive to implement. Block processing of the samples is a possible solution to address this issue [12], and can significantly reduce the computational requirements of ADFs. In this technique, a block of input samples and a block of the desired signal are collected and processed to obtain a block of output samples. However, this requires serial-to-parallel (S-P) conversion of the input data and desired signal, as shown in Fig. 1.8. It can be noted from Fig. 1.8 that it is not very straightforward that the block processing reduce the computational complexity of ADF, but elegant block processing of the data samples such as sharing of the processing time can increase the computational efficiency. Therefore, a good measure to determine the computational complexity in a block processing system is the number of operations to be performed in one block of data divided by the block length.

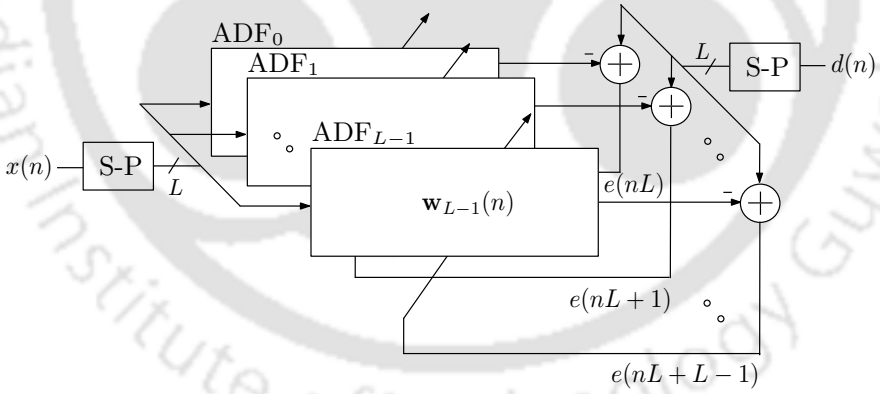


Fig. 1.8: Block diagram of ADF based on BLMS algorithm.

The block of filter output $\mathbf{y}(n)$ of an N^{th} order FIR filter with block length L for an input matrix $\mathbf{X}(n)$ can be computed as

$$\mathbf{y}(n) = \mathbf{X}^T(n)\mathbf{w}(n) \tag{1.19}$$

where n in this case denotes the block iteration, $\mathbf{w}(n) = [w_0(n), w_1(n), ..., w_{N-1}(n)]^T$ is the coefficient vector and $\mathbf{X}(n)$ is the input matrix which is derived from current input block with previous $N-1$

input samples, according to

$$\mathbf{X}(n) = \begin{bmatrix} x(nL) & x(nL-1) & . & . & x(nL-N+1) \\ x(nL-1) & x(nL-2) & . & . & x(nL-N) \\ . & . & . & . & . \\ . & . & . & . & . \\ x(nL-L+1) & x(nL-L) & . & . & x(nL-L-N+2) \end{bmatrix}^T \tag{1.20}$$

and the output vector $\mathbf{y}(n) = [y(nL), y(nL-1), ..., y(nL-L+1)]^T$. Using BLMS criterion, the filter coefficients $w_i(n)$ $(i = 0, 1, 2, ..., N-1)$ are updated, as per

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu_B \mathbf{X}(n)\mathbf{e}(n) \tag{1.21}$$

where μ_B is the step-size of BLMS ADF, $\mathbf{e}(n) = [e(nL), e(nL+1), ..., e(nL+L-1)]^T$ is the block of error samples computed as $\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{y}(n)$, and $\mathbf{d}(n) = [d(nL), d(nL+1), ..., d(nL+L-1)]^T$ is the block of desired response. It is clear that every iteration involves $2NL + L$ multiplications (NL for filtering and $NL + L$ for coefficient adaptation) and $2NL$ additions ($NL - L$ for filtering, NL for coefficient adaptation and L for error computation).

The summary of conventional LMS, CLMS, ADFE and BLMS algorithms in terms of input, output and hardware complexities are listed in Table 1.1.

Table 1.1: Summary of the LMS, CLMS, ADFE and BLMS Presented in Section 1.2

Algorithm	**Inputs**	**Outputs**	**Hardware Complexities**		
			MULT	ADD	REG
LMS	N = filter order	$y(n) = \mathbf{x}^T(n)\mathbf{w}(n)$	N	$N-1$	$N-1$
	μ = step-size	$e(n) = d(n) - y(n)$	–	1	–
	$\mathbf{x}(n)$ = input vector	$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n)$	$N+1$	N	N
CLMS	N = filters order	$y_c(n) = \mathbf{x}^T(n)\mathbf{w}^c(n)$	$2N$	$2N-2$	$2N-2$
	μ_c = step-sizes	$e_c(n) = d(n) - y_c(n)$	–	2	–
	$\mathbf{x}(n)$ = input vector	$\mathbf{w}^c(n+1) = \mathbf{w}^c(n) + \mu_c e_c(n)\mathbf{x}(n)$	$2N+2$	$2N$	$2N$
	$a(n)$ = auxiliary variable	$y(n) = \lambda(n)y_0(n) + \overline{\lambda}(n)y_1(n)$	2	1	–
	$c = 0, 1$ is the filter index	$\lambda(n+1) = 1/(1+\mathrm{e}^{-a(n)})$	COMP: 1	–	–
	$\lambda(n)$ = mixing parameter	$a(n+1) = a(n) + \mu_a e(n)[y_0(n) - y_1(n)]\lambda(n)\overline{\lambda}(n)$	6	2	1
ADFE	N_f = FF filter order	$y_f(n) = \mathbf{x}^T(n)\mathbf{w}^f(n)$	N_f	N_f-1	N_f-1
	N_b = FB filter order	$y_b(n) = \mathbf{b}^T(n)\mathbf{w}^b(n)$	N_b	N_b-1	N_b-1
	$\mathbf{x}(n)$ = input vector	$z(n) = y_f(n) - y_b(n)$	–	1	–
	$\mathbf{d}(n)$ = decision vector	$\delta(n) = Q[z(n)]$	QUANT: 1	–	–
	μ_f = step-size of FF filter	$e(n) = \delta(n) - z(n)$	–	1	–
	μ_b = step-size of FB filter	$\mathbf{w}^f(n+1) = \mathbf{w}^f(n) + \mu_f e(n)\mathbf{x}(n)$	N_f+1	N_f	N_f
		$\mathbf{w}^b(n+1) = \mathbf{w}^b(n) + \mu_b e(n)\mathbf{d}(n)$	N_b+1	N_b	N_b
BLMS	N = filter order, L = block length	$\mathbf{y}(n) = \mathbf{X}^T(n)\mathbf{w}(n)$	NL	$NL-L$	$NL-L$
	μ_B = step-size	$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{y}(n)$	–	L	–
	$\mathbf{X}(n)$ = input matrix	$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu_B\mathbf{X}(n)\mathbf{e}(n)$	$NL+L$	NL	NL

MULT: Multipliers, ADD: Adders, REG: Registers, COMP: Comparator and QUANT: Quantizer.

1.3 Complexity Issues and Related Research in LMS based Systems

Adaptive filters have numerous practical applications in various DSP and communication systems such as system identification, channel equalization, noise cancellation etc [5, 6]. Due to the advancements of technology in recent times, the amount of functionality required in an application is increasing. As a consequence, there is a continuous rise in the area, time and power constraints. This brings in the need for designing low-complexity ADFs with a better trade-off between these parameters. LMS algorithm is usually preferred for the realization of ADFs due to its simplicity and ease of implementation. It consists of a FIR filter which is made of several MAC units depending upon the filter order, as shown in Fig. 1.6. One of the main concerns related with conventional LMS ADF is that its order primarily determines the computational complexities, i.e., the higher the filter order, the higher is the computational complexity. However, higher order ensures better filtering performance.

The first LMS based system considered in this thesis is the design for system identification. In such applications, LMS ADF has to provide both fast convergence rate and low steady-state error in order to approximate the unknown system as close as possible. Convergence rate is an important parameter which indirectly affects the complexity of the LMS algorithm. It is governed by the step-size for the adaptation of filter coefficients. Precisely, the larger the step-size, faster is the convergence and higher is the steady-state error. On the other hand, a low-value of step-size degrades the convergence rate, however reduces the steady state error. A variable step-size LMS algorithm [13] is the viable solution to achieve better trade-off between these parameters. However, it is more complex compared to the fixed step-size LMS algorithm which makes the hardware realization of such filter difficult. In the sequel, many researchers suggested different approaches to address this problem [14–22]. For instance, Kwong et al. in [16] utilized the squared instantaneous error, Aboulnasr et al. in [18] exploited the auto-correlation of time-averaged error signals at adjacent time. Garcia et al. in [11] proposed a different approach which is popularly known as CLMS algorithm. It consists of two LMS ADFs with different step-sizes arranged in parallel. This scheme guarantees better trade-off between the convergence rate and steady-state error. This is because the filter with large step-size (fast filter) provides faster convergence rate, and the filter with small step-size (slow filter) provides lower steady-state error. However, it involves a complex coefficient transfer scheme to switch from the fast filter to the slow filter. Further, due to involvement of two LMS ADFs its complexity is twice that of conventional LMS algorithm. Few attempts have been made in the past to reduce the

complexity of coefficient transfer scheme [23–26]. Garcia et al. in [23] proposed a linear transfer of coefficients by interacting slow filter to fast filter during the transition stage, however it degrades the SNR performance. Later, Ruiz et al. in [24] suggested a new update rule to improve the SNR at the cost of increased computational requirements. Nascimento and de Lamare in [25] presented a low-complexity instantaneous transfer scheme based on sliding window approach for better convergence performance. Lu et al. in [26] proposed a new coefficient transfer scheme based on sign-adaptation approximations for low-complexity realization of CLMS ADF.

The second LMS based system considered in this thesis is ADFE. As discussed in Section 1.1.1.2 that the ADFE consists of two LMS ADFs in series with one in feedforward path and the other in feedback path. The design of ADFE encounters two challenging issues. Firstly, the presence of inherent feedback loop limits the achievable speed. Secondly, the complexity of component filters increases with the transmission data rate. The reason for latter issue is that as the data rate increases the symbol duration becomes shorter and shorter which means lesser and lesser time is available to carry out the computation, while its volume of computation keeps on increasing. As a consequence, it would make the real-time operation of ADFE difficult due to an increase in bandwidth of the signal. Over the years, several researchers tried to reduce the computational complexities while simultaneously achieving the higher speeds for fixed coefficient DFE and ADFE [27–41]. Look-ahead based approaches were found to be popular among the fixed-coefficient DFE and parallel implementation of linear filter. Difficulty arises when the systems contain quantizer loops, Parhi in [28] addressed this problem in an efficient manner. However, this architecture is more complex due to an increase in the number of quantization levels. Parhi et al. in [36] proposed lookahead and relax lookahead techniques to address this issue. The speed limitation of conventional DFE has been overcome by reformulating the architecture into an array of slicers, adders and multiplexers [29–32]. The architecture suffers from high hardware complexity. Later, Lin et al. in [33] suggested a low complexity DFE architecture. It is based on pre-computing and storing the feedback filter coefficients in two look-up tables (LUTs) separated by pipelined registers. Yang et al. in [37] presented a high-speed DFE where an additional FIR filter is employed to cancel the post-cursor ISI component. This scheme maintains almost same hardware complexity while providing a better convergence rate. Lin et al. in [38, 39] proposed a new lookahead technique to break the decision feedback loop to design a multi-gigabit system. Here, the parallelization is chosen to increase the throughput rate since the critical path grows linearly with the

feedback filter order. In addition, the hardware complexity is relatively higher than the multiplexer loop when the order of feedback filter is small. In [41], an efficient finite precision implementation of ADFE using block floating point (BFP) has been suggested. However, it suffers from the round-off noise errors.

The third LMS system considered in this thesis is BLMS based ANC. In such applications, higher order ADFs are desired for effective cancellation of the noise signals. The conventional LMS algorithm is computationally expensive to implement in these scenarios. BLMS algorithm is a fast and efficient approach for realizing higher order ADFs [12, 42]. It processes the block of inputs and computes the block of outputs, unlike the conventional LMS ADF where sample-by-sample processing is performed. In the past, several authors suggested different forms of BLMS ADF [43–47]. In general, the frequency domain approach is more efficient to realize BLMS algorithm, the work in [43] employed frequency domain block FxLMS based on fast Fourier transform (FFT) and fast Hartley transform (FHT). It is shown to be computationally more efficient than its FxLMS counterpart. Baghel et al. in [44] suggested distributed arithmetic (DA) based BLMS ADF using FFT suitable for FPGA. In [45], a time-domain pipelined BLMS algorithm has been presented for high-throughput realization of ADF using concurrent multiplier approach. Jayashri et al. in [46] suggested a low-complexity design for BLMS ADF which utilizes a single MAC cell for the computation of filter output and coefficient adaptation. The architecture suffers from low sampling rate due to less number of computational units. Mohanty and Meher in [47] presented an energy-efficient architecture for BLMS ADF in time-domain. It is based on new DA formulation which requires a single LUT for the computation of filter output and coefficients adaptation. However, the time to update the LUT contents and its size have an exponential dependence on the filter block length. Later, Mohanty et al. in [48] suggested a low-complexity BLMS ADF to achieve low area and low power at the cost of increased computational time.

From the above discussion, it is clear that several authors have made efforts to reduce the hardware complexities of different LMS ADF based systems. In all such systems, the time complexities are mainly determined by the critical path. For instance, the critical path of LMS ADF in Fig. 1.6 can be estimated by summing up the time involved in FIR filter, the error computation and the adaptation of filter coefficients. An N^{th} order LMS ADF requires the computation time of $T_{MUL} + (N-1)T_A$ units in direct-form FIR filter, the computation time of T_A units in error computation, and the computation

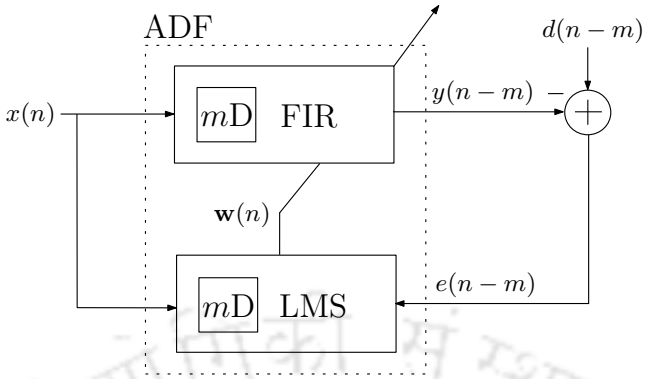


Fig. 1.9: Block diagram of ADF based on DLMS algorithm.

time of $T_{MUL} + T_A$ units in adaptation of filter coefficients, where T_{MUL} and T_A are computational delays of a multiplier and an adder respectively. Thus, the total computation time to realize the LMS ADF becomes $2T_{MUL}+(N+1)T_A$ units. Since the computation time in FIR filter is a function of filter order N which is normally high in many applications. This larger computation time sometimes makes it difficult to achieve the data rates demanded by wireless standard such as fourth generation (4G) communication [49]. This is even more challenging to design the LMS based system in compliance to fifth generation (5G) communication due to high data rate requirements in the order of gigabits [50]. Thus, there is a need to reduce the critical-path delay to meet the desired data rate specifications. Pipelining scheme is successfully employed to increase the computational speed of FIR filter [51]. But, its direct application to conventional LMS ADF is difficult due to the presence of recursive loop. Therefore, its pipelined form known as delayed LMS (DLMS) algorithm [52, 53] is used by inserting the adaptation delays in the structure, as shown in Fig. 1.9. Unlike (1.13), the filter coefficients in case of DLMS ADF are updated as follows

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n-m)\mathbf{x}(n-m) \tag{1.22}$$

where $e(n-m) = d(n-m) - y(n-m)$ with m represents the number of adaptation delays (D). It can be noted that the location of adaptation delays (mD) can be altered to achieve the desired critical path. However, the number of adaptation delays to be employed has an adverse effect on the convergence performance. Hence, the number of adaptation delays are to be kept small in order to obtain satisfactory convergence. On the other hand, if the number of adaptation delays to be employed is high, then the convergence rate becomes slower and the steady-state error becomes larger as compared

to conventional LMS ADF. Over the past two decades, several works have been reported for reducing critical-path delay of the LMS ADF using less number of adaptation delays [54–59]. For instance, Van and Feng in [54] suggested a systolic architecture by employing large number of processing elements (PEs) to reduce the number of adaptation delays without sacrificing the convergence and maintaining the critical path of one MAC operation as compared to other DLMS systolic structures. Later, Yi et al. in [59] proposed a new pipelined structure of an ADF by utilizing the concept of fine-grain in the pipelined implementation of adder tree for direct-form FIR filtering. It limits the critical path to a maximum of one addition time in FIR filter and coefficient update unit. Thus, the architecture supports high sampling rate, but involves large number of adaptation delays. However, this technique suffers from two adverse effects. Firstly, the power consumption increases due to insertion of adaptation delays. Secondly, the large number of adaptation delays degrade the convergence performance. Meher and Maheswari in [60] made further effort for simultaneous reduction of adaptation delays and the critical path with the optimized pipelining of carry-save chain in the forward path. Meher and Park in [57, 58] suggested a 2-bit multiplication cell to minimize the critical path and chip-area. In all the aforementioned works, the necessary design considerations for optimizing the complexities have not been taken into account which leads to higher area and larger computation time. Later, this issue is addressed by a systematic approach for the design of LMS ADF by Meher and Park in [61]. This includes the minimal use of adaptation delays, hence occupies lower area and consumes lesser power without compromising the desired processing throughput. However, there is no discussion on improving the convergence performance. Likewise, the pipelined versions of CLMS, ADFE and BLMS can also be obtained. The summary of the pipelined LMS, CLMS, ADFE and BLMS algorithms in terms of input, output and hardware complexities are listed in Table 1.2.

Based on the above discussion, it can be concluded that the time-complexity of LMS ADFs can be made less by shortening the critical path using pipelining, while hardware complexity is mainly dominated by the number of MAC units. Further, there is a need to reduce the effect of pipelining on the convergence performance of ADF, otherwise the filter would take indefinite time to reach the steady-state and produce the output with more noise. In this process, we found that distributed arithmetic (DA) is a suitable approach for reducing the hardware complexity of pipelined LMS ADF and its variants due to its multiplierless nature. The detailed description of DA and related research is presented in the subsequent sections.

Table 1.2: Summary of the Pipelined LMS, CLMS, ADFE and BLMS Presented in Section 1.2

Algorithm	**Inputs**	**Outputs**	**Hardware Complexities**		
			MULT	ADD	REG
LMS	N = filter order	$y(n-m) = \mathbf{x}^T(n-m)\mathbf{w}(n)$	N	$N-1$	$N-1+m$
	μ = step-size	$e(n-m) = d(n-m) - y(n-m)$	–	1	m
	$\mathbf{x}(n)$ = input vector	$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n-m)\mathbf{x}(n-m)$	$N+1$	N	$N+m$
CLMS	N = filters order	$y_c(n-m) = \mathbf{x}^T(n-m)\mathbf{w}^c(n)$	$2N$	$2N-2$	$2N-2+2m$
	μ_c = step-sizes	$e_c(n-m) = d(n-m) - y_c(n-m)$	–	2	$2m$
	$\mathbf{x}(n)$ = input vector	$\mathbf{w}^c(n+1) = \mathbf{w}^c(n) + \mu_c e_c(n-m)\mathbf{x}(n-m)$	$2N+2$	$2N$	$2N+2m$
	$a(n)$ = auxiliary variable	$y(n) = \lambda(n)y_0(n) + \overline{\lambda}(n)y_1(n)$	2	1	$2m$
	$c = 0, 1$ is the filter index	$\lambda(n+1) = 1/(1+\mathrm{e}^{-a(n)})$	COMP: 1	–	$2m$
	$\lambda(n)$ = mixing parameter	$a(n+1) = a(n) + \mu_a e(n-m)[y_0(n-m) - y_1(n-m)]\lambda(n)\overline{\lambda}(n)$	6	2	$1+2m$
ADFE	N_f = FF filter order	$y_f(n-m) = \mathbf{x}^T(n-m)\mathbf{w}^f(n)$	N_f	N_f-1	N_f-1+m
	N_b = FB filter order	$y_b(n-m) = \mathbf{d}^T(n-m)\mathbf{w}^b(n)$	N_b	N_b-1	N_b-1+m
	$\mathbf{x}(n)$ = input vector	$z(n-m) = y_f(n-m) - y_b(n-m)$	–	1	m
	$\mathbf{d}(n)$ = decision vector	$\delta(n-m) = Q[z(n-m)]$	QUANT: 1	–	–
	μ_f = step-size of FF filter	$e(n-m) = \delta(n-m) - z(n-m)$	–	1	m
	μ_b = step-size of FB filter	$\mathbf{w}^f(n+1) = \mathbf{w}^f(n) + \mu_f e(n-m)\mathbf{x}(n-m)$	N_f+1	N_f	N_f+m
		$\mathbf{w}^b(n+1) = \mathbf{w}^b(n) + \mu_b e(n-m)\mathbf{d}(n-m)$	N_b+1	N_b	N_b+m
BLMS	N = filter order, L = block length	$\mathbf{y}(n-m) = \mathbf{X}^T(n-m)\mathbf{w}(n)$	NL	$NL-L$	$NL-L+Lm$
	μ_B = step-size	$\mathbf{e}(n-m) = \mathbf{d}(n-m) - \mathbf{y}(n-m)$	–	L	Lm
	$\mathbf{X}(n)$ = input matrix	$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu_B\mathbf{X}(n-m)\mathbf{e}(n-m)$	$NL+L$	NL	$NL+Lm$

MULT: Multipliers, ADD: Adders, REG: Registers, COMP: Comparator, QUANT: Quantizer and m is the number of adaptation delays. Note the value of m may be modified if retiming is employed.

1.4 Distributed Arithmetic

Distributed arithmetic (DA) is an efficient technique for the computation of inner product of two vectors with one vector known prior to the implementation [62]. The idea is to pre-compute and store the filter partial products in look-up table (LUT) which are shifted-and-accumulated for certain number of clock cycles in order to produce the filter output.

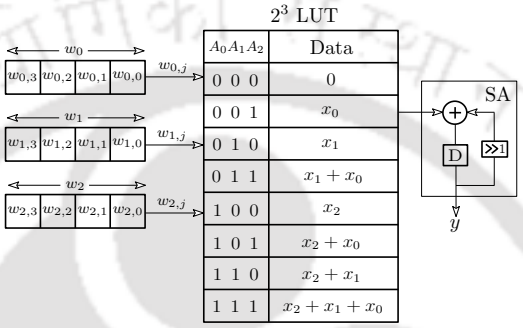


Fig. 1.10: DA architecture for inner product computation of $\mathbf{w} = [w_0, w_1, w_2]$ and $\mathbf{x} = [x_0, x_1, x_2]$.

Consider the inner product computation of two vectors $\mathbf{x} = [x_0, x_1, ..., x_{N-1}]$ and $\mathbf{w} = [w_0, w_1, ..., w_{N-1}]$ as

$$y = \sum_{i=0}^{N-1} x_i w_i \tag{1.23}$$

If every coefficient w_i is represented in W-bit unsigned binary form, then

$$w_i = \sum_{k=0}^{W-1} w_{i,k} 2^{-k} \tag{1.24}$$

where $w_{i,k} \in [0, 1]$. By substituting (1.24) in (1.23) and interchanging the order of summation, we obtain

$$y = \sum_{k=0}^{W-1} \left[\sum_{i=0}^{N-1} x_i w_{i,k} \right] 2^{-k} \tag{1.25}$$

Define

$$d_k = \sum_{i=0}^{N-1} x_i w_{i,k} \tag{1.26}$$

Thus, (1.25) becomes

$$y = \sum_{k=0}^{W-1} d_k 2^{-k} \tag{1.27}$$

From (1.26), it can be noted that k^{th}-bit of filter coefficients w_i would result in 2^N possible combinations of input samples x_i, which are nothing but the filter partial products in binary form. Hence, the

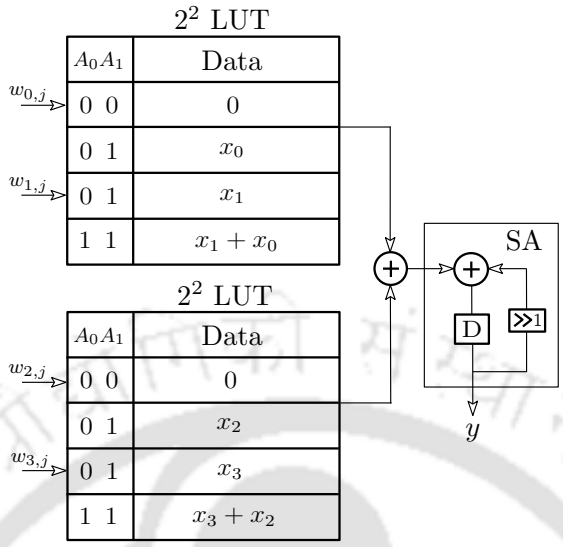


Fig. 1.11: DA architecture for inner product computation of $\mathbf{w} = [w_0, w_1, w_2, w_3]$ and $\mathbf{x} = [x_0, x_1, x_2, x_3]$ with LUT decomposition.

partial products can be stored in a look-up-table (LUT) which is either a read-only memory (ROM) or a random-access memory (RAM). The output y can simply be produced by reading the LUT contents with its address lines as the k^{th}-bit ($k = 0, 1, 2, ..., W-1$) of filter coefficients w_i which are successively passed through a shift-and-accumulate (SA) unit. Note that the least-significant-bit (LSB) of filter coefficients always form address lines to the LUT. The basic DA architecture for the computation of (1.27) for $N = 3$ and $W = 4$ is shown in Fig. 1.10. Clearly, it pre-computes and stores 2^3 combinations of input samples $[x_0, x_1, x_2]$ in LUT with bits of the coefficients $[w_0, w_1, w_2]$ as its addresses. In general, there are total 2^N partial products of input samples which require the LUT size to be 2^N. This implies that if filter order N is large, then the LUT size required would be high. Interestingly, DA provides the flexibility of using multi-LUT blocks [63] which can be exploited by splitting N as $N = M \times L$. Hence, (1.26) can be re-expressed as

$$d_k = \sum_{i=0}^{ML-1} x_i w_{i,k} = \sum_{i=0}^{L-1} x_i w_{i,k} + \sum_{i=L}^{2L-1} x_i w_{i,k} + ... + \sum_{i=(M-1)L}^{ML-1} x_i w_{i,k} \tag{1.28}$$

$$= \sum_{k=0}^{M-1} \sum_{i=jL}^{(j+1)L-1} x_i w_{i,k} \tag{1.29}$$

Now, if dummy variable i is replaced with $i - jL$, then (1.29) would lead to

$$d_k = \sum_{j=0}^{M-1} \sum_{i=0}^{L-1} x_{i+jL} w_{i+jL,k} \tag{1.30}$$

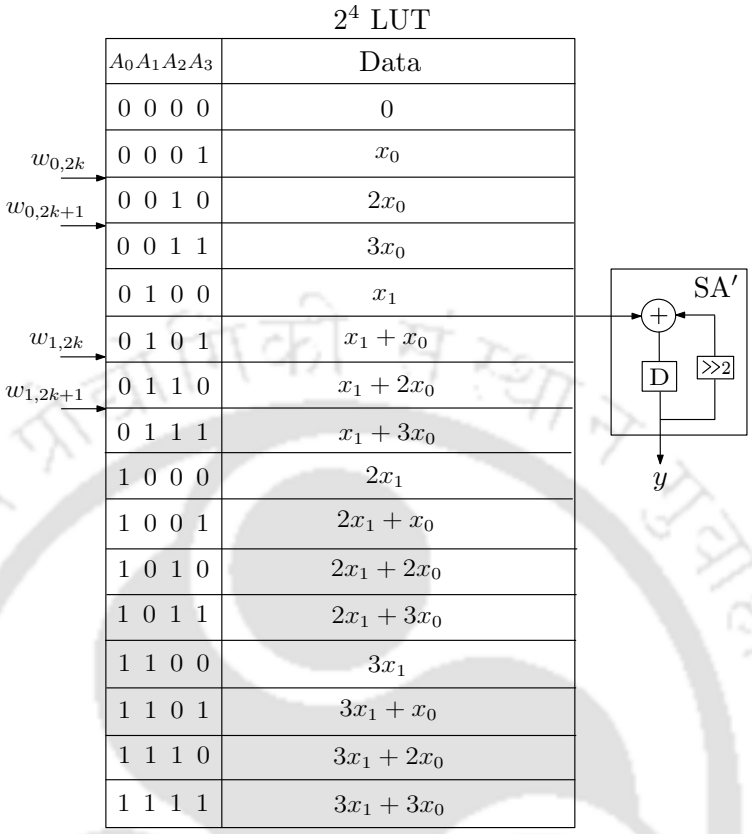


Fig. 1.12: DA architecture for inner product computation of $\mathbf{w} = [w_0, w_1]$ and $\mathbf{x} = [x_0, x_1]$ with radix-4.

where $0 \leq j \leq M-1$. It is clear from (1.30) that M number of LUTs of size 2^L can be used with address lines from k^{th} bit of j sets of w_i. The DA architecture corresponding to (1.30) for $M = 2$ and $L = 2$ is shown in Fig. 1.11. It is important to note that the number of clock cycles to produce the output are still the same as (1.27), however the number of adders is increased. From (1.27), it can be observed that the number of clock cycles to produce the output depends on the precision of filter coefficients, while in case of (1.23) it depends on the filter order N. As the wordlength of filter coefficients increases, the time to produce the output would also goes up. In order to increase the speed of the system, groups of coefficients bits can be used as the address lines to the LUT. Such type of DA design is popularly referred as high-radix DA (or digit-serial DA). However, this comes at the cost of significant increase in LUT size. Now, suppose γ-bit of coefficients are used to address the LUT by splitting W as $W = W_\gamma \times \gamma$. Each γ set of bits in a given k^{th} digit form a binary value of

$w_{i,\gamma k+0}w_{i,\gamma k+1}...w_{i,\gamma k+\gamma-1}$. In this case, (1.27) can be expressed as

$$y = \sum_{k=0}^{W/\gamma-1} d_{k\gamma}2^{-k\gamma} \tag{1.31}$$

According to (1.31), it is clear that the number of clock cycles for computing the output is reduced by a factor of γ. The LUT size is increased by 2^{γ} times as compared to (1.27). This can be understood by re-writing (1.26) in terms of γ as

$$d_{k\gamma} = \sum_{i=0}^{N-1} x_i w_{i,k\gamma} \tag{1.32}$$

Clearly, γ-bits are used as the address lines to the LUT, where $0 \leq k \leq W/\gamma - 1$. In such case, the computation of inner product is completed in W/γ clock cycles. The DA architecture corresponding to (1.32) for $N = 2$ and $\gamma = 2$ is shown in Fig. 1.12.

1.5 DA based FIR Filter

As mentioned, the inner product computation of two vectors using DA replaces multipliers and adders with a LUT and an SA unit respectively. In other words, it performs the inner product computation of two vectors of length N and replaces the N multipliers in the structure with a LUT of size 2^N. The output is generated in W-bit time using $W-1$ additions rather than $N-1$ additions. Further, the LUT decomposition and high-radix can be used in the direct realization of FIR filter. Since the partial products of the filter are nothing but the inner product computation of input samples and filter coefficients. With appropriate design considerations, DA can offer savings up to 80 % in the realization of FIR filters [64]. Based on the representation of coefficients (or input samples) in two's complement (TC) form or offset binary coding (OBC) form, it can be classified as TC-DA or OBC-DA based FIR filter.

1.5.1 TC-DA based FIR Filter

The output $y(n)$ of a N^{th} order FIR filter for an input sequence $x(n-i)$ can be computed as

$$y(n) = \sum_{i=0}^{N-1} x(n-i)w_i \tag{1.33}$$

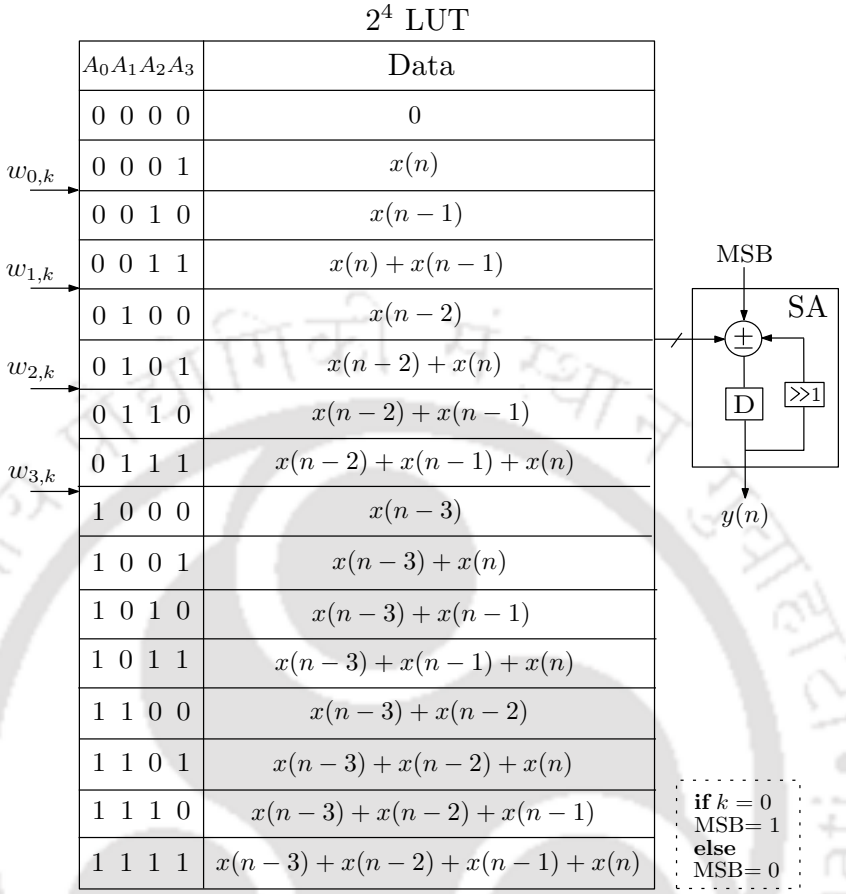


Fig. 1.13: TC-DA architecture of a 4$^{\text{th}}$ order FIR filter.

where w_i $(i = 0, 1, 2, ..., N-1)$ are the filter coefficients. By representing every filter coefficient w_i in two's complement form, we obtain

$$w_i = -w_{i,0} + \sum_{k=1}^{W-1} w_{i,k} 2^{-k} \tag{1.34}$$

Substituting (1.34) in (1.33) and interchanging the order of summation, we arrive at

$$y(n) = \sum_{k=0}^{W-1} d_k(n) 2^{-k} \tag{1.35}$$

where

$$\begin{aligned} d_k(n) &= \sum_{i=0}^{N-1} x(n-i) w_{i,k} \\ d_0(n) &= -\sum_{i=0}^{N-1} x(n-i) w_{i,0} \quad (k=0) \end{aligned} \tag{1.36}$$

Clearly, the sign of term $d_k(n)$ is to be inverted at most-significant-bit (MSB) of filter coefficients. For given set of input samples $x(n-i)$, the term $d_k(n)$ would take 2^N combination of input samples, and out of them only one combination is selected at a time. These are nothing but the filter partial products of input samples which are shifted and accumulated for W number of clock cycles.

1.5.2 OBC-DA based FIR Filter

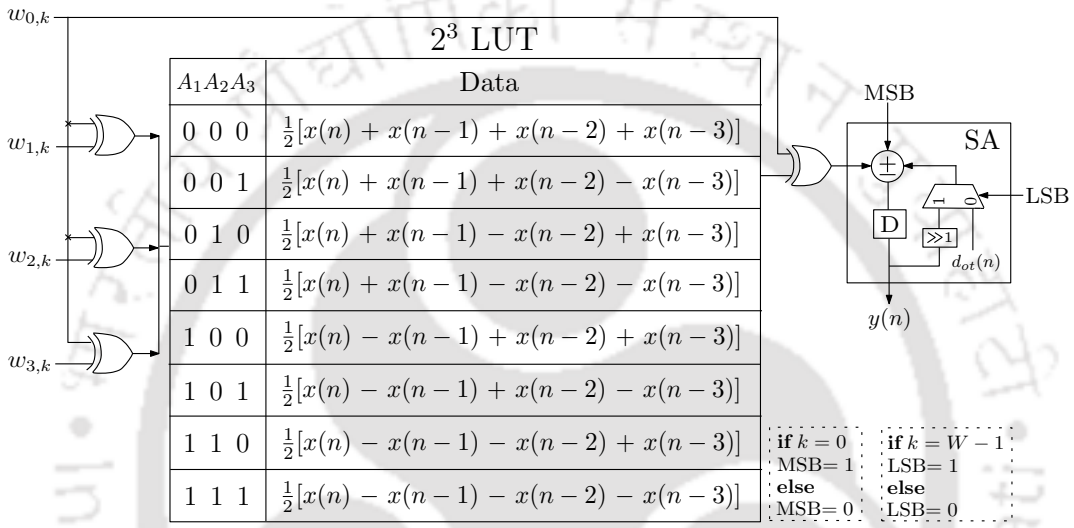


Fig. 1.14: OBC-DA architecture of a 4th order FIR filter.

In TC-DA, the size of LUT to store the partial products of input samples grows exponentially with filter order N. It can be reduced by using OBC technique [65], which represents the filter coefficients as $w_i = \frac{1}{2}[w_i - (-w_i)]$ in addition to (1.34). Hence, one can express each filter coefficient w_i as

$$w_i = \frac{1}{2}\left[-(w_{i,0}-\overline{w}_{i,0}) + \sum_{k=0}^{W-1}(w_{i,k}-\overline{w}_{i,k})2^{-k} - 2^{-(W-1)}\right] \tag{1.37}$$

Substituting (1.37) in (1.23) and interchanging the order of summation, we have

$$y(n) = \sum_{k=0}^{W-1} d_k(n)2^{-k} + d_{ot}(n)2^{-(W-1)} \tag{1.38}$$

where $d_k(n)$ is defined as

$$\begin{aligned} d_k(n) &= \frac{1}{2}\sum_{i=0}^{N-1} x(n-i)c_{i,k} \\ d_0(n) &= -\frac{1}{2}\sum_{i=0}^{N-1} x(n-i)c_{i,0} \quad (k=0) \end{aligned} \tag{1.39}$$

and, $d_{ot}(n)$ is defined as

$$d_{ot}(n) = -\frac{1}{2}\sum_{i=0}^{N-1} x(n-i) \tag{1.40}$$

where $c_{i,k} = w_{i,k} - \overline{w}_{i,k}$ with $\overline{w}_{i,k}$ is one's complement of $w_{i,k}$. It can be noted from (1.39) that only half combination of filter partial products are required to store while other half would be mirror symmetric. Therefore, a ROM or RAM of size 2^{N-1} can be employed whose address lines are obtained through the XOR operation of all the LSB's with the LSB of filter coefficient w_0.

1.6 Literature on DA based Implementations

The study and research on the implementation of various DSP algorithms take us back to early nineteens. Researchers over the time have made significant contributions in the development of DSP architectures. This has been influenced by the demand of applications and available level of integration in the state-of-art IC technology. Many DSP algorithms are characterized in terms of number of computations involved, primarily multiply-add operations. Most of the digital computers in earlier times were equipped with adders, and the complex operations such as multiplication (or division) were realized by shift-and-add (or subtract) operations. This was emphasized for reducing the number of multiplications in any DSP algorithms, as they occupy large chip-area. After a few decades, there was development in the high-speed operation of general purpose microprocessors. This has motivated researchers to look forward to designing fast signal processing systems. During that time, two major concepts have been invented, namely, coordinate rotation digital computers (CORDIC) and distributed arithmetic (DA). CORDIC is primarily employed to implement trigonometric function and rotation operations. While DA is mainly used to compute the inner product of two vectors. DA was first introduced by Croisier [64], and is independently developed by Peled and Liu [66]. Later, DA is successfully employed in various applications such as telephone systems at Bell laboratories, air-to-ground missile digital pilot etc. This is because most of the DSP algorithm in these applications mainly involve convolution and correlation operations which are nothing but inner product of two vectors. Choi et al. in [63] proposed a multi-LUT approach to efficiently implement the DSP algorithms. From then DA has been successfully employed in other DSP algorithms such as Fourier transform, discrete cosine transform, cyclic convolution etc.

As stated earlier, a fixed coefficient FIR filter can be directly implemented using DA by storing the filter partial products in LUT. However, an adaptive FIR filter based on DA requires extra effort

due to coefficient update operation. Moreover, the filtering and coefficient updating operations are mutually coupled which needs that the partial products of filter coefficients must be re-calculated before the filtering operation. In the past, several authors have made effort based on approximations to the standard LMS algorithm [67–71]. Allred et al. in [67] suggested a new technique by employing an external LUT for storing the partial product of input samples. In their work, they showed that by proper selection of system parameters, the throughput can be made independent of the filter orders. However, the use of two LUTs made the whole system computationally time and area intensive. Guo and DeBrunner in [68] addressed this problem using a single LUT for storing the partial products of input samples, while the adaptation of filter coefficients was similar to conventional LMS criterion. A considerable amount of savings in area and time were achieved for both in the filtering and coefficient updating operations. Later, Surya and Rafi in [69] proposed a novel design based on the framework of [67] by storing the partial products of filter coefficients and input samples in two separate LUTs. Further, they showed that storing of the recent sample in the external register would allow decomposition of LUT into two smaller LUTs. Unlike the LUT based realizations discussed above, it is also possible to realize the LMS ADF without LUT [72]. Meher and Park in [70] proposed first LUT-less design for LMS ADF based on DLMS algorithm. Using this scheme, the throughput performance of LMS ADF was improved as compared to [67, 68], however, at the cost of an increase in hardware complexity, especially for the larger order of DA base units. Later, in [71], the same authors proposed a new pipelined architecture of LMS ADF using the frameworks of [68, 70]. It reduces both LUT update time of [68] and complexity requirement of [70] by a novel design of parallel LUT structure. Specifically, LUT is used to generate the partial products of input samples in parallel using hardware elements. Further, the convergence properties are affected by the selected value of step-size and pipelined nature of the structure. In order to increase the speed and reduce the area-complexity of the system, binary carry-save full adder (CSFA) based SA unit was employed. However, it is found that the CSFA based SA unit has carry-loops which may limit the maximum operational speed [71]. Mohanty and Meher in [47] proposed DA based BLMS ADF where a novel LUT sharing technique has been suggested for the computation of filtering and coefficient updating operations. It offers low area and low power for the implementation BLMS ADF. Later, Mohanty et al. in [48] suggested a low-complexity BLMS ADF to achieve better area and power figures, but increased the computational time. Hence, the new challenge would be to reduce the computational complexities of pipelined LMS

ADF and its variants. With DA offers computational savings in the realization of LMS ADF, we feel that there is still scope to realize such filters which account for low-area, low-power, and can provide better throughput performance for higher order filters implementation.

1.6.1 Problem Formulation

Owing to the ubiquitous applications of LMS ADFs, the computational complexity has been increasing rapidly with increasing levels of integration. Thus, the computational complexity poses a major concern for VLSI implementation of LMS ADFs, especially in view of meeting the requirements of a given application. In general, the computational complexity of an algorithm is expressed in terms of hardware and time complexities. In LMS ADF, multipliers primarily contribute to the hardware complexity, while the critical path delay determines the time complexity. As the order of ADF increases, both the number of multipliers and critical path delay go up, which make the real-time operation of ADF difficult. Pipelining is a commonly used technique to reduce the critical path delay, however it slows down the convergence rate and increases the steady state error. In order to improve the convergence performance of ADF, additional hardware is required. Moreover, due to the limitation of standalone LMS ADF in various applications, a combination of LMS ADFs in parallel or in series or in block is normally used to achieve better performance. For instance, the combination of two LMS ADFs in parallel provides fast convergence rate and low steady-state error in system identification, the combination of two LMS ADFs in series achieves better performance in channel equalization and block of LMS ADFs offers superior performance in noise cancellation scenarios. As a consequence, this leads to further increase in hardware complexity. Thus, there is a need for an alternative approach to reduce the overall computational complexity. Distributed arithmetic (DA) is a preferable method for efficient implementation of ADFs as it can eliminate the multipliers from LMS ADFs. However, the complexity of DA based LMS ADF is limited by the size of LUT which is used to store the filter partial products. The determination of redundancies in the partial products could be a possible solution for reducing the complexity of DA based LMS ADF. The complexity reduction of pipelined ADF using DA with the focus on improving its convergence performance without hardware overhead has not been studied in the literature. This motivates us to exploit possible benefits of DA in the realization of low-complexity LMS ADFs. As a proof of concepts, two case studies are considered with one in channel equalization problem for 5G communication system and other in noise cancellation problem for in-ear headphones.

1.7 Organization of the Thesis

Based on the background above, the salient contributions made in the thesis are listed as follows:

Chapter 1

This is an introductory chapter which provides a brief about the significance of ADF in system identification, channel equalization and noise cancellation applications. This chapter also discusses the motivation behind the work and major research in the complexity reduction of LMS ADFs over the last decades. The significance of ADFs realization using DA, and the fundamental issues, challenges and complexity considerations of LMS variants are also presented. The chapter ends by summarizing the overview of its contents and the contributions of the thesis.

Chapter 2

Chapter 2 presents three different optimal complexity architectures of pipelined LMS ADF. The purpose here is to explore the implementation details of LMS ADF which are useful to derive the low-complexity architectures of its variants in subsequent chapters. Although DA based LMS ADF has been well researched in the past [67–71], there was still scope to optimize them in terms of area, power and throughput. So, in this chapter, we exploited OBC scheme to generate the filter partial products serially using hardware elements (LUT-less) instead of storing them in LUT to gain efficiency in terms of area, power and throughput. But, this treatment produces non-OBC terms of input samples at the output during some initial clock cycles due to the pipelined nature of filter. They are subsequently corrected in the error computation unit. This chapter also presents the benefit of generating the partial products and reducing the computational time of the SA unit through high-radix OBC implementation. The work also contributes novel designs of bit-level coefficient update unit, radix-4 OBC partial product generators, minus-minus-plus (MMP) CSFA based SA unit and the offset term. All the proposed designs are extended for large order filter with modification in the correction of non-OBC terms.

Chapter 3

In Chapter 3, a low-complexity CLMS ADF is presented for system identification scenarios. Although DA based LMS ADFs have already been studied [67–71], but no one has discussed the improvement in the convergence properties of pipelined LMS ADF. In Chapter 2, the complexity of LMS ADF is optimized using OBC, but it cannot be used to improve the convergence performance due to the presence of non-OBC terms. In this chapter, TC representation of coefficients is exploited to generate the partial products of input samples in parallel for low-complexity realization of CLMS ADF. The major reduction in hardware complexity is achieved by employing a single DA based LMS ADF. Further reduction in complexity is achieved by sharing of filter partial products. The location of the second adaptation delay is such that it produces the delayed version of adjacent error samples which allows efficient transfer of filter coefficients. Thus, the convergence performance is improved by comparing the time length of the maximum correlation between adjacent error samples with the pre-defined time window. In the sequel, a mathematical derivation is carried out to determine the expression for pre-defined time window in terms of filter parameters such as filter order, coefficients wordlength and step-size.

Chapter 4

In Chapter 4, a low-complexity architecture of ADFE is presented for channel equalization problem in 5G communication system. As discussed in Chapter 2, the SA unit takes a significant amount of time to produce the output which is addressed through high-radix OBC implementation. In this chapter, the representation of decision is therefore taken up in OBC form to derive a SA-less architecture. This chapter first discusses the implementation details of non-ADFE and some design considerations are taken up to reduce the implementation complexities. The complexity of architecture is reduced by two-stage pre-computation and further reduction is achieved by exploiting symmetries in the LUT contents. Moreover, the architecture is pre-speed up by two, retimed and unfolded to achieve the data rate specifications of a 5G communication system. A novel strategy is presented to make the filter adaptive by employing external LUT to pre-compute and store the decisions in OBC-form. In every iteration, the contents of decision LUT has to be updated time-to-time. Further, a new approach is

presented to improve the convergence performance of proposed filter by employing a parallel error multiplexer for the removal of non-OBC terms.

Chapter 5

Having considered the low-complexity LUT-less DA based LMS and CLMS ADFs in Chapters 2 and 3 respectively, and extending the low-complexity SA-less realization of ADFE in Chapter 4, we take up the low-complexity implementation of BLMS ADF in Chapter 5. Here, both physical LUT and SA unit are employed, and their effects on throughput performance are compensated by block processing. The system is considered as a case study for noise cancellation in-ear headphones. It is based on OBC for complexity reduction which stores the partial products of input samples in a LUT. This chapter first presents the mathematical formulation of block LMS algorithm to compute the filter output and coefficient increment terms by decomposing the filter order into input block-length and processing elements. Next, the derivation is extended with OBC formulation followed by splitting of LUT into even and odd components. Then, the architectural details of the proposed filter are presented. Also, a new filter block update strategy and a novel LUT updating scheme are suggested, where LUTs of one processing element are required to update in the set of given processing elements. The proposed architecture does not have only low-complexity, but also have good flexibility in terms of block length with respect to the best existing design. Further, the proposed architecture does not have the problem of non-OBC terms.

Finally, in Chapter 6, the research work is concluded by summarizing up the salient contributions of all the proposed scheme with the suggestion for future work.

2

Optimal Complexity Architectures for Pipelined LMS Adaptive Filter

Contents

2.1 Introduction

ADFs are widely used in many DSP and communication systems such as system identification, channel equalization, noise cancellation etc [5, 6]. Among them, LMS based ADF is usually preferred due to its simplicity and ease of implementation. For effective filtering performance, large orders ADF are always preferred. However, the complexity of such filters is high due to more number of multipliers involved. Structurally, an LMS ADF consists of FIR filter whose coefficients are updated time-to-time. It is made up of several MAC units which occupies more area, consumes high power, and critical path grows linearly with the order of filter. For high-sampling rates, it is necessary to shorten the critical path which calls for its reduction through pipelining [51]. But, the complexity due to MAC units is still a problem for higher order filter implementation. Distributed arithmetic (DA) discussed in Chapter 1, would be a suitable approach for efficient realization of ADF as it can offer low computational complexity without hardware multipliers. It represents the filter coefficients or inputs in TC or OBC form by virtue of which the filter partial products are pre-computed and stored in the LUT, and the output is produced using SA unit. OBC has inherent advantage of less LUT complexity, and is expected to reduce the LUT complexity by $1/r$ for radix-r binary system. Thus, it makes FIR filter more suitable to custom ASIC and FPGA platforms. Towards this objective, three optimal complexity realization of pipelined LMS ADF are presented in this chapter. The purpose here is to study, understand and explore the low-complexity realization of LMS ADF for later use in the subsequent chapters.

A fixed coefficient FIR filter can be easily implemented using DA by storing the partial products in the LUT followed by a SA unit, as discussed in Chapter 1. However, the problem arises when the filter coefficients are updated as the partial products are stored in LUT and re-calculated prior to the computation of filter output. The complexity problem is further pronounced since the size of LUT grows exponentially with the filter order and time to produce the output depends on the wordlength of filter coefficients. Few attempts have been made to realize efficient implementation of LMS ADF using DA [67–71]. Allred et al. in [67] employed an external LUT that is served for the adaptation of LUT contents demanded by conventional DA. Later, Guo and DeBrunner in [68] suggested a new technique in which one LUT is maintained to store the partial products of input samples. As a result, the hardware complexity of system is reduced, but requires the bit-slices of filter coefficients as the address bits to the LUT. Surya and Rafi in [69] employed OBC scheme to reduce the LUT complexity

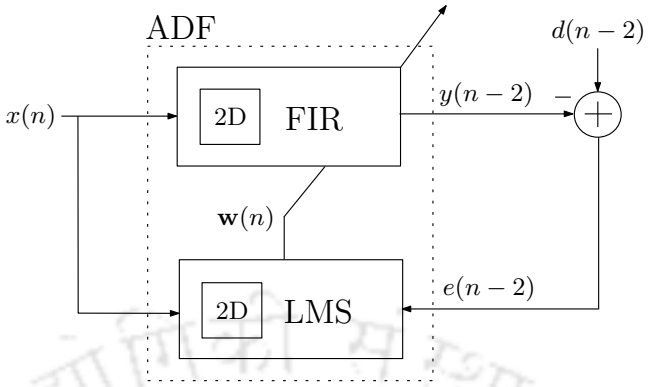


Fig. 2.1: Block diagram of pipelined LMS ADF with two adaptation delays.

based on the framework of [67]. All these schemes have utilized physical LUT, however, it is not necessary to implement filters with LUTs [73]. LUT-less pipelined LMS ADF has been suggested by Meher and Park in [70]. It employs small adaptation delays to pipeline the filter, but its complexity is substantially higher. Later, they presented a new architecture in [71] to reduce the complexity using parallel LUT. The parallelization in structure improves the area, power and throughput performances over [67, 68]. Besides, its hardware complexity grows exponentially with the order of DA base unit. Moreover, binary carry-save full adder (CSFA) based SA unit was employed to reduce the critical path and area complexity. Further, it is found that the CSFA based SA unit has carry-loops which may limit the maximum operational speed. Hence, the new challenge would be to reduce the computational complexity of pipelined LMS ADF. In the following sections, we propose three optimal complexity realization of pipelined LMS ADF based on the OBC-DA which accounts for low-area, low-power, and can provide better throughput performance for higher order filter implementation.

2.2 Mathematical Formulation

Consider a N^{th} order ADF shown in Fig. 2.1 which processes an input sequence $\mathbf{x}(n) = [x(n), x(n-1), ..., x(n-N+1)]^T$, and produces the output $y(n)$ as

$$y(n) = \sum_{i=0}^{N-1} x(n-i)w_i(n) \tag{2.1}$$

where $\mathbf{w}(n) = [w_0(n), w_1(n), ..., w_{N-1}(n)]^T$ is the coefficient vector at current byte-iteration n. An error $e(n)$ is computed by subtracting the output $y(n)$ from the desired response $d(n)$, according to

$$e(n) = d(n) - y(n) \tag{2.2}$$

Using (2.2), the filter coefficients are updated for next byte-iteration based on LMS criterion as

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n) \tag{2.3}$$

where μ is the step-size. It is clear from (2.3) that the filter coefficients require a single clock cycle for the update. In case of DLMS algorithm [54], the filter coefficients are updated as

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n-m)\mathbf{x}(n-m) \tag{2.4}$$

where $e(n-m) = d(n-m) - y(n-m)$ and $\mathbf{x}(n-m) = [x(n-m), x(n-1-m), ..., x(n-N+1-m)]^T$, with m being the adaptation delays. Usually the number of adaptation delays is chosen to be small, in order to maintain the good convergence performance [70, 71]. In this work, the number of adaptation delays are chosen to be two (i.e., $m = 2$). It is worth noting that the number of clock cycles to update the filter coefficients is $m+1$. By representing the filter coefficients $w_i(n)$ in TC form, we obtain

$$w_i(n) = \sum_{k=0}^{W-1} w_{i,k}(n)s_k = \mathbf{w}_i{}^T(n)\mathbf{s} \tag{2.5}$$

where $s_k = (-1)^{\lfloor \frac{W-1-k}{W-1} \rfloor} 2^{-k}$ with $(-1)^{\lfloor \frac{W-1-k}{W-1} \rfloor}$ indicates the sign-reversal operation at the most signifcant bit (MSB) of filter coefficients, $\lfloor \bullet \rfloor$ is the greatest-integer function, $\mathbf{w}_i(n) = [w_{i,0}(n), w_{i,1}(n), ..., w_{i,W-1}(n))]^T$ is the bit-vector of each filter coefficient with $w_{i,k}(n) \in [0, 1]$, $\mathbf{s} = [-2^0, 2^1, \cdots, 2^{-(W-1)}]^T$ is the scaling vector, and W is the wordlength of filter coefficients. The proposed design utilizes OBC scheme as discussed in Chapter 1 to represent the filter coefficients as $w_i(n) = \frac{1}{2}[w_i(n) - (-w_i(n))] = \frac{1}{2}[w_i(n) - \overline{w}_i(n) - 1]$, where $\overline{w}_i(n)$ is one's complement of $w_i(n)$ for all $0 \leq i \leq N-1$. Thus, one can also express the filter coefficients $w_i(n)$ using (2.5) as

$$w_i(n) = \frac{1}{2}\left[\sum_{k=0}^{W-1} c_{i,k}(n)s_k - s_{W-1}\right] = \frac{1}{2}\tilde{\mathbf{w}}_i^T(n)\mathbf{s} \tag{2.6}$$

where $\tilde{\mathbf{w}}_i(n) = [\tilde{w}_{i,0}(n), \tilde{w}_{i,1}(n), ..., \tilde{w}_{i,W-1}(n))]^T$ is the bit-vector of each filter coefficient in OBC-form with $\tilde{w}_{i,k}(n) \in [-1, 1]$. It has two component vectors, namely, $\tilde{\mathbf{w}}_i(n) = \mathbf{c}_i(n) - \mathbf{o}_i$ where $\mathbf{c}_i(n) = [c_{i,0}(n), c_{i,1}(n), ..., c_{i,W-1}(n)]^T$ is the bit difference vector of filter coefficients and its one's complement version with each $c_{i,k}(n) = w_{i,k}(n) - \overline{w}_{i,k}(n)$, and $\mathbf{o}_i = [o_{i,0}, o_{i,1}, ..., o_{i,W-1}]^T$ is the bit offset-vector with each $o_{i,k} = \lfloor W/(k+1) \rfloor$ for all $0 \leq i \leq N-1$. By substituting (2.6) in (2.1) and interchanging

the order of summation, we get

$$y(n) = \sum_{k=0}^{W-1} \left[\sum_{i=0}^{N-1} \frac{1}{2} x(n-i) c_{i,k}(n) \right] s_k - \left[\sum_{i=0}^{N-1} \frac{1}{2} x(n-i) \right] s_{W-1} \tag{2.7}$$

Define

$$d_k(n) = \frac{1}{2} \sum_{i=0}^{N-1} x(n-i) c_{i,k}(n), \;\; d_{ot}(n) = -\frac{1}{2} \sum_{i=0}^{N-1} x(n-i) \tag{2.8}$$

where $d_k(n)$ and $d_{ot}(n)$ represents the partial products and offset term respectively. By substituting (2.8) in (2.7), we have

$$y(n) = \sum_{k=0}^{W-1} d_k(n) s_k + d_{ot}(n) s_{W-1} \tag{2.9}$$

Note that only half of the partial products are required to be stored in LUT, as the other half are redundant [65]. For example, a 4^{th} order OBC based FIR filter stores 2^3 partial products of input samples in LUT, as shown in Fig. 1.14. The remaining partial products are decoded through the use of external exclusive-OR (XOR) gates. Further, LUT content at each address location has some correlation with the contents at nearby address locations. This property is exploited to derive low-complexity optimal LUT architectures. The offset-term $d_{ot}(n)$ is loaded initially into SA unit at least-significant bit (LSB) of filter coefficients using a 2-to-1 multiplexer, as shown in Fig. 1.14. In [69], $d_{ot}(n)$ is obtained from 0^{th} address location in every iteration as both are two's complement of each other. However, this method requires the update of LUT contents in every iteration. Alternatively, an important observation from (2.8) is made to obtain $d_{ot}(n)$, as per

$$d_{ot}(n) = d_{ot}(n-1) - \frac{1}{2}[x(n) + x(n-N)] \tag{2.10}$$

That is, $d_{ot}(n)$ is obtained from its past value $d_{ot}(n-1)$ by subtracting the average of $x(n)$ and $x(n-N)$ samples. This would greatly reduce the computational cost of $d_{ot}(n)$ term. Similar to (2.4), each bit-slice $c_{i,k}(n)$ term is also updated, as it computes $d_k(n)$ in bit-plane. According to (2.6), $w_i(n)$ can also be expressed as $w_i(n) = \frac{1}{2}[\mathbf{c}_i{}^T(n) - \mathbf{o}_i{}^T]\mathbf{s}$, so the coefficient vector would be $\mathbf{w}(n) = \frac{1}{2}[\mathbf{C}^T(n) - \mathbf{O}^T]\mathbf{s}$ with $\mathbf{C} \triangleq [c_{i,k}(n)]_{N \times W}$ as bit-slice difference matrix of filter coefficients, and $\mathbf{O} \triangleq [o_{i,k}(n)]_{N \times W}$ as offset-matrix. Thus, the coefficient update equation given in (2.4) after simplification can be expressed as

$$\mathbf{C}(n+1)\mathbf{s} = \mathbf{C}(n)\mathbf{s} + 2\mu e(n-m)\mathbf{x}(n-m) \tag{2.11}$$

where the matrices $\mathbf{C}(n+1)$ and $\mathbf{C}(n)$ contain the bit-slices of filter coefficients at byte-iterations $n+1$ and n respectively, and the matrix $\mathbf{O}$ has fixed bit-values for all byte-iterations n. Note that the step-size required in (2.11) is twice that of conventional LMS algorithm (2.4). The effect of increased step-size can be compensated by truncating 1-bit of $e(n-m)$ which will be clear in subsequent discussion. It is noted that the byte-iteration n can be expressed in terms of bit-iteration k as $n = Wk$. So, the scaled bit-slice matrix $\mathbf{C}(n)\mathbf{s}$ in byte-iteration n becomes the scaled bit-slice vector $\mathbf{c}_k(n)s_k$ in bit-iteration k for all $0 \leq k \leq W-1$. In order to update filter coefficients at bit-level, coefficient increment terms $2\mu e(n-m)x(n-m-i)$ are to be stored in registers for all $0 \leq i \leq N-1$. This is possible by retiming the coefficient update registers as

$$\mathbf{c}_{k+1}(n)s_k = \mathbf{c}_k(n)s_k + D_k\{2\mu e(n-m)\mathbf{x}(n-m)\} \tag{2.12}$$

where D_k is the unit delay-operator whose contents are accessed at a rate k. It is worth noting that the number of adaptation delays are still the same as the conventional pipelined LMS ADF (2.4). Likewise, the update of offset term, as given in (2.10) can also be derived at bit-level.

2.3 Proposed Scheme

The proposed architecture of LMS ADF based on OBC-DA is shown in Fig. 2.2. It consists of an OBC-DA based FIR filter, a bit-level coefficient update unit (CUU), an adder tree, an error computation unit (ECU), an initial-state-remover (ISR), and a controller. FIR filter comprises of an input update register (IUR), OBC-LUT with an addressing logic (AL), a pipelined MMP-CSFA based SA unit (PSAR), and an offset circuitry (OC). CUU consists of coefficient update register array (CURA) followed by bit-level coefficient accumulators. As mentioned, two adaptation delays D_0 and D_1 are used with locations indicated in Fig. 2.2.

To derive optimal architectures of LUT and SA unit, it is important to revisit the salient features of existing design [71] as discussed below:

C1) It is based on separate generation and selection of filter partial products to make the operation possible with two clocks, namely, $\mathrm{clock_A}$ (bit-iteration) and $\mathrm{clock_B}$ (byte-iteration) such that $\mathrm{clock_B} = W\mathrm{clock_A}$.

C2) Parallel realization of LUT leads to higher complexity, especially when order of DA base unit is large.

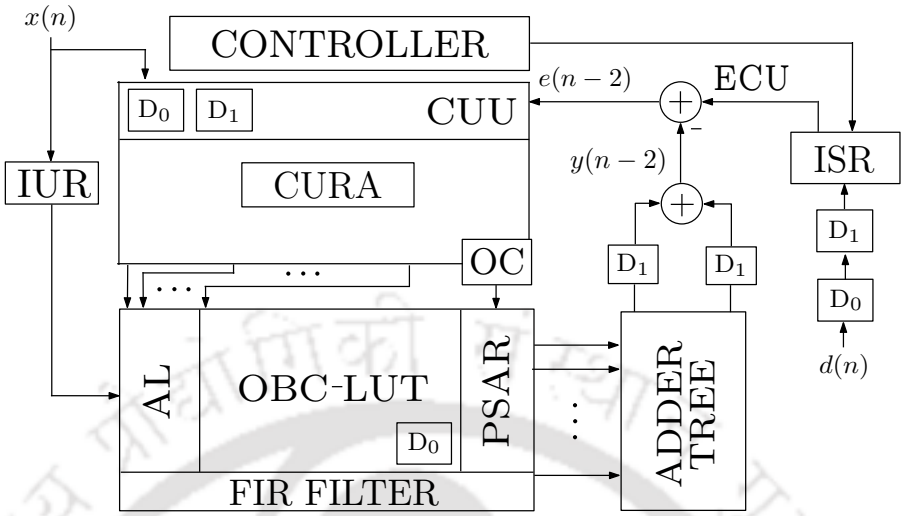


Fig. 2.2: Architecture of pipelined LMS ADF based on OBC-DA.

C3) Area-complexity and sampling period are reduced by employing binary CSFA based SA unit.

C4) The elements of matrix $\mathbf{C}(n) = [c_{i,k}(n)]_{N\times W}$ are zero during initial clock_B cycles ($0 \le n \le m$), that is, $c_{i,k}(n) = 0$ for all $0 \le i \le N-1$ and $0 \le k \le W-1$.

Let us discuss the rationale behind the concept of using clock_A and clock_B in bit-serial DA design. In any digital system, the total power dissipation P_{total} can be estimated as

$$P_{\text{total}} = \alpha C V_{DD}^2 f_{clock_\text{B}} \tag{2.13}$$

where α is the switching activity, C is the total capacitance, V_{DD} is the supply voltage and f_{clock_B} is the operating frequency. According to characteristic C1, clock_A and clock_B are used to operate some functional units with capacitance C_A and other functional units with capacitance C_B respectively. In such a case, the total power dissipation P'_{total} can be expressed as

$$P'_{\text{total}} = \alpha C_\text{B} V_{DD}^2 f_{clock_\text{B}} + \alpha C_\text{A} V_{DD}^2 f_{clock_\text{A}} \tag{2.14}$$

Assuming same switching activity α for both the functional units and using $f_{clock_\text{A}} = W f_{clock_\text{B}}$, the voltage supply of functional units operating with f_{clock_A} can be reduced from V_{DD} to βV_{DD} and $\beta : 0 < \beta < \beta_{max}$ [74] with β_{max} being maximum value of β. Thus, (2.14) can be re-expressed as

$$P'_{\text{total}} = \alpha V_{DD}^2 f_{clock_\text{B}} (C_B + W\beta^2 C_A) \tag{2.15}$$

Therefore, the power dissipation in two-clock system (P'_{total}) over one-clock system (P_{total}) can be

lowered, if $W\beta^2 < 1$, assuming $C = C_{\rm B} + C_{\rm A}$. From (2.15), one can find $\beta_{max} = 1/\sqrt{W}$. To find optimum value of β without affecting the system performance, it is required to consider the duration of clock$_{\rm B}$ as charging time (or discharging time) of gate and stray capacitances. Theoretically, it is expressed as $f_{clock_{\rm B}} = \eta(V_{DD} - V_T)^2/(C_{chrg}V_{DD})$, where C_{chrg} denotes the capacitance to be charged or discharged in the critical path, η is a function of technology parameter and V_T is the threshold voltage. While the effective capacitance to be charged or discharged by clock$_{\rm A}$ would be C_{chrg}/W, therefore, $f_{clock_{\rm A}} = W\eta(\beta V_{DD} - V_T)^2/(C_{chrg}\beta V_{DD})$. If the same clock is used in both the functional units, then optimum β can be found by solving the following quadratic equation

$$W(\beta V_{DD} - V_T)^2 = \beta(V_{DD} - V_T)^2 \tag{2.16}$$

It is clear from (2.16) that the solution to β would result two values as $\beta = -K(1 \mp \sqrt{1 - 1/K^2})$ with $K = (1/(2W))(1 - V_T/V_{DD})^2 - V_T/V_{DD}$, but we have to consider the β which satisfies $\beta V_{DD} > V_T$ or maximum between them. From the above discussion, it is clear that the bit-serial DA design with two clocks consumes less power.

2.3.1 Optimal LUT and SA Architectures

In this section, the design and analysis of the proposed OBC based LUT and SA unit architectures are presented.

2.3.1.1 Design and Analysis of OBC-LUT Architectures

This subsection presents different OBC-LUT architectures, and discusses their time and hardware complexities. The hardware design of different $4^{\rm th}$ order proposed OBC-LUT architectures is shown in Fig. 2.3. Each of them is characterized in terms of feasibility with clock$_{\rm A}$ and clock$_{\rm B}$, critical path delay, LUT output delay, complexity of XOR gates, adders and registers. They are discussed as follows:

- **OBC-LUT$_{\rm I}$**: It has least-critical path, and this corresponds to the last adder in partial-product accumulation [75]. Besides, it cannot be operated with clock$_{\rm A}$ and clock$_{\rm B}$ simultaneously, when the duration of clock$_{\rm A}$ is less than that of clock$_{\rm B}$. It computes the partial-products, according to $\pm D\{\frac{1}{2}x(n+1) \pm D\{\frac{1}{2}x(n+1) \pm D\{\frac{1}{2}x(n+1) \pm D\{\frac{1}{2}x(n+1)\}\}\}\}$, that is, the innermost parenthesis term is first computed, then the next parenthesis term is computed and so on, where 'D' is the unit-delay operator which runs at sampling clock$_{\rm B}$. It is noticed that for a given bit-

Design	OBC-LUT$_{\text{I}}$	OBC-LUT$_{\text{II}}$	OBC-LUT$_{\text{III}}$
clock$_{\text{A}}$ clock$_{\text{B}}$ critical path 1-bit truncator two's complement adder			
Clocks	Only clock$_{\text{B}}$	Both clock$_{\text{A}}$ and clock$_{\text{B}}$	Both clock$_{\text{A}}$ and clock$_{\text{B}}$
CPD	$T'_A + T_X$	$(N-1)T'_A + T_X$	$(\log_2 N)T''_A + T_X$
LOD	$N-1$	1	1
XOR	$\frac{(N+1)(N-2)}{2} + NB'$	NB'	NB'
ADD	$\frac{(N-1)(N-2)}{2} + (N-1)B'$	$\frac{(N-1)(N-2)}{2} + (N-1)B'$	$\frac{N}{4}(\log_2 N - 1) + (N-1)B'$
REG	$\frac{N(N-3)}{2} + NB'$	NB'	NB'

CPD: critical path delay, LOD: LUT output delay, ADD: adders, REG: registers, DEC: decoder, N is the filter order, B is the input wordlength, $B' = B-1$, T'_A and T''_A are adder delays after partial product accumulation in OBC-LUT $_{\text{I}}$, $_{\text{II}}$ and OBC-LUT$_{\text{III}}$ respectively, with $T'_A = T_A + (N-1)T_{FAC}$, $T''_A = T_A + (\log_2 N)T_{FAC}$, T_A is adder delay, T_{FAC} is the carry propogation delay. The notation x_i in above diagrams stand for input sample $x(n-i)$.

Fig. 2.3: Design, analysis and comparison of 4$^{\text{th}}$ order OBC-LUT$_{\text{I, II, III}}$ architectures.

slice of filter coefficients, a total of $N-1$ clock_B cycles delay are required to obtain the correct output.

- **OBC-LUT$_\text{II}$**: Unlike OBC-LUT$_\text{I}$, it has separate filter partial-product generation and selection units. Thus, it is possible to operate with clock_A and clock_B simultaneously. The structure involves less number of registers at bit-level, but its critical path and accumulation of partial products grow linearly with the filter order. It also involves less number of XOR gates, and requires one clock_B cycles to obtain the correct output.

- **OBC-LUT$_\text{III}$**: It reduces the critical path of OBC-LUT$_\text{II}$ by employing a conditional adder tree. For instance, the possible partial products to be stored in LUT for 4^{th} order FIR filter can be expressed as $\pm\frac{1}{2}\{\{x(n)\pm x(n-1)\}\pm\{x(n-2)\pm x(n-3)\}\}$, where $x(n), x(n-1), x(n-2)$ and $x(n-3)$ are the four most recent samples. The structure first computes inner parenthesis terms $\frac{1}{2}\{x(n)\pm x(n-1)\}$ and $\frac{1}{2}\{x(n-2)\pm x(n-3)\}$. Next, their outputs are added or subtracted based on the bit-value of $w_{2,k}$. Finally, the sign of partial products are taken care by the bit-value of $w_{0,k}$. Clearly, the accumulation of partial-products is less severe and involves less adders at bit-level over OBC-LUT$_{\text{I, II}}$.

Based on the above discussion, it is clear that the OBC-LUT$_\text{I}$ has least critical path, and requires more number of adders and registers. Nevertheless, the OBC-LUT$_\text{III}$ has the lowest hardware complexity. Apart from the optimized complexities, these architectures could be useful in reducing the switching activity α. Noticeably, all the proposed architectures employ 1-bit truncation at the input for realizing the scaling factor $\frac{1}{2}$ associated with LUT contents, as shown in Fig. 2.3. This makes the effective wordlength of inputs to be B'-bit with $B'=B-1$. In order to estimate the switching activity α, its expected value is a meaningful factor. This basically indicates the number of logic gates that are switched. For instance, consider OBC-LUT$_\text{II}$ structure with X_i denote the random variable of $(i+1)^{\text{th}}$ partial product accumulation-stage of wordlength $(B'+i)$-bit, and Y denote the random variable of switching activity α. One can express Y as the summation of X_i as $Y=\sum_{i=0}^{N-1}X_i$. Note all X_i have binomial distribution with probability of every bit equal to $\frac{1}{2}$. Using independence assumption [58] and the results derived in [76], the expected value of Y can be obtained as

$$\mathbb{E}[Y]=\sum_{i=0}^{N-1}\mathbb{E}[X_i]=\frac{1}{2}\left[B'N+\frac{N(N+1)}{2}\right] \tag{2.17}$$

where $\mathbb{E}[\bullet]$ denotes the expectation operator. Clearly, the expected value of Y is reduced for 1-bit data truncation which would also reduce the power consumption, as per (2.15). Similarly, the expected value of Y for the other proposed OBC-LUTs can be derived.

2.3.1.2 High Radix TC and OBC based LUTs

Table 2.1: Truth Table for TC and OBC Radix-4 Partial Product Generators

$b_{i,2\rho+1}$	$b_{i,2\rho}$	TC PPG				OBC PPG		
		$w_{i,\rho}$	O_1	O_2	O_3	$\tilde{w}_{i,\rho}$	O_1	O_2
0	0	0	0	0	0	−3	0	1
0	1	1	1	0	0	−1	1	0
1	0	2	0	1	0	+1	1	0
1	1	3	0	0	1	+3	0	1

In radix-r binary system, the digit-slices of filter coefficients in TC form would take the values from the set $W_{i,\rho} \in [0, 1, 2, ..., r-1]$. Any digit $w_{i,\rho}$ in the set $W_{i,\rho}$ can be represented by adjacent γ-bits $b_{i,\gamma(\rho+1)-1}...b_{i,\gamma\rho+1}b_{i,\gamma\rho}$ with $r = 2^\gamma$, according to

$$w_{i,\rho} = \sum_{l=1}^{\gamma} 2^{\gamma-l} b_{i,\gamma(\rho+1)-l} \tag{2.18}$$

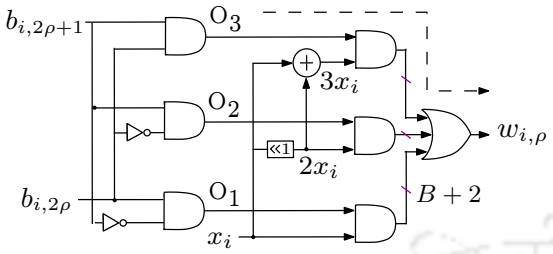
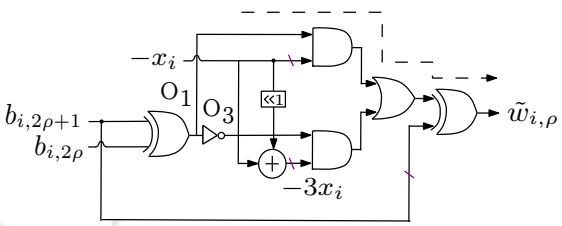
for $\rho = 0, 1, 2, ..., W/\gamma - 1$. In case of OBC, the digit-slices of filter coefficients would take the values from the set $\tilde{W}_{i,\rho} \in [-r+1, .. -3, -1, 1, 3..., r-1]$, in accordance with (2.6). Any digit $\tilde{w}_{i,\rho}$ in the set $\tilde{W}_{i,\rho}$ can be given as

$$\tilde{w}_{i,\rho} = \sum_{l=1}^{\gamma} 2^{\gamma-l+1} b_{i,\gamma(\rho+1)-l} - 2^\gamma + 1 \tag{2.19}$$

It is clear from (2.19) that the signed-digits in radix-r OBC are mirror-symmetric, and therefore only half signed-digits need to be generated. This would lower the hardware complexity for generating the partial products. In multiplier design [77], high-radix coding reduces the number of partial products rows. However, it has an effect on reducing the number of clock cycles involved in SA unit for DA based designs. In such case, the filter output $y(n)$ given in (2.9) for radix-r OBC-DA can be written as

$$y(n) = \sum_{\rho=0}^{W/\gamma-1} d_\rho(n) s_\rho + d_{ot}(n) s_{W/\gamma-1} \tag{2.20}$$

where $s_\rho = (-1)^{\left\lfloor \frac{W/\gamma-1-\rho}{W/\gamma-1} \right\rfloor} 2^{-\rho}$ and $s_{W/\gamma-1} = (2^\gamma - 1)2^{-(W/\gamma-1)}$. It is clear from (2.18) and (2.19)

Radix-4 TC PPG	Radix-4 OBC PPG
	

Design	TC PPG	OBC PPG
CPD	$(\log_2 r - 1)T_A + T_{AO}$	$(\log_2 r - 1)T_A + T_{AO} + T_X - T_O$
AND	$(r-1)(B+2+B_r)$	$(r/2)(B+1+B_r)$
OR	$(r-2)(B+2+B_r)$	$(r/2-1)(B+1+B_r)$
XOR	–	$B+1+B_r$
DEC	AND: $r-1$, NOT: $\frac{r}{2}$	AND: $r\log_2 r$, NOT: $\frac{r}{2}\log_2 r$, OR : $\frac{r}{2}$
ADD	$r/2-1$	$r/2-1$

$B_r = \lceil \log_2(r-1) \rceil$, $T_{AO} = T_{AND} + (\log_2 r)T_O$; T_X, T_{AND} and T_{OR} are delays of a XOR, AND and OR gates respectively

Fig. 2.4: Design, analysis and comparison of radix-4 TC and OBC partial product generators.

that the number of digits to generate in radix-r TC and OBC are $r-1$ and $r/2$ respectively. This is due to the fact that radix-r TC does not have symmetries in digit-slices. Thus, the implementation complexity to generate partial products in OBC-form is relatively lower than TC-form. The truth table for generating the digits in TC and OBC forms is shown in Table 2.1. The corresponding proposed radix-4 partial product generator (PPG) of TC and OBC are shown in Fig. 2.4. They consist of a decoder and an encoder to generate partial product of input x_i with filter coefficients bits $b_{i,2\rho+1}b_{i,2\rho}$. In radix-4 TC PPG, logic gates AND, OR and NOT together decode $b_{i,2\rho+1}b_{i,2\rho}$ bits to three outputs $\mathrm{O}_1 = \overline{b}_{i,2\rho+1}b_{i,2\rho}$, $\mathrm{O}_2 = b_{i,2\rho+1}\overline{b}_{i,2\rho}$ and $\mathrm{O}_3 = b_{i,2\rho+1}b_{i,2\rho}$. In contrast, radix-4 OBC PPG decodes $b_{i,2\rho+1}b_{i,2\rho}$ bits two outputs $\mathrm{O}_1 = \overline{b}_{i,2\rho+1}b_{i,2\rho} + b_{i,2\rho+1}\overline{b}_{i,2\rho}$ and $\mathrm{O}_3 = \overline{b}_{i,2\rho+1}\overline{b}_{i,2\rho} + b_{i,2\rho+1}b_{i,2\rho}$ using XOR and XNOR gates. Note that the decoder complexity of OBC PPG for $r > 4$ may not be simply XOR and XNOR gates rather a more complex logic made of AND, OR and NOT gates. The decoded outputs along with $x_i, 2x_i$ and $3x_i$ in TC-form (or x_i and $3x_i$ in OBC-form) are fed to an AND-OR encoder to compute the possible partial products. Unlike TC PPG, a XOR gate is required for sign-reversal in OBC PPG to decode the remaining partial products. Importantly, input in TC PPG and OBC PPG are sign-extended to have $(B+2)$-bit and $(B+1)$-bit respectively. In

Table 2.2: Performance Comparison for 4th order TC-LUT$_{I,\ III}$ and OBC-LUT$_{I,\ III}$ with Radix-4, 8 using TSMC 90 nm CMOS Library

Design	**TC-LUT$_I$**		**TC-LUT$_{III}$**		**OBC-LUT$_I$**		**OBC-LUT$_{III}$**	
Radix	4	8	4	8	4	8	4	8
Area (μm^2)	4027	8873	3881	8549	3767	7893	3372	6724
Power (mW)	2.23	5.12	1.89	4.91	1.92	4.27	1.76	3.52
MSP (ns)	1.95	2.91	2.59	3.72	1.88	2.79	2.43	3.43
TP (per μs)	512	343	386	268	531	358	411	293
ADP (μm^2.ns)	7853	25820	8757	31802	7082	22021	8194	22929
PDP (mW.ns)	4.35	14.90	4.89	18.26	3.61	11.91	4.28	12.07

MSP: minimum-sampling-period, TP: throughput, ADP: area-delay-product, PDP: power-delay-product.

radix-4 TC representation, the sign-bit of filter coefficients would decode the outputs as $0, 1, -2$ and -1 rather than $0, 1, 2$, and 3 respectively, in accordance with (2.18). As a result, separate PPG unit is required which makes the circuit more complex. In contrast, OBC PPG decodes digits at sign-bit in sign-reversal form i.e., $-3, -1, +1$ and $+3$ instead $+1, +3, -3$ and -1 respectively, in accordance with (2.19). Moreover, the number of inputs to OR gate in the encoder of TC PPG and OBC PPG equal to the possible digits. Hence, the propagation delay in TC PPG is relatively higher than OBC PPG when r increases. Further, more bits in inputs are required for sign-extension in TC PPG when decoded outputs are composite numbers. The comparison of the proposed TC and OBC PPGs in terms of critical path delay (CPD), gates and adder complexities are listed in Fig. 2.4.

For comparison purpose, a 4th order TC-LUT$_{I,\ III}$ and OBC-LUT$_{I,\ III}$ architectures are coded in Verilog for radix-4, 8 with 8-bit wordlength of inputs. ASIC Synthesis is performed using TSMC 90 nm CMOS Library by Cadence RTL Compiler. The corresponding area, power, throughput, minimum-sampling-period (MSP), area-delay-product (ADP) and power-delay-product (PDP) results are listed in Table 2.2. As expected, OBC-LUT$_{I,\ III}$ occupy lesser area and consume lesser power as compared to their TC-LUT$_{I,\ III}$ counterparts. The savings in area and power are more prominent for OBC-LUT$_{I,III}$ than TC-LUT$_{I,III}$ when radix size increases. For example, OBC-LUT$_I$ and OBC-LUT$_{III}$ for radix-8 occupy 11.04% and 21.34% lesser area and consume 16.60% and 28.31% lesser power as compared to TC-LUT$_I$ and TC-LUT$_{III}$ respectively. It is important to note that TC-LUT$_I$ and OBC-LUT$_I$ provide better figures of ADP and PDP than TC-LUT$_{III}$ and OBC-LUT$_{III}$ respectively.

2.3.1.3 Architecture of SA Unit

In [71], it is noticed that significant portion of time is involved in the SA unit, especially when the wordlength of filter coefficients become large. Therefore, they used binary carry-save full adder (CSFA) based SA unit to reduce the computation time of shift-accumulation, but in turn carry-loops are formed. High-radix coding might reduce computation time of CSFA based SA unit. However, it is observed that when the size of radix increases, the delay due to carry propagation also increases. So, it is important to reduce carry-propagation delay of CSFA based SA unit for higher radices. This is achieved with a novel CSFA based on minus-minus-plus (MMP) redundant adder. Redundant adder offer "carry-free" property at word-level addition [78]. It is utilized to reduce the carry-propagation delay for performing bit-level full addition.

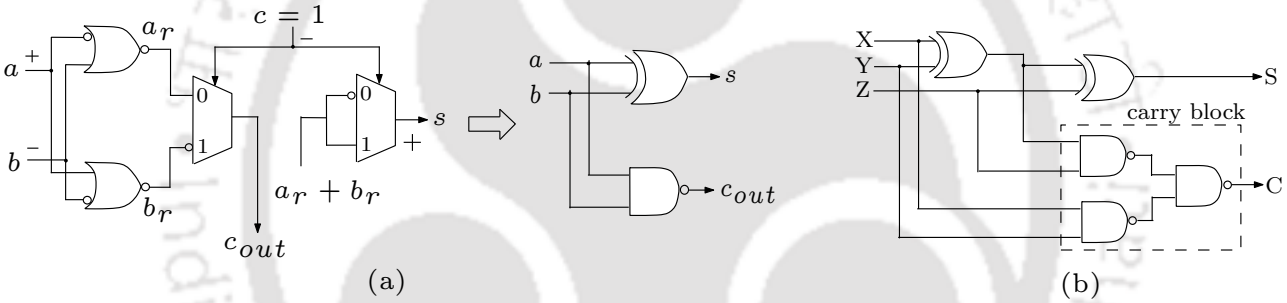


Fig. 2.5: (a) Equivalence between redundant to binary and half addition [78], with simplification towards MMP based carry-save half adder. (b) Gate-level description of MMP based carry-save full adder.

Basically, it consists of two half MMP redundant adders which performs $s = a + b = a - \bar{b} - 1$, where a, b are inputs, s is output sum, and quantities $(a - \bar{b})$ and -1 are redundant digits $(a, \bar{b})$ and $(0, 1)$ respectively. This is performed with three-input MMP adder by setting its third input $c = 1$ corresponding to $(0, 1)$ digit-set, as shown in Fig. 2.5(a). To design smaller and faster circuits, it is required to avoid $(1, 1)$ condition at the inputs of MMP adder [78]. This can be achieved by encoding (a, b) as (a_r, b_r), with $a_r = a\bar{b}$ and $b_r = \bar{a}b$. The simplified structure of MMP based carry-save half adder is shown in Fig. 2.5(a). The corresponding gate-level description of proposed CSFA structure after cascading two redundant MMP based carry-save half adders is shown in Fig. 2.5(b). It can be noted that the final carry output is obtained by passing the individual carry outputs of MMP based carry-save half adders through a NAND gate. Clearly, the carry-propagation delay and gates complexity of proposed MMP-CSFA are reduced over binary CSFA. This can be understood if one

implements the carry block of binary CSFA using NAND gates. Analysis shows that the reduction in carry propagation is $2T_{NAND}$, with T_{NAND} being the delay of NAND gate. The comparison between the proposed and binary CSFA in terms of boolean expression and gates complexity are listed in Table 2.3.

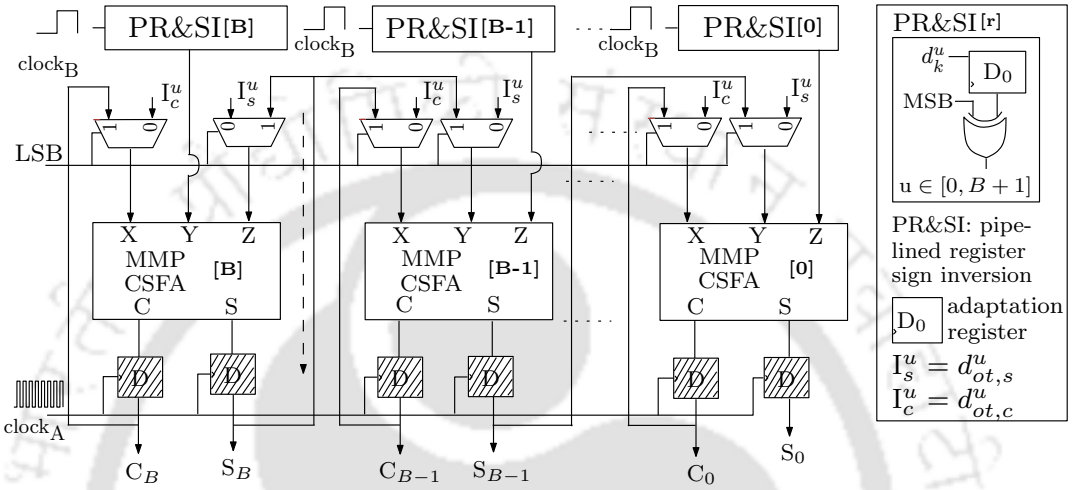


Fig. 2.6: Radix-2 pipelined MMP-CSFA based SA unit for OBC-DA.

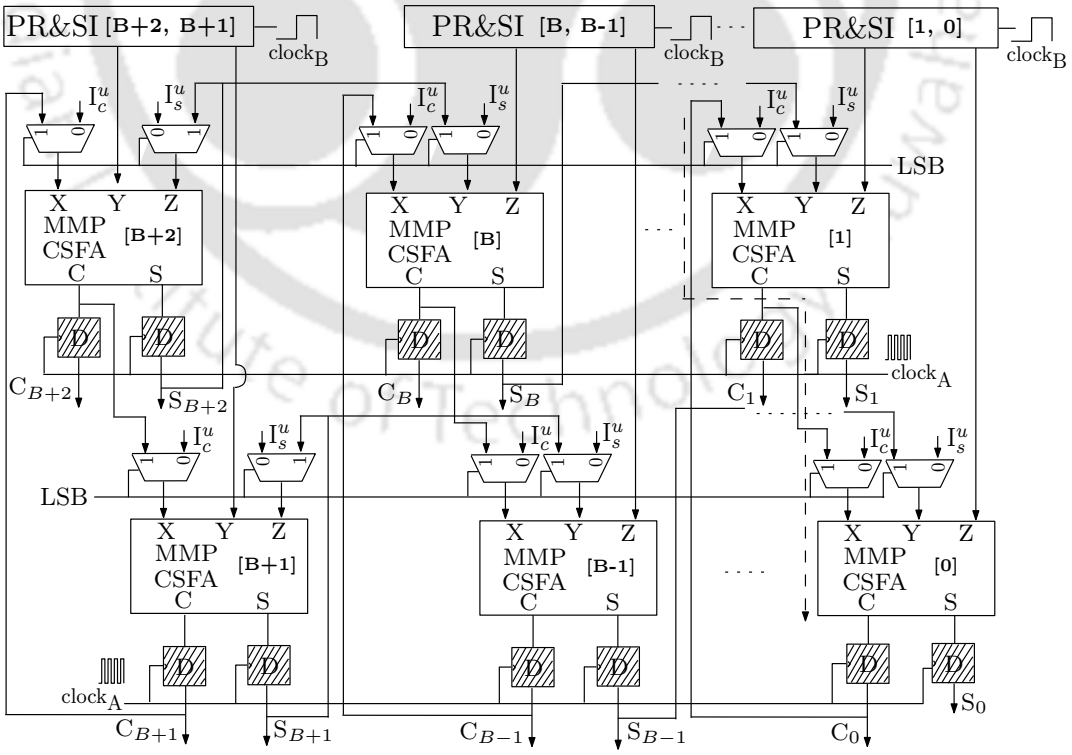


Fig. 2.7: Radix-4 pipelined MMP-CSFA based SA unit for OBC-DA.

The pipelined structure of proposed MMP-CSFA based SA unit (PSAR) is shown in Fig. 2.6. It

consists of one pipelined register and sign inversion (PR&SI) unit, $(B+\frac{r}{2})$ MMP-CSFA, $2(B+\frac{r}{2})$ 2-to-1 multiplexers (bit-level), and $2(B+\frac{r}{2})$ D flip-flops (DFF). PR&SI unit further comprises of $(B+\frac{r}{2})$ XOR gates and D_0 registers. According to (2.9), LSB and MSB signals are used to load $d_{ot}(n)$ term and invert the sign of LUT contents respectively. Unlike in Fig. 1.14, bit-level 2-to-1 multiplexers are employed to load sum $I_s^u = d_{ot,s}^u(n)$ and carry $I_c^u = d_{ot,c}^u(n)$ components of $d_{ot}(n)$ with $u \in [0, B+1]$. Note that the pipelined D_0 register in PR&SI unit operates with clock_B and SA unit runs at clock_A. The duration of clock_A and clock_B are defined as bit-iteration period (BiIP) and byte-iteration period (ByIP) respectively. BiIP is estimated by the delay of XOR gate in PR&SI unit, MMP-CSFA and delay of accumulator. Note that the computation time of radix-2 MMP-CSFA is governed by sum-propagation delay. In contrast, radix-4 PSAR unit performs two bit addition and double right-shift in every clock_A cycle in accordance with (2.20), as shown in Fig. 2.7. Thus, its computation time is wholly determined by the carry-propagation delays of both MMP-CSFA. ByIP is estimated by the wordlength of filter coefficients multiplied with the duration of BiIP. Clearly, the number of clock_A cycles are reduced to half in radix-4 PSAR unit over its radix-2 counterpart. The theoretical and synthesized results of BiIP and ByIP for different CSFA based SA units are listed in Table 2.3.

Table 2.3: Comparison of Computational Complexities between Binary-CSFA and MMP-CSFA using TSMC 90 nm CMOS Library

Type	**Binary CSFA**				**Proposed MMP-CSFA**			
BE	S= X $\oplus$ Y $\oplus$ Z C = (X $\wedge$ Y) $\vee$ (Z $\wedge$ (X $\oplus$ Y))				S = X $\oplus$ Y $\oplus$ Z C = $\neg$ ($\neg$(X $\wedge$ Y) $\wedge\neg$(Z $\wedge$ (X $\oplus$ Y)))			
Gates	$2\times\oplus$, $2\times\wedge$ and $1\times\vee$				$2\times\oplus$ and $3\times(\neg,\wedge)$			
SA	**TC**				**OBC**			
BiIP†	$T_X + max(T_{FAS}, \log_2 r T_{FAC}) + T_D$				$T_{XM} + max(T_{FAS}, \log_2 r T_{FAC}) + T_D$			
BiIP‡	$T_X + max(T_{FAS}, \log_2 r T'_{FAC}) + T_D$				$T_{XM} + max(T_{FAS}, \log_2 r T'_{FAC}) + T_D$			
CT	**CSFA SA**		**MMP-CSFA SA**		**CSFA SA**		**MMP-CSFA SA**	
(ns)	BiIP	ByIP	BiIP	ByIP	BiIP	ByIP	BiIP	ByIP
radix 2	0.43	3.44	0.43	3.44	0.53	4.24	0.53	4.24
radix 4	0.63	5.04	0.53	4.24	0.83	6.64	0.73	5.84

BE: boolean expression, BiIP† : BiIP using binary CSFA, BiIP‡ : BiIP using MMP-CSFA; CT: computation time, ByIP= $W\times$ BiIP; $\oplus, \wedge, \vee, \neg$ and $(\neg, \wedge)$ denote XOR, AND, OR and NOT operators respectively. $T_{XM} = T_X + \log_2 r T_M$, $T_{FAS} = 2T_X, T_{FAC} = T_X + T_{AND} + T_{OR} = T_X + 4T_{NAND}, T'_{FAC} = T_X + 2T_{NAND}; T_M, T_X, T_{AND}, T_{OR}, T_{NAND}$ are delays of multiplexer, XOR, AND, OR and NAND gates respectively.

2.3.2 Architecture for Small Order Filter

The architecture of 4^{th} order proposed filter is shown in Fig. 2.8. It consists of one OBC-LUT, one PSAR, one OC unit, ECU and an ISR, a sign-magnitude logic (SML), four word-parallel to bit-serial (WPBS) converters, four barrel-shifters (BSs), four MMP-CSFA based bit-level accumulators, four registers in CURA and an IUR. It can be observed from Fig. 2.3 that the proposed structures produce few non-OBC terms during initial three clock_B cycles as bit-slices of filter coefficients $c_{i,k}(n)$ remains at zero for all $0 \leq i \leq N-1$ and $0 \leq k \leq W-1$. Hence, it is important to discuss the operation of proposed filter in these clock cycles. In 0^{th} clock_B cycle, the IUR content is updated along with the registers in CURA. In 1^{st} clock_B cycle, PSAR unit is fed with input $\frac{1}{2}x(n)$ from LUT as first non-OBC term since bit-slices of filter coefficients $c_{i,k}(n) = 0$. However, the presence of D_0 and D_1 registers just before the PSAR unit and filter output would produce the output '0'. In 2^{nd} clock_B cycle, the PSAR unit is fed with input $\frac{1}{2}[x(n) + x(n-1)]$ from LUT as second non-OBC term since bit-slices of filter coefficients $c_{i,k}(n) = 0$. However, the output would correspond to 0^{th} input sample $x(n)$ as per, $y(2) = \frac{1}{2}\sum_{k=0}^{W-1}\big(x(n)s_k - d_{ot}(0)s_{W-1}\big)$. In 3^{rd} clock cycle, the PSAR unit is fed with $\frac{1}{2}[x(n)+x(n-1)+x(n-2)]$ from LUT as third non-OBC term. The filter output corresponds to first input sample $x(n-1)$ as $y(3) = \frac{1}{2}\sum_{k=0}^{W-1}\big([x(n) + x(n-1)]s_k - d_{ot}(1)s_{W-1}\big)$. Similarly, in next clock cycle, the filter output corresponds to second input sample $x(n-2)$ as $y(4) = \frac{1}{2}\sum_{k=0}^{W-1}\big([x(n)+x(n-1)+x(n-2)]s_k - d_{ot}(2)s_{W-1}\big)$. However, in same clock cycle, next OBC combination of input samples is selected based on the bit-slices of filter coefficients $c_{i,k}(n)$. These non-OBC terms have to be eliminated, otherwise it would enter into CUU and deviate the actual operation of LMS ADF. Based on above discussion, the output contains $d_{ot}(n)$ which can be implemented, as per (2.10) using byte-level accumulator (ByOC) with $-\frac{1}{2}[x(n)+x(n-N)]$ as its input, as shown in Fig. 2.8. Instead, it can be obtained as $d_{ot}(n) = -d_k(n)$ for initial m clock cycles. Observably, the output of ByOC cannot be loaded to PSAR unit as it operates on clock_A. In order to load $d_{ot}(n)$ in PSAR unit during the start of every byte-iteration, one has to obtain its bit-slices. This is possible using bit-level accumulator (BiOC) in which register is retimed and bit-slices are obtained through WPBS to perform accumulation.

An adaptive equalization problem is considered to observe the effect of non-OBC terms on the convergence performance of the proposed designs. Hereafter, $\text{OBC-LUT}_{\text{I, II, III}}$ are referred as Structure-I, II, III respectively by including ECU and CUU. For reference, conventional DA [67] and pipelined DA [71] based LMS ADFs are considered in the simulations. In all the cases, a linear dispersive

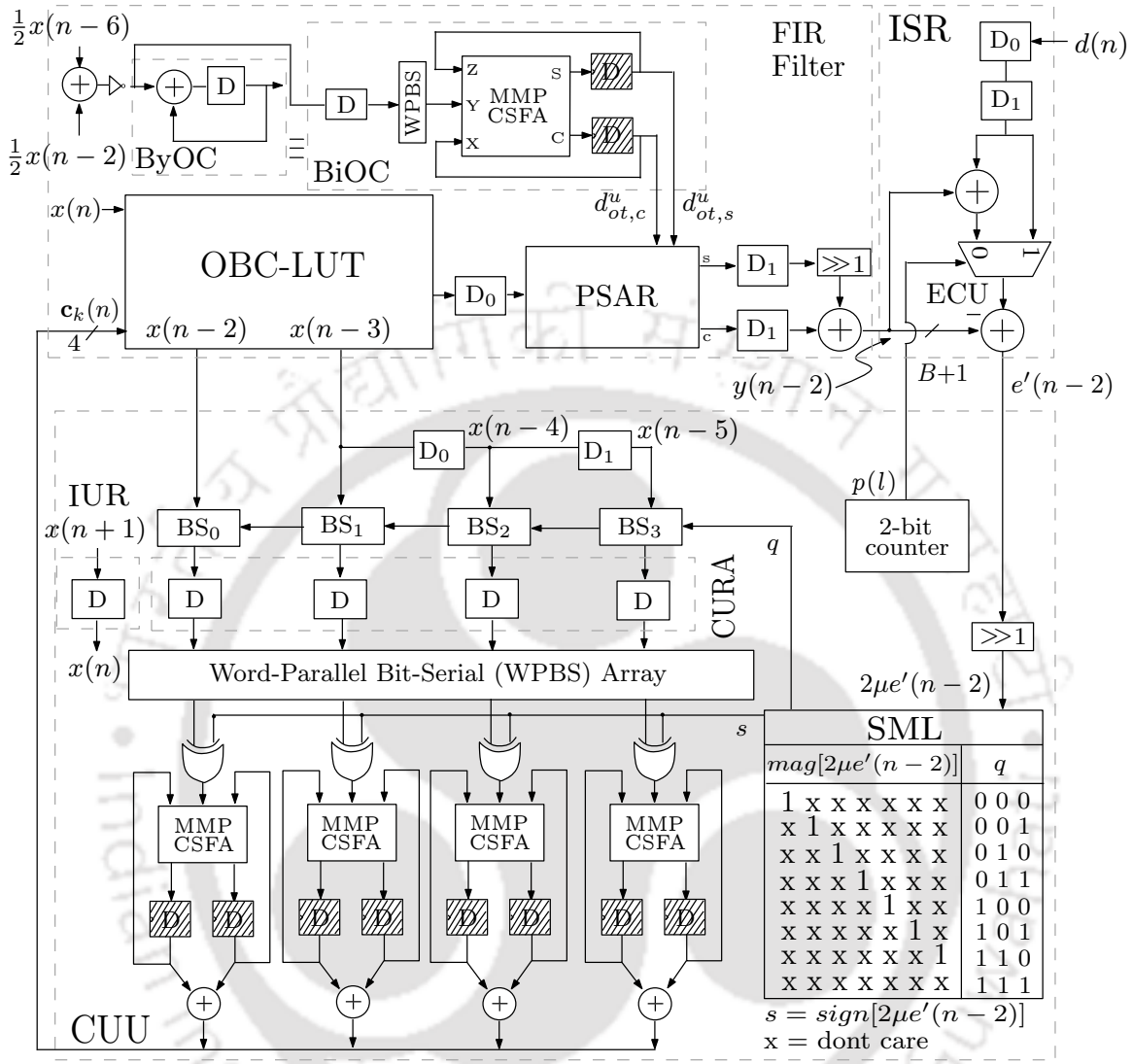


Fig. 2.8: Circuit schematic of 4th order OBC-DA based LMS ADF.

channel is employed to produce random distortion, and an unknown sequence is generated which acts as the source of additive white Gaussian noise of zero mean and unit variance. The impulse response $h(n)$ of the channel is assumed to be raised-cosine [54], as per

$$h(n) = \frac{1}{2}\left[1 + \cos\left(\frac{2\pi(n-2)}{P}\right)\right] \text{ for } n = 1, 2, 3 \tag{2.21}$$

and $h(n) = 0$ otherwise, where P is the distortion parameter to produce random fluctuations in the input amplitude. The values of P and μ are taken to be 3.1 and 0.0625 respectively. The mean squared error (MSE) learning curves of 32nd order proposed and existing [67,71] designs averaged over 50 independent runs are shown in Fig. 2.9(a). As expected, non-OBC terms would have ill-effects on

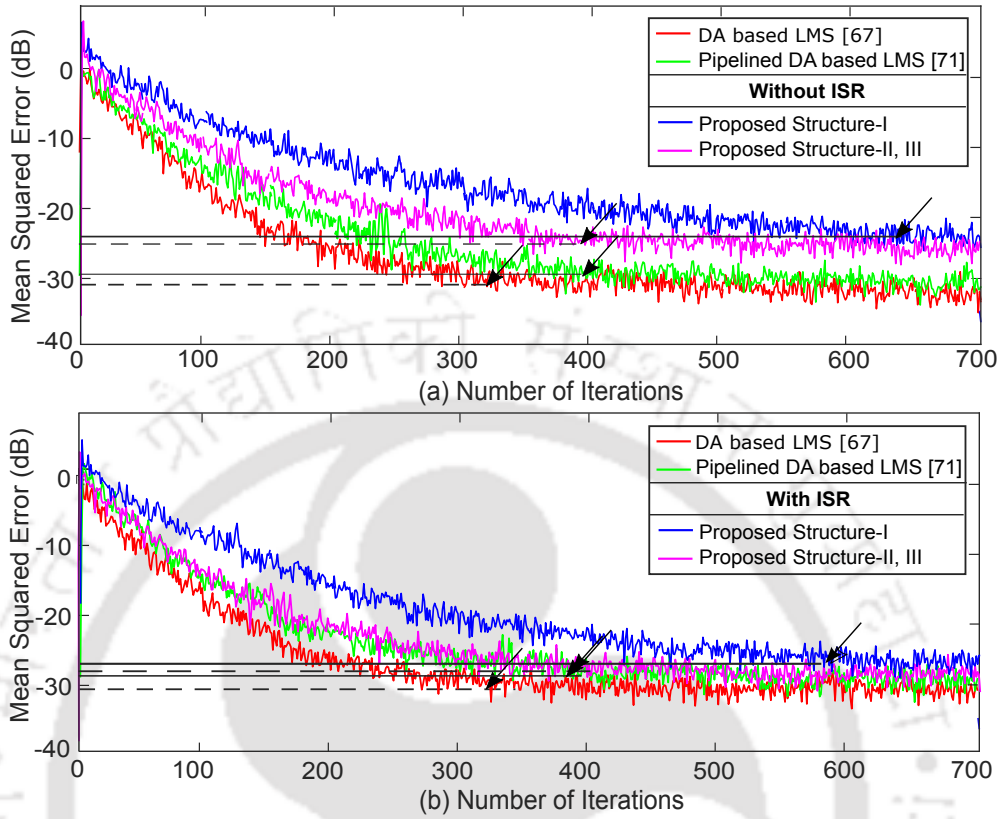


Fig. 2.9: MSE learning curves of adaptive equalization problem for the presented and existing DA based designs [67, 71] (a) without ISR unit (b) with ISR unit.

convergence rate and steady-state error of the proposed designs. However, these are more pronounced for Structure-I as compared to the other proposed structures as it exhibits inherent pipelining. Moreover, the presence of non-OBC terms further deteriorates the convergence performance. On the other hand, the Structure-II, III also have poor convergence rate and steady-state error, but this is mainly due to non-OBC terms. The correct operation of these filters could only be resumed after removal of non-OBC terms. This is possible via modification in ECU as

$$e'(n-2) = [p(l)(\gamma(l) + d(n-2)) + \overline{p}(l)d(n-2)] - y(n-2) \tag{2.22}$$

where $\gamma(l)$ is the correction factor and $p(l)$ is the control signal for enabling correct computation of error signal $e'(n-2)$. The definitions of $\gamma(l)$ and $p(l)$ are respectively, given as

$$\gamma(l) = \frac{1}{2}\sum_{k=0}^{W-1}\left[\sum_{i=0}^{l} x(n-i)\right](s_k - s_{W-1}) \text{ , and} \tag{2.23}$$

$$p(l) = \lfloor n/(l+1) \rfloor \tag{2.24}$$

where $0 \leq l \leq m$ $\forall$ byte-iteration n with $\overline{p}(l) = 1 - p(l)$ and $p(l) \in [0, 1]$. It is worth noting that the correction factor $\gamma(l)$ is the filter output $y(n)$ for initial three clock_B cycles. The convergence performance of the proposed designs after the removal of non-OBC terms are shown in Fig. 2.9(b). Clearly, about 5 dB reduction in steady-state error is observed with slight improvement in convergence rate as compared to their without-ISR counterpart.

2.3.2.1 Complexity Considerations of Coefficient Update Unit

In every byte-iteration, $2\mu e'(n-2)$ is computed by multiplying 2μ with $e'(n-2)$, in accordance with (2.12). This can be performed in more simple manner based on the approximations used in [67]:

- Step-size μ is taken in the orders of $O(1/N)$ to simplify the multiplication of μ and $e'(n-2)$ to a greater extent. For simplicity, we consider $\mu = 2/N$ which is twice that of [71], one could also choose μ as $2^{-v+1}/N$, where v is a positive integer which requires pre-shifting of input samples by t.
- Except the MSB of $e'(n-2)$, all the other bits are taken as don't care ($\times$).

Clearly, $2\mu e'(n-2)$ is nothing but a right-shift version of $e'(n-2)$, if N is assumed in the powers of 2. This can be understood by splitting $2\mu e'(n-2)$ into $s = sign[2\mu e'(n-2)]$ and $t = mag[2\mu e'(n-2)]$, where $sign[\bullet]$ and $mag[\bullet]$ indicate sign and magnitude operators respectively. By utilizing t, each BS provides controlled shifts $\overline{q} = \log_2 t$ on respective input $x(n-i-2)$ for all $0 \leq i \leq N-1$ to produce the coefficient increment terms $2\mu e'(n-2)x(n-i-2)$. The sign of coefficient increment terms in every byte-iteration is decided by s. There are two approaches to update filter coefficients. First, the coefficient increment terms are directly fed to coefficient update accumulators [71]. Subsequently, the bit-slices of filter coefficients are obtained using an array of WPBS. Second, the bit-slices of coefficient increment terms are obtained prior to update of coefficient coefficients, as per (2.12). This needs coefficient update registers CURA to be placed before WPBS, as shown in Fig. 2.8. It would update the coefficient increment terms using MMP-CSFA accumulators. As a result, the complexity of CUU is reduced since update of filter coefficients are performed at bit-level. However, this requires carry-out signals of MMP-CSFA to be flushed out in the end of every byte-iteration.

2.3.3 Architecture for Large Order Filter

It is clear from the previous discussion that OBC reduces the complexity of LUT. However, the number of non-OBC terms grows proportionally with the number of adaptation delays m. This can be

resolved by decomposing N^{th} order LUT into $M(=N/L)$ number of L^{th} order LUTs, where $L < N$. It computes the output $y(n)$ by adding the outputs from M-DA base units, according to

$$y(n) = \sum_{j=0}^{M-1} y_j(n) = \sum_{j=0}^{M-1} \left[\sum_{i=jL}^{(j+1)L} x(n-i)w_i(n) \right] \tag{2.25}$$

Each L^{th} order DA base unit accordingly, have L CUUs. By applying OBC-DA in (2.25), we get

$$y(n) = \sum_{j=0}^{M-1} \left[\sum_{k=0}^{W-1} d_{k,j}(n)s_k + d_{ot,j}(n)s_{W-1} \right] \tag{2.26}$$

Clearly, each DA base unit contains an OBC-LUT and SA unit which are added to produce the final output using an adder tree. The terms $d_{k,j}(n)$, $d_{ot,j}(n)$ are, respectively, given as

$$d_{k,j}(n) = \frac{1}{2} \sum_{i=jL}^{(j+1)L-1} x(n-i)d_{i,k},\ d_{ot,j}(n) = -\frac{1}{2} \sum_{i=jL}^{(j+1)L-1} x(n-i) \tag{2.27}$$

where $j = 0, 1, 2, ..., M-1$. The proposed 16^{th} order filter is shown in Fig. 2.10. It consists of four 4^{th} order DA base units (as shown in Fig. 2.8) whose outputs are added by two separate binary adder trees, corresponding to four $(B+1)$-bit sum and carry words. In this case, truncation of three LSBs of $e'(n-2)$ are required, since $\mu = 0.125$ in order to store coefficient increments within the registers in CURA. The filter performance mainly depends on MSBs of $2\mu e'(n-2)$, therefore, the truncation of LSBs would have negligible effect. The error signal $e'(n-2)$ is given as follows

$$e'(n-2) = [p_j(l)(\gamma_j(l) + d(n-2)) + \overline{p}_j(l)d(n-2)] - y(n-2) \tag{2.28}$$

where $\gamma_j(l)$ and $p_j(l)$ are correction factor and control signal of each DA base unit which are, respectively, defined as

$$\gamma_j(l) = \frac{1}{2} \sum_{k=0}^{W-1} \left[\sum_{i=0}^{l} x(n-i-jL) \right] (s_k - s_{W-1}),\ \text{and} \tag{2.29}$$

$$p_j(l) = \lfloor n/(l+1+jL) \rfloor \tag{2.30}$$

where $j(L-m) \leq l \leq m+jL$ and $0 \leq j \leq M-1\ \forall$ byte-iteration n. For every $m+1$ initial clock cycles of DA base unit $\gamma_j(l)$ is computed by enabling the corresponding registers in CURA using $E_j = 1$ and fed to ISR unit. This requires a M-to-1 multiplexer to pass each $\gamma_j(l)$, with its selection being governed by the control signal $r_j = j$. The control signals $p_j(l)$ and r_j can be realized by using two separate modulo circuits, as shown in Fig. 2.10. The coefficient update equation for j^{th} DA base unit

would be

$$\mathbf{c}_{j,k+1}(n)s_k = \mathbf{c}_{j,k}(n)s_k + D_k\{2\mu e'(n-2)\mathbf{x}_j(n-2)\} \tag{2.31}$$

where $\mathbf{c}_{j,k}(n) = [c_{j0,k}(n), c_{j1,k}(n), ..., c_{jL-1,k}(n)]$, $\mathbf{x}_j(n-2) = [x(n-jL-2), x(n-jL-3), ..., x(n-jL-L-2)]$ and $c_{ji,k}(n)$ are the bit-slices of filter coefficients of j^{th} DA base unit. Algorithm 1 explains the operation of proposed LMS ADFs.

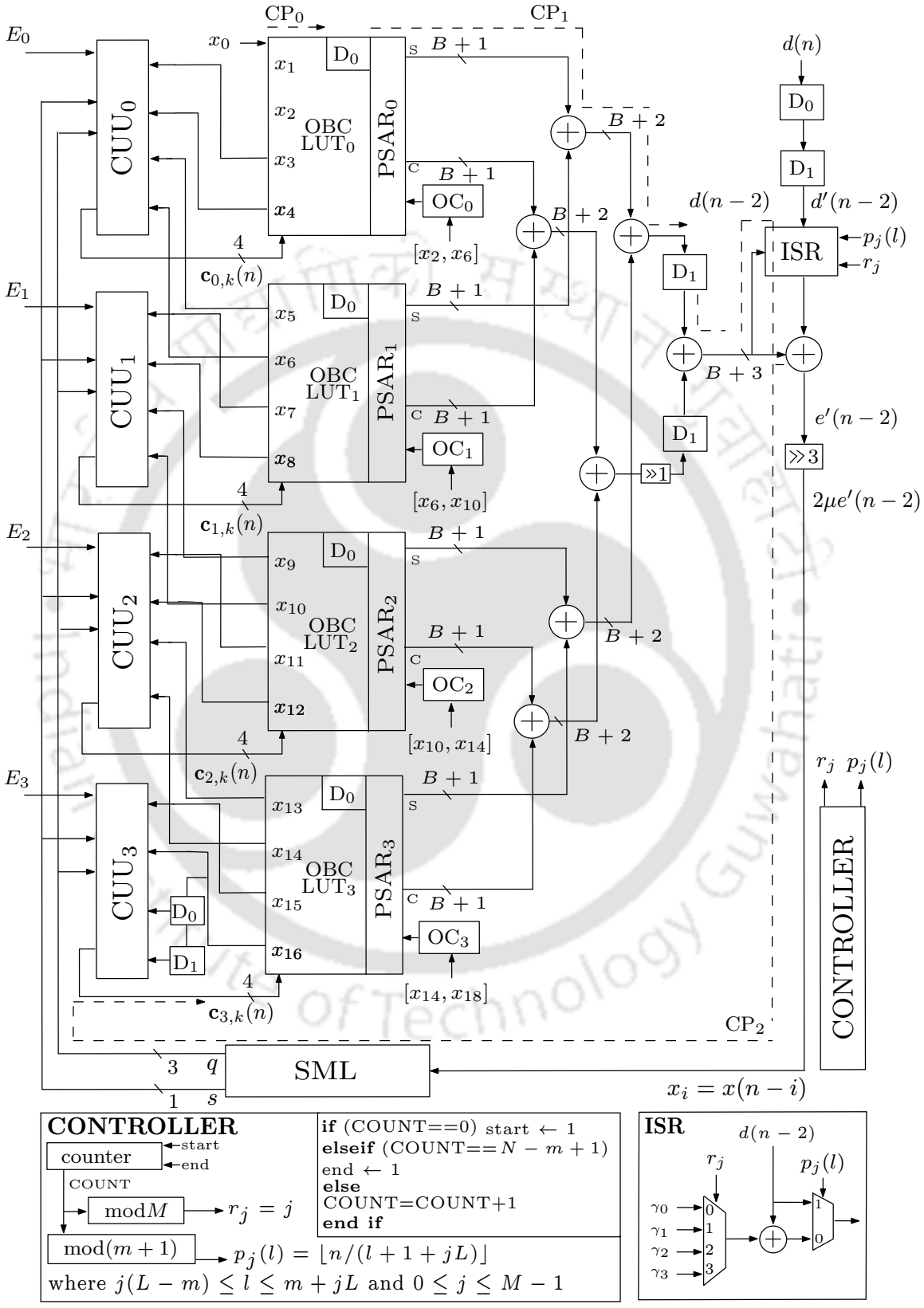


Fig. 2.10: Circuit schematic of 16th order OBC-DA based LMS ADF with an ISR unit and a controller.

Algorithm 1 Algorithm explaining the operation of the proposed LMS ADFs

Initialize:

$m = 2; s = 0, 1, 2, ..., L-1; l = 0, 1, 2, ..., m; j = 0, 1, ...M-1; k = 0, 1, 2, ..., W-1$

loop

$$y(n-m) = \sum_{j=0}^{M-1}\left[\sum_{k=0}^{W-1} d_{k,j}(n-m)s_k + d_{ot,j}(n-m)s_{W-1}\right]$$

  for COUNT= 0 to $N-m$ **do**

    if COUNT mod $(M) == 0$ **then**

      $j \leftarrow j+1$, $r_j \leftarrow j$, $E_j \leftarrow 1$

      $d_{ot,j}(n-m) = d_{ot,j}(n-m-1) - \frac{1}{2}[x(n-jL-m) + x(n-(j+1)L-m)]$

      $\gamma_j(l) = \frac{1}{2}\sum_{k=0}^{W-1}\big[\sum_{i=0}^{l} x(n-i-jL)\big](s_k - s_{W-1})$

      for $i = 0$ to $L-1$ **do**

        $d_{k,j}(n-m+1) = d_{k,j}(n-m) + x_i(n)d_{i,k}$

        for $k = 0$ to $W-1$ **do**

          if $k == 0$ **then**

            $y_j(n-m) = d_{ot,j}(n-m)$

          else $y_j(n-m) = y_j(n-m) + d_{k,j}(n-m)s_k$

          end if $\mathbf{c}_{j,k+1}(n)s_k = \mathbf{c}_{j,k}(n)s_k + \Delta_{j,k}$

        end for

      end for

      $y(n-m) = y(n-m) + y_j(n-m)$

      $\Delta_{j,k} \xleftarrow{D_k} 2\mu e'(n-m)\mathbf{x}_j(n-m)$

      $x(n-jL) \xleftarrow{D} x(n-jL+1)$

    end if

    if COUNT mod $(m+1) == 0$ **then**

      $p_j(i) \leftarrow 0$, $d'(n-m) = \gamma_j(l) + d(n-m)$

    else $p_j(i) \leftarrow 1$, $d'(n-m) = d(n-m)$

    end if $e'(n-m) = d'(n-m) - y(n-m)$

  end for $n+1 \leftarrow n$

end loop

2.4 Performance Comparison

In the following discussion, the existing designs [67], [68], [69], [70] and [71] are referred as DA_0-ADF, DA_1-ADF, DA_2-ADF, DA_3-ADF and DA_4-ADF respectively. For comparison, the radix-2 implementation of the proposed and existing designs are considered.

2.4.1 Computational Complexities

2.4.1.1 Hardware Complexities

The hardware complexities of the proposed and existing designs are listed in Table 2.4 in terms of number of DA base units M and order of DA base unit L, where filter order $N = L \times M$. The complexities of each hardware element is expressed in terms of bit-complexity (BiCom) and byte-complexity (ByCom). BiCom of all hardware elements are normalized with the wordlength of filter coefficients to compare with ByCom components. Unlike DA_3-ADF and DA_4-ADF designs, the ByCom of proposed designs grow linearly with DA base unit L. This is an important advantage when implementing the filter with large order of DA base units. The complexities of DA_0-ADF, DA_1-ADF and DA_2-ADF are limited by the size of their physical LUTs. However, the proposed designs have slightly higher cost due to ISR and OC units. Specifically, an ISR unit comprises of an adder and a 2-to-1 multiplexer. For high order filter, all of them require an extra M-to-2 multiplexer to select $\gamma_j(l)$ from each DA base unit. BiOC unit consists of an adder, a register, a WPBS shift register and a bit-level accumulator. Nevertheless, this increase in hardware is compensated by the proposed low-complexity OBC-LUT designs, as shown in Fig. 2.3. Like DA_0-ADF, DA_1-ADF and DA_2-ADF, all the proposed structures require a controller which involves a counter of $\lceil \log_2(N - m + 1) \rceil$-bit and two modulo circuits. Clearly, none of the proposed structures require multiplexers for LUT implementation, as in the case of DA_3-ADF and DA_4-ADF. Instead, they require XOR gates to generate possible OBC combinations of input samples. Structure-I operates with clock_B, thus, it has higher ByCom over the other proposed structures for all the filter order N. In contrast, it has relatively lower ByCom as compared to DA_3-ADF and DA_4-ADF. Hence, the proposed designs offer substantial savings in hardware as compared to the existing designs, especially when the order of filter becomes large. The comparison chart of registers, adders, multiplexers, XOR gates and LUT words involved in different designs are shown in Fig. 2.11 to Fig. 2.13. Clearly, the proposed Structures-I, II, III for 128^{th} order filter and 4^{th} order DA base unit requires 2.36, 2.36, 4.06 and 1.59, 1.99, 1.99 times less BiCom and

ByCom of adders; 1.38, 1.65, 1.65 and 2.22, 2.22, 2.22 times less BiCom and ByCom of registers; 15.48, 15.48, 15.48 and 14.54, 14.54, 14.54 times less BiCom and ByCom of multiplexers; 0.77, 1.06, 1.06 and 5, 5, 5 times more BiCom and ByCom of XOR gates as compared to DA_4-ADF respectively.

2.4.1.2 Time Complexities

The time complexities of both the proposed and existing designs are listed in Table 2.5 in terms of number of clock cycles (NCC), number of clock used (NOC), adaptation delays (AD), location of adaptation delays (LAD), critical paths (CP_0, CP_1 and CP_2) and throughput. It can be noted that CP_0, CP_1 and CP_2 indicate the time delay from the input sampling registers to D_0 register, the time delay from D_0 register to D_1 register, and the time delay from D_1 register back to sampling registers in CURA respectively, as shown in Fig. 2.10. It can also be noted that the LAD for the proposed designs are before PSAR and ECU units, unlike the case of DA_3-ADF and DA_4-ADF. Furthermore, CP_1 of DA_3-ADF and DA_4-ADF designs are higher, therefore, D_1 register is placed in the proposed designs just before the output of adder tree, as shown in Fig. 2.8. Likewise, D_0 register is placed in the proposed designs between OBC-LUT and PSAR units. Since DA_0-ADF, DA_1-ADF and DA_2-ADF designs are non-pipelined, therefore, they are comprised of single critical path, as indicated by CP_0 in Table 2.5. DA_3-ADF and DA_4-ADF designs, on the other hand are pipelined with two AD. As a result, two critical paths CP_0 and CP_1 are formed, but involve the byte-level CUU. While the proposed designs also have a third component of critical path CP_2 which includes the delay of carry propagate adder, ISR unit and ECU. Note that the delay of SML is ignored as it does not depend on the number of DA base units. Like DA_3-ADF and DA_4-ADF, the CP_0 of the proposed designs involve computational delay of OBC-LUT. However, it also includes the delay from CURA in CUU to adder tree (except Structure-I) of OBC-LUT due to the presence of bit-level CUU. For clarity, let us discuss the critical path CP_0 of Structure-III which includes the delay of WPBS (T_D), bit-level coefficient update accumulator ($T_X + T_D + T_{FA'}$) and OBC-LUT_{III} ($T_X + 2T_A$), where $T_{FA'}, T_X, T_D$ and T_A are computational delays of MMP-CSFA, XOR gate, 1-bit delay FF and adder respectively.

The throughput of DA based ADF is defined as the ratio of clock rate to the number of clock cycles required to produce the filter output. It is measured as the number of samples processed per second (NSPPS). The throughput of different designs are listed in Table 2.5. As discussed Structure-I operates on $clock_B$ and require $N-1$ cycles to produce the correct output for a given bit-slice. Thus, in total $(N-1)W$ $clock_B$ cycles (or alternatively, $W(L-1)\log_2 M$ $clock_B$ cycles for higher

filter order) are required to produce the filter output. The clock rate of the system is decided by maximum of the critical paths CP_0, CP_1 and CP_2. The throughput plots of the proposed and existing designs for various filter orders are shown in Fig. 2.14. It can be noted that DA_3-ADF and DA_4-ADF designs provide highest throughput for small filter orders due to parallel update of LUT contents and their pipelined nature. However, when the order of filter ($N = L \times M$) becomes large, CP_1 dominates over CP_0 due to large DA-base units (M), which is lower than the CP_1 of DA_3-ADF and DA_4-ADF. In case of Structure-I, the throughput performance is relatively higher than DA_0-ADF, DA_1-ADF and DA_2-ADF designs, however it reduces with higher order filter implementation. It is quite obvious that the throughput of DA_0-ADF, DA_1-ADF and DA_2-ADF designs are significantly lesser than the other pipelined designs. This is because several $clock_B$ cycles are required to produce the output depending on the number of DA base units, order of DA base units and wordlength of filter coefficients. For example, the implementation of 128^{th} order filter using 4^{th} order DA base units requires NCC values as 29, 22 and 18 for DA_0-ADF, DA_1-ADF and DA_2-ADF designs respectively. Interestingly, the proposed designs require single clock cycle to produce the output like DA_3-ADF and DA_4-ADF except the proposed Structure-I.

2.4.2 Implementation Results

2.4.2.1 ASIC Synthesis

In order to validate the proposed designs, 16^{th} and 32^{nd} order filters using 4^{th} order DA base units are coded in Verilog. In addition, the existing designs DA_0-ADF, DA_1-ADF, DA_2-ADF, DA_3-ADF and DA_4-ADF are also coded for the same filter orders and the number of DA base units. The wordlengths of input samples and filter coefficients are taken to be 8-bit. ASIC synthesis is performed for both the proposed and existing designs using TSMC 90 nm CMOS Library [79] by Cadence RTL Compiler. Synthesis reports of the proposed and existing designs in terms of area, power, minimum sampling period (MSP) and throughput are listed in Table 2.6. It is clear that all the proposed architectures occupy less area and consume less power over DA_4-ADF. For example, a 32^{nd} order Structure-I, II and III offers 71.13%, 71.83% and 73.08% less area and consumes 68.47%, 70.01% and 72.17% less power as compared to DA_4-ADF. Further, the savings in area and power are much more when the order of DA base unit becomes large. In addition, the proposed structures (except Structure-I) provide superior throughput performance for higher filter orders, as shown in Fig. 2.14. For better comparison, ADP and PDP figures of different designs are also listed in Table 2.6. Evidently, Structure-I has smallest

critical path, however, it occupies more area and consumes more power. Besides, it provides slower convergence rate over the other proposed architectures. In contrast, DA_3-ADF and DA_4-ADF designs offer high throughput, but requires more area and power. Thus, the proposed designs have better ADP and PDP figures and eventually these would be more advantageous for higher orders. For example, 32^{nd} order for Structure-I, II and III become 51.74%, 37.83% and 45.38% less ADP and consume 47.33%, 33.72% and 43.19% less PDP as compared to DA_4-ADF.

2.4.2.2 FPGA Synthesis

DA based FIR filter naturally contains a LUT which is mapped to the LUT of specific FPGA depending upon the filter order. The performance of the proposed and existing designs are also evaluated on Xilinx ZYNQ-XC7Z020-1CLG84C FPGA device. It mainly consists of block random access memory (block RAM), 6-input distributed RAMs and flip-flops (FF). Distributed RAMs are arranged in groups of four to make a slice (referred as SLUT). The physical LUTs of DA_0-ADF, DA_1-ADF and DA_2-ADF designs are mapped onto the FPGA by using the HDL primitive [80]. According to (2.27), LUT size is reduced by computing the partial sums in parallel such that L fits the SLUT size. While the wordsize of SLUT ($\tilde{B}$) is chosen to be $\tilde{B} = B + \log_2 L$ to store the possible partial products corresponding to each DA base unit. Apart from LUT, adders are also mapped to several slices with each containing upto four 6-input distributed RAMs. All the designs are implemented on FPGA device for 16^{th} and 32^{nd} order filters using 4^{th} order DA base unit. The corresponding SLUT and FF results are listed in Table 2.6 by setting the system clock at 100 MHz. Clearly, the proposed structures occupy less number of SLUT and FF over all the other existing designs. The Structure-I, II and III with 16^{th} order filter and 4^{th} order base unit involves 53.26%, 57.61% and 63.04% less number of SLUT and 34.40%, 38.60% and 42.06% less number of FF as compared to DA_4-ADF. While these figures for 32^{nd} order filter and 4^{th} order base unit become 55.11%, 59.66% and 64.20% less number of SLUT and 35.33%, 40.28% and 44.87% less number of FF as compared to DA_4-ADF. Interestingly, the savings in SLUT and FF can be enhanced by using large DA base units. One can also extrapolate these results to find the approximate values of SLUT and FF for higher order implementation of the proposed structures.

2.5 Conclusion

In this Chapter, three different optimal complexity architectures (Structure-I, II and III) for pipelined DA based LMS ADF using OBC have been investigated. The complexity optimization has been performed individually on LUT, SA unit and coefficient update unit. It is found that the LUT optimization has resulted substantial decrease in the hardware complexity. However, non-OBC terms have been generated at the output of filter during initial sampling clock cycles. A novel design for the offset term due to OBC representation of filter coefficients has been presented. These architectures furnished the optimal cost for LMS ADF which makes them amenable to VLSI implementation. Nonetheless, they also offered better throughput figures with similar convergence performance as compared to the design presented in [71]. The proposed architectures are validated through ASIC and FPGA synthesis, and are found to provide significant reduction in area, power, ADP, PDP and logic utilization.

Table 2.4: Comparison of Hardware Complexities between DA Based Designs for N^{th} Order Filter, L^{th} Order Base Unit and W-bit Wordlength of Filter Coefficients

Design	**Adders**		**Registers**		**Multiplexers**		**XOR gates**		**BS**	**LUT**
	BiCom*	**ByCom**	**BiCom***	**ByCom**	**BiCom***	**ByCom**	**BiCom***	**ByCom**		
DA_0-ADF‡ [67]	–	$3M+1$	–	$(1+L)M+2$	$L(L-1)M$	–	–	M	M	$2^{L+1}M$
DA_1-ADF‡ [68]	–	$(L+2)M+1$	–	$(2+3L)M+1$	LM	–	–	M	LM	$2^{L}M$
DA_2-ADF‡ [69]	–	$4M+2$	–	$(3+L)M+2$	$(L-1)(L-2)M$	$2M$	$2(L-1)M$	M	M	$2^{L}M$
DA_3-ADF† [70]	$\alpha_0 M$	$(3.2^{L-1}+1)M$	$\alpha_0 M$	$(3+3.2^{L-1}+L)M+2$	$2(2^{L}-1)M$	$(2^{L}-1)M$	$(L+2)M$	M	$(2^{L}-1)M$	–
DA_4-ADF† [71]	$\alpha_0 M$	$(2+2^{L-1}+L)M$	$\alpha_0 M$	$(3+2^{L}+2L)M+4$	$2(2^{L}-1)M$	$(2^{L}-1)M$	$(L+2)M$	M	LM	–
Structure-I†	$(\alpha_1+1)M+\alpha_3$	$(2L+3)M+1$	$(\alpha_1+L+2)M+\alpha_4$	$(2L+4)M+7$	$2M-2$	$M+1$	$(L(L-3)/2)M+2M-2$	$(L+1)M$	LM	–
Structure-II†	$(\alpha_1+1)M+\alpha_3$	$(L+3)M+1$	$2(L+1)M+\alpha_4$	$(2L+4)M+7$	$2M-2$	$M+1$	$(L+1)M$	$(L+1)M$	LM	–
Structure-III†	$(\alpha_2+1)M+\alpha_3$	$(L+3)M+1$	$2(L+1)M+\alpha_4$	$(2L+4)M+7$	$2M-2$	$M+1$	$(L+1)M$	$(L+1)M$	LM	–

‡ : non-pipelined structure, † : pipelined structure, BS: barrel shifter, BiCom: bit-complexity, ByCom: byte-complexity, BiCom* = ByCom/B, B: wordlength of input samples, W: wordlength of filter coefficients, assumed $B = W$; N: filter order, $M(= N/L)$: number of DA base-units, $\alpha_0 = 17$ for $L = 4$ of DA_3-ADF [70] and DA_4-ADF [71], $\alpha_1 = L(L-1)/2$, $\alpha_2 = L(\log_2 L + 1)/4$, $\alpha_3 = \log_2(LM) - 1$, $\alpha_4 = 2\alpha_3 - 2$. DA_0-ADF, DA_1-ADF and DA_2-ADF designs require a controller which are relatively complex than the proposed designs (a counter $\lceil \log_2(N-m+1) \rceil$-bits and two modulo circuits).

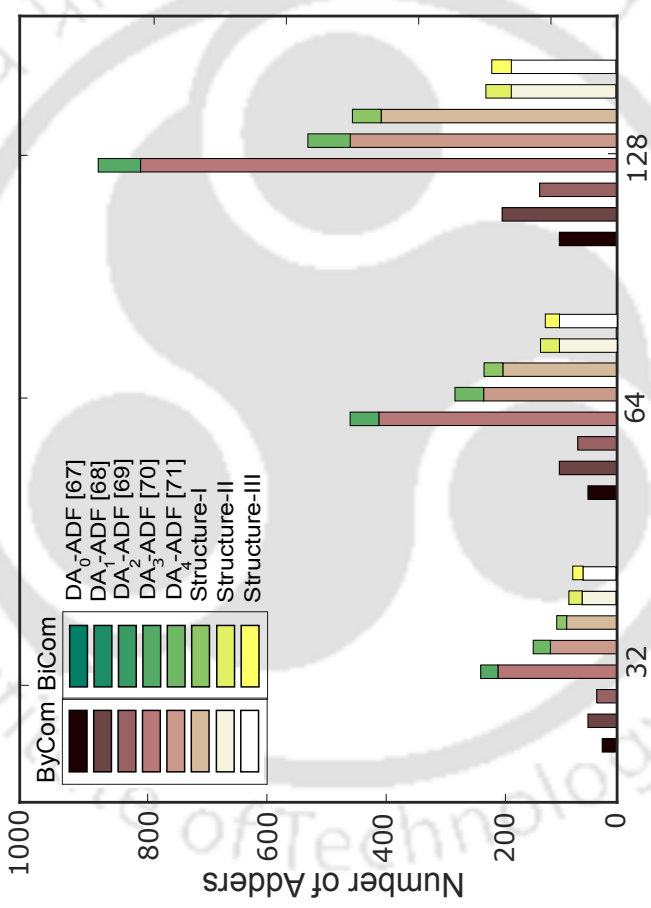


Fig. 2.11: Comparison of adders complexity for the presented and existing DA based designs.

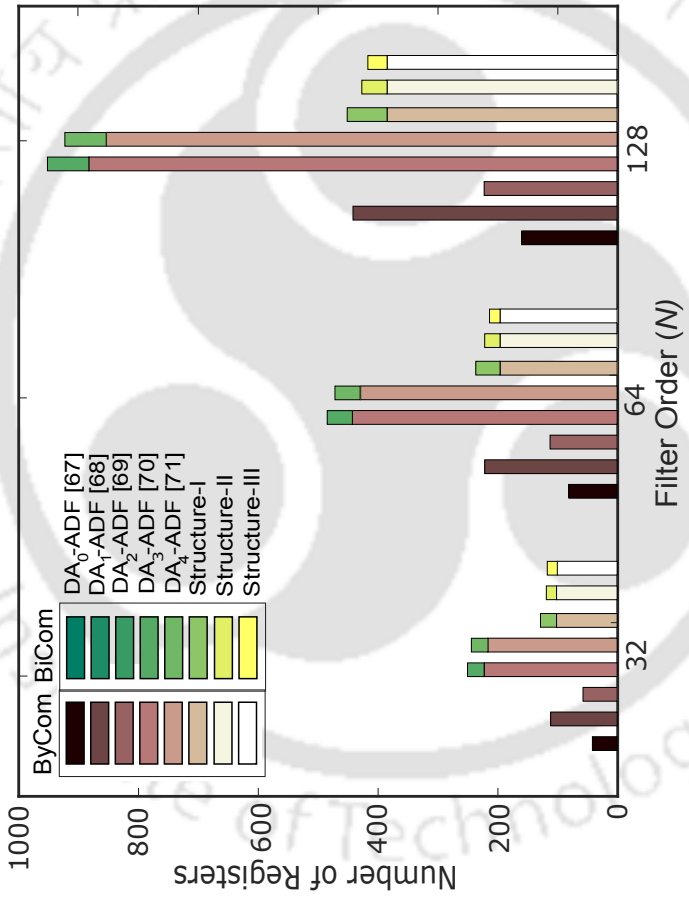


Fig. 2.12: Comparison of registers complexity for the presented and existing DA based designs.

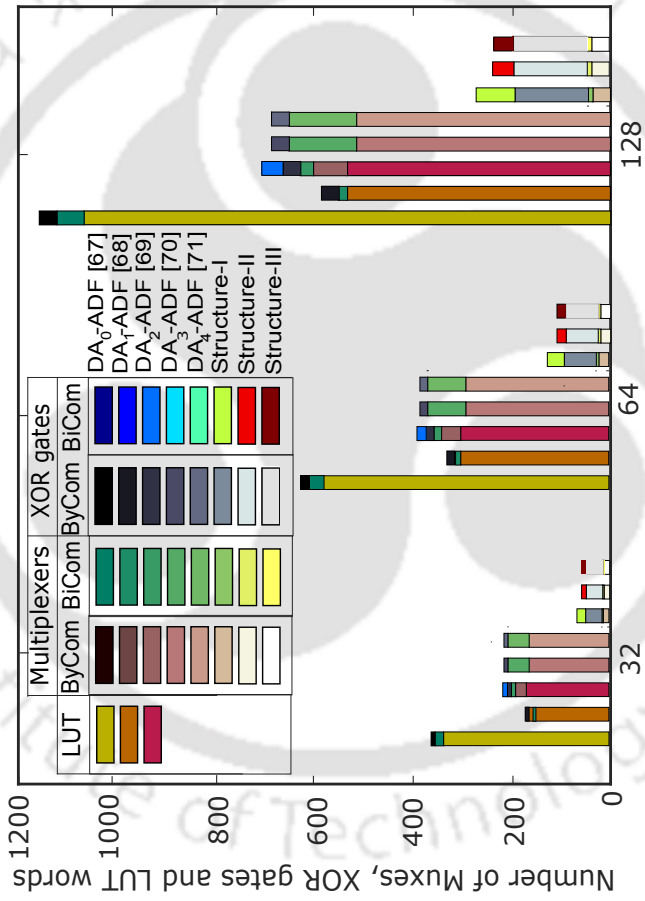


Fig. 2.13: Comparison of multiplexers, XOR gates and LUT words complexities for the presented and existing DA based designs.

Table 2.5: Comparison of Time Complexities between DA Based Designs for N^{th} Order Filter, L^{th} Order Base Unit and W-bit Wordlength of Filter Coefficients

Design	**NOC**	**NCC**	**AD**	**LAD**	**CP$_0$**	**CP$_1$**	**CP$_2$**	**Throughput**
DA$_0$-ADF‡ [67]	1	k_1	0	–	$T_{ACC}+3T_M+T_A$	–	–	$1/k_1\text{CP}_0$
DA$_1$-ADF‡ [68]	1	k_2	0	–	$T_{ACC}+T_A$	–	–	$1/k_2\text{CP}_0$
DA$_2$-ADF‡ [69]	1	k_3	0	–	$T_{ACC}+5T_M+2T_A$	–	–	$1/k_3\text{CP}_0$
DA$_3$-ADF† [70]	2	1	2	a-PSAR, a-ECU	$4T_M+T_{FA}+T_D$	$(2+\log_2 M)T_A$	–	$1/W[\max(\text{CP}_0,\text{CP}_1)]$
DA$_4$-ADF† [71]	2	1	2	a-PSAR, a-ECU	$4T_M+T_{FA}+T_X+T_D$	$(2+\log_2 M)T_A$	–	$1/W[\max(\text{CP}_0,\text{CP}_1)]$
Structure-I†	1	k_4	2	b-PSAR, b-ECU	$T_{FA'}+2T_X+2T_D+T_A$	$T_{FA'}+T_X+T_D+T_M+T_A\log_2 M$	$3T_A+(1+\log_2 M)T_M$	$1/k_4[\max(\text{CP}_0,\text{CP}_1,\text{CP}_2)]$
Structure-II†	2	1	2	b-PSAR, b-ECU	$T_{FA'}+2T_X+2T_D+3T_A$	$T_{FA'}+T_X+T_D+T_M+T_A\log_2 M$	$3T_A+(1+\log_2 M)T_M$	$1/W[\max(\text{CP}_0,\text{CP}_1,\text{CP}_2)]$
Structure-III†	2	1	2	b-PSAR, b-ECU	$T_{FA'}+2T_X+2T_D+2T_A$	$T_{FA'}+T_X+T_D+T_M+T_A\log_2 M$	$3T_A+(1+\log_2 M)T_M$	$1/W[\max(\text{CP}_0,\text{CP}_1,\text{CP}_2)]$

NCC: number of clock cycles, NOC: number of clocks used, AD: adaptation delays (m), LAD: location of adaptation delays, a-PSAR: after PSAR unit, a-ECU: after ECU unit, b-PSAR: before PSAR unit, b-ECU: before ECU unit, $\text{clock}_\text{B} = W\text{clock}_\text{A}$, Throughput=1/[NCC $\times$ critical-path], k_1=$2^L+\max(W,2^{L-1})+\log_2 M$, $k_2=2^{L-1}+\log_2 M+W+1$, $k_3=2^{L/2}+\max(W,2^{(L/2)-1})+\log_2 M+1$, $k_4=W(L-1)\log_2 M$; T_{ACC}, T_M, T_A,T_{FA}, $T_{FA'}$ and T_D are computational delays due to LUT, multiplexer, adder, binary CSA, MMP-CSFA and D flip-flop (FF) respectively.

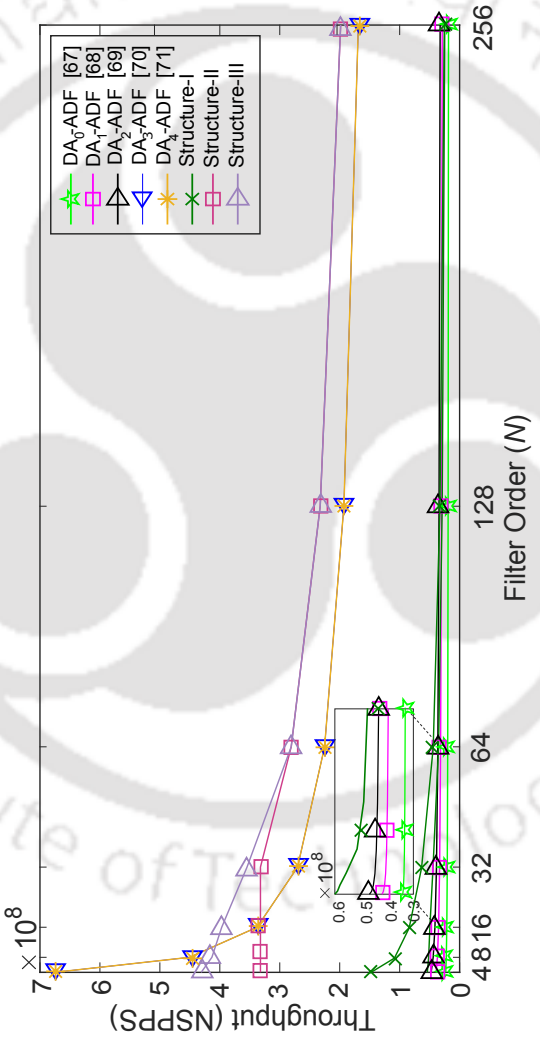


Fig. 2.14: Throughput curves for the presented and existing DA based designs.

Table 2.6: Performance Comparison of Different DA Based Designs with ASIC Synthesis using TSMC 90 nm CMOS Library and FPGA Implementation on Xilinx ZYNQ -XC7Z020-1CLG84C for 8-bit Wordlength of Filter Coefficients and 4^{th} Order Base Unit

Platform	**ASIC Synthesis**												**FPGA Synthesis**			
Design and Filter order	**Area** (μm^2)		**Power** (mW)		**MSP** (ns)		**Throughput** (per μs)		**ADP** (μm$^2\times$ ns)		**PDP** (mW $\times$ ns)		**SLUT** ($\times$ 1000)		**FF** ($\times$ 1000)	
N	16	32	16	32	16	32	16	32	16	32	16	32	16	32	16	32
DA_0-ADF‡ [67]	47938	94188	31.55	62.08	24.28	26.91	41.18	37.16	116334	2534599	766	1670	6.49	11.83	1.65	3.41
DA_1-ADF‡ [68]	39071	77168	26.88	50.89	16.21	18.07	61.69	55.34	633341	1394425	435	919	4.14	7.46	1.53	2.93
DA_2-ADF‡ [69]	29397	57934	15.25	28.11	30.45	33.07	32.84	30.23	895138	1915877	464	929	1.62	2.78	0.72	1.31
DA_3-ADF† [70]	56982	113796	19.23	39.83	4.56	4.56	219.3	219.3	259837	518909	87	181	0.65	1.24	2.67	4.63
DA_4-ADF† [71]	35177	69562	16.34	31.12	5.28	5.45	189.4	183.5	185734	379112	86	169	0.92	1.76	1.45	2.83
Structure-I†	10176	20084	5.19	9.81	8.37	9.11	119.5	109.8	85173	182965	43	89	0.43	0.79	0.95	1.83
Structure-II†	9813	19592	4.98	9.33	11.24	12.03	88.9	83.1	110298	235691	56	112	0.39	0.71	0.89	1.69
Structure-III†	9341	18722	4.76	8.66	10.19	11.06	98.1	90.4	95184	207065	49	96	0.34	0.63	0.84	1.56

MSP: minimum sampling period, ADP: area-delay product, EPS: energy-per-sample.

3

Low Complexity Pipelined Convex Combination LMS Adaptive Filter

Contents

3.1 Introduction

ADFs are widely employed in various DSP and communication systems such as system identification, channel equalization, noise cancellation etc [5,6]. For system identification problem, it is desired that the filter coefficients of ADF reach the optimum solution of unknown system in reasonable time with low steady-state error. LMS and RLS are two popular algorithms for the adaptation of filter coefficients. RLS ADF offers fast initial convergence rate and low steady-state error compared to LMS ADF. But, its computational complexity are significantly higher than LMS ADF which makes the real-time operation difficult. On the other hand, LMS ADF either provides fast convergence rate or low steady-state error based on the selection of step-size. Solving the well known trade-off between the initial convergence rate and the mean-square error in steady state of LMS ADF has gained much attention. Variable step-size LMS ADF is a potential solution which achieves both fast initial convergence and low steady-state error [13]. Due to adaptation of step-size in every iteration, the hardware realization of such filter is difficult. In the sequel, many researchers suggested different approaches to address this problem [14–22]. For instance, Kwong et al. in [16] utilized the squared instantaneous error, Aboulnasr et al. in [18] exploited the auto-correlation of time-averaged error signals at adjacent time. In the past, the idea of combining two LMS ADFs in parallel with different step-sizes has gained significant interest as it is able to optimize the trade-off between convergence speed and steady-state error. It is referred as convex combination of LMS (CLMS) ADF, where filter with large step-size offers fast convergence and filter with small step-size provides low steady-state error [11,26]. But, this requires transfer of filter coefficients from one LMS ADF to other which is difficult from implementation point-of-view. Further, due to involvement of two LMS ADFs its complexity is twice that of conventional LMS algorithm. In addition, the presence of several MAC units in both the LMS ADFs lead to high computational requirements. Few attempts have been made in the past to reduce the complexity of coefficient transfer scheme [23–26]. Garcia et al. in [23] proposed a linear transfer of coefficients by interacting slow filter to fast filter during the transition stage, however it degrades the SNR performance. Later, Ruiz et al. in [24] suggested a new update rule to improve the SNR at the cost of increased computational requirements. Nascimento and de Lamare in [25] presented a low-complexity instantaneous transfer scheme based on sliding window approach for better convergence performance. Lu et al. in [26] proposed a new coefficient transfer scheme based on sign-adaptation approximations for low-complexity realization of CLMS ADF.

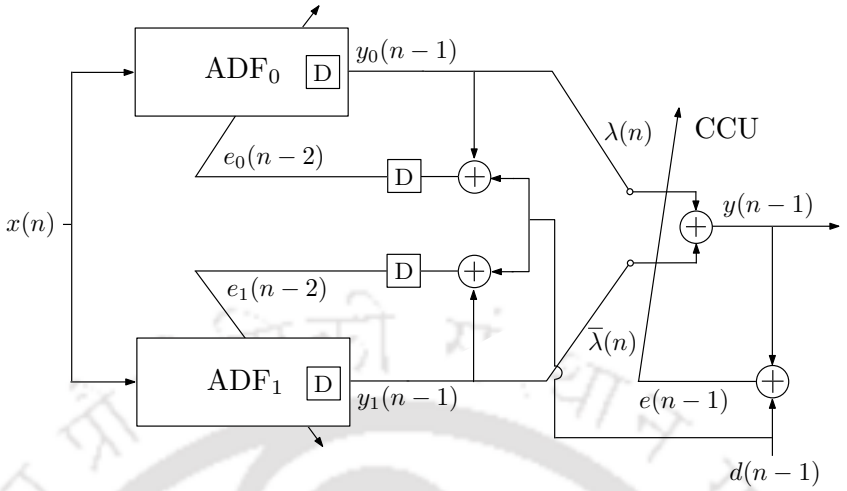


Fig. 3.1: Block diagram of pipelined CLMS ADF with two adaptation delays.

Distributed arithmetic (DA) as discussed in Chapter 1, can be employed for the realization of CLMS ADF since it eliminates the multipliers from the structure. The challenge here is to reduce the implementation complexity of CLMS ADF using DA, while providing the improved convergence characteristics over conventional LMS ADF. In [71], there is a brief discussion about the selection of step-size for the realization of DA based pipelined LMS ADF. For instance, the step-size for N^{th} order filter can be taken in the orders of $O(1/N)$ or $2^{-p}/N$, where p is positive integer. Selecting the step-size as $2^{-p}/N$ leads to low steady-state error as it can be realized by pre-shifting the input samples. To the best of our knowledge, no one has employed DA for low-complexity implementation of CLMS ADF. The purpose here is to explore the potential of DA for the realization of CLMS ADF with low implementation complexity.

3.2 Mathematical Formulation

Consider a N^{th} order pipelined CLMS ADF, as shown in Fig. 3.1 which processes an input sequence $\mathbf{x}(n) = [x(n), x(n-1), x(n-2), ..., x(n-N+1)]$ and produces the output $y_c(n)$, according to

$$y_c(n) = \sum_{i=0}^{N-1} x(n-i) w_i^c(n) \tag{3.1}$$

where $w_i^c(n)$ $(i = 0, 1, 2, ..., N-1)$ are the coefficients of ADF$_0$ and ADF$_1$ for $c = 0$ and $c = 1$ respectively. If every coefficient $w_i^c(n)$ of ADF$_0$ and ADF$_1$ is represented in W-bit two's complement

form, we get

$$w_i^c(n) = -w_{i,0}^c(n) + \sum_{k=1}^{W-1} w_{i,k}^c(n)2^{-k} \tag{3.2}$$

Substituting (3.2) in (3.1) and interchanging the order of summation, we obtain

$$y_c(n) = -\sum_{i=0}^{N-1} x(n-i)w_{i,0}^c(n) + \sum_{k=1}^{W-1}\left[\sum_{i=0}^{N-1} x(n-i)w_{i,k}^c(n)\right]2^{-k} \tag{3.3}$$

Define

$$d_{c,k}(n) = \sum_{i=0}^{N-1} x(n-i)w_{i,k}^c(n) \tag{3.4}$$

Hence,

$$y_c(n) = -d_{c,0}(n) + \sum_{k=1}^{W-1} d_{c,k}(n)2^{-k} \tag{3.5}$$

It is important to note from (3.4) that a single LUT is required to compute the output of ADF_0 and ADF_1 at different time instants. This is because the bit-slices of filter coefficients $w_{i,k}^c(n)$ $(k = 0, 1, 2, ..., W-1)$ would form the address lines to LUT storing the common set of input samples $x(n-i)$. In case of pipelined LMS filter, the feedback error and input samples arrive after certain clock cycles for the adaptation of coefficients. In present case, the outputs of ADF_0 and ADF_1 units are delayed by unit time which implies their respective errors can be computed as $e_c(n-1) = d(n-1) - y_c(n-1)$. The error signals so produced are further delayed by unit time. Thus, the coefficient update equation of pipelined CLMS ADF with two adaptation delays can be expressed as

$$\mathbf{w}^c(n+1) = \mathbf{w}^c(n) + \mu_c e_c(n-2)\mathbf{x}(n-2) \tag{3.6}$$

where $\mathbf{x}(n-2)$ and $e_c(n-2)$ are both delayed version of $\mathbf{x}(n)$ and $e_c(n)$ respectively. As discussed in Chapter 1, if the number of adaptation delays are kept small, then it does not have significant effect on the convergence performance of LMS ADF. Therefore, the number of adaptation delays are two for each component LMS filter with their positions indicated in Fig. 3.1. The purpose of placing the adaptation register between the delayed error signals $e_c(n-1)$ and $e_c(n-2)$ is to extract some property in order to improve convergence performance. For simplicity, the step-size μ_1 of ADF_1 is considered as the scaled version of step-size μ_0 of ADF_0 i.e., $\mu_1 = l\mu_0$ where l denotes the scaling factor such that $l < 1$. Mathematically, the scaling factor can be expressed as: $l = \sum_{q=1}^{K} a_q 2^{-q}$ and a_q is q^{th} bit in K-bit representation of l. For simplicity, l is assumed to be in negative power of two i.e., $\mu_1 = 2^{-p}\mu_0$, where p is a positive integer and $\mu_0 = 1/N$ [71]. Therefore, the coefficient update

equation for the proposed filter would become

$$\mathbf{w}^c(n+1) = \mathbf{w}^c(n) + \frac{2^{-cp}}{N} e_c(n-2)\mathbf{x}(n-2) \tag{3.7}$$

Let $\mathbf{\Delta w}^c(n) = \frac{2^{-cp}}{N} e_c(n-2)\mathbf{x}(n-2)$ be the coefficient increment terms of ADF_0 and ADF_1 corresponding to $c = 0$ and $c = 1$ respectively. Clearly, the coefficients increment terms of ADF_1 can be made right-shifted version of coefficient increment terms of ADF_0, if error computation from both the filters become equal. Thus, the same coefficient update equation could be used for ADF_1 by pre-shifting the input samples with p, in accordance with (3.7). In order to update filter coefficients at bit-level, the component of coefficient increment terms $\mathbf{\Delta w}_c(n)$ are to be stored in registers. This is possible by retiming the coefficient update registers, as per

$$\mathbf{w}^c_{k+1}(n)s_k = \mathbf{w}^c_k(n)s_k + D_k\{\mathbf{\Delta w}^c(n)\} \tag{3.8}$$

where the notations s_k and D_k have the same meaning, as discussed in Chapter 2. The output of combined filter as shown in Fig. 3.1 can be obtained as

$$y(n-1) = \lambda(n)y_0(n-1) + \overline{\lambda}(n)y_1(n-1) \tag{3.9}$$

Note that the convergence rate and steady state error in pipelined implementation are also adjusted by $\lambda(n)$ given in (1.17), however, auxiliary variable $a(n)$ is updated with combined delayed error signal $e(n-1) = d(n-1) - y(n-1)$, as per

$$a(n+1) = a(n) + \mu_a e(n-1)[y_0(n-1) - y_1(n-1)]\lambda(n)\overline{\lambda}(n) \tag{3.10}$$

As mentioned above, ADF_0 converges quickly while ADF_1 takes time to arrive at the steady state since $\mu_0 > \mu_1$. Therefore, the time difference of arrival in the steady-state would produce a time-window of length ζ. In this interval, the coefficients of ADF_0 are to be transferred to ADF_1 subject to the constraints: $n \pmod{\zeta} = 0$ and $a(n+1) = a_+$, where ζ is a length of time-window [25]. If both the constraints are satisfied, then coefficients can be transferred from ADF_0 to ADF_1, as per $\mathbf{w}^1(n+1) = \mathbf{w}^0(n+1)$. Hence, it can be concluded that the separate coefficient adaptation of ADF_1 is not required. But, this imposes critical choice on ζ since it depends on the time-lag of slow filter with respect to the fast filter. It may be noted that the update of auxiliary variable $a(n)$ is not needed after its value reaches to a_+ since it ensures all the coefficients from ADF_0 are transferred to ADF_1.

Based on the above discussion, it can be stated that the cost of CLMS algorithm can be reduced to a single DA based LMS filter since a common LUT is required as per (3.4), with bit-slices of ADF_0 and ADF_1 coefficients can be used as the address lines to the LUT at different time instants. But, the implementation cost is still dominated by the parameters such as $\lambda(n+1)$ and $a(n+1) = a_+$ which require considerable amount of chip-area.

3.3 Proposed Scheme

The architecture of proposed CLMS ADF with two adaptation delays is shown in Fig. 3.2. It consists of a single LMS ADF, a set of external multiplexers, a convex combination unit (CCU) and an adder tree. LMS ADF further comprises of coefficient update unit (CUU) and filtering unit (FU), where FU includes a shared-parallel LUT (S-PLUT) and a CSFA based SA unit. The locations of adaptation delays in the proposed filter are indicated with 'D_0' and 'D_1' registers. Note that the proposed CCU generates a control (CTRL) signal to the external multiplexers based on the delayed error signals $e_c(n-m)$ with $m \in [1, 2]$.

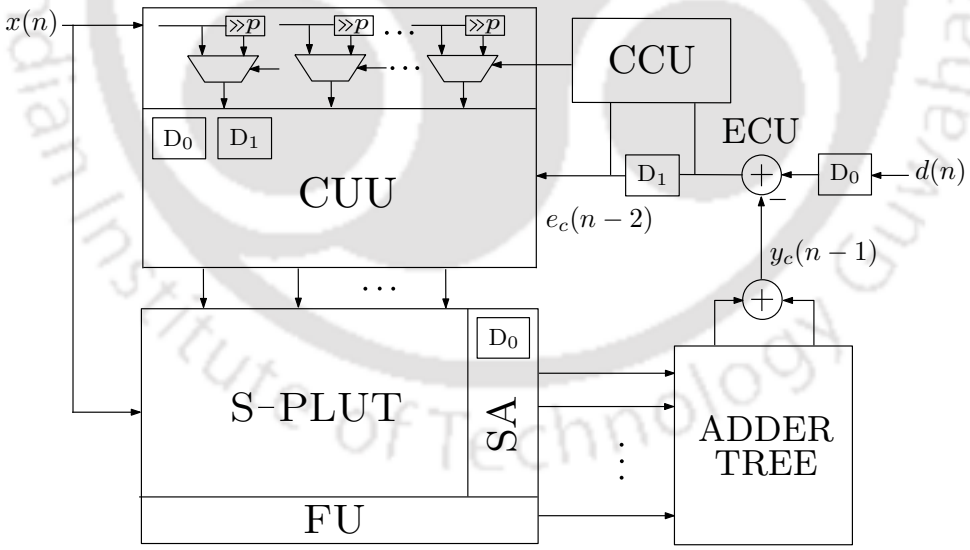


Fig. 3.2: Architecture of pipelined CLMS ADF based on TC-DA with $\mu_1 = 2^{-p}\mu_0$, p is a positive integer.

It can be noted from (3.5) that the output $y_c(n)$ is produced after performing the shift-accumulation on filter partial products $d_{c,k}(n)$ for W successive clock cycles, where its sign must be reversed at W^{th} clock cycle corresponding to MSB of filter coefficients. Therefore, all the bits of LUT output are passed through XOR gates with a sign-control input which is set to '1' only when the MSB slices of filter

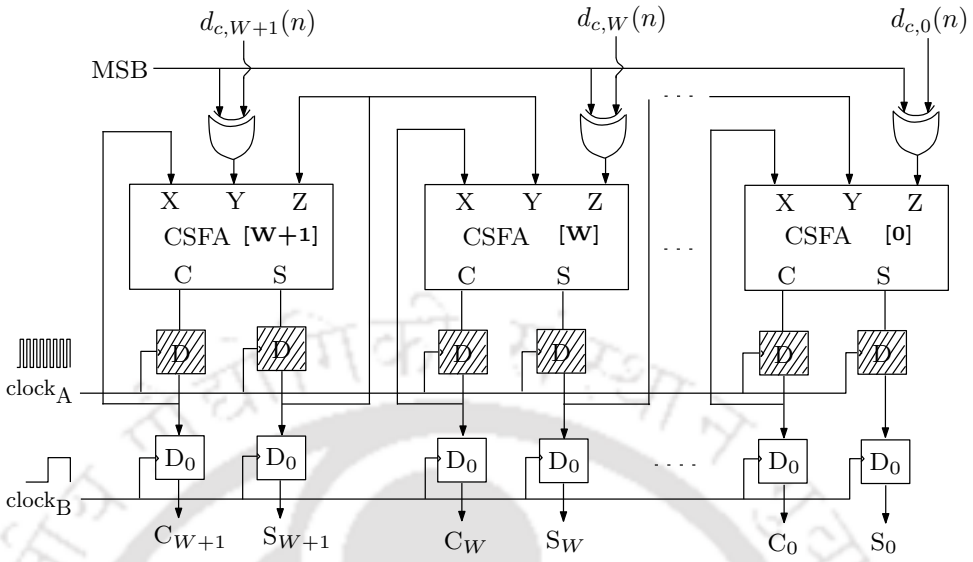


Fig. 3.3: Pipelined CSFA based SA unit for TC-DA.

coefficients appears as address lines to the LUT. Since SA operation involves significant critical path, it is performed using CSFA based accumulator as shown in Fig. 3.3. The sum and carry words obtained after shift-accumulation of W clock cycles are required to be added by a final adder (not shown in the figure), with its input carry to be set to one for the two's complement addition corresponding to the MSB of filter coefficients. Notably, the location of first adaptation delay is chosen after SA unit as indicated in Fig. 3.3. Therefore, the term $d_{c,k}(n)$ can be expressed as

$$d_{c,k}(n) = \sum_{i=0}^{N-1} x(n-i) w_{i,k}^{c}(n) \tag{3.11}$$

where $w_{i,k}^{c}(n) \in [0,1]$ are the bit-slices of coefficients of both the filters, therefore, the term $d_{c,k}(n)$ could take 2^N possible combinations. Clearly, the same number of partial products are required for both the filters with different bit-slices of filter coefficients. These partial products can be stored in LUT such as ROM or RAM. However, it is not necessary to store the partial products in LUT and utilize conventional SA unit. These can be generated using hardware elements in serial or in parallel order based on the OBC or TC of filter coefficients. Although the LUT complexity is optimized by realizing the partial products in serial OBC-form as presented in Chapter 2, there are two problems in such implementation. First, non-OBC terms are produced at the output which does not allow to improve the convergence properties. Second, the critical path grows logarithmically with the order of DA base unit in multiples of the adder delay. These issues can be addressed by implementing

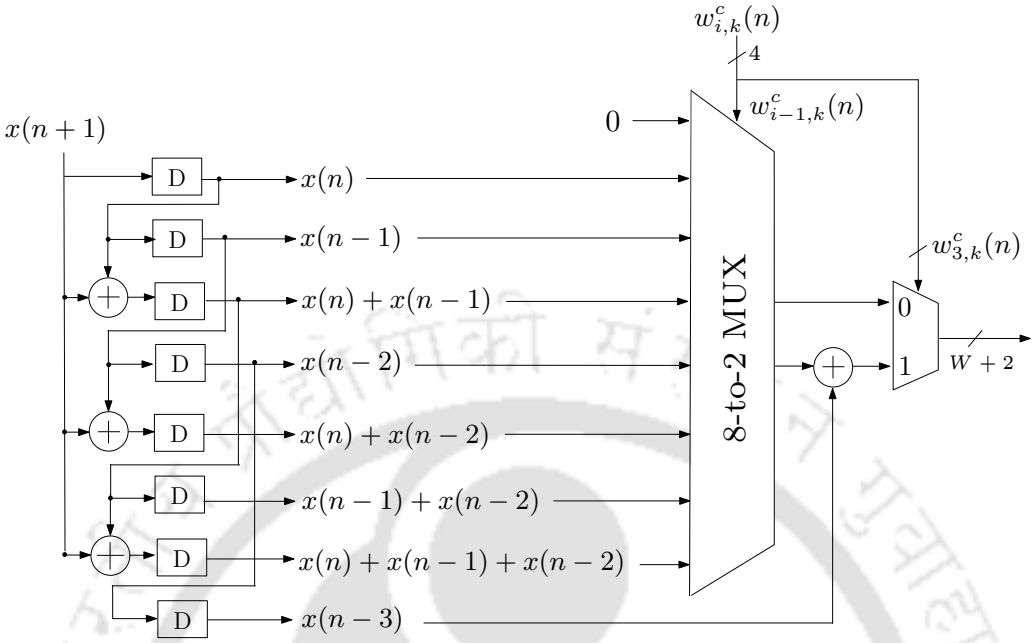


Fig. 3.4: Circuit schematic of 4^{th} order S-PLUT.

the partial products in parallel, as reported in [71]. However, this way of implementing the partial products demand large hardware cost, for example, it can be as high as 2^N for N^{th} order filter. It is noticed that the number of filter partial products to generate can be reduced by expressing (3.11) as

$$d_{c,k}(n) = x(n-N+1)w^c_{N-1,k}(n) + \sum_{i=0}^{N-2} x(n-i)w^c_{i,k}(n) \tag{3.12}$$

It is clear from (3.12) that $d_{c,k}(n)$ could take 2^{N-1} possible combinations while the remaining half can be obtained using an adder and a 2-to-1 multiplexer. Such implementation of LUT is referred as shared-parallel LUT (S-PLUT) where lower half of the partial products are shared to obtain the remaining half. For instance, a 4^{th} order S-PLUT as shown in Fig. 3.4 generates the binary combination corresponding to input vector $\mathbf{x}(n) = [x(n), x(n-1), x(n-2)]$. The remaining half combinations are obtained by adding the input sample $x(n-3)$ and selected through a 2-to-1 multiplexer.

3.3.1 Architecture for Small Order Filter

The proposed architecture of 4^{th} order CLMS ADF is shown in Fig. 3.5. It consists of filtering unit (FU), coefficient update unit (CUU) and error computation unit (ECU). FU comprises of a S-PLUT and a carry-save full-adder based SA unit (CSFA-SA). In CUU, there are four 2-to-1 multiplexers which forms a multiplexer unit (MU); four barrel shifters (BS_i), four parallel-to-serial converters (PS_i), four

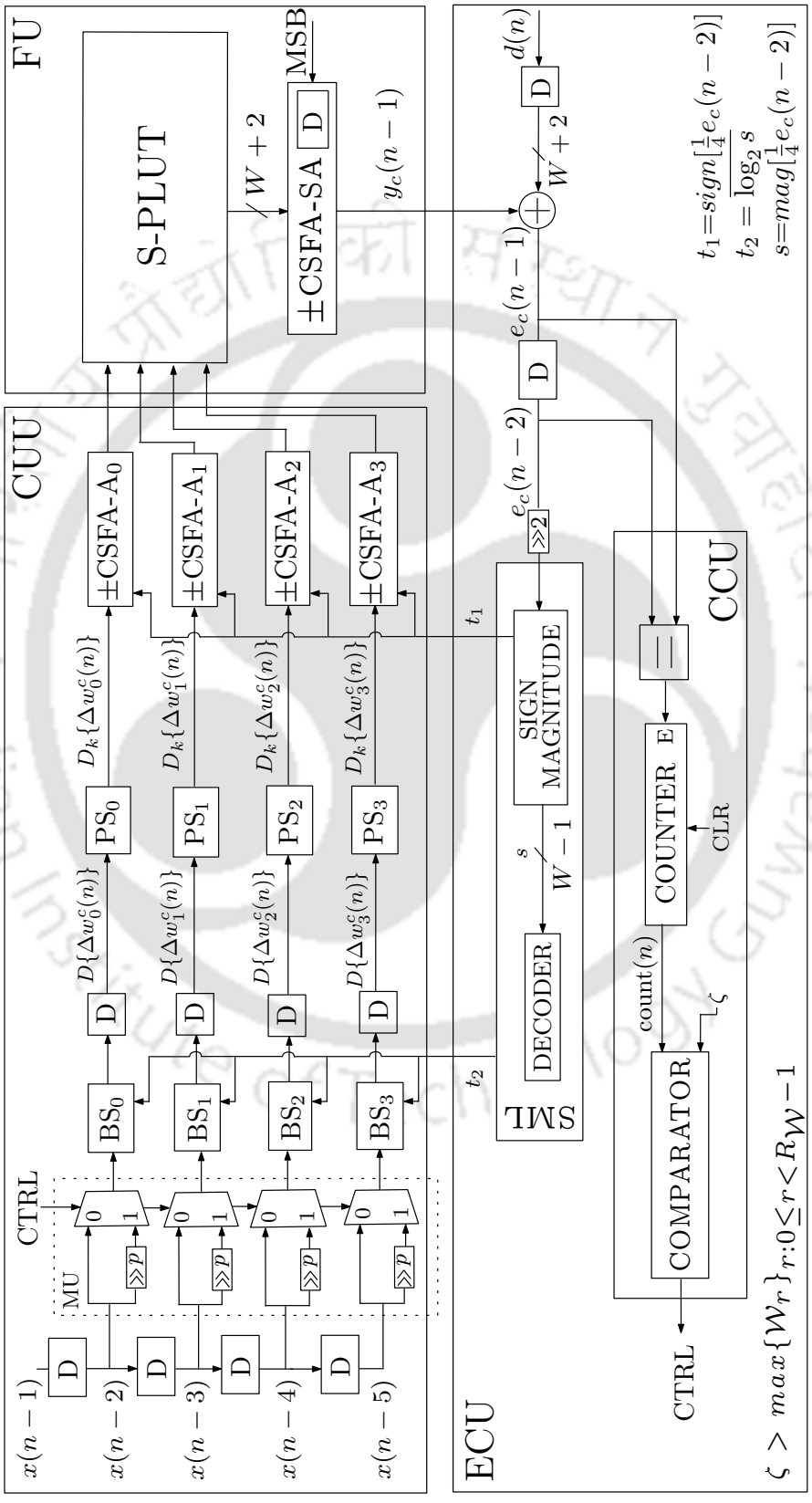


Fig. 3.5: Circuit schematic of 4$^{\text{th}}$ order LMS ADF based on TC-DA.

CSFA bit-accumulators (CSFA-A$_i$) and eight 1-bit D flip-flops with $0 \leq i \leq 3$. ECU further consists of a subtractor, a register (word-level) to obtain $e_c(n-2)$ from $e_c(n-1)$, convex combination unit (CCU), and a sign-magnitude logic (SML). The filter coefficients are transferred from ADF$_0$ to ADF$_1$ through MU using CTRL signal, as shown in Fig. 3.5. All the coefficient-increment terms $\mathbf{\Delta w}^c(n)$ are produced at the output of BS$_i$ units, in accordance with (3.6). These coefficient increment terms are first bit-sliced using PS$_i$ units to produce $D_k\{\mathbf{\Delta w}^c(n)\}$ and then updated at bit-level using CSFA-A$_i$ accumulators with $0 \leq i \leq N-1$, $0 \leq j \leq 1$ and $0 \leq k \leq W-1$. For given coefficient bit-slices, the corresponding S-PLUT content would be selected, and undergoes SA operation through CSFA-SA unit for W number of clock cycles. Notably, the sign of S-PLUT content must be reversed at W^{th} clock cycle, in accordance with (3.3). The CSFA-SA unit is pipelined with first adaptation delay which results the output to produce after one clock cycle i.e., $y_c(n-1)$. Subsequently, it is used to compute error $e_c(n-1)$ by subtracting from the desired signal $d(n-1)$. The second adaptation delay is placed after $e_c(n-1)$ computation so as to obtain $e_c(n-2)$ from $e_c(n-1)$, as shown in Fig. 3.5. It is then multiplied with the step-size $\mu_c = 2^{-cp}/N$ to obtain scaled error $\mu_c e_c(n-2)$. All coefficient increment terms $\mathbf{\Delta w}^c(n)$ are computed through the control shifts on input samples using BS$_i$ for $0 \leq i \leq N-1$. The controlled shifts are obtained by considering the MSB of $\mu_c e_c(n-2)$ signal and such approximation does not affect the convergence of LMS ADF [67]. SML unit separates the sign (*sign*) and magnitude (*mag*) components of $\mu_c e_c(n-2)$ into two signals t_1 and s respectively. The control bits for the selection lines of BS$_i$ units are generated from the decoder using s and the sign of each bit-level accumulator in CUU is adjusted by t_1. From mathematical analysis, the expressions for t_1 and t_2 can be found as $t_1 = sign[\mu_c e_c(n-2)]$ and $t_2 = \overline{\log_2 s}$ respectively, where $s = mag[\mu_c e_c(n-2)]$, and overline represents one's complement operator. This can be understood by considering 4^{th} order proposed filter with wordlength of coefficients as 8-bit and step-size $\mu_0 = 1/4$ which implies s and t_1 to be 7-bit and 1-bit respectively. As stated, it is sufficient to consider MSB of $\frac{1}{4}e_c(n-2)$ which takes 128, 64, 32, 16, 8, 4 and 2 as possible values, and the corresponding control bits t_2 for BS$_i$ would be 0, 1, 2, 3, 4, 5 and 6.

The implementation of convex combination unit (CCU) for coefficient transfer indicated by (1.17) and (3.10) is difficult due to high complexity constraints. This is overcome by the proposed efficient criterion through which the coefficients from one filter to other can be transferred. Now, consider the operation of proposed CCU as shown in Fig. 3.5. It is based on the maximum correlation between

time-adjacent errors $e_c(n-1)$ and $e_c(n-2)$. The correlation between $e_c(n-1)$ and $e_c(n-2)$ signals is found to be maximum when ADF_0 reaches to steady state, i.e., both error signals have same value over a longer period of time. However, there exists some undesired correlation between $e_c(n-1)$ and $e_c(n-2)$ signals during the initial adaptation period of ADF_0. In order to transfer the coefficients from ADF_0 to ADF_1, the removal of such undesired correlations are necessary. Simulation study suggests that the duration of time windows of undesired correlations are always less as compared to duration of time window of desired correlation (in steady state). The internal schematic of proposed CCU includes an equality checker, a counter with enable (E) and clear (CLR) inputs and a comparator, as shown in Fig. 3.5. An equality checker generates an enable signal for the counter whenever $e_c(n-1) = e_c(n-2)$ exists. The counter begins its operation and will keep counting until $e_c(n-1) \neq e_c(n-2)$. When this condition arrives, the last count will be on hold in count(n) and then compared with a pre-defined time window ζ. From above discussion, ζ has to be greater than the maximum of all possible time window lengths before ADF_0 reaches to steady state. Mathematically, $\zeta > max\{\mathcal{W}_r\}_{r:0\leq r<R_{\mathcal{W}}-1}$, where $\mathcal{W}_r$ is the time length of r^{th} window in the set of $R_{\mathcal{W}}$ windows. This will ensure ADF_0 to be in steady-state. Simulations are performed for different values of ζ in different situations and found that the proposed method is not very sensitive to the selection of ζ. It is noticed that the values of ζ do not affect the performance of filter, but if its value is larger than the required value then the time lag between slow and fast filter increases. When ADF_0 reaches the steady state, the counter begins its operation since $e_c(n-1) = e_c(n-2)$, and would count upto its maximum value. Note that the condition $e_c(n-1) \neq e_c(n-2)$ would never arrive. After certain number of iterations, the count(n) input of comparator becomes greater than ζ, and sets CTRL signal to logic '1'. Thereby, transferring the coefficients of ADF_0 to ADF_1 via MU by pre-shifting the input samples. It should be noted that the counter has to be cleared for every new count using CLR signal. The size of counter is decided by ζ, whose value is given as $\lceil \log_2 \zeta \rceil$, where $\lceil \bullet \rceil$ is the least-integer function.

3.3.2 Architecture for Large Order Filter

It is noticed from Fig. 3.4 that the complexity of S-PLUT is reduced to almost half, but is still high for larger order filters. In such a case, the complexity of S-PLUT is reduced by splitting the filter into smaller order filters whose outputs are combined using an adder tree. For instance, consider an N^{th} order ADF which is to be splitted into M number of smaller ADFs of L^{th} order such that

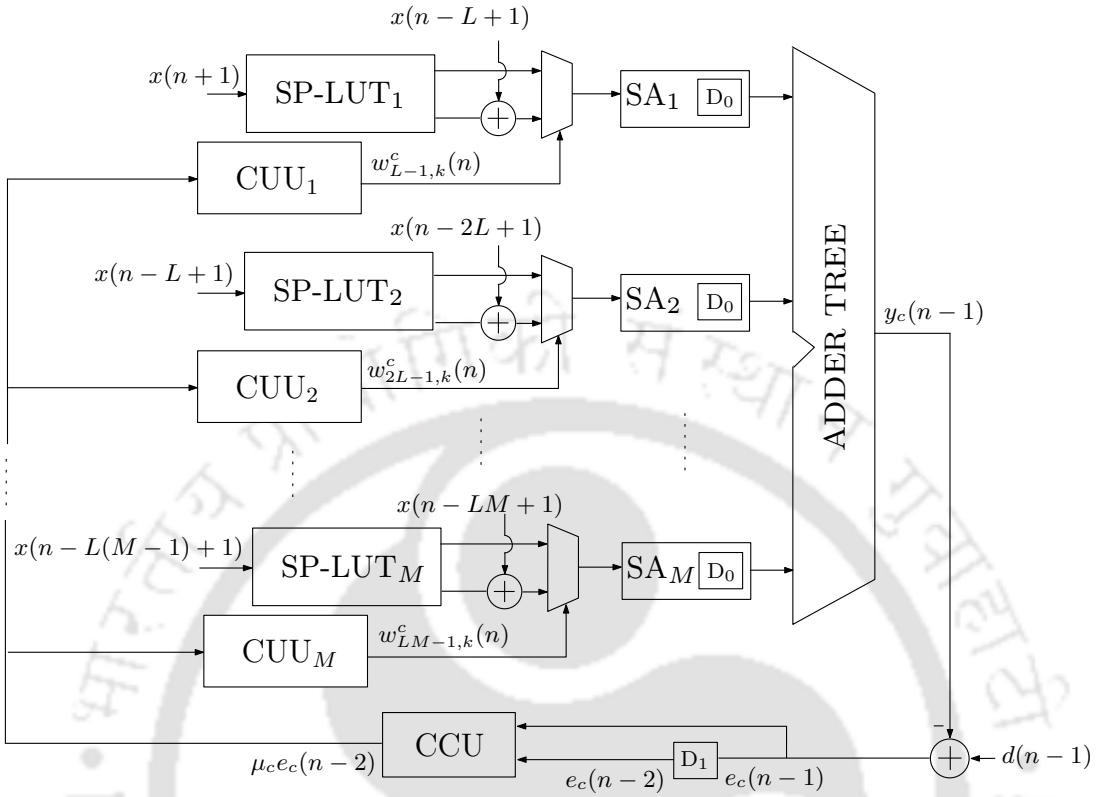


Fig. 3.6: Circuit schematic of 16$^{\text{th}}$ order LMS ADF based on TC-DA.

$N = L \times M$ (assuming L to be composite). Therefore, the term $d_{c,k}(n)$ can be expressed as

$$d_{c,k}(n) = \sum_{s=0}^{M-1} [x(n-n_s) w^c_{n_s,k}(n) + \sum_{r=n_s}^{n_s+L-2} x(n-r+L-1) w^c_{r-L+1,k}(n)] \tag{3.13}$$

where $x(n_s) = x(n-n_s)$ and $n_s = (s+1)L-1$ for $s \in [0, M-1]$. Clearly, the size of each S-PLUT is reduced to 2^{L-1} which is substantially lower as compared to the design in [71]. The proposed DA based architecture for filter order $N = 16$ and DA base unit $L = 4$ is shown in Fig. 3.6. Note that each L^{th} order DA base unit have a coefficient update unit.

Assuming the step-size $\mu = 1/N$, the four LSBs of $e_c(n-1)$ can be truncated for filter order $N = 16$ so as to make the wordlength of SML unit W-bit. It must be noted that this truncation does not affect the performance of the ADF since the design needs the location of the most significant one of $\mu_c e_c(n-2)$ as others are don't care case.

3.3.3 Determination of Pre-defined Time Window

To determine the pre-defined time window (ζ), a general system identification problem is considered for an unknown system with coefficient vector $\mathbf{w}^u(n) = [w_0^u(n), w_1^u(n), ..., w_{L-1}^u(n)]$, as shown in Fig. 1.2. Following the procedure presented in [81], the assumptions made are listed below:

- $x(n)$ and $d(n)$ are zero-mean, wide-sense stationary, and mutually-independent processes.
- $x(n)$ is a white process.
- The difference between the coefficients vector $\mathbf{w}^u(n) - \mathbf{w}(n)$ is independent of $x(n)$ and $d(n)$.

Both the proposed filter and unknown system are excited with the input vector $\mathbf{x}(n)$, and the error obtained after the computation of output can be given as

$$e(n) = d(n) + \epsilon(n) \tag{3.14}$$

where $\epsilon(n) = \mathbf{x}^T(n)\mathbf{w}^u(n) - \mathbf{x}^T(n)\mathbf{w}(n)$ is the residual error. Let us denote the mean squared value of $x(n), d(n)$ and $\epsilon(n)$ as $\sigma_x^2 = \mathbb{E}[x^2(n)]$, $\sigma_d^2 = \mathbb{E}[d^2(n)]$ and $\sigma_\epsilon^2 = \mathbb{E}[\epsilon^2(n)]$ respectively. Many practical system are analyzed with their mean square error (MSE) curves $\xi(n)$ as a function of n, according to

$$\xi(n) = \frac{\sigma_\epsilon(n)}{\sigma_d} \tag{3.15}$$

It can be noted from (3.15) that $\xi(n)$ depends on the choice of adaptation algorithm. If LMS algorithm is chosen, then analytical expression of $\xi(n)$ in accordance with [82] can be given as

$$\xi^2(n+1) = \xi^2(n)(1 - 2\mu\sigma_x^2 + \mu^2 N\sigma_x^4) + \mu^2 N\sigma_x^4 \tag{3.16}$$

In compact form, it can be re-written as

$$\xi^2(n) = \rho^n[\xi^2(0) - \xi^2(\infty)] + \xi^2(\infty) \tag{3.17}$$

where

$$\rho \triangleq 1 - 2\mu\sigma_x^2 + \mu^2 N\sigma_x^4 \tag{3.18}$$

$$\xi^2(\infty) \triangleq \frac{\mu^2 N\sigma_x^4}{1-\rho} = \frac{\mu N\sigma_x^2}{2 - \mu N\sigma_x^2} \tag{3.19}$$

Note that $\xi^2(0)$ and $\xi^2(\infty)$ are the initial and mean steady-state errors respectively. By combining (3.18) and (3.19), ρ can also be expressed as

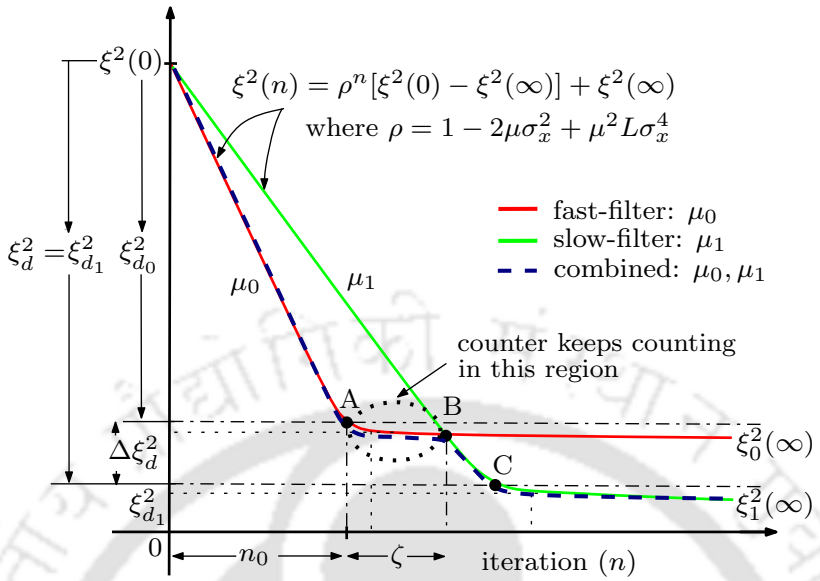


Fig. 3.7: Assumed MSE curves for the presented design to determine ζ.

$$\rho = 1 - \frac{4\xi^2(\infty)}{(1 + \xi^2(\infty)^2 N} \tag{3.20}$$

Usually, it is difficult to define the exact number of iterations required by the LMS ADF to converge at $\xi^2(\infty)$. This is because the error power would fluctuate around $\xi^2(\infty)$ with some variance. So, for many practical applications, the convergence (threshold-point) of LMS ADF is assumed to be $3-6$ dB higher than the $\xi^2(\infty)$ [82]. Hence, it is sufficient to consider the convergence of LMS ADF about threshold-point of desired value ξ_d^2. To derive the pre-defined time window ζ and extra mean steady-state error performance $\Delta\xi_d^2$, three points: 'A', 'B' and 'C' are considered, as shown in Fig. 3.7. The point 'A' is the threshold point of LMS ADF$_0$, point 'B' is the crossover point of LMS ADF$_0$ and LMS ADF$_1$ and point 'C' is the 'A' is the threshold point of LMS ADF$_1$. The iterations between point 'A' to point 'B' gives the estimate of pre-defined time window ζ and extra steady-state error performance $\Delta\xi_d^2$ can be obtained by taking difference of error at point 'A' and point 'C'. For clarity, let us assume the threshold-points of ADF$_0$ and ADF$_1$ as $\xi_{d_0}^2$ and $\xi_{d_1}^2$ respectively, where $\xi_{d_1}^2 = \xi_d^2$ is the desired steady-state error performance. At point 'A', one can apply (3.16) for ADF$_0$ to obtain $\xi_{d_0}^2$, accordingly as

$$\xi_{d_0}^2 = \rho_0^{n_0}[\xi^2(0) - \xi_0^2(\infty)] + \xi_0^2(\infty) \tag{3.21}$$

where n_0 denotes time at which ADF_0 enters into the steady-state, and $\xi_0^2(\infty)$ denotes its mean steady-state error. From (3.21), n_0 can be expressed as

$$n_0 = \frac{1}{\log\rho_0}\log\frac{\xi_{d_0}^2 - \xi_0^2(\infty)}{\xi^2(0) - \xi_0^2(\infty)} \tag{3.22}$$

At point 'B', it is assumed that ADF_0 is in the steady-state and value of ADF_1 becomes equal to its mean steady-state error due to crossover, that is, $\xi_1^2(n_0+\zeta) = \xi_0^2(\infty)$. Mathematically,

$$n_0 + \zeta = \frac{1}{\log\rho_1}\log\frac{\xi_0^2(\infty) - \xi_1^2(\infty)}{\xi^2(0) - \xi_1^2(\infty)} \tag{3.23}$$

Subtracting (3.22) from (3.23) would lead to the expression for pre-defined time window ζ as

$$\zeta = \frac{1}{\log\rho_1}\log\frac{\xi_0^2(\infty) - \xi_1^2(\infty)}{\xi^2(0) - \xi_1^2(\infty)} - \frac{1}{\log\rho_0}\log\frac{\xi_{d_0}^2 - \xi_0^2(\infty)}{\xi^2(0) - \xi_0^2(\infty)} \tag{3.24}$$

It is fair to assume that $\log(1-x) \approx -0.434x$ since both $\xi_0^2(\infty)$ and $\xi_1^2(\infty) << 1$ which simplifies (3.24) to

$$\zeta = \frac{-0.57N}{\xi_1^2(\infty)}\log\frac{\xi_0^2(\infty) - \xi_1^2(\infty)}{\xi^2(0) - \xi_1^2(\infty)} - \frac{-0.57N}{\xi_0^2(\infty)}\log\frac{\xi_{d_0}^2 - \xi_0^2(\infty)}{\xi^2(0) - \xi_0^2(\infty)} \tag{3.25}$$

Although analysis seems to be simpler, the expression of ζ obtained in (3.25) is very complex. Based on the derivation [81], the optimum step-size μ_c^* for ADF_0 and ADF_1 in terms of coefficients wordlength W-bit and filter order N can be given as

$$\mu_c^* = \frac{2^{-(cp+W)}}{2}\sqrt{\frac{N}{3\xi^{*2}_c(\infty)\text{tr}\boldsymbol{R}}} \tag{3.26}$$

or

$$\xi^{*2}_c(\infty) = \frac{2^{-2(cp+W)}N}{12\mu_c^*\text{tr}\boldsymbol{R}} = \frac{2^{-2(cp+W)}}{12\mu_c^*\sigma_x^2} \tag{3.27}$$

where $\text{tr}\boldsymbol{R} = N\sigma_x^2$ denotes the trace of matrix $\boldsymbol{R} = \mathbb{E}[\mathbf{x}(n)\mathbf{x}^T(n)]$. Therefore, one can find optimum values of steady-state errors $\xi^{*2}_c(\infty)$ for both the filters. By substituting the value of $\xi^{*2}_c(\infty)$ in (3.20), we get

$$\rho_c^* = 1 - \frac{4\xi^{*2}_c(\infty))}{(1+\xi^{*2}_c(\infty))^2 N} \tag{3.28}$$

Following the steps as obtained for (3.24), one can find the optimum value of pre-defined time window ζ_{opt} with dependence on coefficient wordlength W and filter order N, according to

$$\zeta_{opt} = \frac{1}{\log\rho_1^*}\log\frac{\xi^{*2}_0(\infty) - \xi^{*2}_1(\infty)}{\xi^2(0) - \xi^{*2}_1(\infty)} - \frac{1}{\log\rho_0^*}\log\frac{\xi^{*2}_{d_0} - \xi^{*2}_0(\infty)}{\xi^2(0) - \xi^{*2}_0(\infty)} \tag{3.29}$$

It is clear from (3.29) that $\xi^2(\infty)$ does not depend on the input signal, therefore, ζ can be estimated when $\mu_0, \mu_1, N, \sigma_x^2, \xi(0)$ and ξ_{d_0} are specified. For clarity, different plots of ζ for 32^{nd} order filter with respect to ξ_{d_0} and W are shown in Fig. 3.8. Clearly, about 150 iterations are needed to switch from ADF_0 to ADF_1 when the initial error $\xi(0)$ is 0 dB. Further, higher the value of desired error of ADF_0, the number of iterations are more required to switch from ADF_0 to ADF_1, as indicated in Fig. 3.8. Assuming ${\xi^*}_{d_1} = \sqrt{2}\xi_1^*(\infty)$, the excess steady-state error can be expressed as ${\Delta\xi^*}_d = \sqrt{2}\Delta\xi^*(\infty) = \sqrt{2}(\xi_1^*(\infty) - \xi_0^*(\infty))$. The effect of increasing coefficients wordlength on ζ is quite obvious, as shown in Fig. 3.8. This is because the number of iterations would increase to obtain lower steady-state error. Algorithm 2 explains the operation of proposed CLMS ADF.

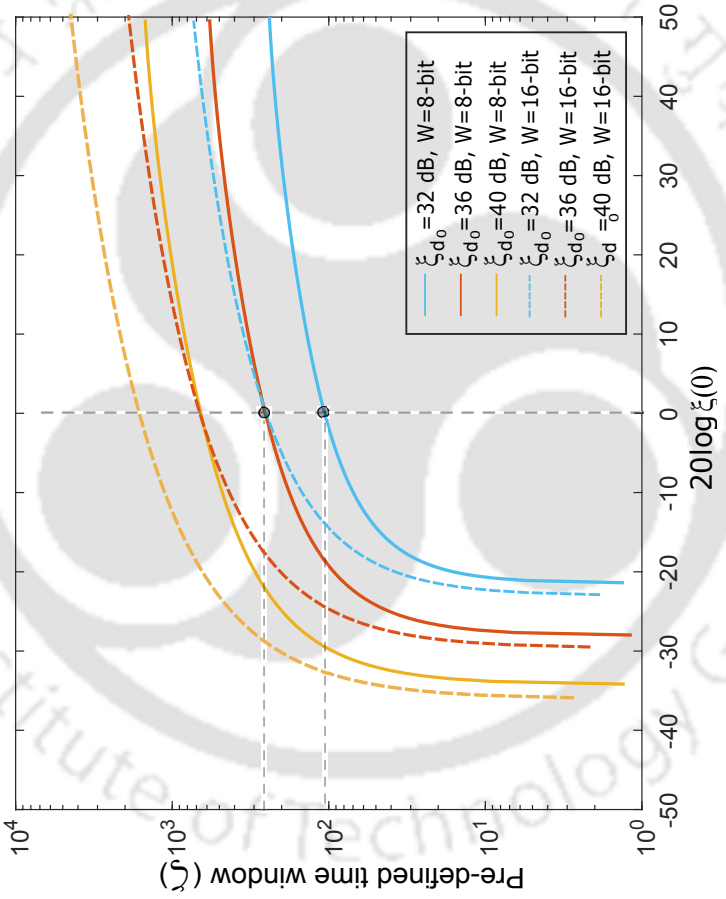


Fig. 3.8: Variation of ζ with respect to initial error for 32^{nd} order filter and 8, 16-bit wordlengths of filter coefficients.

Algorithm 2 Algorithm explaining the operation of the proposed CLMS ADF

Initialize:

$m = 2; k = 0, 1, ..., W-1;\ y_c(-1), d(-1), e_c(-1), e_c(-2), \text{CLR}, \text{CTRL} = 0$

loop

$y(n) = \sum_{k=0}^{W-1} d_{c,k} 2^{-k}$

  for $k = 0$ to $W-1$ **do**

    if $k == W-1$ **then**

      MSB=1, $y_c(n) = y_c(n) - d_{c,k}2^{-k}$

    else

      MSB=0, $y_c(n) = y_c(n) + d_{c,k}2^{-k}$

    end if

  end for

  $e_c(n-1) = d(n-1) - y_c(n-1)$

  $e_c(n-2) \leftarrow e_c(n-1)$

  for $k = 0$ to $W-1$ **do**

    $\mathbf{w}_{k+1}^0(n) = \mathbf{w}_k^0(n) + D_k\{t_1 t_2 \mathbf{x}(n-2)\}$

  end for

  if $e_c(n-1) = e_c(n-2)$ **then**

    $\text{count}(n+1) \leftarrow \text{count}(n) + 1$

    if $\text{count}(n) == 2^{\lceil \log_2 \zeta \rceil} - 1$ **then**

      $\text{CTRL} \leftarrow 1$

      for $k = 0$ to $W-1$ **do**

        $\mathbf{w}_{k+1}^1(n) = \mathbf{w}_k^1(n) + D_k\{2^{-p} t_1 t_2 \mathbf{x}(n-2)\}$

      end for

    else

      $\text{CLR} \leftarrow 1$

    end if

  end if

  $n \leftarrow n+1$

end loop

3.4 Performance Comparison

In this section, the performance of the proposed and existing designs are compared in terms of computational complexities, convergence, area, power and throughput. For clarity, the existing designs are referred as DA_0-ADF [67], $DA_{1(a),1(b)}$-ADF (two designs) [68], DA_2-ADF [69], DA_3-ADF [70], DA_4-ADF [71].

3.4.1 Computational Complexities

As discussed in Section 3.2, several approximations have been made into basic-CLMS ADF [11] in accordance with [25] and [26]. The conventional CLMS ADF has two LMS units which require a total of $(4N + 2)$-MAC units for the implementation. In addition, it needs six multipliers in the computation of combined output and update of auxiliary variable. Later, this has been reduced to three multipliers in [26] by sign-adaptation scheme. While in case of proposed algorithm, there is no need of auxiliary variable and mixing parameter, but it is still capable of providing instantaneous transfer of filter coefficients. This is because it estimates the duration of correlation between time-adjacent errors $e_c(n-m)$ for $m \in [1, 2]$ in every iteration. And, as soon as it approaches the pre-defined time-window ζ, the switching of coefficients from fast filter to slow filter is possible. Moreover, the multiplierless feature of the proposed architecture makes it more amenable for hardware realization than the conventional CLMS ADF.

The hardware complexities of the proposed and existing DA based designs with $N = L \times M$ are listed in Table 3.1. The proposed design consists of M-FUs, M-CUUs and a ECU. Each FU further consists of a S-PLUT with 2^{L-2} adders, 2^{L-1} registers and $2^{L-1} + 1$ multiplexers, and a pipelined CSFA-SA unit. Each CUU consists of L barrel shifters, L parallel-to-serial converters and L CSFA based bit-level accumulators. ECU consists of a subtractor, two delay registers, SML and CCU units. It can be noted from Table 3.1 that the complexity of DA_0-ADF, $DA_{1(a),1(b)}$-ADF and DA_2-ADF designs are limited by the size of physical LUTs. While the complexity of DA_3-ADF and DA_4-ADF designs are dominated by the number of adders, multiplexers and registers. By sharing the partial products, as in the case of proposed filter, the number of hardware units for the implementation of proposed filter is almost reduced to half. But, this comes at the cost of slight increase in the critical path as compared to DA_4-ADF. For example, the proposed filter with 32^{nd} order and 4^{th} base order involves 46.42% less adders, 36.69% less registers, 18.75% less multiplexers and same number of

barrel-shifters with slight increase in the critical path delay over DA_4-ADF. Compared with existing design [71], the savings in adders, multiplexers and registers for the proposed design are found to be significant, especially when the order of DA base unit is large.

3.4.2 Convergence Performance

Two different experiments are performed to determine the accurate convergence behaviour of the proposed and existing designs. In first experiment, the effect of hardware approximations on convergence characteristics with respect to basic CLMS ADF is studied. In sequel, the convergence of proposed filter is compared with floating-point CLMS ADF without any hardware approximations. In second experiment, the improvements in convergence performance of the proposed design over the existing DA based design are examined.

The fixed-point realization of the proposed filter with wordlength of coefficients and inputs as 20-bit, and the floating-point realization of convex combination and its component filters with wordlength of coefficients and inputs as 64-bit are simulated for a system identification problem. All the systems are excited by zero mean white Gaussian signal of unit variance with a random white Gaussian noise of zero mean and 0.01 variance is added to desired signal which are operated at 20 dB signal-to-noise ratio (SNR) with the simulation parameters $p = 2$, $m = 2$, $\mu_0 = 1/N$, $\mu_1 = 2^{-p}\mu_0$, $\zeta \approx 1780$ and $N = 128$ for 200 independent runs. The mean squared error (MSE) learning curves of different designs in fixed-point and floating-point realization are depicted in Fig. 3.9(a). As expected, the convergence of the proposed design with respect to conventional CLMS ADF [11] and their component filters is found to be satisfactory. It is interesting to note from Fig. 3.9(a) that the convergence performance of proposed design in fixed-point realization is almost similar to the conventional CLMS ADF [11]. This can be understood by knowing the fact that the proposed design does not utilize the approximations made in [25] and [26], rather depends on the correlation of adjacent errors $e_c(n-m)$ for $m \in [1, 2]$. In addition, it is stable and show similar numerical properties with respect to floating-point CLMS counterpart. On the other hand, it exhibits slightly higher perturbations in the steady-state as the implementation complexity of floating-point CLMS ADF is extremely higher than the proposed filter.

The MSE learning curves of different DA based LMS ADFs with $\zeta = 332$ and $N = 32$ for the same system identification problem are illustrated in Fig. 3.9(b). Clearly, the convergence rate of DA_3-ADF and DA_4-ADF are slower and involve several misadjustments as compared to DA_0-ADF, $DA_{1(a),1(b)}$-ADF and DA_2-ADF. This is mainly due to the pipelined nature of filters which create

instability in the coefficients, especially when step-size $\mu_0 = 1/N$. However, if one employs smaller step-size $\mu_1 = 2^{-p}/N$ then both misadjustments and steady-state error would be reduced, but it makes the convergence rate slower. On the other hand, the proposed architecture is although pipelined, the convergence rate and minimum steady-state error are found to be better, since it is based on the two different step-sizes. For example, the proposed filter has employed step-size $\mu_0 = 1/N$ in the initial adaptation period and step-size $\mu_1 = 2^{-p}/N$ in the steady-state. According to the simulation results, the improvements in steady-state error performance for the proposed filter could be as high as 6 dB when compared with DA_3-ADF and DA_4-ADF. In contrast, DA_0-ADF, $DA_{1(a),1(b)}$-ADF and DA_2-ADF are based on conventional LMS algorithm, therefore, they have better convergence rate as compared to DA_3-ADF and DA_4-ADF.

3.4.3 ASIC Synthesis

In order to estimate area, power and throughput of different DA based designs, all are coded in Verilog for 32^{nd} and 64^{th} orders filter with wordlength of coefficients as 8-bit and 20-bit separately. ASIC synthesis is performed by Cadence RTL Compiler using TSMC 65 nm CMOS Library, and corresponding results are listed in Table 3.2. Additionally, the area-delay-product (ADP) and the energy-per-sample (EPS) are estimated to compare the actual performance of different designs. As expected, the proposed design occupies relatively lesser area and consumes lesser power at the cost of slightly lower throughput as compared to DA_4-ADF. However, the throughput of proposed filter is still higher as compared to non-pipelined DA_0-ADF, $DA_{1(a),1(b)}$-ADF and DA_2-ADF designs irrespective of filter order and size of DA base unit. Notably, the savings in area and power for the proposed design are significant when order of filter and wordlength of coefficients become large. For example, a 64^{th} order proposed filter with 4^{th} order DA base-units occupies nearly 37.10% less area, consumes 24.79% less power, provides 20.35% less ADP and 4.76% less EPS as compared to DA_4-ADF. The savings are even higher when the proposed filter is implemented with large DA base unit. The proposed design for 64^{th} order filter with 4^{th} order DA base units and 20-bit wordlength of coefficients occupies 29.03% less area, consumes 58.73% less power, provides 89.91% less ADP and 94.13% less EPS as compared to DA_2-ADF design.

3.5 Conclusion

In this Chapter, a low-complexity and improved convergent DA based pipelined CLMS ADF has been presented. Although the complexity of pipelined LMS ADF is optimized using OBC in Chapter 2, it produces non-OBC terms at the output which does not allow to improve its convergence performance. The improvements in convergence properties for the proposed filter have been obtained by implementing the convex combination using TC-form of coefficients. In order to reduce the hardware complexity of CLMS ADF, it has been implemented with single LMS ADF. Further reduction in hardware complexity has been achieved by sharing the filter partial products and using the bit-level coefficient-update accumulators. A novel coefficient transfer has been proposed by comparing the time duration of correlation between the adjacent errors with a pre-defined time-window. An analytical expression has been derived for pre-defined time-window and simulated for different filter parameters. Compared with the best existing design, it is found that hardware savings for the proposed design are significant, and exhibits slightly inferior convergence performance over the conventional CLMS ADF [11].

Table 3.1: Comparison of Computational Complexities between DA based designs for N^{th} Order Filter, L^{th} Order Base Unit and W-bit Wordlength of Filter Coefficients

Design	**Type**	**Adders**	**Registers**	**Shifters**	**LUT/MUX**	**Throughput**
DA_0-ADF [67]	Non-Pip/Par†	$3M+1$	$M+N+2$	M	$2^{L+1}M$-LUT	$1/[k_0(T_{ACC}+(L-1)T_M+T_A)]$
$DA_{1(a)}$-ADF [68]	Non-Pip/Par†	$2M+N+1$	$2N+3L+1$	LM	$2^{L}M$-LUT	$1/[k_1(T_{ACC}+T_A)$
$DA_{1(b)}$-ADF [68]	Non-Pip/Par*†	$3M+N+3$	$2M+3N+1$	LM+NM'-XOR	$2^{L-1}M$-LUT	$1/[k_1(T_{ACC}+T_M+T_A)$
DA_2-ADF [69]	Non-Pip/Par*‡	$4M+3$	$M+3N+2$	M+$2NM'$-XOR	$2^{L}M$-LUT	$1/[k_2(T_{ACC}+(L+1)T_M+2T_A)]$
DA_3-ADF [70]	Pip*/Par†	$(3.2^{L-1}+1)M$	$3M(1+2^{L-1})+N+2$	$\sim 2^{L}M$	$2^{L}M$-MUX	$1/[W(LT_M+T_{FA}+T_D)]$
DA_4-ADF [71]	Pip*/Par†	$M(2+2^{L-1})+N$	$(3+2^{L})M+2N+4$	LM	$2^{L}M$-MUX	$1/[W(LT_M+T_{FA}+T_X+T_D)]$
Proposed	Pip*/Par†¶	$M(3+2^{L-2})+N'$	$(4+2^{L-1})M+N$ $+2N'+4$	LM $(1+L)M$-MUX	$(2^{L-1})M$	$1/[W(LT_M+T_A+T_{FA}+T_X$ $+T_d)]$

Pip$*$: Pipelined architecture with two adaptation delays, Par: Parallel LUT, $\dagger$: concurrent SA and CUU operations, $\ddagger$: concurrent SA, CUU and LUT operations, $\star$: OBC based schemes, ¶ : partial-products sharing. The designs in [67–69] have controller, while the proposed design involves a CCU which consists of an equality-checker, a counter and a comparator; $N=LM$, L : base order, M : number of base units; B : denotes the wordlength of both input samples and coefficients; $N'=N/W$; $L'=L-1+W$; $k_0=2^{L}+max(W,2^{L-1})+\log_2 M$, $k_1=2^{L-1}+W+\log_2 M+1$, $k_2=2^{L/2}+max(W,2^{L/2-1})+\log_2 M+1$; $T_{ACC}, T_M, T_A, T_{FA}$ and T_D are delays of a look-up table, a 2-to-1 multiplexer, an adder, a full-adder and a register respectively.

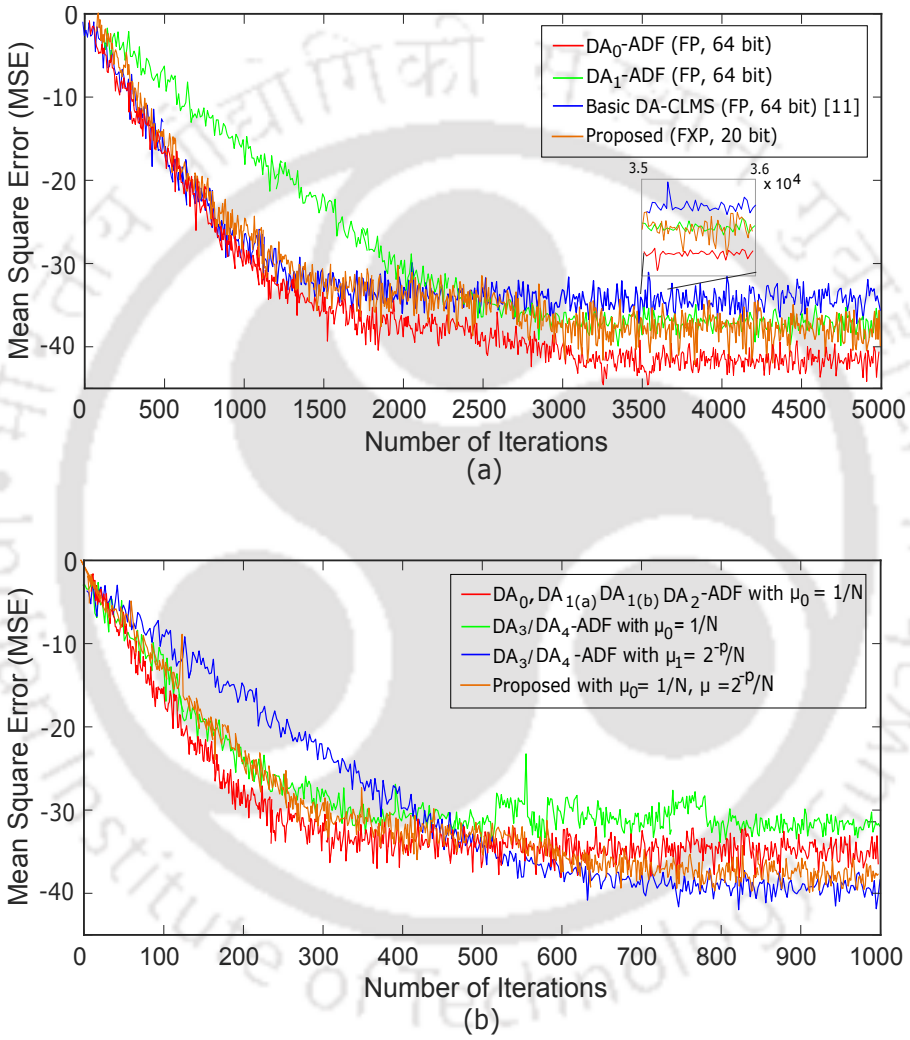


Fig. 3.9: (a) Finite-precision simulation of the presented DA based design, slow-ADF, fast-ADF and basic-CLMS ADF [11] for 128^{th} order, FP: floating-point and FXP: fixed-point (b) MSE learning curves of DA_0 [67], DA_1 [68], DA_2 [69], DA_3 [70], DA_4 [71] ADFs and the proposed design for 32^{nd} order filter with $p = 2$, $m = 2$, $\mu_0 = 1/N$, $\mu_1 = 2^{-p}/N$ and $\zeta = 332$.

Table 3.2: Performance Comparison of Different DA Based Designs with ASIC Synthesis using TSMC 65 nm CMOS Library for $32^{nd}, 64^{th}$ Orders Filter and 8, 20-bits Wordlength of Filter Coefficients

Design	**Filter Order** (N)	**Wordlength** (W)	**Area** (μm^2)	**Power** (mW)	**MSP** (ns)	**DAT** (ns)	**Throughput** (per μsec)	**ADP** ($\mu m^2 \times$ ns)	**EPS** (mW $\times$ ns)
DA_0-ADF [67]	32	8	49159	46.58	21.47	0.81	46.57	1055443.73	1000.07
	64	8	97284	90.25	23.87	0.89	41.89	2322169.08	2154.26
	32	20	114704	108.68	53.67	0.96	18.63	6156163.68	5832.85
	64	20	226996	210.58	57.82	1.05	17.29	13124908.72	12175.73
$DA_{1(a)}$-ADF [68]	32	8	40235	41.05	14.32	0.73	69.83	576165.2	587.83
	64	8	79867	80.13	15.63	0.78	63.97	1248321.21	1212.37
	32	20	93881	95.78	35.80	0.85	27.93	3360939.8	3428.92
	64	20	186356	186.97	37.28	0.93	26.82	4998067.92	6970.24
$DA_{1(b)}$-ADF [68]	32	8	33770	33.92	9.30	0.58	107.52	314061	315.45
	64	8	66863	65.94	10.76	0.62	92.93	719445.88	709.51
	32	20	78796	79.14	23.25	0.68	43.01	1832007.00	1840.00
	64	20	156013	153.86	25.38	0.76	39.40	3959609.94	3904.96
DA_2-ADF [69]	32	8	32252	32.48	13.05	0.69	76.62	420888.6	423.86
	64	8	63857	63.43	14.31	0.74	69.88	913793.67	907.68
	32	20	75254	75.78	32.62	0.79	30.65	2454785.48	2471.94
	64	20	149100	148.01	34.12	0.85	29.30	5087292.00	5050.10
DA_3-ADF [70]	32	8	59567	24.19	2.85	2.85	350.87	169765.95	68.94
	64	8	109756	47.82	2.85	2.85	350.87	312804.6	136.287
	32	20	138989	56.44	7.13	7.13	140.25	990991.57	402.41
	64	20	256097	111.58	7.29	7.29	137.17	1866947.13	813.41
DA_4-ADF [71]	32	8	37013	17.82	3.46	3.46	289.01	128064.98	61.65
	64	8	72105	34.81	3.68	3.68	271.17	265346.40	128.10
	32	20	86367	40.97	3.52	3.52	284.09	304011.84	144.21
	64	20	168245	81.22	3.83	3.83	261.09	644378.35	311.07
Proposed	32	8	23351	13.49	4.29	4.29	233.10	100175.79	57.87
	64	8	45349	26.12	4.62	4.62	216.45	209512.38	120.67
	32	20	54318	29.76	4.43	4.43	225.73	240628.74	131.84
	64	20	105814	61.08	4.85	4.85	206.18	513197.9	296.24

MSP: minimum sampling period, DAT: data arrival time, ADP: area-delay product, EPS: energy-per-sample.

4

Low Complexity Pipelined LMS based Adaptive Decision Feedback Equalizer

Contents

4.1 Introduction

Mobile communication technology has come a long way since its innovation. It has transformed our societies, especially in ways as to how human in the current societal scenarios access, exchange, and share information with each other. The unprecedented human needs for superior communication led to the deployment of different generations of mobile system across the globe. For instance, first-generation (1G) of analogue mobile phone system to the latest commercial fourth-generation (4G) long-term evolution (LTE) networks [83]. At present, we live in an era of mobile internet with growing demands for higher data rates which calls for a next-generation mobile communication systems. Fifth-generation (5G) mobile communication is fast emerging to tackle the challenges such as to deliver high data rates in demanding applications, to overcome exponential increase in wireless data traffic, to establish connection between billions of smart devices such as surveillance cameras, smart-home/grid devices, connected sensors, etc [84–89]. One of the main goals of a 5G network is to deliver 1-10 gigabit-per-second (Gbps) data rate to individual users [89]. To achieve these data rates, drastic innovations are being made in physical layer of mobile network architecture. Multi-carrier modulation technique such as orthogonal frequency division multiplexing (OFDM) and its variants are potential 5G waveform candidates [90, 91]. Quadrature amplitude modulation (QAM) when coupled with OFDM can provide greater robustness for forthcoming 5G wireless networks. However, at these high data rates 5G is particularly vulnerable to inter-symbol interference (ISI). Hence, there is a call for channel equalization to mitigate the ISI from the received pulses in 5G scenarios. The block diagram of typical OFDM-QAM system with time-domain equalization (TEQ) is shown in Fig. 4.1.

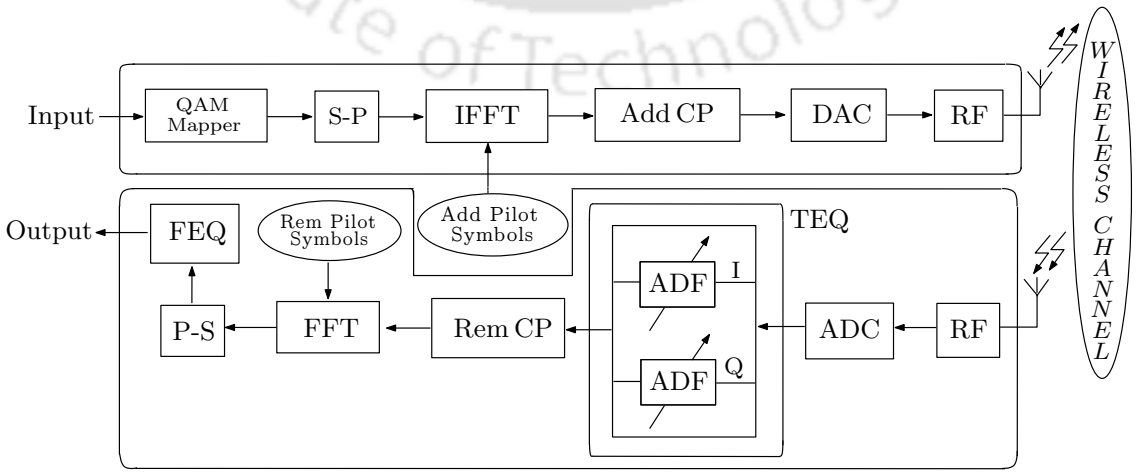


Fig. 4.1: Block diagram of OFDM-QAM with TEQ.

ADF is widely used in channel equalization for estimating the unidentified wireless channel impulse response. It merely balances non-ideal features of wireless channel by stirring up with the supplementary filtering. However, the direct use of ADF in channel equalization can lead to noise amplification at the output, especially when the channel has spectral nulls. This can be overcome by ADFE discussed in Chapter 1, where two LMS ADFs are combined in series which are referred as FF and FB filters, as shown in Fig. 1.4. It offers superior performance against ISI and work well for the channels containing the spectral nulls. However, it has two main design issues. Firstly, the speed is limited by decision feedback loop (DFL). Secondly, the complexity of FF and FB ADFs grow drastically with the transmission data rate. The reason of latter issue is that as the transmission data rate increases, more and more symbols get overlapped. High order FF and FB ADFs are therefore necessary for effective ISI cancellation. This in turn leads to several MAC units which are both area and power hungry, and makes the real-time operation ADFE difficult. In the past, several high-speed multiplierless architectures have been suggested to overcome the speed limitations of DFE [27–41]. They are based on reformulating the DFE architecture into an array of slicers, adders and multiplexers [29–32]. Although, the issue of speed limitation is overcome, the hardware complexity is significantly increased. Later, Lin et al. in [33] suggested a low-complexity fixed-coefficient DFE architecture by pre-computing and storing the coefficients of FB filter in two LUTs separated by pipelined registers. Most of the aforementioned works are applicable to constant or slowly varying channels. However, if the characteristics of channel changes rapidly, then it is necessary to adapt the filter coefficients. Distributed arithmetic (DA) as discussed in Chapter 1 is an efficient technique for the realization of LMS ADFs where the filter partial products are pre-computed and stored in a LUT followed by a SA unit. Over the years, several authors presented efficient LMS ADF architectures using DA [67–71]. They either employed physical LUT or realized LUT with hardware elements (LUT-less). However, the direct realization of FB filter in DFE using DA would increase the critical path since SA unit contribute significant amount of time in the feedback loop. Hence, SA-less approach would be more suitable to design high-throughput ADFE. In this chapter, a case study on channel equalization problem for 5G communication system using ADFE is considered.

Some important design considerations for the formulation of FB filter of an ADFE using DA are listed as:

- As discussed in Chapter 1, when the wordlength of coefficients become equal to the radix size,

the FIR filter architecture does not have SA unit. Thus, the tentative decisions are chosen to be coded in OBC form since their wordlength require fewer bits, and is suitable for low-complexity realization.

- The fact that M-QAM is considered as two independent pulse-amplitude modulations (PAMs) with the tentative decisions belonging to the set $\phi \in [-(M-1), -(M-3), ..., +(M-3), +(M-1)]$, or $\phi = 2m-1-M$ with $m = 0, 1, 2, ..., M-1$. Hence, the number of bits to represent the possible decision values would be $B = \log_2 M$.

- It is observed that the decision values in the set ϕ are same as the digit values in the radix-r OBC-DA except scaling by $1/2$, as discussed in Chapter 2. The scaling by $1/2$ provides data truncation of 1-bit for computing the inner product of two vectors on hardware, while in case of LUT based OBC-DA structures, the word-size of LUT could be reduced by 1-bit.

4.2 Mathematical Formulation and Background

Consider the conventional architecture of ADFE as shown in Fig. 4.2. The critical path as indicated by dashed line includes the computational delay of a multiplier, two adders and a slicer, that is, $T_{MUL} + 2T_A + T_S$, where T_{MUL}, T_A and T_S are the computational delays of a multiplier, an adder and a slicer respectively. The multipliers present in FB filter of ADFE can be replaced by a 2-to-1 multiplexer, and the computational time of slicer can be ignored since tentative decision $\delta(n)$ belong to the set $\phi \in [-1, 1]$ for in-phase (I) and quadrature-phase (Q) channels of 4-QAM. The critical path delay of 4-QAM ADFE can be further reduced by replacing with parallel carry-save full adder and followed by two-input carry propagate adder. Hence, the critical path of 4-QAM ADFE becomes $T_M + T_{FA} + T_A$, where T_M and T_{FA} are the computational delays of a 2-to-1 multiplexer and a carry-save full adder respectively. In TSMC 90 nm CMOS Library [79], the values of T_{MUL}, T_A, T_S, T_M and T_{FA} are 1.54 ns, 0.74 ns, 0.05 ns, 0.10 ns and 0.21 ns respectively, with inputs and coefficients of FB filter assumed to be 8-bit wide. Similarly, the critical path of 16-QAM ADFE can be estimated as $2T_M + T_{FA} + T_A + T_S$ which is 1.20 ns. It is below the throughput requirement of 5G communication system. For the sake of clarity, we restrict our discussion to I-channel of 4-QAM transmitted signals.

Let us assume that the coefficients of FF and FB filters are fixed. In such case, the output of

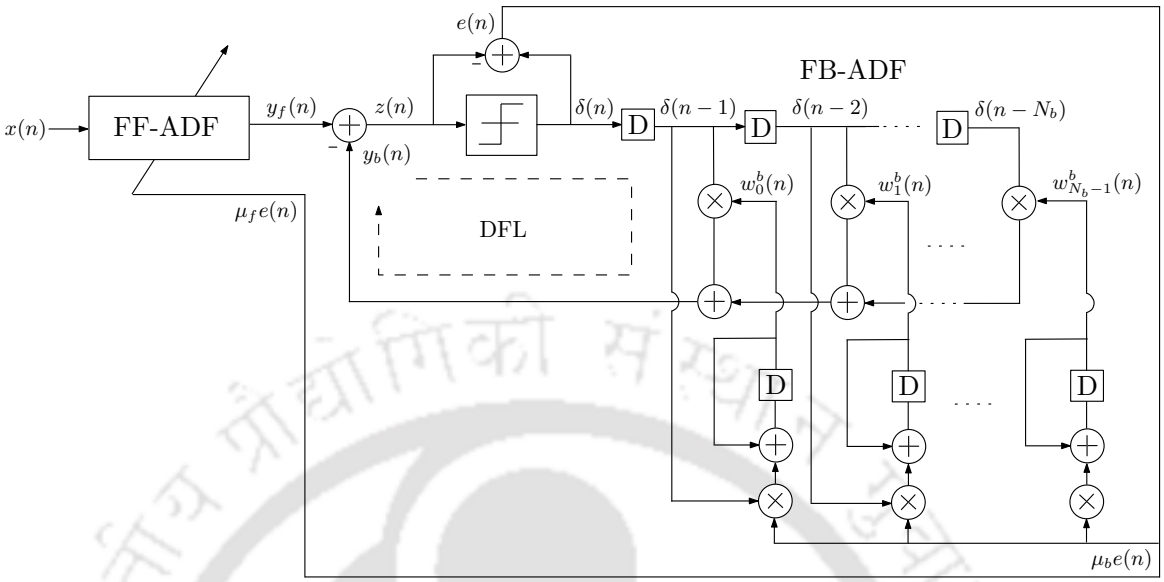


Fig. 4.2: Architecture of conventional ADFE for N_b^{th} order FB filter.

ADFE $z(n)$ can be given as

$$z(n) = y_f(n) - y_b(n) = \sum_{i=0}^{N_f-1} w_i^f x(n-i) - \sum_{i=1}^{N_b} w_i^b \delta(n-i) \tag{4.1}$$

where $y_f(n)$ is the output of FF filter and $y_b(n)$ is the output of FB filter. The decision device (or quantizer) would map the value of $z(n)$ into one of the elements in the set ϕ, according to

$$\delta(n) = Q[z(n)] \tag{4.2}$$

where $\delta(n)$ is the most recent decision. It is important to note that the FF filter can be realized in a straightforward manner using DA as discussed in Chapter 2. However, the new challenge is to lower the computational complexity of FB filter while simultaneously meeting the throughput requirement of 5G communication system. Thus, it is important to formulate FB filter with tentative decisions using OBC-DA as it offers low implementation complexity. Suppose, if every tentative decision $\delta(n-i)$ is represented in B-bit signed two's complement form, we get

$$\delta(n-i) = -\delta_{i,0}(n) + \sum_{k=0}^{B-1} \delta_{i,k}(n) 2^{-k} \tag{4.3}$$

where $i = 1, 2, ..., N_b$ and $\delta_{i,k}(n)$ denotes the k^{th}-bit of tentative decision $\delta(n-i)$. Using the framework of OBC discussed in Chapter 2, one can express each tentative decision as $\delta(n-i) = \frac{1}{2}[\delta(n-i) -$

$(-\delta(n-i))]$ which gives,

$$\delta(n-i) = \frac{1}{2}[-\delta'_{i,0}(n) + \sum_{k=0}^{B-1} \delta'_{i,k}(n)2^{-k} - 2^{-(B-1)}] \tag{4.4}$$

where $\delta'_{i,k}(n) = \delta_{i,k}(n) - \overline{\delta}_{i,k}(n)$ and $\overline{\delta}_{i,k}(n)$ is one's complement of $\delta_{i,k}(n)$. Substituting (4.4) in (4.1), we get

$$z(n) = y_f(n) - \left\{ \sum_{k=0}^{B-1} \left[\frac{1}{2} \sum_{i=0}^{N_b-1} \delta'_{i,k}(n) w_i^b \right] 2^{-k} - \left[\frac{1}{2} \sum_{i=0}^{N_b-1} w_i^b \right] 2^{-(B-1)} \right\} \tag{4.5}$$

In case of radix-r binary system with $r = 2^\gamma$ where γ denotes adjacent-bit to represent the digit-value of a tentative decision, as discussed in Chapter 1. Hence, one can re-express (4.5) as

$$z(n) = y_f(n) - \left\{ \sum_{k=0}^{B/\gamma-1} \left[\frac{1}{2} \sum_{i=0}^{N_b-1} \delta'_{i,k}(n) w_i^b \right] 2^{-k} - (2^\gamma - 1) \left[\frac{1}{2} \sum_{i=0}^{N_b-1} w_i^b \right] 2^{-(B/\gamma-1)} \right\} \tag{4.6}$$

It is interesting to note that when the radix-size becomes equal to wordlength of tentative decision, i.e., $\gamma = B$, then (4.6) leads to

$$z(n) = y_f(n) - \left\{ \frac{1}{2} \sum_{i=0}^{N_b-1} w_i^b [\delta'_{i,k}(n) - (2^\gamma - 1)] \right\} \tag{4.7}$$

where the term $\frac{1}{2} \sum_{i=0}^{N_b-1} w_i^b [\delta'_{i,k}(n) - (2^\gamma - 1)]$ can be represented in 1-bit less than actual wordlength of LUT since the scaling by 1/2 has nothing to do with the filtering operation, as discussed in Chapter 2. It rather allows data (or filter coefficients) truncation by 1-bit if implemented on hardware or reduce the word-size of LUT. This idea has been addressed in [29] by reformulating the architecture of conventional DFE into an array of slicers, adders and multiplexers. This is because the decision values are known in prior, and thus coefficients of FB filter can be pre-computed and stored in LUT, but OBC-DA principles have not been applied to derive the architecture. If the channel is constant or varying slowly, then it would allow the output of FF filter to be pre-computed and stored along with the coefficients of FB filter in the same LUT. Based on the above discussion, the output of DFE $z(n)$ for 4-QAM in terms of LUT address index a can be re-expressed as

$$z(n) = y_f(n) - \mathbf{a}^T \mathbf{w}^b = y_f(n) - \sum_{i=1}^{N_b} w_i^b (-1)^{(r^a_{N_b-i}+1)}, \quad 0 \le a \le 2^{N_b} - 1 \tag{4.8}$$

where $\mathbf{a} = [(-1)^{r^a_{N_b-1}+1}, (-1)^{r^a_{N_b-2}+1}, ..., (-1)^{r^a_0+1}]^T$, $\mathbf{w}^b = [w_0^b, w_1^b, ..., w_{N_b-1}^b]^T$, ${r_i}^a$ is i^{th} bit representation ($r^a_{N_b-1} = 0$) of address index $a = \sum_{i=0}^{N_b-1} {r_i}^a 2^i$. Depending upon the value of a, the term $z(n)$ would take one out of 2^{N_b} possible combinations. As an example, a reformulated DFE (R-DFE) for 3$^{\text{rd}}$

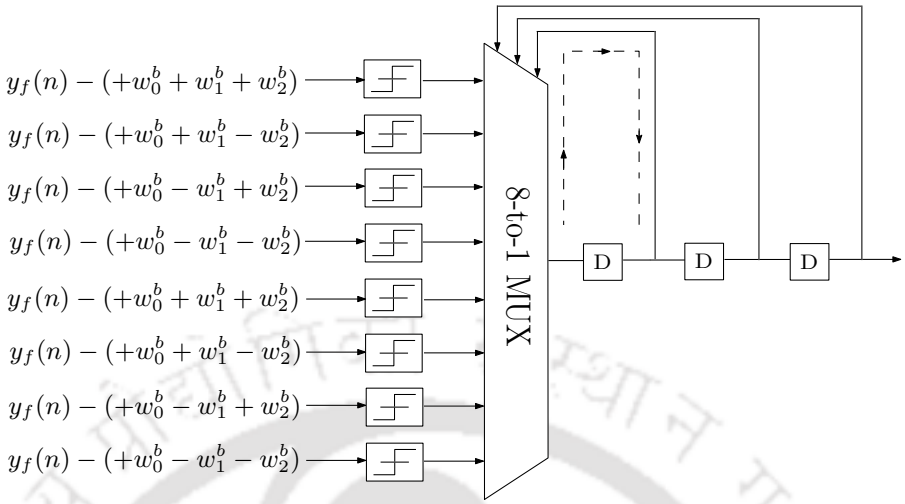


Fig. 4.3: Architecture of R-DFE for 3$^{\text{rd}}$ order FB filter [29].

order FB filter with 4-QAM is shown in Fig. 4.3, where the critical path is the computational delay of one 2-to-1 multiplexer. Although the critical path of R-DFE is much lower than that of conventional DFE, the complexity of multiplexers and LUT grow exponentially with N_b, which is not desirable from implementation point-of-view. Hence, there has been a call for the complexity reduction of R-DFE.

In [33], partial pre-computation (PP) DFE scheme was proposed to pre-compute and store the first few FB filter coefficients (*say*, P) in LUT. For instance, a PP-DFE with $P = 2$ pre-computes and stores the first two FB filter coefficients in LUT as shown in Fig. 4.4(a). As a consequence of partial pre-computation, P number of registers are inserted in the inner loop with a vector merge adder (VMA) comprising of 2^P adders. The presence of these additional registers in the inner loop speeds up by $1/(P+1)$ times over the conventional DFE architecture. Noticeably, the complexity of multiplier is reduced to $N_b - P + 1$, while the complexity of adders and multiplexers become $N_b - P + 2^P$ and 2^P respectively. Based on the above discussion, it is clear that the complexity of multipliers is reduced by pre-computing and storing the coefficients in LUT. However, the number of multipliers corresponding to remaining FB filter coefficients $b_{i|i\in[P+1,P+2,\ldots,N_b]}$ are still high and occupies more area. This has been addressed by the same authors in [33] by pre-computing and storing the remaining FB filter coefficients in another LUT, as shown in Fig. 4.4(b). The scheme is popularly known as two stage pre-computation DFE (TSP-DFE). Clearly, the scheme is free of multipliers, and minimizes the hardware complexity by choosing $P = N_b/2$. The complexity of adders, multiplexers and LUT are $2^{N_b/2+1}$, $2^{N_b/2+1} - 2$ and $2^{N_b/2+1}$ respectively. Mathematically, the contents of LUT for stage-I is

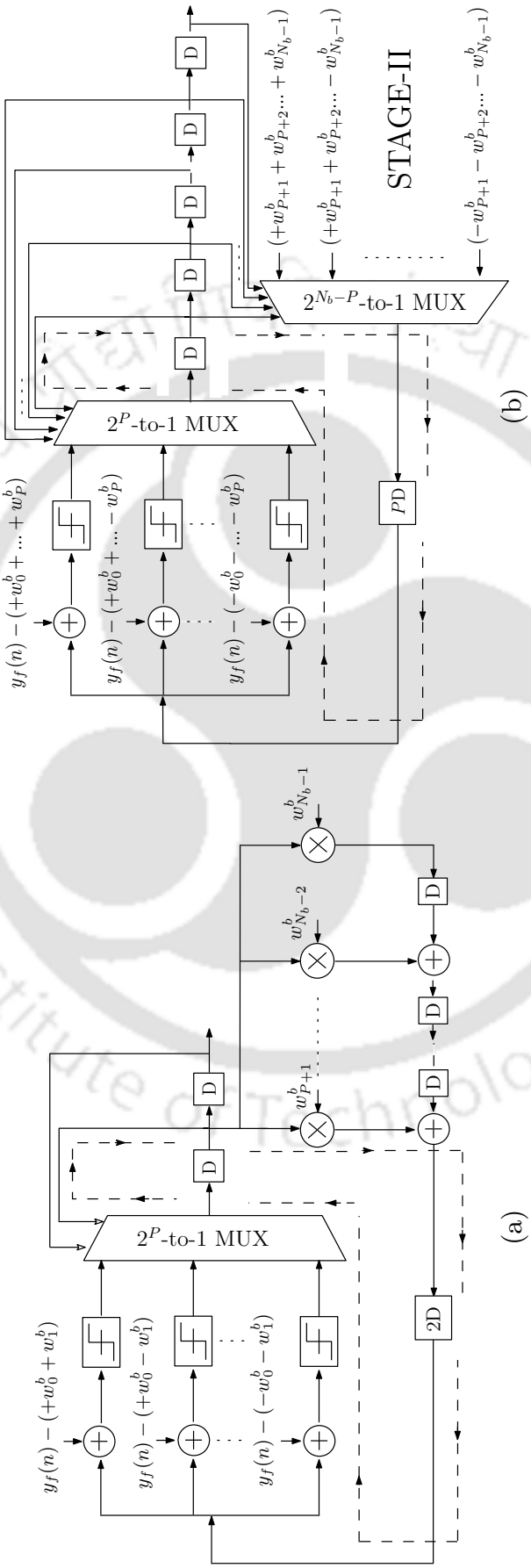


Fig. 4.4: (a) Partial pre-computation DFE scheme for N_b^{th} order FB filter with a speed-up factor of two [33]. (b) Two stage pre-computation DFE for N_b^{th} order FB filter with a speed-up factor of P [33].

$a_3a_2a_1a_0$	LUT_I Data
0 0 0 0	$y_f(n)-(\ w_1^b + w_2^b + w_3^b + w_4^b)$
0 0 0 1	$y_f(n)-(\ w_1^b + w_2^b + w_3^b - w_4^b)$
0 0 1 0	$y_f(n)-(\ w_1^b + w_2^b - w_3^b + w_4^b)$
0 0 1 1	$y_f(n)-(\ w_1^b + w_2^b - w_3^b - w_4^b)$
0 1 0 0	$y_f(n)-(\ w_1^b - w_2^b + w_3^b + w_4^b)$
0 1 0 1	$y_f(n)-(\ w_1^b - w_2^b + w_3^b - w_4^b)$
0 1 1 0	$y_f(n)-(\ w_1^b - w_2^b - w_3^b + w_4^b)$
0 1 1 1	$y_f(n)-(\ w_1^b - w_2^b - w_3^b - w_4^b)$
1 0 0 0	$y_f(n)+(\ w_1^b - w_2^b - w_3^b - w_4^b)$
1 0 0 1	$y_f(n)+(\ w_1^b - w_2^b - w_3^b + w_4^b)$
1 0 1 0	$y_f(n)+(\ w_1^b - w_2^b + w_3^b - w_4^b)$
1 0 1 1	$y_f(n)+(\ w_1^b - b_2 + w_3^b + w_4^b)$
1 1 0 0	$y_f(n)+(\ w_1^b + w_2^b - w_3^b - w_4^b)$
1 1 0 1	$y_f(n)+(\ w_1^b + w_2^b - w_3^b + w_4^b)$
1 1 1 0	$y_f(n)+(\ w_1^b + w_2^b + w_3^b - w_4^b)$
1 1 1 1	$y_f(n)+(\ w_1^b + w_2^b + w_3^b + w_4^b)$

$a_3a_2a_1a_0$	LUT_II Data
0 0 0 0	$(+\ w_5^b + w_6^b + w_7^b + w_8^b)$
0 0 0 1	$(+\ w_5^b + w_6^b + w_7^b - w_8^b)$
0 0 1 0	$(+\ w_5^b + w_6^b - w_7^b + w_8^b)$
0 0 1 1	$(+\ w_5^b + w_6^b - w_7^b - w_8^b)$
0 1 0 0	$(+\ w_5^b - w_6^b + w_7^b + w_8^b)$
0 1 0 1	$(+\ w_5^b - w_6^b + w_7^b - w_8^b)$
0 1 1 0	$(+\ w_5^b - w_6^b - w_7^b + w_8^b)$
0 1 1 1	$(+\ w_5^b - w_6^b - w_7^b - w_8^b)$
1 0 0 0	$(-\ w_5^b - w_6^b - w_7^b - w_8^b)$
1 0 0 1	$(-\ w_5^b - w_6^b - w_7^b + w_8^b)$
1 0 1 0	$(-\ w_5^b - w_6^b + w_7^b - w_8^b)$
1 0 1 1	$(-\ w_5^b - w_6^b + w_7^b + w_8^b)$
1 1 0 0	$(-\ w_5^b + w_6^b - w_7^b - w_8^b)$
1 1 0 1	$(-\ w_5^b + w_6^b - w_7^b + w_8^b)$
1 1 1 0	$(-\ w_5^b + w_6^b + w_7^b - w_8^b)$
1 1 1 1	$(-\ w_5^b + w_6^b + w_7^b + w_8^b)$

Fig. 4.5: LUT contents of stage-I and stage-II for 8^{th} order FB filter.

expressed as

$$\mathcal{C}_a^I = y_f(n) - \sum_{i=1}^{P} w_i^b (-1)^{(r_{P-i}^a+1)}, \quad 0 \le a \le 2^P - 1$$

$$\mathcal{C}_a^I = y_f(n) - \mathbf{a_I}^T \mathbf{w_I}^b \tag{4.9}$$

where $\mathbf{a_I} = [(-1)^{r_{P-1}^a+1}, (-1)^{r_{P-2}^a+1}, ..., (-1)^{r_0^a+1}]^T$ and $\mathbf{w_I}^b = [w_1^b, w_2^b, ..., w_P^b]^T$. Similarly, the contents of LUT for stage-II is expressed as

$$\mathcal{C}_a^{II} = \sum_{i=P+1}^{N_b} w_i^b (-1)^{(r_{N_b-P-i}^a+1)}, \quad 2^P \le a \le 2^{N_b} - 1$$

$$\mathcal{C}_a^{II} = \mathbf{a_{II}}^T \mathbf{w_{II}}^b \tag{4.10}$$

where $\mathbf{a_{II}} = [(-1)^{r_{N_b-P-1}^a+1}, (-1)^{r_{N_b-P-2}^a+1}, ..., (-1)^{r_{N_b}^a+1}]^T$ and $\mathbf{w_{II}}^b = [w_{P+1}^b, w_{P+2}^b, ..., w_{N_b}^b]^T$. It is interesting to note from Fig. 4.5 that the content of LUT_II at 0^{th} address location is two's complement of the content at 15^{th} address location, the content of LUT_II at 1^{st} address location is two's complement of the content at 14^{th} address location and so on. This implies that the contents of LUT_II can be

written as

$$\mathcal{C}_a^{II} = -\mathcal{C}_{2^{N_b}-(a+1)}^{II}, \qquad 2^P \leq a \leq 2^{N_b-1} - 1 \tag{4.11}$$

It is clear from (4.11) that the size of LUT_{II} can be reduced to half. While the presence of FF filter output $y_f(n)$ in LUT_{I} makes it difficult to reduce its size. Hence, this scheme is particularly suitable for slowly varying channels.

4.3 Proposed Scheme

In this section, the contents of LUT_{I} and LUT_{II} of existing TSP-DFE [33] shown in Fig. 4.5 are first transformed using the framework of OBC-DA. Next, the proposed architecture is retimed and then unfolded to achieve throughput of 2.88 Gbps and 2.15 Gbps for 4-QAM and 16-QAM respectively. Lastly, a new strategy is presented to update the coefficients of TSP-DFE by storing the recent decisions in a separate LUT.

4.3.1 Transformation of LUT Contents

As mentioned, TSP-DFE offers low-implementation complexity when the number of pre-computed filter coefficients becomes half of the FB filter order. Further, it is observed that the contents of LUT_{II} have symmetries, therefore, its size can be reduced to half, in accordance with OBC. Thus, the contents of LUT_{II} can be transformed as per

$$\mathcal{C}_a^{II} \rightarrow (-1)^{\left\lfloor \frac{a}{2^{N_b-P-1}} \right\rfloor + 1} \mathcal{C}_{a/2}^{II} \tag{4.12}$$

From (4.12), it is clear that the size of LUT_{II} is reduced to half by reversing sign of the other half contents. This is achieved by using few XOR gates for addresses LUT_{II} with $a \geq 2^{N_b-P-1}$. For example, an 8^{th} order FB filter requires 4-XOR gates for LUT_{II} to obtain the remaining half terms, as shown in Fig. 4.6. The savings achieved by exploiting symmetry are significant, especially when the order of FB filter is large. The presence of $y_f(n)$ in the contents of LUT_{I} disturbs the symmetry. However, if $y_f(n)$ is added external to LUT_{I} contents, then mirror symmetry can be achieved similar to LUT_{II}. The contents of LUT_{I}, therefore, can be transformed according to

$$\mathcal{C}_a^{I} \rightarrow y_f(n) + (-1)^{\left\lfloor \frac{a}{2^{P-1}} \right\rfloor} \mathcal{C}_{a/2}^{I} \tag{4.13}$$

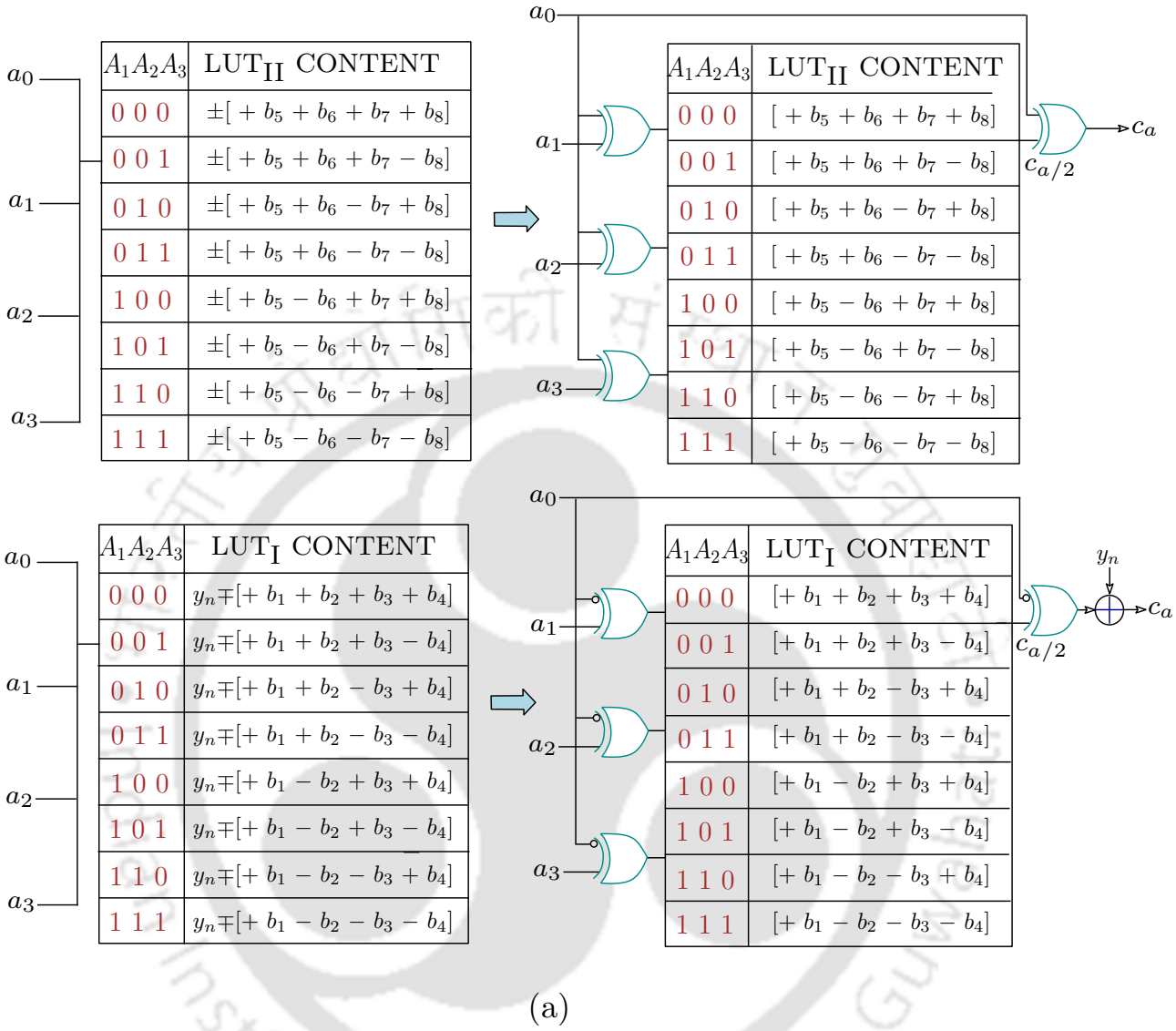


Fig. 4.6: Transformation of LUT contents of stage-I and stage-II for 8$^{\text{th}}$ order FB filter.

For an 8$^{\text{th}}$ order FB filter, the implementation of (4.13) requires 4-XOR gates and an adder, as shown in Fig. 4.6. In general, the contents of LUT$_{\text{I}}$ and LUT$_{\text{II}}$ for N_b^{th} order FB filter after transformation can be expressed as

$$\mathcal{C}_{a/2}^{I,II} = (-1)^{\mathcal{P}}\left\{w_{\mathcal{P}}^{b} + \sum_{i=1}^{N_b/2} w_{\mathcal{P}+i}^{b}(-1)^{\left(r_{N_b/2+\mathcal{P}-i}^{(a/2)}+1\right)}\right\} \tag{4.14}$$

with

$$\mathcal{P} = \begin{cases} 1, & \text{for LUT}_{\text{I}} \\ N_b/2, & \text{for LUT}_{\text{II}} \end{cases} \tag{4.15}$$

where N_b is assumed to be an integer multiples of 2. In order to obtain the other half combinations of LUT_I and LUT_II contents, the relation $\mathcal{C}^{I,II}_{a/2} = (-1)^{\mathcal{P}+1}\mathcal{C}^{I,II}_{a/2}$ holds good.

4.3.2 Proposed Architecture

It can be noted from Fig. 4.6 that the LUT sizes of stage-I and stage-II are reduced to half. The architecture obtained by combining the reduced size LUTs with TSP-DFE (in Fig. 4.4(b)) and keeping $P = N_b/2 + 1$ is shown in Fig. 4.7(a). In this case, one can determine the iteration bound as $(2T_A + T_S)/(N_b/2 + 2) + T_M \log_2 M$. Note the addition of $y_f(n)$ in DFL has slightly increased the iteration bound. This could be significant for unfolded architecture, for example, if the proposed architecture is unfolded by a factor of 2, then resulting latency due to adders would be $10T_M$ [78], with T_M being the computational delay of a 2-to-1 multiplexer. In order to overcome this issue, the register corresponding to $P = 1$ is retimed to pipeline the VMA unit, as shown in Fig. 4.7(b). Some of the salient features of proposed architecture are listed as:

- Iteration bound of proposed architecture for $N_b{}^{\text{th}}$ order FB filter is $(T_A + 0.5T_S)/(N_b/4 + 1) + T_M \log_2 M$. It can be noted that its value decreases asymptotically to $T_M \log_2 M$, especially when order of FB filter is large. While the hardware complexity is almost reduced by $(M/2)^{-1}$ for M-QAM scheme as compared to the design presented in [33].

- In wireless communication, it is difficult to store $y_f(n)$ in the LUT due to rapid channel variations. Moreover, the time and power required to update the LUT contents would be more due to its large size. Adding $y_f(n)$ outside the LUT not only reduces its time and power consumptions but also adapts the channel variations.

- In existing TSP-DFE [33], the fan-out number of stage-I and stage-II multiplexers grow exponentially with $N_b/2$. However, this seems to be high and can cause fan-out impairment in stage-I and stage-II for large order filters. This is almost reduced by a factor of two using the proposed approach.

4.3.3 Design of High-Throughput Architecture

In order to derive the high-throughput architecture, an example of 4-QAM constellation is considered with orders of FF and FB filters as 10 and 8 respectively. The wordlengths of filter coefficients and inputs are assumed to be 8-bit. With these parameters, the proposed design has iteration bound

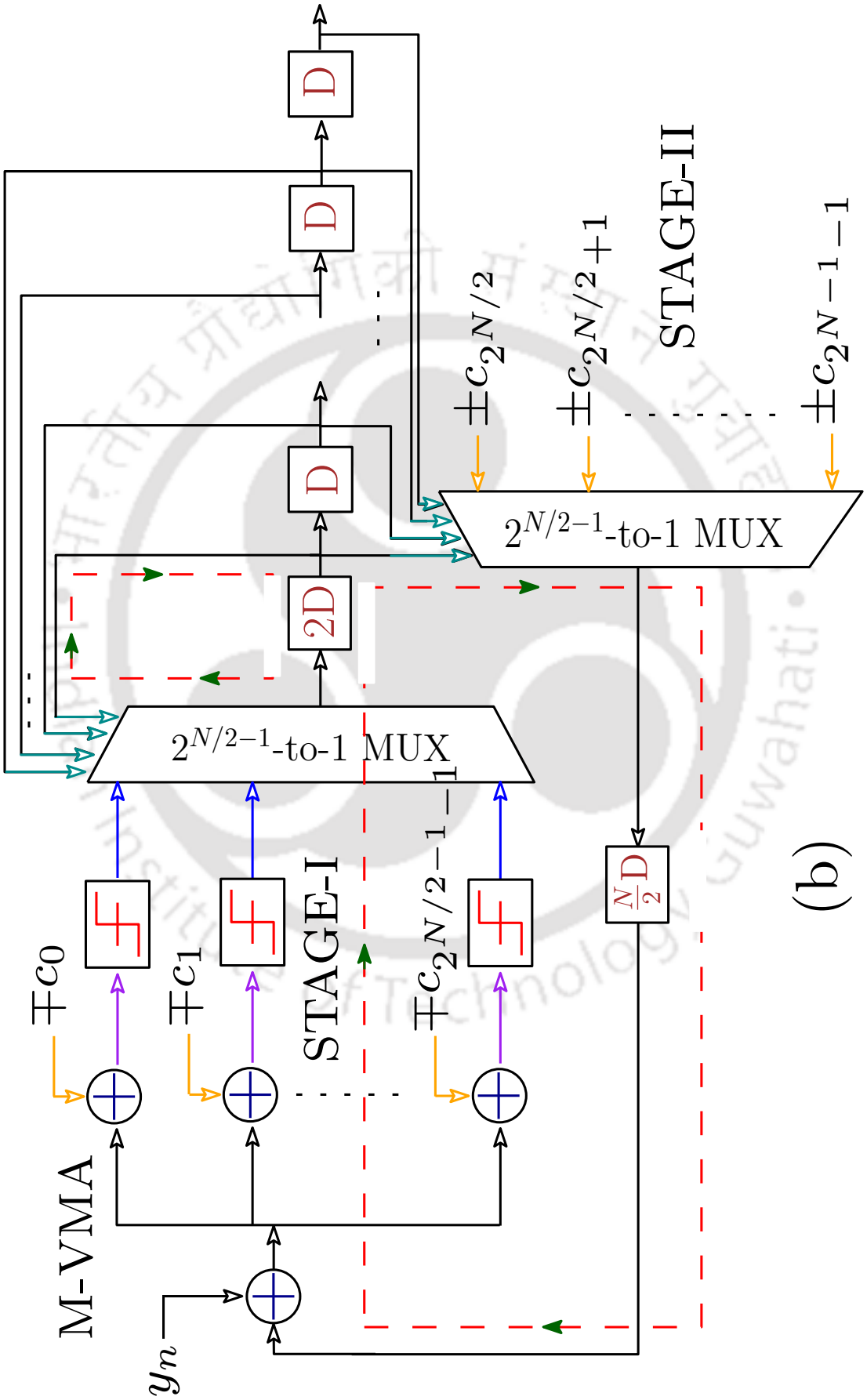
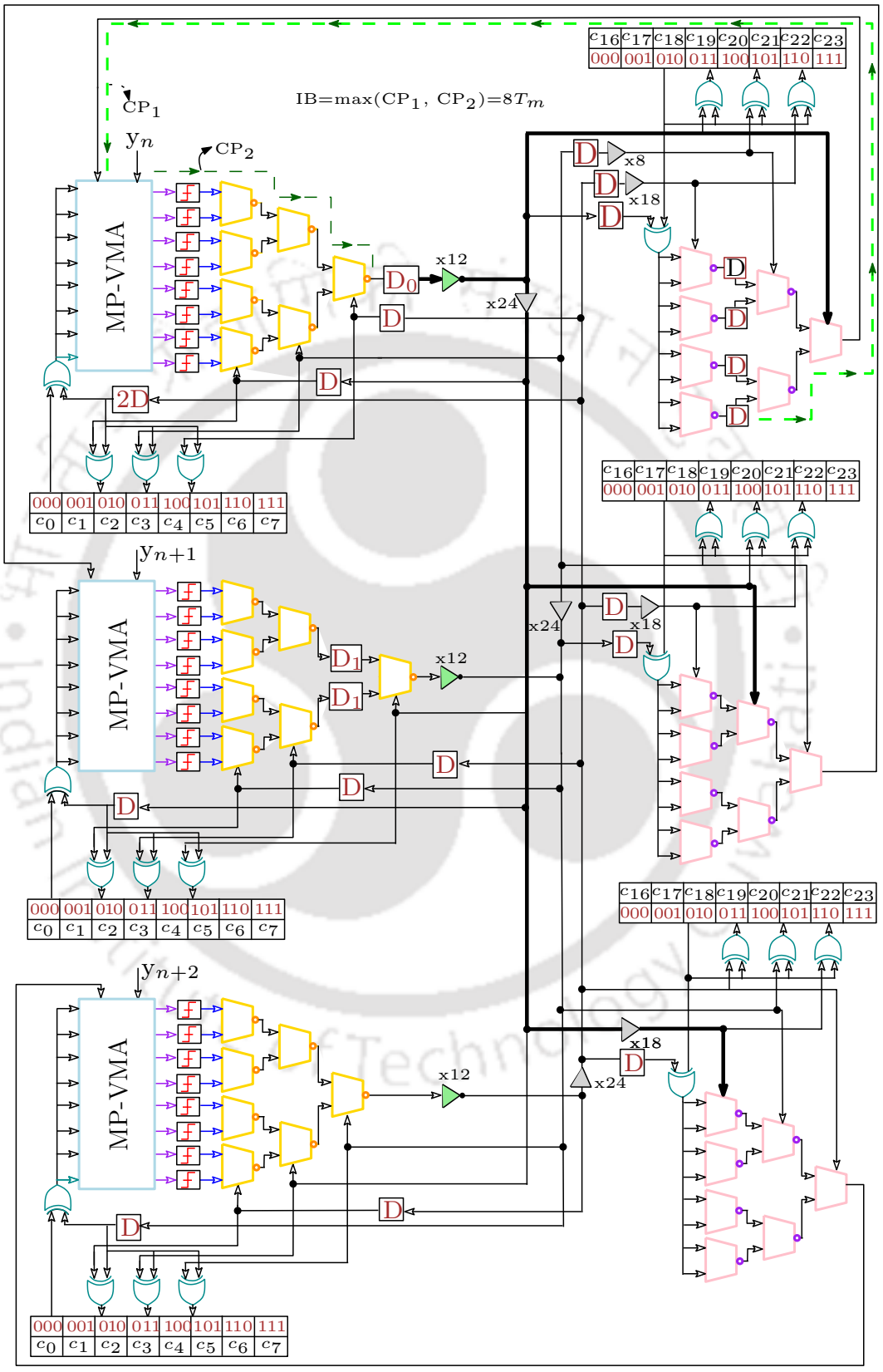


Fig. 4.7: (a) Architecture of TSP-DFE after the LUTs' transformation. (b) Retimed architecture of TSP-DFE after the LUTs' transformation.

gfx/fig32(a,b)_jet-eps-converted-to.pdf

Fig. 4.8: Data flow graph (DFG) of TSP-DFE architecture with unfolding factor of three, where $X_k - x_k$, $Y_k - y_k$ $(0 \leq k \leq 2)$ denote the operation of 8-to-1 multiplexer - XOR gate for stage-I and stage-II respectively; Z_k denotes the combined operation of slicer and pipelined vector merge adder.

Fig. 4.9: Circuit schematic of unfolded architecture with unfolding factor of three.



(c)

Fig. 4.10: Detailed circuit schematic of unfolded architecture with unfolding factor of three, and necessary modifications with inverted multiplexers, buffers/inverters and retiming.

$8T_M/3$ and require 17 adders, 16 multiplexers and 10 registers for the implementation. In general, the iteration bound of unfolded architecture for FB filter order N_b, constellation size M and unfolding factor J can be estimated as $\left(T_A + T_S + \log_2 M(\frac{N_b}{2} - 1)T_M\right)/J$. To achieve the throughput requirements of 5G communication system while maintaining the low-implementation complexity, it is important to select a small value of unfolding factor J, and satisfying the timing constraints of VMA. This is because every rise in the unfolding factor would proportionally increase the hardware complexity. And, the maximum computation time of pipelined VMA unit is $5T_M$, which is larger than iteration bound of the proposed TSP-DFE. As a result, retiming technique cannot be employed for achieving the sampling period corresponding to iteration bound [74]. Hence, the proposed architecture is unfolded by a factor of 3 to meet the throughput requirements of 5G communication system. This requires a systematic approach which involves the development of data flow graph (DFG) and corresponding architecture for the unfolded design. The proposed unfolded DFG is shown in Fig. 4.8, where the symbols $X_k - x_k$, $Y_k - y_k$ $(0 \leq k \leq 2)$ denote the operation of 8-to-1 multiplexer - a XOR gate for stage-II and stage-I respectively; and Z_k denotes the combined operation of slicer and modified pipelined VMA (MP-VMA). The unfolded architecture shown in Fig. 4.9 has iteration bound of $8T_M$. Since 3-unfolded architecture process 3 samples simultaneously, the effective iteration bound becomes $8T_M/3$. In similar arguments, the iteration bound of the proposed unfolded architecture for 16-QAM would be $11T_M/3$. However, the above analysis assumed the computational delay of slicer as zero and the computational delay of adder as $T_A = 5T_M$ [74], therefore, the actual iteration bound of unfolded architecture for 16-QAM must be $(T_A + T_S + 6T_M)/3$. Note that the cells used in the unfolded architecture for multiplexer, slicer, XOR-gate, adder and D-flip flop are from TSMC 90 nm CMOS Library with computational delays as 0.1 ns, 0.05 ns, 0.1 ns, 0.74 ns and 0.19 ns respectively. In order to reduce latency and increase the fan-out number of stage-I and stage-II of the proposed unfolded architecture, the following efforts have been made:

4.3.3.1 Use of Inverted Multiplexers

In TSMC 90 nm CMOS Library, the latency of inverted 2-to-1 multiplexer is nearly half than that of normal 2-to-1 multiplexer. By carefully observing the architecture obtained in Fig. 4.10, several 2-to-1 multiplexers are replaced with inverted 2-to-1 multiplexers without affecting its functionality in order to obtain the low latency architecture. Note that the inputs of 2-to-1 multiplexers need to be interchanged for the inverted output.

4.3.3.2 Use of Buffers/Inverters

A set of inverters or buffers (or both) are always useful to increase the fan-out of output stages. The proposed design uses several XOR gates for sign reversal operation, therefore, the fan-out needs to be maintained large as compared to the design presented in [33]. By trial and error method, the strengths of buffers and inverters are fixed, as indicated in Fig. 4.10.

4.3.3.3 Use of Retiming Technique

The inverted 2-to-1 multiplexers and buffers/inverters have reduced the latency from stage-I to stage-II for each unfolded level. However, the latency of stage-II is increased by a 2-to-1 multiplexer for every unfolded level (shown by bold line). This is overcome by retiming the D_1 register as shown in Fig. 4.10. In similar manner, D_2 register can also be retimed to obtain the same latency for each unfolded level.

4.3.4 Coefficient Update Unit

Recall that, in TSP-DFE, the first P filter coefficients are pre-computed and stored in LUT, and the remaining $N_b - P$ filter coefficients are also pre-computed and stored in another LUT. However, a pipelined inner loop is formed between the LUTs. The convergence performance worsens with increasing values of P, due to more errors in the TSP-DFE output. In order to improve the convergence performance of TSP-DFE, the following mathematical analysis is carried out. By re-writing (4.1), we get

$$z(n) = y_f(n) - \mathbf{d}^T(n)\mathbf{w}^b(n) \tag{4.16}$$

where $y_f(n) = \mathbf{x}^T(n)\mathbf{w}^f(n)$, $\mathbf{d}(n) = [\delta(n-1)...\ \delta(n-P), \delta(n-P+1)...\delta(n-N_b)]^T$ and $\mathbf{w}^b(n) = [w_1^b(n), ..., w_P^b(n), w_{P+1}^b(n), ..., w_{N_b}^b(n)]^T$. From (4.16), $\mathbf{d}^T(n)\mathbf{w}^b(n)$ can be decomposed into two smaller inner products corresponding to the stages of TSP-DFE as

$$\mathbf{d}^T(n)\mathbf{w}^b(n) = \mathbf{d}_{\mathrm{I}}^T(n)\mathbf{w}_{\mathrm{I}}^b(n) + \mathbf{d}_{\mathrm{II}}^T(n)\mathbf{w}_{\mathrm{II}}^b(n) \tag{4.17}$$

where $\mathbf{w}_{\mathrm{I}}^b(n) = [w_1^b(n), w_2^b(n), ..., w_P^b(n)]^T$, $\mathbf{w}_{\mathrm{II}}^b(n) = [w_{P+1}^b(n), w_{P+2}^b(n), ..., w_{N_b}^b(n)]^T$, $\mathbf{d}_{\mathrm{I}}(n) = [\delta(n-1),\ \delta(n-2)...\delta(n-P)]^T$ and $\mathbf{d}_{\mathrm{II}}(n) = [\delta(n-P-1), \delta(n-2)...\delta(n-N_b)]^T$. If the proposed architecture shown in Fig. 4.7(b) is speedup by a factor of P, then (4.17) becomes

$$\mathbf{d}^T(n-P)\mathbf{w}^b(n-P) = D_P\{\mathbf{d}_{\mathrm{II}}^T(n)\mathbf{w}_{\mathrm{II}}^b(n)\} - \mathbf{d}_{\mathrm{I}}^T(n-P)\mathbf{w}_{\mathrm{I}}^b(n-P) \tag{4.18}$$

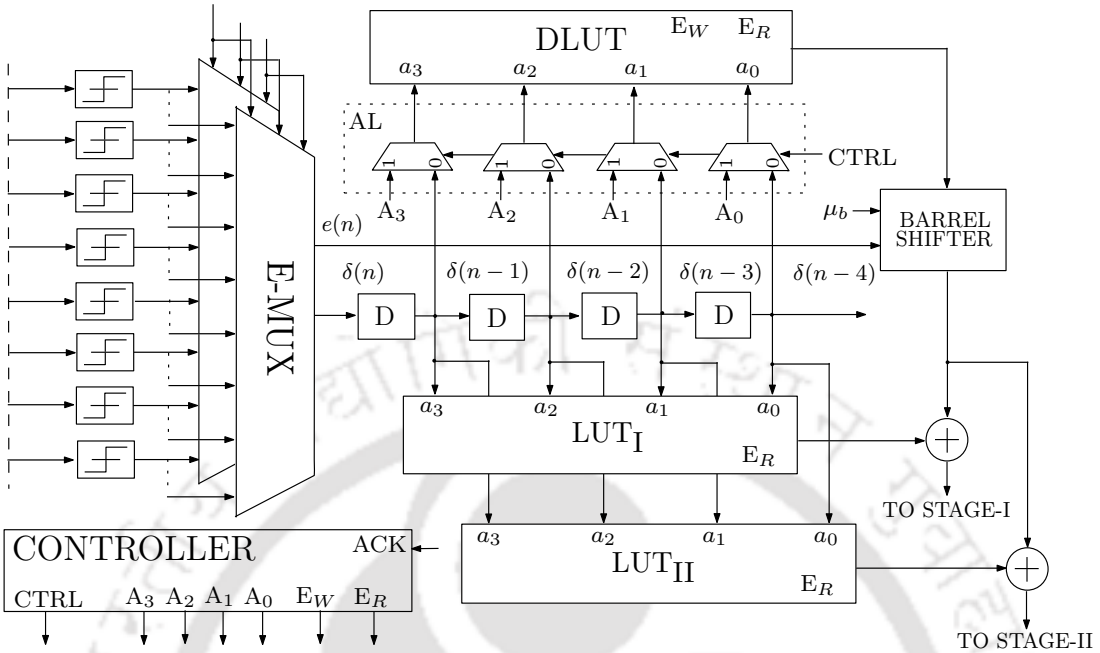


Fig. 4.11: Circuit schematic of coefficient update unit for 8^{th} order FB filter and speed-up factor of four.

where D_P denotes the delay operator to be applied on stage-II output whereas the output of LUT_I need to be inverted in accordance with (4.13). As stated, the convergence of TSP-DFE worsens with increasing values of P, on the other hand, it speeds up the inner loop. Therefore, it can be considered as a constraint optimization problem i.e., $min\{\mathbb{E}[(\delta(n)-z(n))^2]\}$ with constraint on P. By substituting (4.18) in (1.7) and re-arranging, we get

$$e(n) = y_f(n) + [\delta(n) - \mathbf{d}_\text{I}^T(n-P)\mathbf{w}_\text{I}^b(n-P)] + D_P\{\mathbf{d}_\text{II}^T(n)\mathbf{w}_\text{II}^b(n)\} \tag{4.19}$$

According to (4.19), $y_f(n)$ is added explicitly in the inner loop and the term $\mathbf{d}_\text{I}^T(n-P)\mathbf{w}_\text{I}^b(n-P)$ indicates the dependencies due to past decisions corresponding to first P filter coefficients. These are nothing but non-OBC terms due to past decision as a result of pipelined architecture discussed in Chapter 2. To improve the convergence of proposed TSP-DFE against these non-OBC terms, the effect of $\mathbf{d}_\text{I}^T(n-P)\mathbf{w}_\text{I}^b(n-P)$ has to be eliminated. This can be achieved by combining the slicer output $\delta(n)$ with $\mathbf{d}_\text{I}^T(n-P)\mathbf{w}_\text{I}^b(n-P)$. Clearly, it has an advantage of minimizing $e(n)$ when $[\delta(n) - \mathbf{d}_\text{I}^T(n-P)\mathbf{w}_\text{I}^b(n-P)]$ approaches to least value, as per aforesaid constraint. Structurally, it is achieved by placing an error multiplexer (E-MUX) in parallel to the stage-I multiplexer for computing $e(n)$, as shown in Fig. 4.11. Further, the additional hardware of 2^P adders and 2^P-to-1 multiplexers

are required which slightly increases the critical path. It can be easily seen that the hardware overhead is compensated by the proposed low-complexity design as discussed in Section 4.3.1.

The coefficients of FB filter $w_i^b(n)$ $(i = 0, 1, 2, ..., N_b - 1)$ are updated using the scaled error signal $\mu_b e(n)$ and tentative decision $\delta(n - i)$, in accordance with (1.9). In every iteration, E-MUX and stage-I multiplexer generates $e(n)$ and $\delta(n)$ in parallel, where the choice of μ_b decides the rate of convergence and steady-state error performance. The multiplication of step-size μ_b and error $e(n)$ can be obtained by quantizing μ_b in negative powers of two and considering the most significant bit (MSB) of $e(n)$ [67]. In case of TSP-DFE, the contents stored in LUT_I and LUT_II need to be re-calculated. This is achieved by a separate LUT storing the OBC combination of tentative decisions referred as decision LUT (DLUT). In the proposed scheme, DLUT is operated in two separate modes, namely, decisions update (DU) and coefficients update (CU). The sequence of operations in DU and CU mode are explained as follows. In DU mode, write enable (E_W) and CTRL signal are first set to logic '1'. This would start the process of updating the contents of DLUT using addressing logic (AL) unit. Note that the time required to update the contents of DLUT depends on the number of clock cycles corresponding to its address locations. For $N_b{}^{\text{th}}$ order FB filter, the total number of clock cycles to update DLUT would be $2^{N_b/2-1}$. An acknowledge (ACK) signal is sent back to controller, after all the contents of DLUT are updated. Controller in turn disables E_W and CTRL by setting them to logic '0'. In CU mode, the read enable (E_R) signal of all the LUTs are set to logic '1'. This would allow the LUTs to perform read operation on their stored contents. It is clear from the proposed algorithm that an updated DLUT is used to update LUT_I and LUT_II. Therefore, the contents of each address location a corresponding to LUT_I and LUT_II are updated according to following expressions

$$\sum_{i=1}^{P} r_i^a w_i^b(n+1) = \sum_{i=1}^{P} r_i^a w_i^b(n) + \mu_b e(n) \sum_{i=1}^{P} r_i^a \delta(n-i) \tag{4.20}$$

$$\sum_{i=P+1}^{N_b} r_i^a w_i^b(n+1) = \sum_{i=P+1}^{N_b} r_i^a w_i^b(n) + \mu_b e(n) \sum_{i=P+1}^{N_b} r_i^a \delta(n-i) \tag{4.21}$$

respectively, where ${r_i}^a$ is i^{th} bit representation $({r_P}^a, r_{N_b}^a = 0)$ of address index $a = \sum_{i=1}^{P} {r_i}^a 2^i$ for stage-I and $a = \sum_{i=P+1}^{N_b} {r_i}^a 2^i$ for stage-II. It is clear from (4.20) and (4.21) that the pattern of decisions to be stored in DLUT must be of same type as that of LUT_I and LUT_II. At time instant $n + 1$, the contents of LUT_I and LUT_II can be obtained by adding with the corresponding DLUT contents using

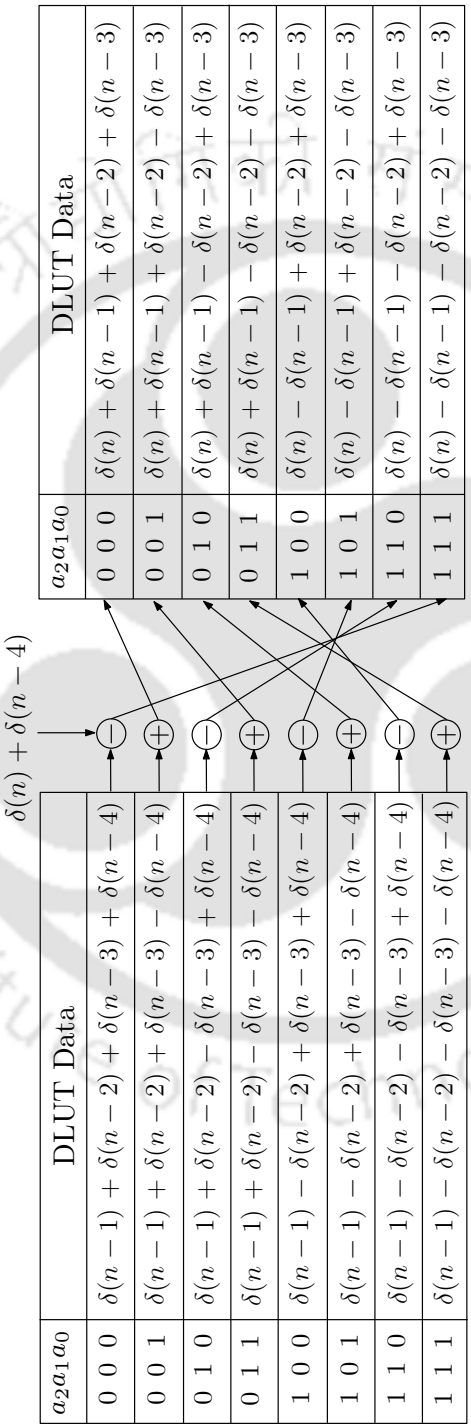


Fig. 4.12: DLUT update scheme from time n to $n+1$ for 8^{th} order FB filter and speed-up factor of four.

the adders shown in Fig. 4.11. The update scheme of DLUT from time instant n to $n+1$ for 8^{th} order FB filter is shown in Fig. 4.12. To update DLUT contents, the recent decision $\delta(n)$ must be added while the oldest decision $\delta(n-4)$ must be eliminated. It can be seen that the oldest decision $\delta(n-4)$ is present corresponding to LSB of DLUT address line whose sign alternates between $'+'$ and $'-'$ corresponding to even and odd address locations. In order to eliminate oldest decision $\delta(n-4)$, the external term $\delta(n)+\delta(n-4)$ is either subtracted or added depending upon even or odd address locations. Further, it requires the re-mapping of DLUT contents at time instant $n+1$, as illustrated in Fig. 4.12. For example, the content at 0^{th} address location of DLUT is $\delta(n-1)+\delta(n-2)+\delta(n-3)+\delta(n-4)$ which contains the oldest decision $+\delta(n-4)$, when subtracted from $\delta(n)+\delta(n-4)$ would give $\delta(n)-\delta(n-1)-\delta(n-2)-\delta(n-3)$. Careful observation suggests that it is nothing but the content of 7^{th} address location at time instant $n+1$. In similar manner, all the remaining address locations of DLUT can also be updated. Mathematically, the contents of DLUT at any time n can be written as

$$\begin{aligned} D_a(n+1) &= [\delta(n)+\delta(n-4)] + D_{2a+1}(n) && a < 2^{N_b/2-1} \\ D_a(n+1) &= [\delta(n)+\delta(n-4)] - D_{2(2^{N_b/2}-1-a)}(n) && a \geq 2^{N_b/2-1} \end{aligned} \tag{4.22}$$

In the previous discussion, it is mentioned that the updated DLUT contents are used to update LUT_I and LUT_II contents. It would result in significant power savings while keeping the computation time less to update the coefficients of FB filter. Algorithm 3 explains the operation of proposed ADFE.

Algorithm 3 Algorithm explaining the operation of proposed ADFE

Initialize:

$a = a', D_P = P = N_b/2$

loop

$z(n) = y_f(n) - y_b(n)$

$e(n) = \delta(n) - z(n)$ with $\delta(n) = Q[z(n)]$

for $i = 0$ to $N_f - 1$ **do**

$y_f(n) = y_f(n) + w_i^f(n)x(n-i)$

$w_i^f(n+1) = w_i^f(n) + \mu_b e(n)x(n-i)$

end for

if $CTRL\&\&E_W\&\&\overline{E_R} == 1$ **then**

for $a = 0$ to $2^{N_b/2}$ **do**

if $a < 2^{N_b/2-1}$ **then**

$D_a(n+1) = [\delta(n) + \delta(n - N_b/2)] + D_{2a+1}(n)$

else

$D_a(n+1) = [\delta(n) + \delta(n - N_b/2)] - D_{2(2^{N_b/2}-1-a)}(n)$

end if

end for

set $ACK = 1$

end if

if $\overline{CTRL}\&\&\overline{E_W}\&\&E_R == 1$ **then**

$\mathcal{C}_a^I(n+1) \leftarrow \mathcal{C}_a^I(n) + \mu_b e(n)D_a(n)$

$\mathcal{C}_a^{II}(n+1) \leftarrow \mathcal{C}_a^{II}(n) + \mu_b e(n)D_a(n)$

end if

for $i = 0$ to D_P **do**

$\mathcal{C}_a^{II}(n-i+1) \leftarrow \mathcal{C}_a^{II}(n-i)$

end for

$y_b(n) \leftarrow \mathcal{C}_a^{II}(n) + \mathcal{C}_a^I(n)$

$e(n) \leftarrow [\delta(n) - \mathcal{C}_a^I(n)] + \mathcal{C}_a^{II}(n)$

$a' \leftarrow a$

$n \leftarrow n+1$

end loop

4.4 Performance Comparison

In this section, the performance comparison between the proposed and existing designs are carried out. For non-adaptive DFE [29, 33], same coefficient update unit is considered, as derived in Section 4.3.4. Next, the convergence performance and bit-error rate (BER) with different speedup factors are evaluated for both the proposed and existing designs. Finally, the ASIC and FPGA synthesis results of the proposed and existing designs are compared.

4.4.1 Computational Complexities

4.4.1.1 Hardware Complexity

The hardware complexities of the proposed and existing designs are listed in Table 4.1. As expected, the proposed non-adaptive TSP-DFE has reduced the hardware complexity and the LUT size by almost half for 4-QAM as compared to the existing non-adaptive TSP-DFE. In general, the reduction in hardware complexity of the proposed non-adaptive TSP-DFE over existing non-adaptive TSP-DFE would be even higher for larger order filter and constellation size. However, an extra adder and few XOR gates are involved in the transformation of LUT_I and LUT_II contents. As depicted in Fig. 4.10, the number of unfolding levels for the proposed architecture are also less over the existing non-adaptive TSP-DFE. Hence, it also provides hardware savings in the number of functional units. A comparison plot of hardware complexity and LUT size for the proposed and existing designs for 16-QAM with 14 unfolding factor is shown in Fig. 4.13. Clearly, the proposed technique has reduced hardware complexity and LUT size by several fold for large FB filter order over the existing R-DFE and TSP-DFE designs. This is particularly important for 5G communication systems due to huge amount of computation involved in TEQ. In contrast, the designs in [38, 39] are more complex for low FB filter orders and less complex for high FB filter orders. Thus, despite the fact that they have low hardware complexity, the proposed design involves relatively lower unfolding factor as compared to the design in [38]. Furthermore, these designs are based on the principle of virtual fixing of FB filter coefficients in every iteration, which makes the design of controller highly complex. In proposed adaptive TSP-DFE, the hardware complexity is slightly increased due to coefficient update unit, which comprises of an AL unit, a DLUT, two adders, a conditional adder, a barrel-shifter and a controller, as listed in Table 4.1. Interestingly, the overhead due to coefficient update unit is not significant; and can be even less as compared to the existing non-adaptive TSP-DFE.

4.4.1.2 Time Complexity

The critical path, throughput and latency of the proposed and existing designs are listed in Table 4.1. For ADFE system, the throughput is defined as the ratio of clock rate to the time required for processing an input sample. The maximum clock rate of a ADFE is limited by its iteration bound [74], while the time required to process an input sample depends on the order of FF and FB filters. However, in case of adaptive TSP-DFE, the processing time additionally depends on the number of clock cycles used in the inner loop and the update of DLUT contents. Thus, the number of clock cycles required for the proposed adaptive TSP-DFE design with 4-QAM can be estimated as $N_f + P + 2^{N_b-P-1} + 2$; where P and 2^{N_b-P-1} are the number of clock cycles used in the inner loop and DLUT respectively. It can be noted that the proposed design also requires an extra clock cycle to pre-speedup the architecture. Similarly, the number of clock cycles required for the update of DLUT in adaptive TSP-DFE and R-DFE designs would be 2^{N_b-P} and 2^{N_b} respectively. Hence, it is clear that the number of clock cycles to update the contents of DLUT for the proposed design is significantly reduced. In contrast, the number of clock cycles required for the design in [39] depends on the buffer size and slow-down factor. A comparison plot of critical paths between the proposed and existing designs for 16-QAM with various FB filter orders is shown Fig. 4.14. Clearly, the critical path of proposed design shows very less dependence on FB filter order like that of R-DFE and TSP-DFE designs, whereas the critical path of designs in [38,39] have linear dependence on FB filter order. As a result, a large parallelization factor is always needed for such designs. For better throughput comparison, a parallelization factor (p) is defined which is the ratio of critical paths of the proposed ADFE to that of design in [39], that is, $p = (((N_b+1)T_A + T_S)/\frac{1.5T_A+T_S}{N_b/4+1} + T_M \log_2 M)$. A throughput comparison plot between the proposed and existing designs for $L = 11$, $P = N_b/4$ and $N_b/2$ with 4-QAM and 16-QAM is shown in Fig. 4.15. It can be observed that the existing designs suffer from the throughput bottleneck due to more system processing time. However, this variation is reduced for the proposed design, which leads to higher throughput over the existing designs. For example, an 8^{th} order filter of proposed adaptive FB filter with $P = N_b/2$ for 4-QAM offers nearly 3 times higher throughput over the existing adaptive TSP-DFE, while it nearly provides 15 times higher throughput over the design in [39].

4.4.2 Error Performance

4.4.2.1 Convergence Performance

In order to determine the effectiveness of the proposed design over the existing designs, different simulations are carried out to estimate the performance in terms of convergence rate and mean steady-state error. The ITU Pedestrian B channel model is selected for generating fast fading [92], which comprises of four delay paths at 3 km/h. Different mean squared error (MSE) learning curves corresponding to each design for 16-QAM are plotted in Fig. 4.16(a) by ensemble averaging over 200 independent runs at SNR= 14 dB with speed-up factor $P = 4$. Compared with R-DFE, the proposed adaptive TSP-DFE has slightly more steady-state error due to an extra register in the pre-speedup of proposed architecture. In contrast, the convergence rate and steady-state error of design in [39] are proportionally degraded with slow down factor, with no control over it. In order to see the effect of first P coefficients of TSP-DFE on the convergence characteristics, a separate simulation is carried out at SNR= 28 dB. The comparison of convergence rate and steady-state error with various P values for both the proposed and existing TSP-DFEs is shown in Fig. 4.16(b). For the proposed design, it can be observed that as the speed-up factor P increases, the effect of non-OBC terms on the convergence rate and mean steady-state error are reduced. This is basically due to the minimization of error signal via parallel E-MUX which makes FF filter less intend towards cancelling of post-cursor ISI elements. Interestingly, the steady-state error values lie in the range of 4G standard specifications [93], and can be altered with SNR requirement.

4.4.2.2 BER Performance

In order to evaluate BER performance of the proposed design, extensive simulations are carried out based on the specifications provided in [87–89]. In the simulations, a system with 18 MHz bandwidth, 2 GHz carrier frequency, 2048-point IFFT/FFT, the length of 144 cyclic prefix is considered. Further, same channel model is adopted as described for MSE measurements. In addition, 16-QAM modulation scheme is adopted for transmitting data and training sequences. Several BER curves of the proposed adaptive TSP-DFE design for AWGN and Rayleigh fading channels are plotted in Fig. 4.17 by varying signal-to-noise ratio (SNR) from 0 to 30 dB with 8^{th} order filter and speed-up factors $P = 3, 4$. It is clear that the proposed design improves the BER performance in both AWGN and Rayleigh fading channels, when compared to the case of no equalizer. However, it is observed that the large value of P greatly impacts the SNR requirement for Rayleigh fading channel as compared to AWGN channel.

For example, if BER value is fixed at 10^{-3} and P is varied from 3 to 4, then nearly 2 dB more SNR is required for Rayleigh fading channel while this is just 1.2 dB for AWGN channel. Compared with R-DFE and TSP-DFE, the proposed design ensures better BER value since it eliminates the effect of first P coefficients of FB filter in parallel due to E-MUX in the stage-I, as shown in Fig. 4.11. In contrast, the design in [39] is associated with slow-down factor whose value greatly impacts the BER result. Additionally, this design also needs extra clocks for system initialization which further degrades the BER performance.

4.4.3 Implementation Results

4.4.3.1 ASIC Synthesis

The proposed unfolded design shown in Fig. 4.10 is realized using standard cells of TSMC 90 nm CMOS technology. It is found that the proposed design achieves 2.88 Gbps and 2.15 Gbps throughput for 4-QAM and 16-QAM respectively. In order to compare area and power consumptions of different designs, ASIC synthesis is carried out using the same technology node by Cadence RTL Compiler. The synthesis of the proposed and existing designs for 4-QAM and 16-QAM with various speed-up factors and FB filter orders are listed in Table 4.2. It can be noted that the proposed non-adaptive DFE and ADFE occupy relatively less area and consume less power compared to the existing designs. The savings in area and power are even more for larger FB filter order and constellation size. For example, an 8^{th} order FB filter of proposed non-adaptive TSP-DFE design for 4-QAM occupies nearly 1.92 times less area and consumes 1.95 times less power while these figures increase by several folds for 16-QAM over existing non-adaptive TSP-DFE. Similar type of advantages are observed for the proposed adaptive TSP-DFE. For example, an 8^{th} order FB filter of proposed adaptive TSP-DFE for 4-QAM occupies 2.22 times less area and consumes 1.71 times less power, whereas for 16-QAM, it occupies nearly 4.26 times less area and consumes nearly 3.51 times less power over the existing adaptive TSP-DFE. On the other hand, the proposed adaptive DFE design with 6^{th} order for 4-QAM occupies nearly 1.59 times less area and consumes 1.39 times less power, while for 16-QAM it occupies 2.14 times less area and consumes 3.43 times less power over the design in [39]. The area-delay product (ADP) is considered as a good figure of merit for comparison since it includes both cost and performance factors. The estimated ADP values of the proposed and existing designs are also listed in Table 4.2. Clearly, the proposed designs have better ADP figures over the existing TSP-DFE designs for all filter orders and constellation sizes. In contrast, when the proposed design is compared with the

design in [39], it is found that the ADP value depends on the constellation size. For example, an 8^{th} order proposed adaptive TSP-DFE for 4-QAM has 1.27 times more ADP and consumes nearly 2.13 times less power, whereas for 16-QAM, the proposed ADFE has nearly 1.05 times less ADP, however, it consumes 2.48 times less power over the design presented in [39].

4.4.3.2 FPGA Synthesis

The synthesis of proposed and existing designs are also carried out with a Xilinx ZYNQ XC7Z020-1CLG84C FPGA device for both 4-QAM and 16-QAM with different speedup factors and filter orders. The logic utilization are obtained in terms of slice LUTs (SLUT) and flip-flops (FF) by setting the system clock at 100 MHz, as listed in Table 4.2. From the synthesis results, it is clear that the proposed non-adaptive TSP-DFE for 4-QAM with 8^{th} order has almost 2.06 times less SLUT and 2.29 times less FF over the existing non-adaptive TSP-DFE. On the other hand, the proposed non-adaptive TSP-DFE for 4-QAM and 6^{th} order offers 1.46 times less SLUT and 1.58 times less FF over the design in [38]. Note that the reduction in SLUT and FF for the proposed non-adaptive TSP-DFE are even more in 16-QAM case. From Table 4.2, it is evident that the proposed adaptive TSP-DFE design for both 4-QAM and 16-QAM always involve significantly less number of SLUT and FF over the adaptive TSP-DFE. In contrast, the proposed adaptive TSP-DFE for 16-QAM with 6^{th} order nearly requires 2.53 times less SLUT and 2.88 times less FF; and with 8^{th} order it nearly requires 1.93 times less SLUT and 2.04 times less FF over the design reported in [39].

4.5 Conclusion

In this Chapter, a low-complexity energy efficient architecture of ADFE in compliance to 5G communication system has been presented. Extending the idea of [33] to the frequency selective channels, a new design for ADFE has been suggested. It is based on OBC scheme in which a separate LUT has been used to store the tentative decisions for the adaptation of FB filter coefficients. The effect of non-OBC terms due to pipelining are eliminated using parallel error multiplexer. The proposed design has reduced the hardware complexity and LUT size by several times, while achieving a throughput of nearly 2.15 Gbps for 16-QAM. It is also found that the reduction in hardware complexity becomes prominent, when the proposed architecture is unfolded. This is an important advantage, especially in 5G applications where the volume of computations would be large. Simulation results showed that the BER performance and convergence characteristics of the proposed ADFE are better than the existing

designs, with slight increase in SNR requirement for Rayleigh fading channel. ASIC and FPGA synthesis results showed significant improvements in the area, power and logic utilization for the proposed design as compared to the best existing design.

Table 4.1: Comparison of Computational Complexities between DFE Designs for N_f^{th} Order FF filter and N_b^{th} Order FB filter with Constellation Size M

Design	**Type**	**Adders**	**Multiplexers**	**Registers**	**Critical Path/Throughput**	**Latency**	**LUT/MULT**
R-DFE† [29]	non-A	$(M)^{N_b}$	$(M)^{N_b}-1$	$N_b\log_2 M$	$T_M\log_2 M$	0	0
	A	$2(M)^{N_b}+1$	$2(M)^{N_b}-2$ $+(M-1)N_b$	$N_b\log_2 M$	$1/[k_0 T_M \log_2 M]$	0	2^{N_b+M}
PP-DFE†⋆ [33]	non-A	2^{P+M} $+(N_b-P+1)$	$2^{P+M}-1$	$(N_b+P)\log_2 M$	$T_{FA}\sim T_{MUL}$	P	$(N_b-P+1)\log_2 M$
TSP-DFE† [33]	non-A	$2(M)^{N_b/2}$	$2(M)^{N_b/2}-2$	$3N_b\log_2 M/2$	$\frac{T_A+T_S}{N_b/2+1}+T_M\log_2 M$	$N_b/2$	$2^{(N_b+M)/2+1}$
	A	$3(M)^{N_b/2}+2$	$3(M)^{N_b/2}$ $+(M-1)N_b/2$	$(3N_b\log_2 M)/2$	$1/[k_1(\frac{2T_A+T_S}{N/2+1}+\log_2 M T_M)]$	$N_b/2$	$3(2)^{(N_b+M)/2}$
DFE† [34]	A	$(M)^{N_b}$	$(M)^{N_b}$	$(M)^{N_b}+N_b$	$J\log_2(N_b+1)T_M$	$\beta+\gamma$	0
DFE† [35]	non-A	$(M)^{N_b}$	$\sim M^{N_b}N_b$	${N_b}^2+(M)^{N_b}$	$\log_2(N_b+1)T_M/(P+\log_2 N-1)$	$R-1$	0
DFE⋆ [38]	non-A	$\sim[(N_b+1)\log_2 M]^2$	–	$N_b\log_2 M$	$J'[(N_b+1)T_A+T_S]$	γ	$\sim(N_b\log_2 M)^2$
DFE⋆ [39]	A	$\sim 2[(N_b\log_2 M)^2+1]$	–	$N_b\log_2 M$	$1/[k_2((N+1)T_A+T_S)]$	$\beta+\gamma$	$\sim 2(N_b\log_2 M)^2$
DFE† [40]	non-A	$(N_b\log_2 M)^2/2$	$(M-1){N_b}^2/2$	$(N_b\log_2 M)^3/6$	$N_bT_A+T_S+T_M\log_2 M$	$Q-1$	–
Proposed†	non-A	$2(M)^{N_b/2-1}+1$	$2(M)^{N_b/2-1}$	$(3N_b/2+2)\log_2 M$	$\frac{T_A+0.5T_S}{N_b/4+1}+T_M\log_2 M$	$N_b/2+1$	$2^{(N_b+M)/2}$
	A	$3(M)^{N_b/2-1}+3$	$3(M)^{N_b/2-1}-1$ $+(M-1)N_b/2$	$(3N_b/2+2)\log_2 M$	$1/[k_3(\frac{1.5T_a+0.5T_s}{N_b/4+1}+T_M\log_2 M)]$	$N_b/2+1$	$3(2)^{(N_b+M)/2-1}$

† : Multiplierless architecture; ⋆ : Multiplier-based architecture; A: Adaptive, P: Speed-up factor, J: Unfolding (or parallelization) factor, R: Incremental factor, Q: Iteration factor; β: Slow-down factor; γ: System initialization clock cycles, for simplicity $\gamma=0$; K_M : Input buffer size; $K_M=2^M K$ with $K=N_f+N_b+1$, Throughput = Clock rate/Processing time per sample, Clock rate = 1/Critical path, $k_0=N_f+\alpha_0 2^{N_b}+1$; $k_1=N_f+P+\alpha_1 2^{N_b-P}+1$; $k_3=N_f+P+\alpha_2 2^{N_b-P-1}+2$; with $\alpha_0=2^M$, $\alpha_1=2^{M/2}$ and $\alpha_2=2^{M/2-1}$; $k_2=J'K_M$; with $J'=pJ$ and p denotes extra parallelization involved in DFE⋆ [38] and DFE⋆ [39]. T_M T_A, T_{MUL} and T_{FA} are computational delays of 2-to-1 multiplexer, adder, multiplier and carry-save full-adder respectively. In addition to the listed hardware complexities, the proposed non-adaptive DFE and ADFE require 1-bit $2((M-1)N_b/2-1)$ and $3((M-1)N_b/2-1)$ XOR gates respectively, while proposed ADFE involves a conditional adder and a barrel-shifter to update the DLUT contents and FB filter coefficients respectively. Note that the designs in [34, 39] are based on the same principle and involves large sized buffers. The above listed hardware complexities of proposed and existing designs are estimated for I-channel of M-QAM system, which is to be multiplied by a factor of two to determine the overall complexity.

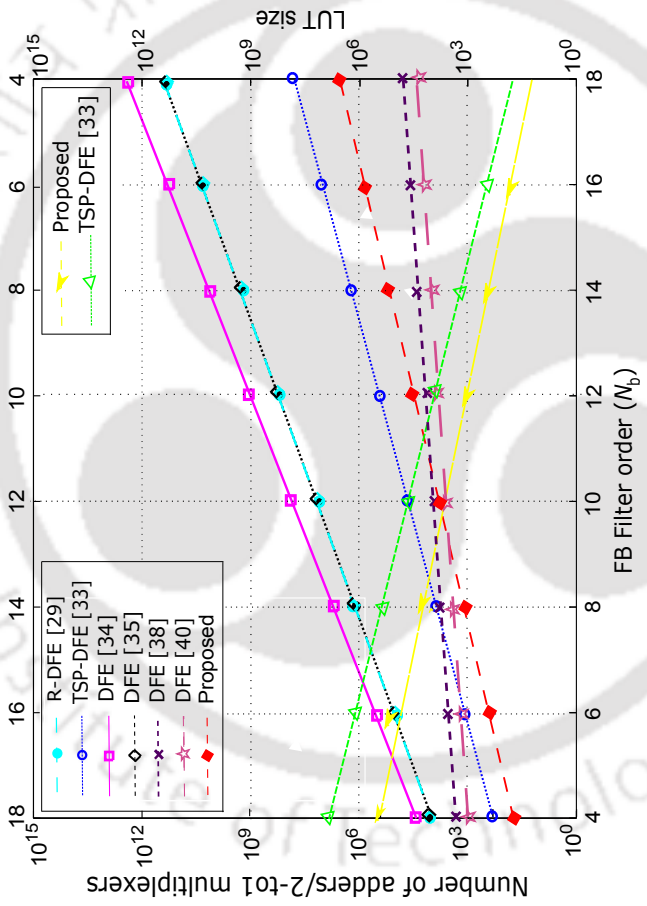


Fig. 4.13: Comparison of hardware complexity and LUT size between proposed and existing design for 16-QAM and 14 unfolding factor, assuming complexity of multiplier $\approx 6\times$ complexity of adder [38].

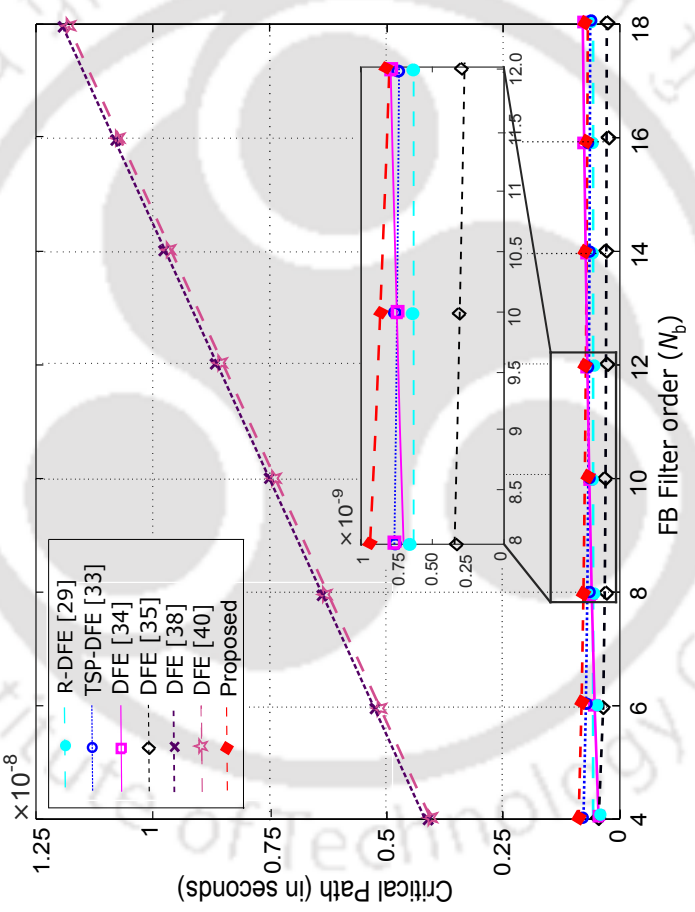


Fig. 4.14: Comparison of critical paths between proposed and existing designs for 16-QAM.

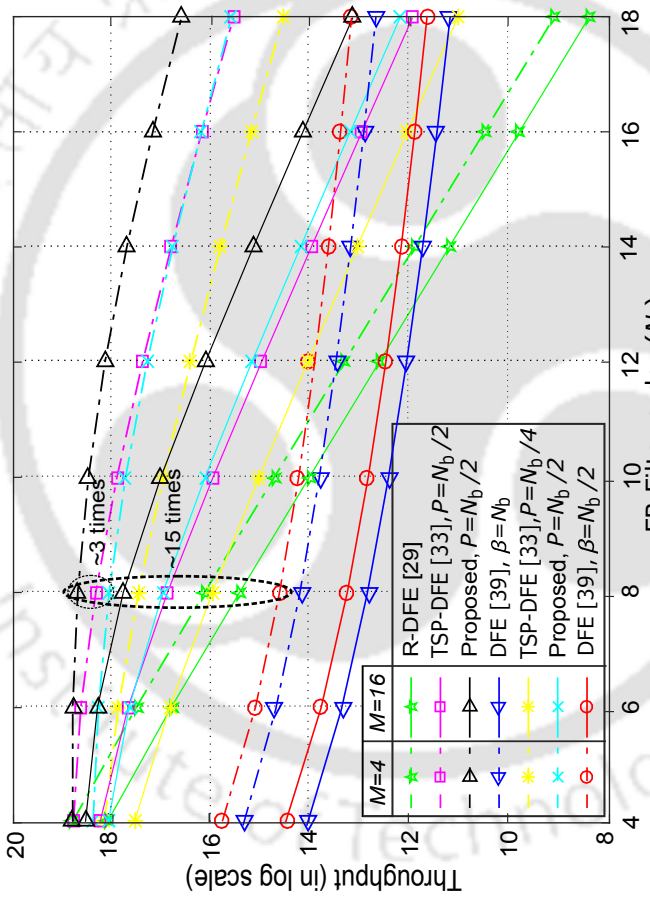


Fig. 4.15: Comparison of throughput for the presented and existing designs with 11^{th} order FF filter, speed-up factors $N_b/2, N_b/4$ and 4, 16-QAM.

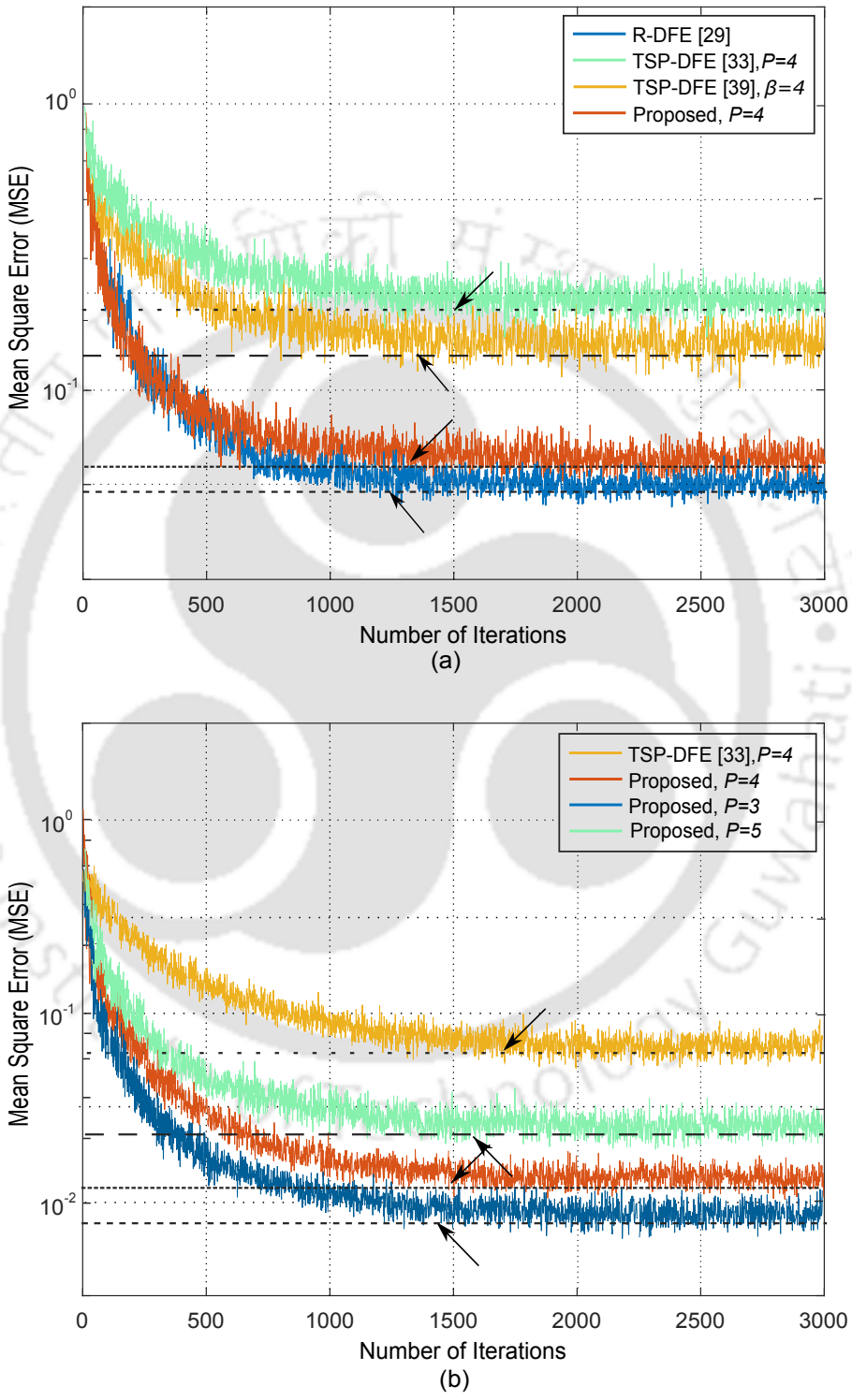


Fig. 4.16: Comparison of MSE learning curves for the presented and existing designs with 11^{th} order FF filter, 8^{th} order FB filter, step-size 0.0008 and 16-QAM at (a) SNR= 14 dB (b) SNR= 28 dB.

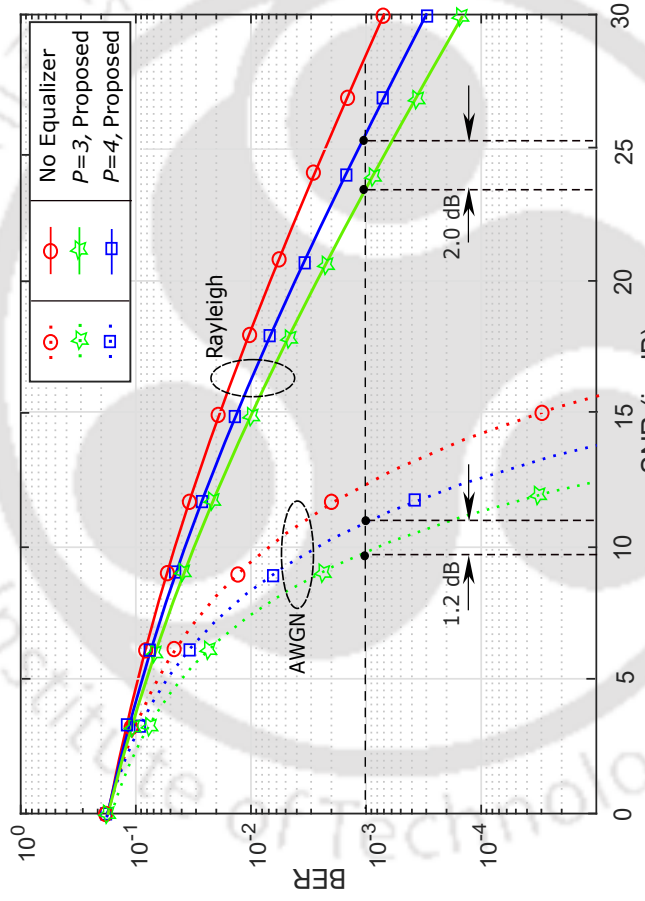


Fig. 4.17: BER curves for the presented design in AWGN and Rayleigh fading channels with 8^{th} order FB filter and speed-up factors 3, 4.

Table 4.2: Performance Comparison of Different Designs with ASIC Synthesis using TSMC 90 nm CMOS Library and FPGA Synthesis on Xilinx ZYNQ XC7Z020-1CLG84C for Various U, N_b and M

Design	**Type**	U, N_b	**ASIC Synthesis using TSMC 90 nm CMOS Library**								**FPGA Synthesis on Xilinx XC7Z020-1CLG84C at 100 MHz**			
			$M = 4$				$M = 16$				$M = 4$		$M = 16$	
			Area (μm^2)	**Power** (mW)	**TS** (ns)	**ADP** (μm$^2\times$ns)	**Area** (μm^2)	**Power** (mW)	**TS** (ns)	**ADP** (μm$^2\times$ns)	**SLUT**	**FF**	**SLUT**	**FF**
R-DFE† [29]	non-A	0, 6	18121	2.14	1.32	23919	1218612	45.67	0.68	828656	1821	1039	124810	67921
	A	0, 6	46782	3.84	0.58	27134	4811317	112.86	0.32	1539621	3132	2460	205781	161786
PP-DFE†* [33]	non-A	2, 6	2832	0.14	1.41	3993	11675	0.48	0.98	11441	238	169	962	548
		2,8	3437	0.23	1.23	4228	13752	0.59	0.85	11689	296	198	1262	840
TSP-DFE† [33]	non-A	3, 6	4899	0.45	2.29	11219	42583	3.82	1.02	43435	428	310	3431	2562
		4, 8	9832	0.86	2.17	21335	161646	11.23	0.94	151947	849	692	13861	11225
	A	3, 6	8725	0.67	2.10	17378	72893	4.98	1.05	76538	721	610	43562	34278
		4, 8	16214	1.08	1.97	31942	266190	15.88	0.88	234247	1437	1132	23566	18452
DFE† [34]	A	2, 4	10810	2.18	1.63	17620	533832	42.15	0.70	373682	1120	548	48891	18560
DFE† [38]	non-A	0, 6	7782	0.45	0.69	5370	26350	4.41	0.78	20553	363	278	1782	1244
DFE* [39]	A	3, 6	11544	0.88	0.48	5541	42481	5.57	0.56	23789	782	667	4186	3188
		4, 8	17838	1.45	0.63	11238	74380	11.26	0.87	64710	1312	1107	8267	6387
DFE* [40]	non-A	0, 6	6731	0.20	0.78	5250	29576	4.22	0.93	27506	318	229	1638	952
Proposed	non-A	3, 6	2952	0.28	2.31	6819	14248	1.04	1.17	16670	248	176	924	639
		4, 8	5123	0.44	2.19	11219	45961	1.82	1.09	50097	412	301	3568	2234
	A	3, 6	4691	0.42	2.09	9804	19789	1.62	1.07	21174	382	279	1652	1108
		4, 8	7236	0.68	1.98	14327	62538	4.53	0.99	61912	521	388	4272	3127

U denotes any of P, R, Q, or β for comparison purpose depending on the design. TS: timing slack, ADP: area-delay product, SLUT: slice LUTs, FF: flip-flops. Note that synthesis of all designs are performed with unfolding (or parallelization) factor as 1. The synthesis of design in [35] is skipped since it can be derived from R-DFE [29] for large parallelization factor.

5

Low Complexity Pipelined Block LMS Adaptive Filter

Contents

5.1 Introduction

Adaptive noise cancellation (ANC) is a well-known technique introduced by Widrow [94]. It is employed in headphones to mitigate the effects of noise. Depending on the availability of reference channel, the headphone may consists of two or more microphones to characterize the correlated samples. The conceptual diagram of ANC used in in-ear headphones is shown in Fig. 5.1. As discussed in Chapter 1, the ADF is the basic building block in noise canceller to estimate the noise for cancellation. The output of ADF is subtracted from the microphone output (signal+noise) to adjust its coefficients based on the adaptation algorithm. Such applications demand higher order ADF, power intensive hardware, and significant time to estimate and synthesize anti-noise signals in order to remove the noise signals in real time. ANC techniques can attenuate low-frequency noise in headphones [95–100], ducts [101, 102] etc. FxLMS algorithm is generally used for the adaptation of filter coefficients. This algorithm exhibits slower convergence and the noise cancelling performance may be affected by the causality unbalance [103]. The feedback adaptive noise canceller algorithm and its variants have been suggested for headphones to remove the narrowband noise at low frequencies [96–98]. Later, an efficient approach has been suggested in [99] for effective attenuation of the broadband noise. To achieve low-cost noise canceller headphones, Chang et al. in [104] suggested microcontroller as the core component. But, the problem of such system is that as the bandwidth of noise increases, the noise cancelling performance starts to deteriorate.

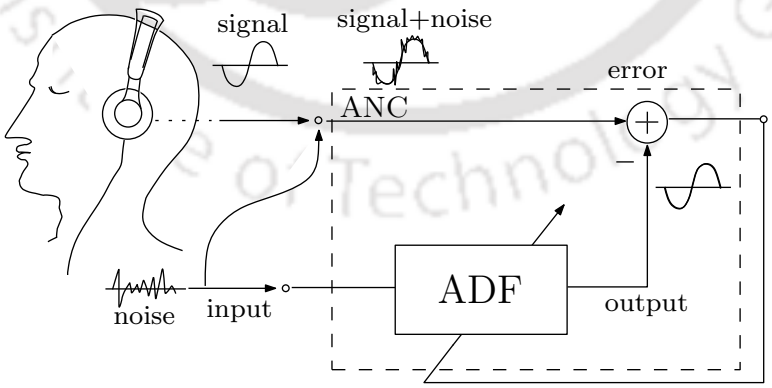


Fig. 5.1: Conceptual diagram of adaptive noise cancellation for in-ear headphones.

BLMS algorithm is a computationally fast and efficient approach for realizing the FIR ADF of higher orders, as discussed in Chapter 1. It processes a block of inputs and computes a block of filter outputs unlike conventional LMS algorithm where sample-by-sample processing is performed.

Over the years, many works have been reported to efficiently realize BLMS filter. For example, FFT based frequency-domain block FxLMS is proposed in [43]. DA based BLMS ADF architecture and its corresponding FPGA based implementation is presented in [44]. Mohanty et al. in [47] suggested an energy-efficient time-domain architecture for BLMS ADF. It is based on new DA formulation to compute the filter output and coefficient increment terms. But, the complexity of LUTs and clock cycles to update them grows exponentially with the input block length.

From the above discussion, it is clear that achieving low area/power and better noise cancellation in modern portable devices still remains a challenging task. In this chapter, a dedicated noise canceller circuit implementation based on the BLMS algorithm using OBC-DA for in-ear headphones is investigated. The BLMS algorithm is optimized and a low-complexity architecture is proposed for realizing the required convolution operations. To save the hardware resources and clock cycles, splitting of LUT into two LUTs is adopted for storing the partial OBC combinations of input samples which are shared to compute the filter output and coefficient increment terms. Besides, an external register is employed for pipelined architectural design to update its content along with error computation in each block. By doing so, there will not be any problem of non-OBC terms like those encountered in Chapters 2 and 4. Altogether, the proposed design outperforms in terms of noise reduction and area/power performances.

5.2 Mathematical Formulation

The coefficient update equation for BLMS algorithm as indicated in (1.21) can also be expressed as

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{\Delta w}(n) \tag{5.1}$$

where $\mathbf{\Delta w}(n) = [\Delta w_0(n), \Delta w_1(n), ..., \Delta w_{N-1}(n)]$ denotes the vector for coefficient increment terms, and is defined as

$$\mathbf{\Delta w}(n) = \mu_B \mathbf{X}(n)\mathbf{e}(n) \tag{5.2}$$

where $\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{y}(n)$ is the error vector, $\mathbf{d}(n)$ is the desired response vector and μ_B is the step-size. By separately defining $\mathbf{\Delta w}(n)$ has an advantage since it contains the input matrix $\mathbf{X}(n)$ which is also a part of filtering operation as given in (1.19). For pipelined BLMS algorithm as listed in Table 1.2,

the coefficient update equation can be re-written as

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{\Delta w}(n-m) \tag{5.3}$$

where m is block adaptation delay, $\mathbf{\Delta w}(n-m) = [\Delta w_0(n-m), \Delta w_1(n-m), ..., \Delta w_{N-1}(n-m)]$ denotes the vector for delayed coefficient increment terms, and is defined as

$$\mathbf{\Delta w}(n-m) = \mu_B \mathbf{X}(n-m)\mathbf{e}(n-m) \tag{5.4}$$

where $\mathbf{e}(n-m) = \mathbf{d}(n-m) - \mathbf{y}(n-m)$ is the delayed error vector, $\mathbf{d}(n-m)$ is the delayed desired response vector.

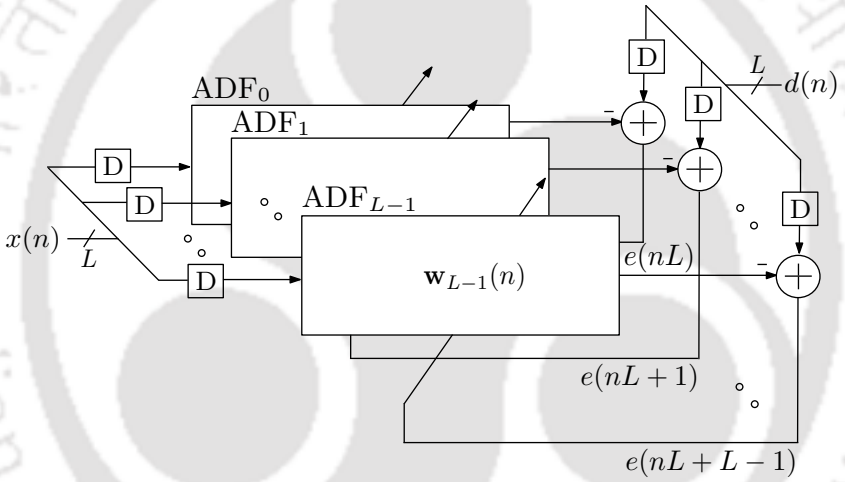


Fig. 5.2: Block diagram of pipelined ADF based on BLMS algorithm with one adaptation delay.

The value of block adaptation delay is taken to be one (i.e., $m = 1$) whose locations are shown in Fig. 5.2. Note that the location of block adaptation delays are external to the system for storing the recent input samples and desired response. This would not produce any non-OBC terms in the system, so there is no requirement of correcting them in order to improve the convergence performance. Therefore, both the input matrix $\mathbf{X}(n)$ and coefficient increment vector $\mathbf{\Delta w}(n)$ are delayed by unit time. Further, the input matrix $\mathbf{X}(n)$ of order $L \times N$ can be splitted into M square matrices $\mathbf{X}_j(n)$ of order $L \times L$ such that $N = L \times M$. Similarly, the coefficient vector $\mathbf{w}(n)$ and coefficient increment vector $\mathbf{\Delta w}(n)$ are to be splitted into M small vectors $\mathbf{w}j(n)$ and $\mathbf{\Delta w}_j(n)$ of size $L \times 1$ respectively. Note the unit delayed square input matrices $\mathbf{X}_j(n)$ can also be expressed as $\mathbf{D}_j\{\mathbf{X}_j(n)\}$, where $\mathbf{D}_j$ are the delay operators storing the most recent sample in each block. For the sake of simplicity, the notation of $\mathbf{D}_j$ is dropped for the derivation purpose. The unit delayed input matrix $\mathbf{X}_j(n)$, coefficient

vector $\mathbf{w}_j(n)$ and coefficient increment vector $\mathbf{\Delta w}_j(n)$ are, respectively, defined as

$$\mathbf{X}_j(n) = \begin{bmatrix} x(j'L) & x(j'L-1) & . & . & x(j'L-N+1) \\ x(j'L-1) & x(j'L-2) & . & . & x(j'L-N) \\ . & . & . & . & . \\ . & . & . & . & . \\ x(j'L-L+1) & x(j'L-L) & . & . & x(j'L-L-N+2) \end{bmatrix}^T \tag{5.5}$$

$$\mathbf{w}_j(n) = \begin{bmatrix} w_{jL}(n) & w_{jL+1}(n) & . & . & w_{jL+L-1}(n) \end{bmatrix}^T \tag{5.6}$$

$$\mathbf{\Delta w}_j(n) = \begin{bmatrix} \Delta w_{jL}(n) & \Delta w_{jL+1}(n) & . & . & \Delta w_{jL+L-1}(n) \end{bmatrix}^T \tag{5.7}$$

where $j' = n - j - 1$. Based on the above formulation, (1.19), (5.1) and (5.4) can be then computed as

$$\mathbf{y}(n) = \sum_{j=0}^{M-1} \mathbf{X}_j^T(n)\mathbf{w}_j(n) \tag{5.8}$$

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \mathbf{\Delta w}_j(n) \tag{5.9}$$

$$\mathbf{\Delta w}_j(n) = \mu_B \mathbf{X}_j(n)\mathbf{e}(n) \tag{5.10}$$

where $0 \leq j \leq M-1$. For clarity, the expression of each filter output can be given as

$$y(nL-i) = \sum_{j=0}^{M-1} u_{i,j}(n) \quad \text{and} \quad u_{i,j}(n) = \mathbf{x}_{i,j}^T(n)\mathbf{w}_j(n) \tag{5.11}$$

where $u_{i,j}(n)$ indicates the inner-product of input vector $\mathbf{x}_{i,j}(n)$ and coefficient vector $\mathbf{w}_j(n)$. Note that $\mathbf{x}_{i,j}(n)$ is the input-vector corresponding to $(i+1)$-row of input matrix $\mathbf{X}_j(n)$. Similarly, each coefficient increment term can be expressed as

$$\Delta w_{jL+i}(n) = \mu_B v_{i,j}(n) \quad \text{and} \quad v_{i,j}(n) = \mathbf{x}_{i,j}^T(n)\mathbf{e}(n) \tag{5.12}$$

where $v_{i,j}(n)$ represents the inner-product of input vector $\mathbf{x}_{i,j}(n)$ and error vector $\mathbf{e}(n)$.

Let us denote $(r+1)^{\text{th}}$ component of vector $\mathbf{w}_j(n)$ and $\mathbf{e}(n)$ as $w_{r,j}(n)$ and $e_r(n)$ respectively, and they are assumed to be W-bit wide. By representing each $w_{r,j}(n)$ in two's complement form, we get

$$w_{r,j}(n) = -\{w_{r,j}(n)\}_0 + \sum_{k=1}^{W-1} \{w_{r,j}(n)\}_k 2^{-k} \tag{5.13}$$

Alternatively, $w_{r,j}(n)$ can also be expressed as: $w_{r,j}(n) = \frac{1}{2}[w_{r,j}(n) - (-w_{r,j}(n))] = \frac{1}{2}[w_{r,j}(n) - \overline{w}_{r,j}(n)]$, where $\overline{w}_{r,j}(n)$ is two's complement of $w_{r,j}(n)$. This representation reduces the LUT size. The above simplification for $w_{r,j}(n)$ when combined with (5.13) leads to

$$w_{r,j}(n) = \frac{1}{2}\Bigg[- (\{w_{r,j}(n)\}_0 - \{\overline{w}_{r,j}(n)\}_0) + \sum_{k=1}^{W-1} (\{w_{r,j}(n)\}_k - \{\overline{w}_{r,j}(n)\}_k) 2^{-k} - 2^{-(W-1)} \Bigg] \tag{5.14}$$

Assuming $\{\delta w_{r,j}(n)\}_k = \{w_{r,j}(n)\}_k - \{\overline{w}_{r,j}(n)\}_k$, substituting (5.14) in (5.11) would lead $u_{i,j}(n)$ to

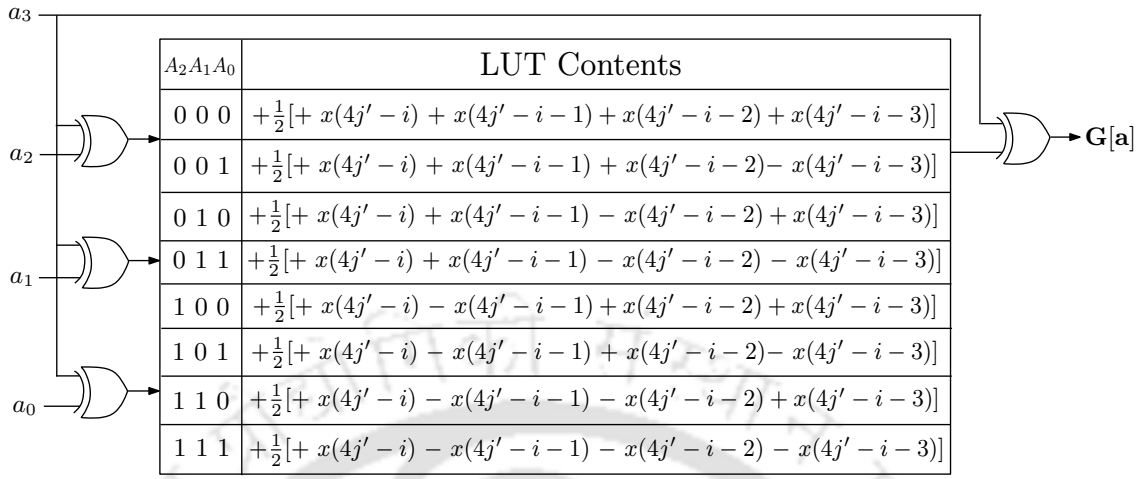
$$u_{i,j}(n) = \sum_{k=0}^{W-1} \mathbf{\Phi}\Bigg[\sum_{r=0}^{L-1} \frac{1}{2} x(j'L - i - r) \{\delta w_{r,j}(n)\}_k \Bigg]_k 2^{-k} - \frac{1}{2}\Bigg[\sum_{r=0}^{L-1} x(j'L - i - r) \Bigg] 2^{-(W-1)} \tag{5.15}$$

where $\mathbf{\Phi}[\bullet]_k = -1$ for $k = W-1$, otherwise $\mathbf{\Phi}[\bullet]_k = 1$ to indicate the SA operation corresponding to MSB of filter coefficients, and $-\frac{1}{2}\sum_{r=0}^{L-1} x(j'L - i - r)$ is loaded into SA corresponding to LSB of filter coefficients. Note that the terms in the inner sum of (5.15) represents the filter partial products in terms of input vector $\boldsymbol{\chi}_{i,j}(n) = [x(j'L - i), x(j'L - i - 1), ..., x(j'L - i - L + 1)]$ which are to be stored in LUT, whose address bits are the difference of filter coefficients bit-slices and their two's complement, that is, $\{\delta w_{r,j}(n)\}_k$ for $0 \le r \le L-1$ and $0 \le j \le M-1$, as per (5.15). In general, there would be 2^L OBC combinations of input samples to be stored in LUT, however, there exists mirror symmetry in these combinations. As a result, LUT size can be reduced to half, the remaining half combinations are taken care by external XOR gates. For example, when the filter block-length $L = 4$, the contents to be stored in LUT are shown in Fig. 5.3. It consists of four XOR gates and a 2-to-1 multiplexer to decode the remaining OBC combinations of input samples. In simplest form, the computation of (5.15) could be expressed in terms of LUT read and SA operations, according to

$$u_{i,j}(n) = \sum_{k=0}^{W-1} \mathbf{G}[\{\boldsymbol{\delta}\mathbf{w}_j(n)\}_k] 2^{-k} - \mathbf{G}[0] 2^{-(W-1)} \tag{5.16}$$

where $\mathbf{G}[\bullet]$ is the LUT read-operation, and its argument $\{\mathbf{w}_j(n)\}_k$ represents the address vector for $0 \le k \le W-1$, that is, $\{\boldsymbol{\delta}\mathbf{w}_j(n)\}_k = [\{\delta w_{0,j}(n)\}_k, \{\delta w_{1,j}(n)\}_k, ..., \{\delta w_{L-1,j}(n)\}_k]$ with each $\{\delta w_{r,j}(n)\}_k \in [-1, 1]$ and $\mathbf{G}[0]$ is a matrix whose components are defined as the LUTs content present at 0^{th} address location. Specifically, $\mathbf{G}[0] = [g_{0,j}(n)[0], g_{1,j}(n)[0], ..., g_{L-1,j}(n)[0]]$ for all $0 \le j \le M-1$, where $g_{i,j}(n)[0] = -\frac{1}{2}\sum_{r=0}^{L-1} x(j'L - i - r)$.

In a similar manner, the expression for coefficient increment terms (5.12) can be derived. Mathe-



$A_2A_1A_0$	LUT Contents
0 0 0	$+\frac{1}{2}[+x(4j'-i)+x(4j'-i-1)+x(4j'-i-2)+x(4j'-i-3)]$
0 0 1	$+\frac{1}{2}[+x(4j'-i)+x(4j'-i-1)+x(4j'-i-2)-x(4j'-i-3)]$
0 1 0	$+\frac{1}{2}[+x(4j'-i)+x(4j'-i-1)-x(4j'-i-2)+x(4j'-i-3)]$
0 1 1	$+\frac{1}{2}[+x(4j'-i)+x(4j'-i-1)-x(4j'-i-2)-x(4j'-i-3)]$
1 0 0	$+\frac{1}{2}[+x(4j'-i)-x(4j'-i-1)+x(4j'-i-2)+x(4j'-i-3)]$
1 0 1	$+\frac{1}{2}[+x(4j'-i)-x(4j'-i-1)+x(4j'-i-2)-x(4j'-i-3)]$
1 1 0	$+\frac{1}{2}[+x(4j'-i)-x(4j'-i-1)-x(4j'-i-2)+x(4j'-i-3)]$
1 1 1	$+\frac{1}{2}[+x(4j'-i)-x(4j'-i-1)-x(4j'-i-2)-x(4j'-i-3)]$

Fig. 5.3: LUT contents for BLMS filter based on OBC-DA with block-length of four, where $0 \leq i \leq L-1$, $0 \leq j \leq M-1$ and $j' = n-j-1$.

matically, it can be given as

$$v_{i,j}(n) = \sum_{k=0}^{W-1} \mathbf{G}[\{\boldsymbol{\delta}\mathbf{e}(n)\}_k]2^{-k} - \mathbf{G}[0]2^{-(W-1)} \tag{5.17}$$

where the address vector becomes the difference of bit-slices of error signals and their two's complement, that is, $\{\boldsymbol{\delta}\mathbf{e}(n)\}_k = [\{\delta e(nL)\}_k, \{\delta e(nL+1)\}_k, ..., \{\delta e(nL+L-1)\}_k]$ with each $\{\delta e(nL+r)\}_k \in [-1, 1]$. In this case, the address vector for LUT becomes $\{\boldsymbol{\delta}\mathbf{e}(n)\}_k$. Interestingly, (5.16) and (5.17) are in similar form except that the address vectors for LUTs are different. Suppose for any address vector $\mathbf{a} = [a_0, a_1, ..., a_{L-1}]$ with $a_r \in [-1, 1]$ for $0 \leq r \leq L-1$, the contents of LUT can be written as

$$\mathbf{G}[\mathbf{a}] = \sum_{r=0}^{L-1} \frac{1}{2} x(j'L - i - r)a_r \tag{5.18}$$

where $\mathbf{G}[\mathbf{a}]$ indicate matrix of LUTs arranged in L-rows and M-columns. As discussed previously, it is required to store only lower half OBC combinations in LUT whereas upper half OBC combinations can be obtained using external XOR gates. Mathematically, it can be written as

$$\mathbf{G}^p[\mathbf{a}] = (-1)^p \left[\frac{1}{2}x(j'L - i) + \sum_{r=0}^{L-2} \frac{1}{2}x(j'L - i - r + 1)a_{r-1}\right] \tag{5.19}$$

where $\mathbf{G}^p[\mathbf{a}]$ denote the lower and upper components of LUT for $p = 0$ and $p = 1$ respectively. In addition, it can also be seen from Fig. 5.3 that LUT ($\mathbf{G}^p[\mathbf{a}]$) can be splitted into even and odd address

locations, according to

$$\mathbf{G}^{pq}[\mathbf{a}] = (-1)^p \left[\frac{1}{2} x(j'L - i) + \sum_{r=0}^{L-3} \frac{1}{2} x(j'L - i - r + 2) a_{r-2} + \frac{(-1)^q}{2} x(j'L - i - L + 1) \right] \tag{5.20}$$

where $\mathbf{G}^{pq}(\mathbf{b})$ denotes the even and odd components of LUT for $q = 0$ and $q = 1$ respectively. It can be noted from (5.20) that the contents present at lower half even address location for $p = 0, q = 0$ are two's complement of the contents present at upper half odd address location for $p = 1, q = 1$. Similarly, the contents present at lower half odd address location for $p = 0, q = 1$ are two's complement of the contents present at upper half even address location for $p = 1, q = 0$. Mathematically, the above observation can be written as

$$\mathbf{G}^{00}[\mathbf{a}] = -\mathbf{G}^{11}[\mathbf{a}] \quad \text{and} \quad \mathbf{G}^{01}[\mathbf{a}] = -\mathbf{G}^{10}[\mathbf{a}] \tag{5.21}$$

Again, it is required to store half input OBC combinations into two separate LUTs based on even and odd address locations: LUT_0 ($\mathbf{G}^{00}[\mathbf{a}]$) and LUT_1 ($\mathbf{G}^{01}[\mathbf{a}]$) and their corresponding halves can be obtained by taking two's complement, in accordance with (5.21).

5.3 Proposed Scheme

In this section, the strategy for filter block update based on the mathematical formulation as discussed in Section 5.2 is presented. Next, the architectural details of the various operations involved in the proposed scheme such as LUT update, filter output, error computation and filter block update are discussed.

5.3.1 Filter Block Update Strategy

Consider a BLMS adaptive filter which receive the input-matrix $\mathbf{X}(n)$ with coefficient-vector $\mathbf{w}(n)$ and coefficient increment-vector $\mathbf{\Delta w}(n)$. As discussed, $\mathbf{X}(n)$, $\mathbf{w}(n)$ and $\mathbf{\Delta w}(n)$ are respectively decomposed into smaller components $\mathbf{X}_j(n)$, $\mathbf{w}_j(n)$ and $\mathbf{\Delta w}_j(n)$ for $j = 0, 1, 2, ..., M-1$. It can be noted from (5.11) and (5.12) that coefficient and error vectors are multiplied independently with smaller input-matrices. For the sake of clarity, an example of filter order $N = 6$ and block-length $L = 2$ is considered. This corresponds to two rows ($L = 2$) and three columns ($M = 3$), with indices denoted by $i = 0$ and 1 and $j = 0, 1$ and 2, as shown in Fig. 5.4. In n^{th} iteration, the filter receives two input samples $[x(2n), x(2n-1)]$ and produces the output samples $[y(2n), y(2n-1)]$. In $(n+1)^{\text{th}}$ iteration, two new input samples $[x(2n+2), x(2n+1)]$ will be received by the filter to compute L new

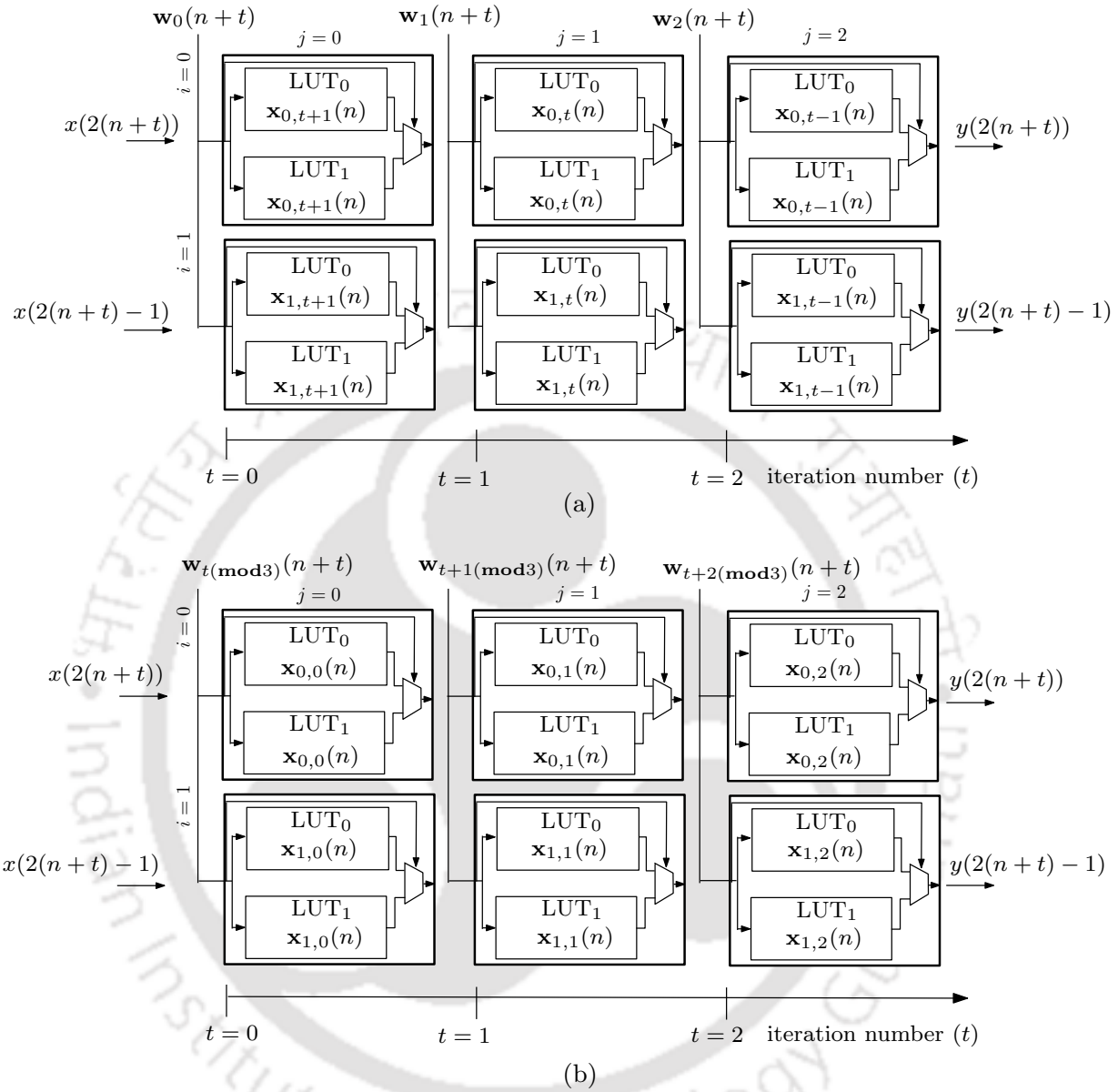


Fig. 5.4: Block update scheme for the presented DA based BLMS ADF (only LUTs are shown with subscripts 0 and 1 indicate even and odd components respectively) of 6^{th} order filter and block-length of two. (a) Update via right-shifting of input vectors (b) Update via circular left-shifting of splitted coefficient vectors.

inner-products. Similarly, in every iteration, two new samples will be received by the filter. But, it is required to discard two oldest input samples from the last column ($j = 2$). Let t be the iteration number, then shifting of input-vector across the columns in i^{th}-row for any iteration $(n+t)^{\text{th}}$ can be written as

$$\mathbf{x}_{i,(t-j)}(n) \leftarrow \mathbf{x}_{i,(t+1-j)}(n) \quad \forall t, j \tag{5.22}$$

Notably, the column-index of coefficient vectors does not change with t to compute the inner-products of a particular column, as shown in Fig. 5.4(a). This way of updating the filter block requires physical shifting of input-vectors across rows via LUTs. It requires more time and consumes significant power. The other way to update filter blocks is circular left-shifting of coefficient vectors, that is, the coefficient vectors are left-shifted block-by-block in every iteration. For example, the splitted coefficient vector $\mathbf{w}_j(n)$ for $0 \leq j \leq M-1$ is left-shifted circularly by changing the column-index j as a function of iteration number t, that is, $(j+t)(\text{mod } M)$. The splitted coefficient vector $\mathbf{w}_j(n)$ in n^{th} iteration are aligned with column index j, that is, $\mathbf{w}_0(n), \mathbf{w}_1(n), \mathbf{w}_2(n)$ and the pattern becomes $\mathbf{w}_1(n), \mathbf{w}_2(n), \mathbf{w}_0(n)$ in $(n+1)^{\text{th}}$ iteration, as shown in Fig. 5.4(b). From the pattern, it can be observed that coefficient vectors obtained for every next iteration are nothing but circular left-shifted versions of previous iteration, for example, $\mathbf{w}_2(n+1) \leftarrow \mathbf{w}_0(n)$, $\mathbf{w}_0(n+1) \leftarrow \mathbf{w}_1(n)$ and $\mathbf{w}_1(n+1) \leftarrow \mathbf{w}_2(n)$. In $(n+t)^{\text{th}}$ iteration, the coefficient vectors for M-columns can be obtained as

$$\mathbf{w}_{(t+j)(\text{mod } M)}(n+t) \leftarrow \mathbf{w}_{(t+j-1)(\text{mod } M)}(n+t-1) \quad \forall t, j \tag{5.23}$$

From the discussion, it is clear that the second approach to update the filter block is better, as it does not require the shifting of input-vectors via LUTs, thereby, both time and power can be saved.

5.3.2 Proposed Architecture

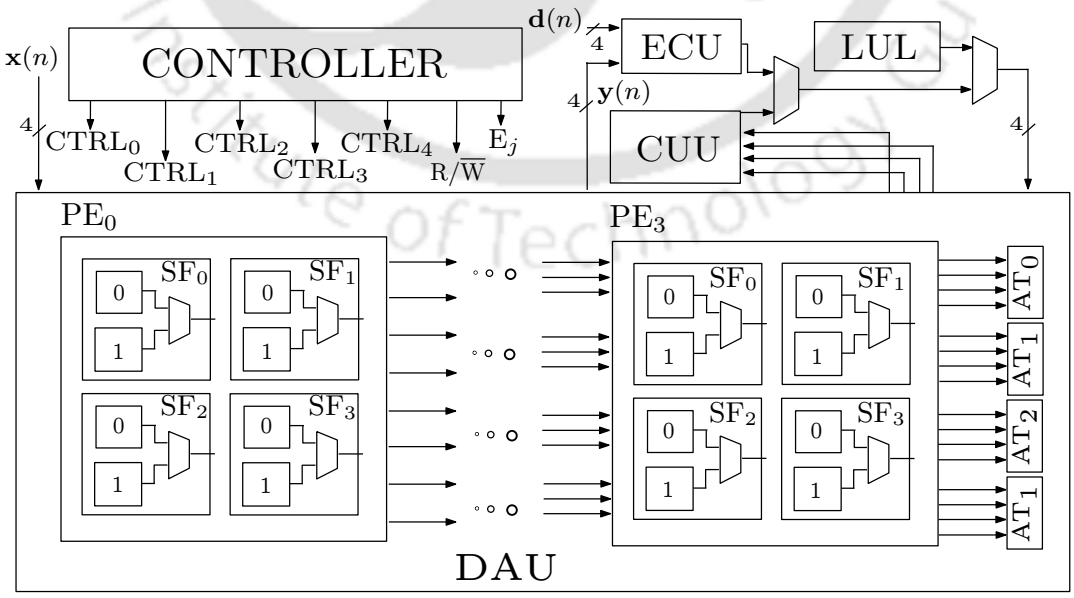


Fig. 5.5: Architecture of BLMS ADF based on OBC-DA for 16^{th} order filter and block-length of four, where $\mathbf{x}(n) = [x(4n), x(4n-1), x(4n-2), x(4n-3)]$, $\mathbf{y}(n) = [y(4n), y(4n-1),\ y(4n-2), y(4n-3)]$ and $\mathbf{d}(n) = [d(4n), d(4n-1),\ d(4n-2), d(4n-3)]$.

The block schematic of the proposed BLMS ADF for filter order $N = 16$ and block-length $L = 4$ is shown in Fig. 5.5. It consists of one DA-unit (DAU), one error computation unit (ECU), coefficient update unit (CUU) and a controller. In DAU, there are M processing elements (PE) and each PE has L sub-filters (SF), one LUT update logic (LUL) and L adder tree (AT) with M-inputs word for $0 \leq i \leq L-1$ and $0 \leq j \leq M-1$. Each SF further consists of one $\text{LUT}_0/\text{LUT}_1$ and one SA unit. The addresses for $\text{LUT}_0/\text{LUT}_1$ are obtained by bit-slicing the filter coefficients $\{\mathbf{w}_j(n)\}_k$ and error signal $\{\mathbf{e}(n)\}_k$ at the outputs of CUU and ECU respectively. The proposed design for updating the filter block requires that the LUTs of a particular column are updated. These are further utilized to compute filter output and coefficient increment terms. A controller is designed to generate control signals such as CTRL_0, CTRL_1, CTRL_2, CTRL_3, CTRL_4, $R/\overline{\text{W}}$ and $\mathbf{E} = [\text{E}_0, \text{E}_1, ..., \text{E}_{M-1}]$ to enable or disable processing units.

5.3.2.1 LUT Update Scheme

The proposed update scheme for LUT_0 and LUT_1 is explained as follows: Averaging of contents present at even and odd address locations of LUT_0 and LUT_1 would result a term independent of oldest sample $x(j'L - i - L + 1)$. With subsequent addition or subtraction of recent sample $\frac{1}{2}x(j'L - i)$ as stored in register $\mathbf{D}_j$ would generate contents corresponding to upper or lower half address locations, respectively. For example, the average of the contents at 0^{th} address locations (even) of LUT_0 and LUT_1 for block-length $L = 4$ would give $+\frac{1}{2}[+x(4j' - i - 1) + x(4j' - i - 2)]$. Notably, it does not contain any oldest sample $\frac{1}{2}x(4j' - i - 3)$. When this term is added or subtracted with content of $\mathbf{D}_j$ will generate two contents $\frac{1}{2}[+x(4j' - i) + x(4j' - i - 1)] + \frac{1}{2}x(4j' - i - 2)$ and $\frac{1}{2}[-x(4j' - i) + x(4j' - i - 1)] + \frac{1}{2}x(4j' - i - 2)$, respectively. Clearly, these corresponds to the contents of LUT_0 present at 0^{th} (lower half) and 2^{nd} (upper half) address locations for next iteration. Similarly, the average of the contents at 1^{st} address locations (odd) of LUT_0 and LUT_1 would give $+\frac{1}{2}[+x(4j' - i - 1) - x(4j' - i - 2)]$. Adding and subtracting of $\mathbf{D}_j$ would generate two contents $\frac{1}{2}[+x(4j' - i) + x(4j' - i - 1)] - \frac{1}{2}x(4j' - i - 2)$ and $\frac{1}{2}[-x(4j' - i) + x(4j' - i - 1)] - \frac{1}{2}x(4j' - i - 2)$ which correspond to the contents of LUT_1 present at 0^{th} (lower half) and 2^{nd} (upper half) address locations for next iteration, respectively. In a similar manner, the remaining contents of LUT_0 and LUT_1 can be updated. Mathematically, the contents of LUT_0 and LUT_1 for next-iteration from the

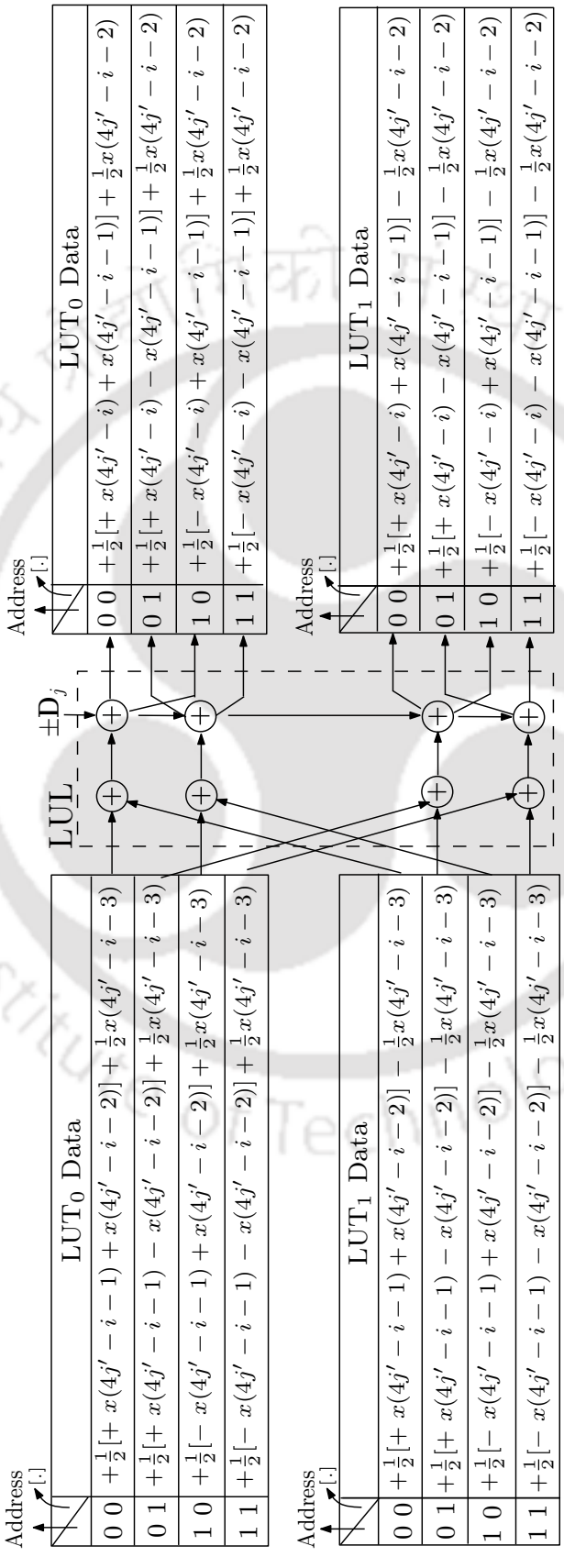


Fig. 5.6: (a) LUT_0 and LUT_1 update scheme for block-length of four, where $0 \leq i \leq L-1$, $0 \leq j \leq M-1$ and $j' = n-j-1$.

contents of LUT_0 and LUT_1 from current iteration can be expressed as

$$\begin{aligned}\mathbf{G}_a^{00}[\mathbf{a}] &= (-1)^{\left\lfloor \frac{a}{2^{L-3}} \right\rfloor}\mathbf{D}_j + \frac{1}{2}\left\{\mathbf{G}^{00}_{2\left\lfloor \frac{a}{2} \right\rfloor}[\mathbf{a}] + \mathbf{G}^{01}_{2\left\lfloor \frac{a}{2} \right\rfloor}[\mathbf{a}]\right\} \\ \mathbf{G}_a^{01}[\mathbf{a}] &= (-1)^{\left\lfloor \frac{a}{2^{L-3}} \right\rfloor}\mathbf{D}_j + \frac{1}{2}\left\{\mathbf{G}^{00}_{2\left\lfloor \frac{a}{2} \right\rfloor+1}[\mathbf{a}] + \mathbf{G}^{01}_{2\left\lfloor \frac{a}{2} \right\rfloor+1}[\mathbf{a}]\right\}\end{aligned} \tag{5.24}$$

where a is the address index with $a \in [0, 2^{L-2}-1]$ and $\pm$ sign of $\mathbf{D}_j$ is decided by the upper and lower half address locations of LUT. It is clear from the above discussion that the parallel update of LUTs by storing the recent sample in register of every block has resulted into lesser computational time.

5.3.2.2 Architecture of Sub-Filter Unit

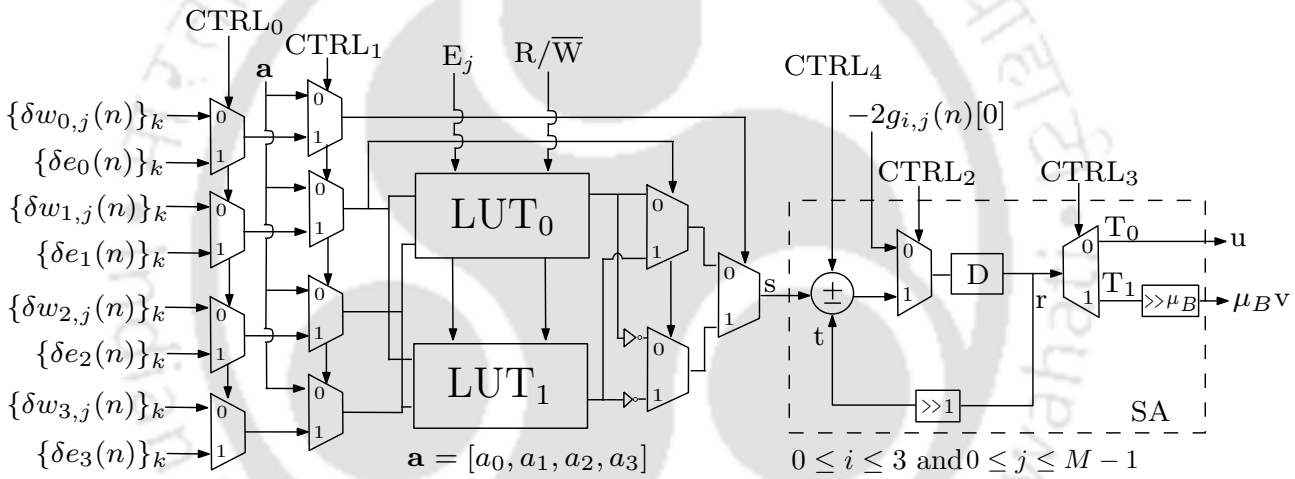


Fig. 5.7: Detailed architecture of SF unit of a particular PE for block-length of four.

After the completion of LUT update, the filtering operation is performed using the bit-slice coefficient-vector $\{\mathbf{w}_j(n)\}_k$ as addresses for LUT, in LSB to MSB order, as per (5.11). It must be noted that $-2g_{i,j}(n)[0]$ (-2, since it is placed before fixed right-shifter) is loaded initially by setting $\text{CTRL}_2 = 0$, in accordance with (5.16). As discussed, the same LUTs of a particular PE are used to compute the coefficient increment terms. Therefore, a two-level multiplexer (TLM) is necessary to select the LUT addresses for computation of filter output and coefficient increment terms. The first-level multiplexer in TLM either selects the bit-slices of filter coefficient $\{\mathbf{w}_j(n)\}_k$ or error signals $\{\mathbf{e}(n)\}_k$. However, it is always required to update the LUTs prior to the computation of partial filter output and coefficient increment terms in every iteration. This is taken care by second-level multiplexers in TLM, where its one input is the output of first-level multiplexers and other input as address vector $\mathbf{a} = [a_0, a_1, a_2, ..., a_{L-1}]$ for LUT-update. The LUT is splitted into LUT_0 and LUT_1 which are utilized

for the computation of filter output and coefficient increment term in same iteration. This has resulted into two more 2-to-1 multiplexers to read-out their contents, out of them, one has inverted inputs, in accordance with (5.21). In every iteration, the LUTs are read-out to perform shift-accumulation in successive clock cycles, according to DA principle. Like the computation of filter output, SA unit has to be loaded again initially with $-2g_{i,j}(n)[0]$ by setting $\text{CTRL}_2 = 0$ using a 2-to-1 multiplexer to compute the coefficient increment terms. The control signal CTRL_4 subtracts the contents of LUT from the accumulator at $(W-1)^{\text{th}}$ clock cycle. It must be noted that in every block iteration all initial terms $-2g_{i,j}(n)[0]$ are to be updated along with LUT-update, as they are nothing but two's complement of the content present at 0^{th} address locations of all LUTs. A 1-to-2 de-multiplexer is required to select the computed filter output and coefficient increment by setting $\text{CTRL}_3 = 0$ and 1 respectively, as shown in Fig. 5.7.

5.3.2.3 Architectures of Error Computation Unit and Coefficient Update Unit

The error computation unit (ECU) consists of an array of subtractors to compute the block of error $\mathbf{e}(n)$ from the block of filter output $\mathbf{y}(n)$ and desired response $\mathbf{d}(n)$, as shown in Fig. 5.8. In order to compute the coefficient increment terms of a particular PE, it is required to obtain the bit-slices of error block as addresses $\{\mathbf{e}(n)\}_k = [\{e(nL)\}_k, \{e(nL-1)\}_k, ..., \{e(nL-L+1)\}_k]$ for LUTs in LSB to MSB order. This is achieved by placing parallel-in serial-out (PISO) shift-registers after the error computation.

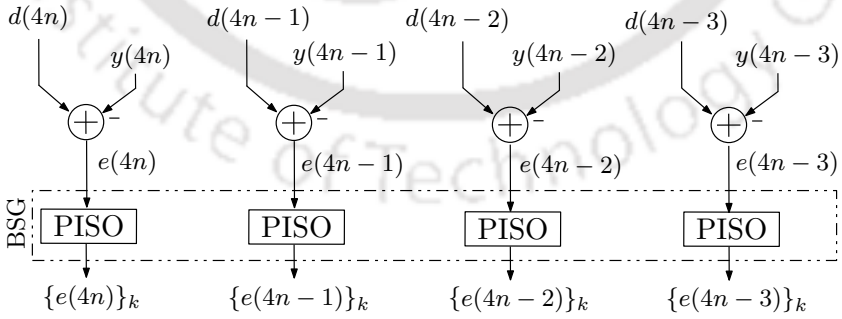


Fig. 5.8: Architecture of error-computation unit for block-length of four.

After the computation of coefficient increment terms with bit-slices $\{\mathbf{e}(n)\}_k$ as addresses for LUT, it is then required to compute the filter coefficients for next-iteration. Meanwhile, coefficient increment terms are scaled by μ_B, in accordance with (5.6). As mentioned in (5.23), the coefficient vector of the current iteration are used to update the coefficient-vector for the next iteration. This is achieved

with the bit-serial CUU architecture, as discussed in Chapters 2 and 3. It has an advantage of low-complexity over its counterpart word-parallel structure. This is because the bit-serial realization of CUU requires bit-slices of coefficient increments. The proposed CUU consists of an array of serial-in serial-out (SISO) shift-registers along with CSFAs and XOR gates, as shown in Fig. 5.9. It is important to note that the XOR gates are included so as to generate the bit-slices in OBC form, where CTRL_4 takes care of the sign-bit of filter-coefficients, in accordance with (5.15). The bit-slices of coefficient increments are stored column-wise and coefficient vectors are generated for the next-iteration from the first column. Subsequently, they are fed to the PE, whose LUTs are to be updated. In every iteration, the coefficient vector are aligned with the corresponding PE. For example, in n^{th} iteration, suppose LUTs of PE_1 are to be updated, then the second column SISO-array must contain the coefficient vector $\mathbf{w}_1(n)$. The coefficient increment values of $(j+1)^{\text{th}}$ column of filter coefficients are obtained from $(j+2)^{\text{th}}$ column, as shown in Fig. 5.9. The coefficient increment terms are added to filter coefficients in bit-serial fashion using CSFA. Importantly, the output of CSFA in $(j+1)^{\text{th}}$ column constitute a bit-vector of filter coefficients $\{\mathbf{w}_j(n)\}_k$. In successive clock cycles, the SISO contents are left-shifted to generate coefficient-vectors, in accordance with (5.23). In every iteration, the shifting in SISO-array begins after $2^{L/2-1}+1$ clock cycles and continues upto W clock cycles. Algorithm 4 explains the operation of proposed BLMS ADF.

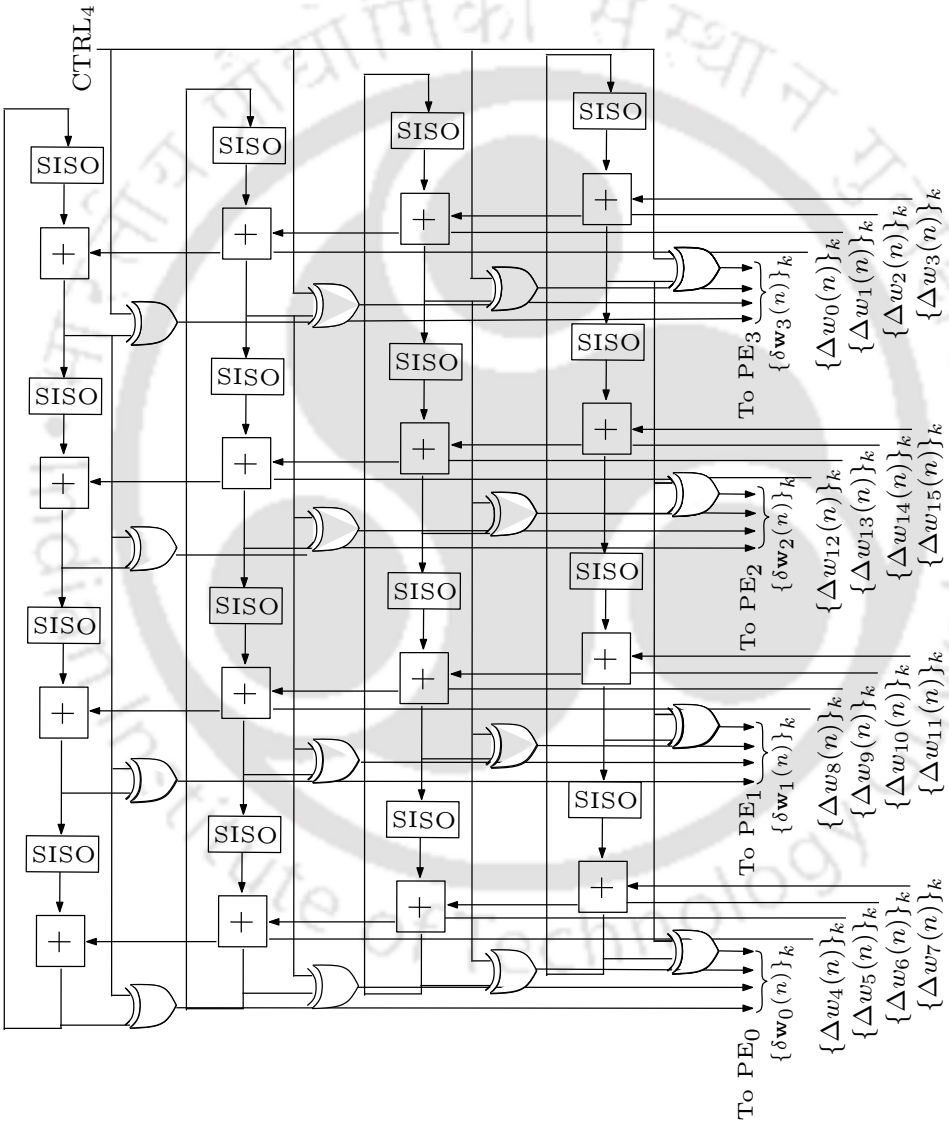


Fig. 5.9: Bit-serial architecture of coefficient-update unit for 16^{th} order filter and block-length of four.

Algorithm 4 Algorithm explaining the operation of the proposed BLMS ADF

Initialize:
$N = L \times M, i = 0, 1, ..., L-1, j = 0, 1, ..., M-1, j' = n-j-1$, **addr**
loop
$\mathbf{y}(n) = \sum_{j=0}^{M-1} \mathbf{X}_j^T(n)\mathbf{w}_j(n)$, $u_{i,j}(n) = \mathbf{x}_{i,j}^T(n)\mathbf{w}_j(n)$
$\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta\mathbf{w}(n)$ with $\Delta\mathbf{w}(n) = \mu_B \mathbf{X}_j(n)\mathbf{e}(n)$, $v_{i,j}(n) = \mathbf{x}_{i,j}^T(n)\mathbf{e}(n)$
  for $j = 0$ to $M-1$ **do**
    $u_{i,j}(n) = \sum_{k=0}^{W-1} \mathbf{G}(\{\boldsymbol{\delta}\mathbf{w}_j(n)\}_k)2^{-k} - \mathbf{G}(0)2^{-(W-1)}$
    $v_{i,j}(n) = \sum_{k=0}^{W-1} \mathbf{G}(\{\boldsymbol{\delta}\mathbf{e}(n)\}_k)2^{-k} - \mathbf{G}(0)2^{-(W-1)}$
    if E_j==1 **then**
      if $\mathrm{R}/\overline{\mathrm{W}}$==0 **then**
        if CTRL_1==0 **then addr** = **a**
          for $a = 0$ to $2^{L-2}-1$ **do**
            $\mathbf{G}_a^{00}[\mathbf{a}] \leftarrow (-1)^{\lfloor \frac{a}{2^{L-3}} \rfloor}\mathbf{D}_j + \frac{1}{2}\{\mathbf{G}_{2\lfloor \frac{a}{2} \rfloor}^{00}[\mathbf{a}] + \mathbf{G}_{2\lfloor \frac{a}{2} \rfloor}^{01}[\mathbf{a}]\}$
            $\mathbf{G}_a^{01}[\mathbf{a}] \leftarrow (-1)^{\lfloor \frac{a}{2^{L-3}} \rfloor}\mathbf{D}_j + \frac{1}{2}\{\mathbf{G}_{2\lfloor \frac{a}{2} \rfloor+1}^{00}[\mathbf{b}] + \mathbf{G}_{2\lfloor \frac{a}{2} \rfloor+1}^{01}[\mathbf{a}]\}$
          end for
          $\mathbf{w}_{j+1} \leftarrow$ **circleftshift**$\{\mathbf{w}_j, L\}$
        else
          if CTRL_0==0 **then addr** = $\{\boldsymbol{\delta}\mathbf{w}_j(n)\}_k$
            for $k = 0$ to $W-1$ **do**
              if $k == 0$ **then**
                CTRL_2=0, $u_{i,j}(n) = -\mathbf{G}[0]$
              else
                $u_{i,j}(n) = u_{i,j}(n) + \mathbf{G}[\{\boldsymbol{\delta}\mathbf{w}_j(n)\}_k]2^{-k}$
              end if
            end for
            $\mathbf{D}_j = \frac{1}{2}x(j'L - i)$, $\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{y}(n)$
          else addr = $\{\boldsymbol{\delta}\mathbf{e}(n)\}_k$
            for $k = 0$ to $W-1$ **do**
              if $k == 0$ **then**
                CTRL_2=0, $v_{i,j}(n) = -\mathbf{G}[0]$
              else
                $v_{i,j}(n) = v_{i,j}(n) + \mathbf{G}[\{\boldsymbol{\delta}\mathbf{e}(n)\}_k]2^{-k}$
              end if
            end for
          end if
        end if
      end if
    end if
  end for
  $n \leftarrow n+1$
end loop

5.4 Performance Comparison

In this section, the computational complexities involved in different designs are first discussed. Next, the performance of proposed BLMS ADF in terms of noise reduction is evaluated. Finally, the comparison between the proposed and existing designs are made in terms of area, power, throughput and logic utilization by carrying out FPGA and ASIC synthesis. The existing designs in [67], [68], [69], [44] and [47] are referred as DA_0-BADF, DA_1-BADF and DA_2-BADF, DA_3-BADF and DA_4-BADF respectively. So far, the design considerations of proposed filter for block length $L = 4$ is considered, it is important to discuss the design of proposed filter for higher block-lengths. A simple method is to express the filter block-length L in terms of smaller block-lengths of size 4, that is, $L = 4P$, where P is an integer. In such case, the architecture of SF has to be modified with respect to integer P. For example, the SF unit consists of P number of LUT_0 and LUT_1 of size four words, an adder tree is needed to add the outputs from the LUTs. Correspondingly, the bit-slices of filter coefficients and error vectors are splitted into P small vectors.

5.4.1 Computational Complexities

The proposed design with filter order $N = L \times M$ and block-length $L = 4P$ consists of M PEs with each PE comprises of L SFs (LP number of LUT_0 and LUT_1 of size four words, $L(P-1)$ adders, L 1-to-2 line de-multiplexer, Lw' number of XOR gates for sign reversal and L 2-to-1 multiplexers to load initial terms in SA unit, where $w' = W' + \log_2 L$, $2LP$ 2-to-1 multiplexers in LUT addressing-logic, $3L$ 2-to-1 multiplexers in LUT-read), one LUL ($2LP$ adders), one ECU (L subtractors followed by L PISO shift-registers of size W-bit), and one CUU (LM CSFAs, LM XOR gates, LM SISO shift registers of size $W+1$-bit). It is required to obtain bit-slices of coefficient increment terms, therefore, L PISO shift-registers of size W-bit are needed. Hence, the proposed filter requires $(PM+1)L+2PL$ adders, $N(3W+1)+LW$ registers, $4LM+M+1$ multiplxers/de-multiplexers, $8PN$ LUT words and $LM+LMw'$ XOR gates. The hardware complexities of the proposed and existing designs are listed in Table 5.1. Unlike DA_4-BADF, the LUL does not involve adder trees and input-delay unit to update the LUTs, therefore, the number of adders and registers are reduced. Also, the OBC combinations of input samples reduce the number of LUT words by half at the expense of slight increase in 2-to-1 multiplexers. Notably, the LUL of the proposed design logic does not require $W'LM$ AND gates, as in the case of DA_4-BADF, it rather uses LM XOR gates. While DA_0-BADF, DA_1-BADF, DA_2-BADF

and DA_3-BADF designs do not contain block length L term in their listed hardware complexities, since they were derived using LMS algorithm.

Table 5.1 also compares the time-complexities of the proposed and existing designs. The parallel update of LUT_0 and LUT_1 reduces the number of clock cycles. As an example, the proposed BLMS ADF with block-length L requires $2^{L/2-1}$ clock cycles to update LUTs. The computation of filter coefficient increment term and output require $2W$ clock cycles. In the proposed design, the contents of registers $\mathbf{D}_j = [D_0, D_1, ..., D_{M-1}]$ for a particular column are updated with the block of error computations. Hence, the total number of clock cycles for the proposed design would be $2^{L/2-1} + 2W + 1$. Moreover, the parallel access of LUT_0 and LUT_1 also leads to lesser access delays over DA_4-BADF design. Due to BLMS algorithm, the proposed design offers higher throughput than DA_0, DA_1, DA_2 and DA_3-BADF designs. Nonetheless, the effect of LUT access time and computation time of SA unit on throughput performance can be selectively overcome by choosing the appropriate block length. The complete filtering operation involve $2L$ write and $(2W)LM$ read operations, respectively, to update LUTs and to compute filter output and coefficient increment terms. Therefore, the number of LUT access per output (LAPO) in case of proposed filter would be $2L+2WLM$. From the results in Table 5.1, the proposed filter offers less LAPO since splitted LUTs are used as compared to DA_4-BADF design.

5.4.2 Noise Reduction Performance

A theoretical comparison between the adders, LUT size and noise reduction of the proposed and existing design [47] for two cases (Case A and Case B) is shown in Table 5.2. It is to be noted that Case A corresponds to large order filter design, while Case B corresponds to small order filter design with same input block-length to decide the number of PEs (and SFs) in the design of adaptive noise-canceller. It can be noted that the proposed design for Case B offers 43.24 % less adders, 87.5 % LUT words as compared to Case A for almost same noise reduction performance. Further, the existing design occupies more hardware and LUT resources over the proposed design for same noise reduction performance. Thus, it is concluded that the adopting large order filter does not always bring better noise reduction performance [103]. Hence, a 32^{nd} order filter is considered for the implementation of proposed design suitable to in-ear headphone applications.

To verify the validity of the proposed filter, the design is coded in Verilog along with the other existing works. The proposed filter is synthesized using Xilinx ZYNQ FPGA device XC7Z020-1CLG84C

for filter order $N = 32$, block-length $L = 8$ and $\mu = 0.04$ by setting the system clock at 20 MHz. An audio sample of bandwidth 0-4000 Hz and sampling frequency 8000 Hz is considered as the desired response $\mathbf{d}(n)$ [105]. An additive white gaussian noise (AWGN) is generated to produce the actual signal for in-ear headphone and fed to the input of the proposed filter, as shown in Fig. 5.1. Based on the above specifications, the proposed design has to finish all the filter computations within 2500 clock cycles. While the proposed design takes only 41 clock cycles (8 clock cycles in LUT_0 and LUT_1 update, 32 clock cycles in SA unit, and 1 clock cycle in block error computation and the update of external registers content). It is capable of providing a throughput performance of 1.95×10^6 samples / sec for $L = 4$ at 20 MHz. Note that the exceeded performance of the proposed design makes it capable to operate in higher frequency data converters. The corresponding results of noise reduction performance for the proposed design are illustrated in Fig. 5.10(a)−5.10(c). The output desired speech signal after noise cancellation is shown in Fig. 5.10(a). Also, the proposed filter keeps the estimate of input noise time-to-time, as shown in Fig. 5.10(b). To identify the noise cancelling performance of proposed filter at different frequencies, the noise reduction (in dB) is plotted and indicated by green-line in Fig. 5.10(c), which is the ratio of error signal to the desired signal. Note that the red-line in Fig. 5.10(c) depicts the spectrum of desired response $\mathbf{d}(n)$, plotted as $-\log_{10}(1 - \mathcal{D}(k))$, where $\mathcal{D}(k)$ is the FFT of $\mathbf{d}(n)$. From the results, it is found that the proposed filter performs well at certain frequencies. For instance, the proposed filter provides a maximum attenuation of 20 dB when the input frequency lies in the range 2000 Hz to 2700 Hz. In general, 6−20 dB noise reduction is always achieved by the proposed filter. For comparison purpose, the noise reduction along with logic utilization in terms of sliced LUTs (SLUT) and flip-flops (FF) for different designs in frequency range 2000-2700 Hz are listed in Table 5.3. Simulation results show that the proposed filter outperforms DA_0-BADF, DA_1-BADF, DA_2-BADF, DA_3-BADF for noise cancellation in in-ear headphones. For example, the noise reduction achieved by DA_4-BADF in the frequency range 2000−2700 Hz is lower as compared to the proposed design. From Table 5.3, it is quite evident that the proposed filter always has less logic utilization. For example, the proposed filter with order $N = 32$ and block-length $L = 4$ requires 1.80 times less SLUT and 1.71 times less FF than the DA_4-BADF design.

5.4.3 ASIC Synthesis

ASIC synthesis for the proposed and existing designs is carried out by Cadence RTL Compiler using TSMC 90 nm CMOS library and the corresponding results are listed in Table 5.4. The proposed

filter with filter order $N = 32$ and block-length $L = 8$ occupies 1.91 times less area and consumes 1.78 times less power as compared to DA_4-BADF. Furthermore, it provides nearly 3.91 and 1.4 times higher throughput as compared to DA_0-BADF and DA_4-BADF respectively. For better comparison, area-delay-product (ADP) and energy-per-output (EPO) of different designs are also estimated. Clearly, the proposed filter provides 3.83, 2.41, 2.17, 2.81, 1.98 times less ADP as compared to DA_0-BADF, DA_1-BADF, DA_2-BADF, DA_3-BADF and DA_4-BADF designs respectively. While the proposed filter offers 10.81, 3.57, 1.93, 4.32, 2.61 times less EPO over DA_0-BADF, DA_1-BADF, DA_2-BADF, DA_3-BADF and DA_4-BADF designs respectively. This is due to the reason that the clock cycles for the proposed filter to produce the output is $3 + 2W$. All the experimental results are in good agreement with theoretical results as listed in Table 5.1.

5.5 Conclusion

In this Chapter, a low-complexity pipelined BLMS ADF based on OBC-DA suitable to in-ear headphones application has been presented. Here, both physical LUT and SA unit are employed, and their effect on throughput performance has been compensated by block processing. A block of external registers are used whose contents are updated along with the block of error computation. As a consequence, there has not been any problem of non-OBC terms like those encountered in Chapters 2 and 4. The proposed approach employed two splitted LUTs based on even and odd address locations for computing the filter output and coefficient increment terms in the same-iteration. Also, a new filter block update strategy and a novel LUTs updating scheme have been suggested. The block update strategy needs to update only one processing element in the given set of processing elements, where the splitted LUTs of each processing element are updated in parallel with the use of external registers. Compared with the best existing design, the proposed filter offers less adders, less registers and half LUT words with simultaneous reduction in the number of clock cycles. Simulation results showed that the proposed design offers more noise reduction, occupies less area, consumes less power and utilizes less number of SLUT and FF as compared to the best existing design.

Table 5.1: Comparison of Computational Complexities of Different DA Based Designs with N^{th} Order Filter and L^{th} Order Block Length

Design	**Adders**	**Registers**	**Mux/Dmux**	**LUT words**	**Clock period**	**TR**	**LAPO**
DA_0-BADF [67]	$3N/4$	$2(N-2)W$	$4+11N/4$	$8N$	$T_{MA}+T_A\log_2 M$	$1/T$	$N(W+64)/4$
DA_1-BADF [68]	$(3N+2)/2$	$(9W+4W')N/4$	$2+3N/2$	$4N$	$T_{MA}+T_A\log_2 M$	$1/T$	$N(W+16)/4$
DA_2-BADF [69]	$(3N+4)/2$	$(9W+4W')N/4$	$2+3N$	$4N$	$T_{MA}+T_A\log_2 M$	$1/T$	$N(W+16)/4$
DA_3-BADF* [44]	$32N+2.5\gamma+2$	$4NW'+(10N+\gamma)W$	–	$10[N_1 2^{\alpha_1}+N_2 2^{\alpha_2}]$	$T_{MA}+3T_A$	N/T_0	$20W$
DA_4-BADF [47]	$(PM+1)L+3PL$	$LM(3W+1)+$ $L(W+W')-W'$	$(2LM+M+1)$	$16PN$	τ_1	L/T_1	$16P+2NW/L$
Proposed	$(PM+1)L+2PL$	$N(3W+2)+LW$	$(4LM+M+1)$	$8PN$	τ_2	L/T_2	$2P+2NW/L$

TR: throughput rate, LAPO: LUT access per output, W' and W are wordlength of $x(nL-i)$ and, $e(nL-i)$ and $w(nL-i)$ respectively, τ_1 =max$[(T_{ACC_1}+T_A\log_2 P), T_A(1+\log_2 M)]$, τ_2 =max$[(T_{ACC_2}+T_A\log_2 P), T_A(1+\log_2 M)]$, $T=k[(T_{MR}+T_A(1+\log_2 M)]$ with $k=48$ and $k=32$ for DA_0-BADF and DA_1-BADF respectively; $T_0=2(N_1+N_2+2)W$, $T_1=\tau_1(17+2W)$, $T_2=\tau_2(3+2W)$, where T_{ACC_1} and T_{ACC_2}, T_{MR}, and T_A are computational delays of LUT access in DA_0-BADF and DA_1-BADF designs, LUT read operation (MR) and W'-bits adder respectively, with $T_{ACC_1} > T_{ACC_2}$. DA_3-BADF* [44] is FFT-based design, where $\gamma=N_1+N_2$, $2N=N_1N_2$, $\alpha_1=\frac{N_1-1}{2}$, $\alpha_2=\frac{N_2-1}{2}$ and requires $8(N+2)$ multipliers. Proposed and DA_4-BADF designs need $LM+LM(W'+\log_2 L)$-XOR gates and $W'LM$-AND+$LM(W'+\log_2 L)$-XOR gates respectively.

Table 5.2: Comparison of Adders, LUT size and Noise Reduction of Presented and Existing Design [47] for different Filter Orders

Filter Order (N)	**Proposed Design**			**Existing Design [47]**		
	ADD	LUT	NR	ADD	LUT	NR
Case A $N = 128, L = 8, P = 2$	296	2048	14.5	312	4096	14.5
Case B $N = 32, L = 8, P = 2$	168	512	13.4	184	1024	13.4

ADD: Additions, LUT: Look-up Table, NR: Noise Reduction (in dB) and TR: Throughput Rate.

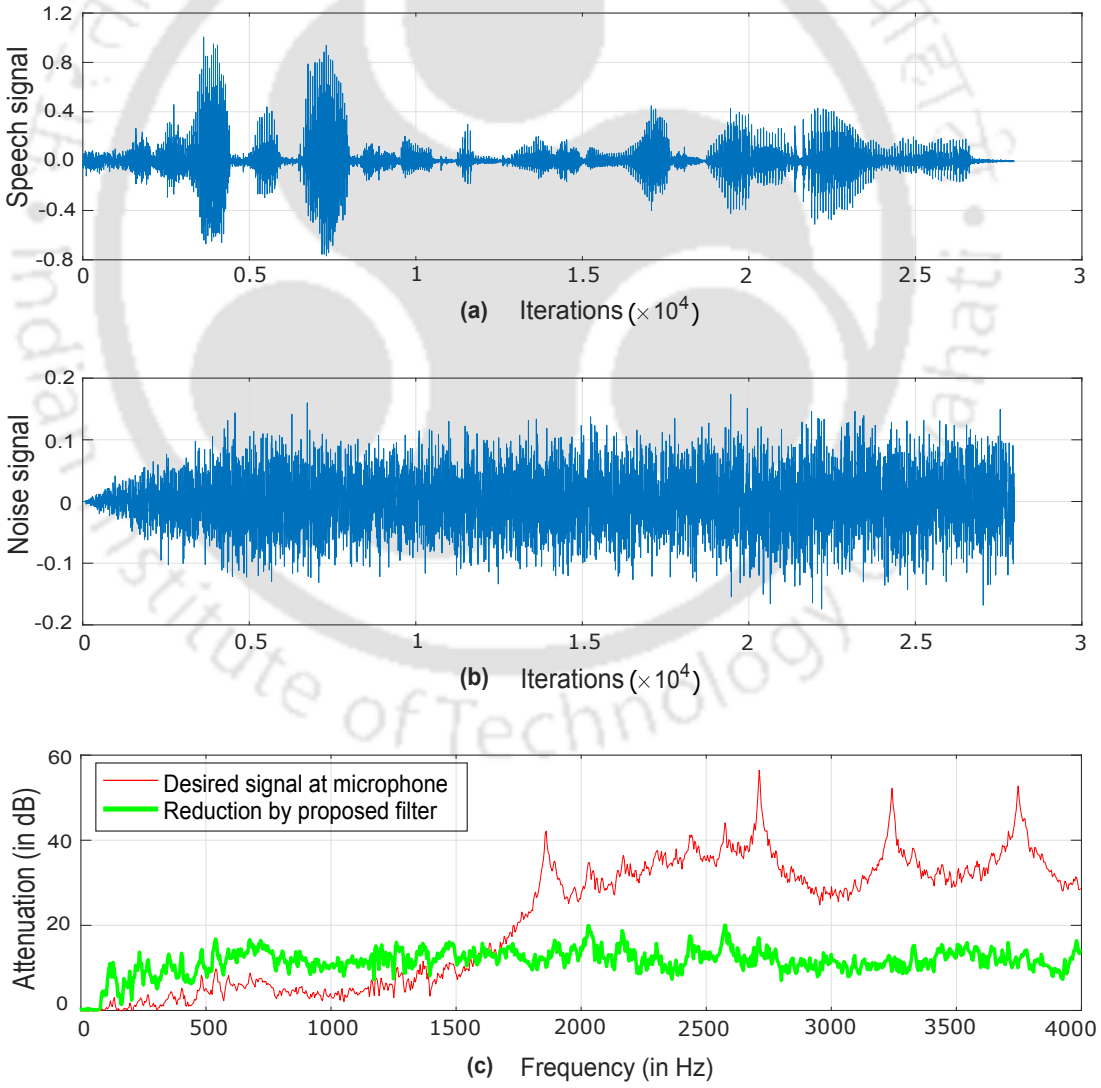


Fig. 5.10: (a) Desired speech signal (b) Estimated noise signal (c) Noise reduction at different frequency components.

Table 5.3: Noise Performance Evaluation for 32nd Order Filter, 4th Order Block Length and 16-bit Wordlength of Filter Coefficients on Xilinx ZYNQ XC7Z020-1CLG84C

Design	**Platform**	**FR** (Hz)	**NR** (dB)	**SLUT**	**FF**
DA_0-BADF [67]	FPGA	2000-2700	8-13	49563	15258
DA_1-BADF [68]	FPGA	2000-2700	8-14	31632	9723
DA_2-BADF [69]	FPGA	2000-2700	9-15	20129	7263
DA_3-BADF [44]	FPGA	2000-2700	11-14	18237	6968
DA_4-BADF [47]	FPGA	2000-2700	8-19	25783	8637
Proposed	FPGA	2000-2700	6-20	14275	5043

FR: frequency range, NR: noise-reduction, SLUT: sliced LUTs, FF: flip-flops.

Table 5.4: Performance Comparison of Different DA Based Designs with ASIC Synthesis using TSMC 90 nm CMOS Library for 32nd Order Filter, 4th Order Block Length and 16-bit Wordlength of Filter Coefficients

Design	**Area** (sq. μm)	**Power** (mW)	**MSP** (ns)	**ADP** (sq. μm.ms)	**EPO** (nJ)
DA_0-BADF [67]	272553.2	2.925	28.34	349.30	7.03
DA_1-BADF [68]	211562.7	1.452	32.52	219.16	2.32
DA_2-BADF [69]	176122.3	1.573	35.11	197.87	1.26
DA_3-BADF [44]	262367.8	3.124	27.52	256.23	2.81
DA_4-BADF [47]	452166.5	2.781	32.63	180.73	1.70
Proposed	236728.3	1.558	31.42	91.11	0.65

MSP: minimum sampling period, ADP: Area delay product (area $\times$ MSP/L), EPO: Power/Throughput (energy per output).

6

Summary and Conclusions

Contents

6.1 Summary of the Present Work

The objective of the work presented in this thesis is to develop low-complexity pipelined architectures for LMS adaptive filters. This has been achieved using DA approach. Low-complexity DA based pipelined LMS adaptive filters are then applied in several applications such as system identification, channel equalization and noise cancellation.

In Chapter 2, three architectures for pipelined LMS adaptive filter based on OBC scheme have been presented with an objective of optimizing the hardware complexity. The proposed design uses the philosophy of LUT-less design in [71] to generate the partial products. Here, the partial products have been generated serially in different ways by representing the coefficients in OBC-form. However, non-OBC terms have been produced at the output during initial clock cycles which have been subsequently corrected in error computation unit. The reason for choosing OBC scheme is to exploit symmetries between the digits in high-radix for the realization of low-complexity partial product generator and high-speed SA unit. Further, the novel implementations of minus-minus-plus based carry save full adder, offset term, coefficient update and SA units have also been presented. All the proposed designs have been extended for large filter order with the modification in correction scheme of non-OBC terms.

A general system identification problem with CLMS adaptive filter has been considered in Chapter 3. Although the complexity of pipelined LMS adaptive filter has been optimized using OBC scheme in Chapter 2, non-OBC terms have been produced at the output. A correction scheme has been suggested for the removal of non-OBC terms, but it cannot improve the convergence performance. This has been addressed by CLMS adaptive filter which is realized using a single DA based LMS unit by representing the filter coefficients in TC-form. Further reduction in complexity has been achieved by sharing of partial products in parallel. To transfer the filter coefficients from one LMS unit to another in CLMS adaptive filter, the correlation between adjacent errors has been exploited. In every iteration, the correlation between the adjacent errors has been compared with the pre-defined time window. As the duration of correlation of adjacent errors exceeds the pre-defined time window, the transfer of filter coefficients has been made possible by pre-shifting the input samples. In the sequel, an expression for pre-defined time window has been derived in terms of filter order, coefficients wordlength and step-size. It is found that the proposed design for 32^{nd} order filter provides about 6 dB better steady-state error performance over the best existing design.

A case study on channel equalization problem for 5G communication system using ADFE has

been considered in Chapter 4. This design is derived using low-complexity implementation of feedback filter based on the OBC scheme. Although LUT-less designs are already discussed in Chapter 2 and 3, the idea of SA-less design is more suitable for the realization of feedback filter and hence it has been taken up in this chapter. It is based on the fact that when radix-size equals the wordlength of tentative decisions, the implementation of feedback filter can be made SA-less. The reason for choosing the tentative decisions in OBC-form due to its low-complexity realization. The complexity has been reduced by pre-computing and storing the coefficients of feedback filter in two different LUTs separated by pipelined registers. Further reduction in complexity has been achieved by exploiting the symmetries between the contents of each LUT. The presented architecture has been pre-speed up by two, retimed and unfolded to meet the throughput requirements of 5G communication. To implement the coefficient updating unit of feedback filter, a novel strategy has been presented to update the contents of both the LUTs by adding them with the contents of decision LUT. In every iteration, the contents of decision LUT are updated and the effect of non-OBC terms due to pipelining has been eliminated using parallel error multiplexer. The proposed design offers a throughput of nearly 2.15 Gbps for 16-QAM while the hardware complexity and LUT size have been reduced over the best existing design.

Finally, in Chapter 5, a BLMS algorithm based noise cancellation problem for in-ear headphone applications has been considered. Here, both physical LUT and SA unit have been employed, and their effects on throughput performance have been compensated by block processing. In this chapter, the mathematical formulation is carried out for BLMS algorithm based on OBC to compute the filter output and coefficient increment terms by decomposing the filter order into input-block length and processing elements. The proposed approach employed two splitted LUTs to compute filter output and coefficient increment terms in the same-iteration. Also, a new filter block update strategy and a novel LUT update scheme have been suggested, where LUTs of one processing element require to update in the set of given processing elements. The proposed architecture does not have the problem of non-OBC terms as pipelined registers are external to the adaptive filter. Compared with the best existing scheme, the proposed filter offers fewer adders, fewer registers, half LUT words while simultaneously reduces the number of clock cycles. Simulation results showed that the proposed design for 32^{nd} order filter with block-length 8 offers 6-20 dB noise reduction while occupies less area, consumes less power and utilizes less SLUT and FF as compared to the best existing design.

6.2 Suggestions for Future Research

In this thesis, we have shown the effectiveness of DA for low-complexity realization of least-mean-square (LMS) based adaptive filters. However, we still felt that there may exist considerable scope to extend this work in various directions for future investigation. Some of them are listed as follows

(i) Approximate computing is an emerging area for VLSI design which basically relaxes the algorithm precision constraints to improve digital circuit performance. Hence, there may be some possibility of reducing the chip area and power dissipation at the algorithmic-level implementation of adaptive filters using approximate computing along with DA.

(ii) This thesis is primarily concentrated on complexity reduction of pipelined LMS adaptive filter and its variants. The other variants such as normalized LMS (NLMS), variable step-size LMS (VSS-LMS), sign-sign LMS (SS-LMS) etc. may also be considered for future research to reduce their complexity using DA. Further, recursive-least-square (RLS) adaptive filters which exhibit superior performance over LMS adaptive filter have not been considered. Thus, it will be pertinent to take similar methodology for DA based realization of RLS adaptive filter.

(iii) To improve the convergence of the proposed filter as described in Chapter 3, the present work has employed the concept of pre-defined time window and compared it with the duration of correlation between consecutive error signal. The convergence of pipelined filter may be improved by parallel elimination, but this may require extra hardware, and one may try for low-complexity implementation of such schemes.

(iv) The performance of proposed ADFE as introduced in Chapter 4 in terms of hardware complexity may be enhanced by dynamic adaptation of filter coefficients. This is because the LUT complexities of both the stages have still exponential dependence on the filter order with base as QAM constellation size. For this purpose, thorough assessment of the coefficient adaptation needs to be done since ADFE structure has two loops.

Bibliography

[1] G. E. Moore, "Cramming more components onto integrated circuits, reprinted from electronics, vol. 38, no. 8, april 19, 1965, pp. 114 ff." *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, 2006.

[2] L. R. Vega and H. Rey, *A Rapid Introduction to Adaptive Filtering.* Springer Science & Business Media, 2012.

[3] S. Rudich and A. Wigderson, *Computational Complexity Theory.* American Mathematical Soc., vol. 10, 2004.

[4] D. M. Markovic, "A power/area optimal approach to VLSI signal processing," *University of California, Berkeley*, 2006.

[5] S. Haykin, *Adaptive Filter Theory (3rd Ed.).* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[6] J. Benesty and Y. Huang, *Adaptive Signal Processing: Applications to Real-World Problems.* Springer Science & Business Media, 2013.

[7] J. A. Apolinário and S. L. Netto, "Introduction to adaptive filters," in *QRD-RLS Adaptive Filtering*, pp. 1–27, Springer, 2009.

[8] B. Widrow and S. D. Stearns, *Adaptive Signal Processing.* Pearson Education, 1985.

[9] L. C. Wood and S. Treitel, "Seismic signal processing," *Proceedings of the IEEE*, vol. 63, no. 4, pp. 649–661, 1975.

[10] D. George, R. Bowen, and J. Storey, "An adaptive decision feedback equalizer," *IEEE Transactions on Communication Technology*, vol. 19, no. 3, pp. 281–293, 1971.

[11] J. Arenas-García, A. R. Figueiras-Vidal, and A. H. Sayed, "Mean-square performance of a convex combination of two adaptive filters," *IEEE Transactions on Signal Processing*, vol. 54, no. 3, pp. 1078–1090, 2006.

[12] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications.* John Wiley & Sons, 2013.

[13] R. Harris, D. Chabries, and F. Bishop, "A variable step (VS) adaptive filter algorithm," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 2, pp. 309–316, 1986.

[14] S. Cheng, Y. Wei, Y. Chen, S. Liang, and Y. Wang, "A universal modified LMS algorithm with iteration order hybrid switching," *ISA Transactions*, vol. 67, pp. 67–75, 2017.

[15] A. I. Sulyman and A. Zerguine, "Convergence and steady-state analysis of a variable step-size NLMS algorithm," *Signal Processing*, vol. 83, no. 6, pp. 1255–1273, 2003.

[16] R. H. Kwong and E. W. Johnston, "A variable step size LMS algorithm," *IEEE Transactions on Signal Processing*, vol. 40, no. 7, pp. 1633–1642, 1992.

[17] W. P. Ang and B. Farhang Boroujeny, "A new class of gradient adaptive step-size LMS algorithms," *IEEE Transactions on Signal Processing*, vol. 49, no. 4, pp. 805–810, 2001.

[18] T. Aboulnasr and K. Mayyas, "A robust variable step-size LMS-type algorithm: analysis and simulations," *IEEE Transactions on Signal Processing*, vol. 45, no. 3, pp. 631–639, 1997.

[19] M. H. Costa and J. C. M. Bermudez, "A robust variable step size algorithm for LMS adaptive filters," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 3, pp. III–III, 2006.

[20] A. Mader, H. Puder, and G. U. Schmidt, "Step-size control for acoustic echo cancellation filters–an overview," *Signal Processing*, vol. 80, no. 9, pp. 1697–1719, 2000.

[21] M. T. Silva, V. H. Nascimento, and J. Arenas-García, "A transient analysis for the convex combination of two adaptive filters with transfer of coefficients," in *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*, pp. 3842–3845, 2010.

[22] C.-L. Wang and R.-Y. Chen, "Optimum design of the LMS algorithm using two step sizes for adaptive FIR filtering," *Signal processing*, vol. 26, no. 2, pp. 197–204, 1992.

[23] J. Arenas-García, M. Martínez-Ramón, Á. Navia-Vázquez, and A. R. Figueiras-Vidal, "Plant identification via adaptive combination of transversal filters," *Signal Processing*, vol. 86, no. 9, pp. 2430–2438, 2006.

[24] L. A. Azpicueta-Ruiz, A. R. Figueiras-Vidal, and J. Arenas-Garcia, "A normalized adaptation scheme for the convex combination of two adaptive filters," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3301–3304, 2008.

[25] V. H. Nascimento and R. C. de Lamare, "A low-complexity strategy for speeding up the convergence of convex combinations of adaptive filters," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3553–3556, 2012.

[26] L. Lu, H. Zhao, Z. He, and B. Chen, "A novel sign adaptation scheme for convex combination of two adaptive filters," *AEU-International Journal of Electronics and Communications*, vol. 69, no. 11, pp. 1590–1598, 2015.

[27] M. Renfors and Y. Neuvo, "The maximum sampling rate of digital filters under hardware speed constraints," *IEEE Transactions on Circuits and Systems*, vol. 28, no. 3, pp. 196–202, 1981.

[28] K. K. Parhi, "Pipelining in algorithms with quantizer loops," *IEEE Transactions on Circuits and Systems*, vol. 38, no. 7, pp. 745–754, 1991.

[29] S. Kasturia and J. H. Winters, "Techniques for high-speed implementation of nonlinear cancellation," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 5, pp. 711–717, 1991.

[30] K. K. Parhi, "Pipelining of parallel multiplexer loops and decision feedback equalizers," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, Proceedings (ICASSP)*, vol. 5, pp. V–21, 2004.

[31] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters- Part I and II." *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 7, pp. 1099–1117, 1989.

[32] K. K. Parhi, "Design of multigigabit multiplexer-loop-based decision feedback equalizers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 4, pp. 489–493, 2005.

[33] C. H. Lin, A. Y. A. Wu, and F. M. Li, "High-performance VLSI architecture of decision feedback equalizer for gigabit systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 9, pp. 911–915, 2006.

[34] C. S. Lin, Y. C. Lin, S. J. Jou, and M. T. Shiou, "Concurrent digital adaptive decision feedback equalizer for 10GBase-LX4 ethernet system," in *IEEE Custom Integrated Circuits Conference (CICC)*, pp. 289–292, 2007.

[35] D. Oh and K. K. Parhi, "Low complexity design of high speed parallel decision feedback equalizers," in *International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 118–124, 2006.

[36] N. R. Shanbhag and K. K. Parhi, "Pipelined adaptive DFE architectures using relaxed look-ahead," *IEEE Transactions on Signal Processing*, vol. 43, no. 6, pp. 1368–1385, 1995.

[37] M. D. Yang, A. Y. Wu, and J. T. Lai, "High-performance VLSI architecture of adaptive decision feedback equalizer based on predictive parallel branch slicer (PPBS) scheme," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 2, pp. 218–226, 2004.

[38] Y. C. Lin, M. T. Shiue, and S. J. Jou, "10 Gbps decision feedback equalizer with dynamic lookahead decision loop," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1839–1842, 2009.

[39] Y. C. Lin, S. J. Jou, and M. T. Shiue, "High throughput concurrent lookahead adaptive decision feedback equaliser," *IET circuits, devices & systems*, vol. 6, no. 1, pp. 52–62, 2012.

[40] A. L. Pola, J. E. Cousseau, O. E. Agazzi, and M. R. Hueda, "A low-complexity decision feed-forward equalizer architecture for high-speed receivers on highly dispersive channels," *Journal of Control Science and Engineering*, vol. 2013, p. 10, 2013.

[41] R. Shaik, M. Chakraborty, and S. Chattopadhyay, "An efficient finite precision realization of the block adaptive decision feedback equalizer," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1910–1913, 2008.

[42] G. Clark, S. K. Mitra, and S. R. Parker, "Block implementation of adaptive digital filters," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 29, no. 3, pp. 744–752, 1981.

[43] D. Das, G. Panda, and S. Kuo, "New block filtered-x LMS algorithms for active noise control systems," *IET Signal Processing*, vol. 1, no. 2, pp. 73–81, 2007.

[44] S. Baghel and R. Shaik, "FPGA implementation of fast block LMS adaptive filter using distributed arithmetic for high throughput," in *IEEE International Conference on Communications and Signal Processing (ICCSP)*, pp. 443–447, 2011.

[45] B. K. Mohanty and P. K. Meher, "Delayed block LMS algorithm and concurrent architecture for high-speed implementation of adaptive FIR filters," in *TENCON IEEE Region 10 Conference*, pp. 1–5, 2008.

[46] R. Jayashri, H. Chitra, S. Kusuma, A. Pavithra, and V. Chandrakanth, "Memory based architecture to implement simplified block LMS algorithm on FPGA," in *International Conference on Communications and Signal Processing (ICCSP)*, 2011, pp. 179–183.

[47] 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 Transactions on Signal Processing*, vol. 61, no. 4, pp. 921–932, 2013.

[48] B. K. Mohanty, P. K. Meher, and S. K. Patel, "LUT optimization for distributed arithmetic-based block least mean square adaptive filter," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 5, pp. 1926–1935, 2016.

[49] P. Channels, "Modulation (3gpp ts 36.211 version 10.4. 0 release 10),," *ETSI TS*, vol. 136, no. 211, p. v10, 2012.

[50] P. Banelli, S. Buzzi, G. Colavolpe, A. Modenini, F. Rusek, and A. Ugolini, "Modulation formats and waveforms for 5G networks: Who will be the heir of OFDM?: An overview of alternative modulation schemes for improved spectral efficiency," *IEEE Signal Processing Magazine*, vol. 31, no. 6, pp. 80–93, 2014.

[51] P. M. Kogge, *The architecture of pipelined computers*. CRC Press, 1981.

[52] G. Long, F. Ling, and J. G. Proakis, "The LMS algorithm with delayed coefficient adaptation," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 9, pp. 1397–1405, 1989.

[53] M. D. Meyer and D. P. Agrawal, "A modular pipelined implementation of a delayed LMS transversal adaptive filter," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1943–1946, 1990.

[54] L. D. Van and W. S. Feng, "An efficient systolic architecture for the DLMS adaptive filter and its applications," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, no. 4, pp. 359–366, 2001.

[55] L. K. Ting, R. Woods, and C. F. Cowan, "Virtex FPGA implementation of a pipelined adaptive LMS predictor for electronic support measures receivers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 1, pp. 86–95, 2005.

[56] E. Mahfuz, C. Wang, and M. O. Ahmad, "A high-throughput DLMS adaptive algorithm," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 3753–3756, 2005.

[57] P. K. Meher and S. Y. Park, "Low adaptation-delay LMS adaptive filter part-i: Introducing a novel multiplication cell," in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1–4, 2011.

[58] Meher, P. K. and Park, S. Y, "Area-delay-power efficient fixed-point LMS adaptive filter with low adaptation-delay," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 2, pp. 362–371, 2014.

[59] Y. Yi, R. Woods, L.-K. Ting, and C. Cowan, "High speed FPGA-based implementations of delayed-LMS filters," *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, vol. 39, no. 1-2, pp. 113–131, 2005.

[60] P. K. Meher and M. Maheshwari, "A high-speed FIR adaptive filter architecture using a modified delayed LMS algorithm," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 121–124, 2011.

[61] P. K. Meher and S. Y. Park, "Critical-path analysis and low-complexity implementation of the LMS adaptive algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 3, pp. 778–788, 2014.

[62] S. P. Matcha, "High performance architectures for adaptive equalizers using distributed arithmetic," Ph.D. dissertation, 2016.

[63] J.-P. Choi, S.-C. Shin, and J.-G. Chung, "Efficient ROM size reduction for distributed arithmetic," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2, pp. 61–64, 2000.

[64] A. Croisier, D. Esteban, M. Levilion, and V. Riso, "Digital filter for PCM encoded signals," Dec. 4 1973, uS Patent 3,777,130.

[65] W. Huang and D. V. Anderson, "Adaptive filters using modified sliding-block distributed arithmetic with offset binary coding," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 545–548, 2009.

[66] A. Peled and B. Liu, "A new hardware realization of digital filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 22, no. 6, pp. 456–462, 1974.

[67] D. J. Allred, H. Yoo, V. Krishnan, W. Huang, and D. V. Anderson, "LMS adaptive filters using distributed arithmetic for high throughput," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 7, pp. 1327–1337, 2005.

[68] R. Guo and L. S. DeBrunner, "Two high-performance adaptive filter implementation schemes using distributed arithmetic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 9, pp. 600–604, 2011.

[69] M. S. Prakash and R. A. 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, 2013.

[70] P. K. Meher and S. Y. Park, "High-throughput pipelined realization of adaptive FIR filter based on distributed arithmetic," in *IEEE International Conference on VLSI and System-on-Chip (VSLI-SOC)*, pp. 428–433, 2011.

[71] S. Y. Park and P. K. Meher, "Low-power, high-throughput, and low-area adaptive FIR filter based on distributed arithmetic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 6, pp. 346–350, 2013.

[72] P. K. Meher and T. Stouraitis, *Arithmetic Circuits for DSP Applications*. Wiley Online Library, 2017.

[73] R.-Y. Chen and C.-L. Wang, "On the optimum step size for the adaptive sign and LMS algorithms," *IEEE Transactions on Circuits and Systems*, vol. 37, no. 6, pp. 836–840, 1990.

[74] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. John Wiley & Sons, 2007.

[75] X. Lou, Y. J. Yu, and P. K. Meher, "Analysis and optimization of product-accumulation section for efficient implementation of FIR filters," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 10, pp. 1701–1713, 2016.

[76] K. Han, B. L. Evans, and E. Swartzlander, "Low-power multipliers with data wordlength reduction," in *Proc. Asilomar Conference on Signals, Systems and Computers (ACSSC)*, pp. 1615–1619, 2005.

[77] X. Cui, W. Liu, X. Chen, E. E. Swartzlander, and F. Lombardi, "A modified partial product generator for redundant binary multipliers," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1165–1171, 2016.

[78] K. K. Parhi, "Fast VLSI binary addition," in *IEEE Workshop on Signal Processing Systems (SIPS), Design and Implementation,* pp. 232–241, 1997.

[79] "TSMC 90 nm General-Purpose CMOS Standard Cell Libraries−tcbn90ghp," [Online] Available: www.tsmc.com/.

[80] V. Sklyarov, I. Skliarova, A. Barkalov, and L. Titarenko, *Synthesis and Optimization of FPGA-based Systems.* Springer Science & Business Media, vol. 294, 2014.

[81] C. Caraiscos and B. Liu, "A roundoff error analysis of the LMS adaptive algorithm," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 1, pp. 34–41, 1984.

[82] N. Verhoeckx and T. Claasen, "Some considerations on the design of adaptive digital filters equipped with the sign algorithm," *IEEE Transactions on Communications*, vol. 32, no. 3, pp. 258–266, 1984.

[83] E. Dahlman, S. Parkvall, and J. Skold, *4G: LTE/LTE-advanced for mobile broadband.* Academic press, 2013.

[84] C.-X. Wang, F. Haider, X. Gao, X.-H. You, Y. Yang, D. Yuan, H. Aggoune, H. Haas, S. Fletcher, and E. Hepsaydir, "Cellular architecture and key technologies for 5G wireless communication networks," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 122–130, 2014.

[85] F.-L. Luo and C. Zhang, *Signal Processing for 5G: Algorithms and Implementations.* John Wiley & Sons, 2016.

[86] G. Wunder, P. Jung, M. Kasparick, T. Wild, F. Schaich, Y. Chen, S. Ten Brink, I. Gaspar, N. Michailow, A. Festag *et al.*, "5GNOW: non-orthogonal, asynchronous waveforms for future mobile applications," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 97–105, 2014.

[87] H. Lin, "Flexible Configured OFDM for 5G air interface," *IEEE Access*, vol. 3, pp. 1861–1870, 2015.

[88] R. Gerzaguet, N. Bartzoudis, L. G. Baltar, V. Berg, J.-B. Doré, D. Kténas, O. Font-Bach, X. Mestre, M. Payaró, M. Färber *et al.*, "The 5G candidate waveform race: a comparison of complexity and performance," *EURASIP Journal on Wireless Communications and Networking*, vol. 2017, no. 1, p. 13, 2017.

[89] W. Xiang, K. Zheng, and X. S. Shen, "5G mobile communications," Springer, 2016.

[90] H. Schulze and C. Lüders, *Theory and Applications of OFDM and CDMA: Wideband Wireless Communications.* John Wiley & Sons, 2005.

[91] L. L. Hanzo and T. Keller, *OFDM and MC-CDMA: A Primer*. John Wiley & Sons, 2007.

[92] L. Korowajczuk, *LTE, WiMAX and WLAN network design, optimization and performance analysis*. John Wiley & Sons, 2011.

[93] M. Meidlinger and Q. Wang, "Performance evaluation of LTE advanced downlink channel estimators," in *IEEE International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 252–255, 2012.

[94] B. Widrow, J. R. Glover, J. M. McCool, J. Kaunitz, C. S. Williams, R. H. Hearn, J. R. Zeidler, J. E. Dong, and R. C. Goodlin, "Adaptive noise cancelling: Principles and applications," *Proceedings of the IEEE*, vol. 63, no. 12, pp. 1692–1716, 1975.

[95] S. M. Kuo and D. R. Morgan, "Active noise control: a tutorial review," *Proceedings of the IEEE*, vol. 87, no. 6, pp. 943–973, 1999.

[96] W. S. Gan, S. Mitra, and S. M. Kuo, "Adaptive feedback active noise control headset: implementation, evaluation and its extensions," *IEEE Transactions on Consumer Electronics*, vol. 51, no. 3, pp. 975–982, 2005.

[97] Y. Song, Y. Gong, and S. M. Kuo, "A robust hybrid feedback active noise cancellation headset," *IEEE Transactions on Speech and Audio Processing*, vol. 13, pp. 607–617, 2005.

[98] S. M. Kuo, S. Mitra, and W.-S. Gan, "Active noise control system for headphone applications," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 2, pp. 331–335, 2006.

[99] L. Zhang, L. Wu, and X. Qiu, "An intuitive approach for feedback active noise controller design," *Applied Acoustics*, vol. 74, no. 1, pp. 160–168, 2013.

[100] S. Hu, R. Rajamani, and X. Yu, "Invisible speakers in home windows for simultaneous auxiliary audio playback and active noise cancellation," *Mechatronics*, vol. 22, no. 8, pp. 1031–1042, 2012.

[101] S. M. Kuo, I. Panahi, K. M. Chung, T. Horner, M. Nadeski, and J. Chyan, "Design of active noise control systems with the TMS320 family," *Texas Instruments*, 1996.

[102] L. Wu, X. Qiu, and Y. Guo, "A simplified adaptive feedback active noise control system," *Applied Acoustics*, vol. 81, pp. 40–46, 2014.

[103] H.-S. Vu and K.-H. Chen, "A low-power broad-bandwidth noise cancellation VLSI circuit design for in-ear headphones," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2013–2025, 2016.

[104] C. Y. Chang and S. T. Li, "Active noise control in headsets by using a low-cost microcontroller," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 5, pp. 1936–1942, 2011.

[105] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1," *NASA STI/Recon technical report n*, vol. 93, 1993.

List of Publications

International Peer Reviewed Journals

1. **M.T. Khan** and **R.A. Shaik**, "Optimal complexity architectures for pipelined distributed arithmetic based LMS adaptive filter," *IEEE Transaction on Circuits and Systems I, vol. 66, no. 2, pp. 630-642, Feb. 2019.*

2. **M.T. Khan**, **R.A. Shaik** and Surya Prakash Matcha, "Improved convergent distributed arithmetic based low complexity pipelined least-mean-square filter," *IET Circuits, Devices & Systems, April 2018, DOI: 10.1049/iet-cds.2018.0041* [Available Online]

3. **M.T. Khan** and **R.A. Shaik**, "An energy efficient VLSI architecture of decision feedback equalizer for 5G communication system," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 7, no. 4, pp. 569-581, Dec. 2017.*

4. **M. T. Khan** and **R.A. Shaik**, "High-Performance Hardware Design of Block LMS Adaptive Noise Canceller for In-ear Headphones," *IEEE Consumer Electronics Magazine, July 2018.* [Accepted]

Manuscripts Under Preparation

1. **M. T. Khan** and **R.A. Shaik**, "High-Throughput, Fast-Convergent and Low-Steady State Error Pipelined Architecture of Adaptive DFE," in *IEEE Transactions on Circuits and Systems II.*

2. **M. T. Khan** and **R.A. Shaik**, "High-Performance Pipelined Architecture of Distributed Arithmetic based LMS Adaptive Filter," in *IEEE Signal Processing Letters.*

3. **M. T. Khan** and **R.A. Shaik**, "Finite Precision Analysis of Two Non-Pipelined Distributed Arithmetic based LMS Adaptive Filters," in *IEEE Transaction on Circuits and Systems I.*

Conference Publications

1. **M.T. Khan** and **R.A. Shaik**, "Optimal complexity architectures for pipelined distributed arithmetic based LMS adaptive filter", International Symposium on Circuits and Systems (ISCAS 2019), Japan, May 2019 [Accepted].

2. **M.T. Khan** and **R.A. Shaik**, "Analysis and Implementation of Block Least Mean Square Adaptive Filter using Offset Binary Coding", International Symposium on Circuits and Systems (ISCAS 2018), Italy, pp. 1-5, May 2018.

3. **M.T. Khan** and **R.A. Shaik**, "Enhanced Convergence Distributed Arithmetic Based LMS Adaptive Filter Using Convex Combination", National Conference on Communication (NCC 2018), India.

4. **M.T. Khan** and **R.A. Shaik**, "Area and Power Efficient VLSI Architecture of Distributed Arithmetic Based LMS Adaptive Filter", 31st International Conference on VLSI Design and 17th International Conference on Embedded Systems (VLSID 2018), India. pp. 283-288, Jan. 2018.

5. **M.T. Khan** and **R.A. Shaik**, "VLSI Realization of Low Complexity Pipelined LMS Filter Using Distributed Arithmetic", in IEEE TENCON 2017, Malaysia, pp. 433-438, Nov. 2017.

6. **M.T. Khan** and **R.A. Shaik**, "A New High Performance VLSI Architecture for LMS Adaptive Filter using Distributed Arithmetic", IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2017), Germany, pp. 219-224, July 2017.

7. **M.T. Khan** , **R.A. Shaik**, "VLSI Implementation of Throughput Efficient Distributed Arithmetic Based LMS Adaptive Filter," in Springer Communications in Computer and Information Science, VLSI Design and Test. VDAT 2017, vol 711, Jun. 2017.

8. **M.T. Khan**, **R.A. Shaik** and B. Forrest, "Low Complexity and Critical Path Based VLSI Architecture for LMS Adaptive Filter Using Distributed Arithmetic", 30th International Conference on VLSI Design and 16th International Conference on Embedded Systems (VLSID 2017), India, pp. 127-132, Jan. 2017.

9. **M.T. Khan**, **R.A. Shaik** and A. Chatterjee, "Efficient Implementation of Concurrent Lookahead Decision Feedback Equalizer using Offset Binary Coding", in IEEE 20th International Symposium on VLSI Design and Test (VDAT 2016), India, pp. 1-6, May 2016.

10. **M.T. Khan** and **R.A. Shaik**, "Low Cost Implementation of Concurrent Decision Feedback Equalizer using Distributed Arithmetic", 1st India International Conference on Information Processing (IICIP 2016), India, pp. 1-6, Aug. 2016.

11. P.K. Sharma, **M.T. Khan** and **R.A. Shaik**, "An Alternative Approach To Design Reconfigurable Mixed Signal VLSI DA Based FIR Filter", IEEE Students Technology Symposium (TechSym 2016), India, pp. 284-288, Sep. 2016.

MOHD. TASLEEM KHAN

- AIM
  - To strive for attaining academic and research excellence through continuous learning
- PERSONAL DETAILS
  - *Born :* 17 June 1991 | Aligarh, India
  - *Contact Info. :* tasleem@iitg.ac.in, khantasleem91@gmail.com, +91-9085060735
- EDUCATION:
  - *2013-2019 :* Doctor of Philosphy (Ph.D) from Indian Institute of Technology Guwahati | Research area on DSP Architectures | Course work CPI: 9.450/10
  - *2009-2013 :* Bachelor of Technology (B.Tech) from Zakir Hussain College of Engineering and Technology, Aligarh Muslim University | Specialization in Electronics | Overall CPI: 9.355/10
- TECHNICAL SKILLS
  - *Simulation Tools :* Verilog, System Verilog, VHDL, C, MATLAB.
  - *Languages :* Windows, Linux, Latex (scripting language)
  - *Synthesis Tools :* Cadence RTL Compiler, Cadence Virtuoso, Synopsis Design Compiler
  - *Hardware Kits :* Xilinx FPGAs (Xilinx Spartan-3E XC3S500E, Xilinx ZYNQ - XC7Z020-1CLG84C), CPLD Kit (IIT Bombay) by Altera Quartus FPGA JTAG Software
- ACHIEVEMENTS
  - Reviewer: *IEEE* Transaction on Circuits and Systems I (TCAS I), *Springer* Circuits, Systems and Signal Processing (CSSP), International Symposium on Circuits and Systems (ISCAS).
  - *Secretary*, Departmental Postgraduate Programme Committee, (DPPC), EEE Dept., IIT Guwahati