Linköping Studies in Science and Technology
Dissertations, No 1716

Techniques for Efficient Implementation of FIR and Particle Filtering

Syed Asad Alam

Division of Computer Engineering
Department of Electrical Engineering
Linköping University
SE–581 83 Linköping, Sweden

Linköping 2016

Linköping Studies in Science and Technology
Dissertations, No 1716

Syed Asad Alam
syed.asad.alam@liu.se
www.da.isy.liu.se/
Division of Computer Engineering
Department of Electrical Engineering
Linköping University
SE–581 83 Linköping, Sweden

Copyright © 2016 Syed Asad Alam, unless otherwise noted.
All rights reserved.

Alam, Syed Asad
Techniques for Efficient Implementation of FIR and Particle Filtering
ISBN 978-91-7685-915-5
ISSN 0345-7524

Typeset with LaTeX
Printed by LiU-Tryck, Linköping, Sweden 2016

To my mother and wife

Abstract

Finite-length impulse response (FIR) filters occupy a central place many signal processing applications which either alter the shape, frequency or the sampling frequency of the signal. FIR filters are used because of their stability and possibility to have linear-phase but require a high filter order to achieve the same magnitude specifications as compared to infinite impulse response (IIR) filters. Depending on the size of the required transition bandwidth the filter order can range from tens to hundreds to even thousands. Since the implementation of the filters in digital domain requires multipliers and adders, high filter orders translate to a large number of these arithmetic units for its implementation. Research towards reducing the complexity of FIR filters has been going on for decades and the techniques used can be roughly divided into two categories; reduction in the number of multipliers and simplification of the multiplier implementation.

One technique to reduce the number of multipliers is to use cascaded sub-filters with lower complexity to achieve the desired specification, known as frequency-response masking (FRM). One of the sub-filters is a upsampled model filter whose band edges are an integer multiple, termed as the period L, of the target filter's band edges. Other sub-filters may include complement and masking filters which filter different parts of the spectrum to achieve the desired response. From an implementation point-of-view, time-multiplexing is beneficial because generally the allowable maximum clock frequency supported by the current state-of-the-art semiconductor technology does not correspond to the application bound sample rate. A combination of these two techniques plays a significant role towards efficient implementation of FIR filters. Part of the work presented in this dissertation is architectures for time-multiplexed FRM filters that benefit from the inherent sparsity of the periodic model filters.

These time-multiplexed FRM filters not only reduce the number of multipliers but lowers the memory usage. Although the FRM technique requires a higher number delay elements, it results in fewer memories and more energy efficient memory schemes when time-multiplexed. Different memory arrangements and memory access schemes have also been discussed and compared in terms of their efficiency when using both single and dual-port memories. An efficient

pipelining scheme has been proposed which reduces the number of pipelining registers while achieving similar clock frequencies. The single optimal point where the number of multiplications is minimum for non-time-multiplexed FRM filters is shown to become a function of both the period, L and time-multiplexing factor, M. This means that the minimum number of multipliers does not always correspond to the minimum number of multiplications which also increases the flexibility of implementation. These filters are shown to achieve power reduction between 23% and 68% for the considered examples.

To simplify the multiplier, alternate number systems like the logarithmic number system (LNS) have been used to implement FIR filters, which reduces the multiplications to additions. FIR filters are realized by directly designing them using integer linear programming (ILP) in the LNS domain in the minimax sense using finite word length constraints. The branch and bound algorithm, a typical algorithm to implement ILP problems, is implemented based on LNS integers and several branching strategies are proposed and evaluated. The filter coefficients thus obtained are compared with the traditional finite word length coefficients obtained in the linear domain. It is shown that LNS FIR filters provide a better approximation error compared to a standard FIR filter for a given coefficient word length.

FIR filters also offer an opportunity in complexity reduction by implementing the multipliers using Booth or standard high-radix multiplication. Both of these multiplication schemes generate pre-computed multiples of the multiplicand which are then selected based on the encoded bits of the multiplier. In transposed direct form (TDF) FIR filters, one input data is multiplied with a number of coefficients and complexity can be reduced by sharing the precomputation of the multiplies of the input data for all multiplications. Part of this work includes a systematic and unified approach to the design of such computation sharing multipliers and a comparison of the two forms of multiplication. It also gives closed form expressions for the cost of different parts of multiplication and gives an overview of various ways to implement the select unit with respect to the design of multiplexers.

Particle filters are used to solve problems that require estimation of a system. Improved resampling schemes for reducing the latency of the resampling stage is proposed which uses a pre-fetch technique to reduce the latency between 50% to 95% dependent on the number of pre-fetches. Generalized division-free architectures and compact memory structures are also proposed that map to different resampling algorithms and also help in reducing the complexity of the multinomial resampling algorithm and reduce the number of memories required by up to 50%.

Populärvetenskaplig Sammanfattning

Digitala filter är signalbehandlingsalgoritmer som används i många olika typer av applikationer och system. Som i de flesta fall finns det ett generellt intresse att göra saker enklare och effektivare. I denna avhandlingen studeras två olika klasser av filter och förbättringar föreslås för effektivare implementering av dessa filter.

Den första klassen av filter är så kallade FIR filter. Dessa kräver typiskt många operationer när det finns strikta krav på filtreringen. Då multiplikationer är klart mer komplexa än additioner, både var gäller area, tid och effektförbrukning, så fokuseras arbetet på dessa. Traditionellt finns det två spår för att förbättre detta: antingen minskar man antalet multiplikationer eller så förenklar man multiplikationerna. Metoder för bägge områdena föreslås i detta arbetet.

Ett effektivt sätt att konstruera FIR filter med väldigt smalt övergångsband, dvs avståndet mellan frekvenser som släpps igenom och som dämpas, är att använda frekvensmaskning. I denna typen av filter så använder man ett filter där många av multiplikationerna är noll och därmed inte behöver beräknas. Detta filter har nollorna fördelade i ett periodiskt mönster vilket leder till att beteendet i frekvensdomänen också blir periodiskt. Fördelen är att komplexiteten för att skapa ett väldigt smalt övergångsband skalar omvänt proportionellt med perioden, så ju högre period desto lägre komplexitet. Då det oftast inte är ett periodiskt filter man vill ha i slutänden så behövs det ytterligare filter som tar bort de oönskade delarna i frekvensdomänen. Dessa har typiskt högre komplexitet för högre period, så en lagom avvägning måste hittas.

Trots mycket tidigare arbete på att konstruera sådana filter har väldigt lite arbete lagts på att implementera dem effektivt. Här har vi speciellt tittat på fallet där datatakten, som bestäms av applikationen, och kretsens klockfrekvens, som bestäms av implementeringsteknologin, inte är samma. Specifikt det mest realistiska fallet att datatakten är lägre än klockfrekvensen. En arktitektur som tar hänsyn till det periodiska filtret har föreslagits och olika möjligheter har utretts i detalj. Resultaten visar att vid implementering på en FPGA så minskar mängden minne som används, tvärtemot vad man kan tro. Ett ytterligare resultat är att effekten minskar ca 80% av minskningen i antal multiplikationer.

Ett alternativ för att minska komplexiteten på multiplikationerna är att använda logaritmiska talsystem (LNS). I dessa blir multiplikationerna bara en addition av exponenterna. I detta sammanhanget har för första gången optimala filter konstruerats direkt i den logaritmiska domänen med ändlig ordlängd. Att kunna konstruera optimala filter i både den linjära och logaritmiska domänen är en förutsättning för att kunna göra korrekta jämförelser mellan filter implementerade i de bägge domänerna.

Ytterligare ett alternativ för att minska komplexiteten på multiplikationerna är att använda multiplikatorer med högre radix och dela vissa delar mellan flera multiplikatorer. Vid implementering av FIR filter uppkommer ett mycket gynnsamt fall för detta. I avhandlingen föreslår vi ett enhetligt sätt att konstruera och utvärdera denna typ av multiplikatorer. På så sätt visar vi att tidigare arbeten bara var specialfall av denna generella metod. Då tidigare arbeten inte visat hur parametrar valt eller i vissa fall ej ens insett kopplingen till multiplikatorer med högre radix, kan vi visa hur man bör välja parametrar för bästa effektivitet.

Till sist behandlas en helt annan typ av digitala filter. Dessa så kallade partikelfilter används för att skatta tillstånd i dynamiska system. Den kritiska delen för effektiv implementering här är omsamplingssteget. Vi förslår tre metoder för att förbättra implementeringen av detta. En direkt implementering av omsampling bygger på att man jämför innehållet i två minnen. Varje cykel läser man från ett av minnena, vilket beroende på resultatet av tidigare jämförelse. Den förbättrade metoden bygger på att man läser in extra data från det ena minnet och kan på så sätt utföra den total jämförelsen snabbare. Med bara ett extra jämförelseblock kan man statistiskt minska den kritiska tiden med 56%. Den andra metoden löser problemet med att normalisering av värdena som jämförs, vilket normalt kräver en division. Istället används enbart multiplikationer och som en bieffekt så kan vi även skapa sekvenser av sorterade slumptal på ett nytt effektivt sätt. Till sist visar vi att upp till hälften av minnet som används för att spara sekvenserna som ska jämföras kan sparas genom att beräkna resultaten i real-tid.

Acknowledgments

Praise be to Al-Mighty Allah, the most Compassionate, the most Merciful, who gave me an opportunity to contribute to the vast body of knowledge. Peace and blessings of Allah be upon the Holy Prophet Muhammad (Peace be upon Him), the last Prophet of Allah, who has always exhorted his followers to seek knowledge and whose life is the glorious model for the humanity.

There are a lot of people to whom I would like to express my gratitude. The following is certainly not exhaustive but an effort to thank those who had the most impact on my research and life in Linköping:

- My advisor, Dr. Oscar Gustafsson, for having confidence in me by giving me an opportunity to complete my PhD. I am greatly indebted to him for his inspiring and valuable guidance, enlightening discussions, his patience when ever I was short of his standards and constant encouragement in difficult times, kind and dynamic supervision through out and in all the phases of this thesis. Working and learning from him was always a pleasure. Thank you Oscar.
- My co-supervisor, Dr. Kent Palmkvist, for help with FPGA, VHDL and Linux related issues.
- The former and present colleagues at the old Division of Electronics Systems, Department of Electrical Engineering, Linköping University for creating a very friendly environment. They always were kind enough to do their best to help you.
- The former and present colleagues at the Division of Computer Engineering, Department of Electrical Engineering, Linköping University for making me feel welcome when I joined their division and helping out when needed.
- A special thanks to our current and past secretary Gunnel Hässler and Susanna von Sehlen for helping out in various administrative tasks.
- A special thanks to Doktorand Syed Ahmed Aamir for his support, both materialistic and spiritual, during my early days in Linköping which immensely helped me in settling in this city.
- My present and former room-mates, Dr. Fahad Qureshi, Doktorand Carl Ingemarsson and Doktorand Fahim-ul-Haque for putting up with me,

building a good working environment and having discussions on mundane and technical issues.

- Dr. Muhammad Abbas for his help and guidance at the start of my PhD studies.
- Doktorand Muhammad Touqir Pasha and Doktorand Fahim-ul-Haque for proof reading and giving valuable advice while I was writing my thesis.
- Dr. Fahad Qazi, Doktorand Muhammad Touqir Pasha, Dr. Jawad ul Hassan, Dr. Hafiz Muhammad Sohail, Dr. Nadeem Afzal, Dr. Irfan Kazim, Dr. Usman Dastageer, Dr. Muhammad Junaid and others for building a nice social circle without which it would have been hard for me and my family to live here.
- Finally a special thanks to my family
  - My mother, Tanweer Alam, for her immense love, sacrifices, guidance, support and upbringing. Without her I would not be where I am. Without her life means nothing to me. It was on her encouragement that I took the decision to pursue PhD even though living so far apart was always troubling for her. Thanks Maa, because without your prayers and support, I would not have been able to complete my studies.
  - My wife, Eyshaa Zehra, you have been a true life partner, bearing the load of taking care of home and children while I spent long hours in office, for always showing love, patience, affection to me and for your immense support and cooperation. You made taking a lot of tough decisions easy for me.
  - A special thanks to my children, Muhammad and Arwaa, whom I am sure will read this when they grow up, for taking away all the tiredness and stress with their beautiful smiles, small giggles and playful gestures while welcoming me home.
  - My whole extended family in Pakistan, Canada and the U.S.A, my in-laws, my aunts and uncles, my cousins etc., for always being there for me and for making me feel special.
- To those not listed here, I say profound thanks for bringing pleasant moments in my life.

Syed Asad Alam,
January 21, 2016,
Linköping Sweden.

Abbreviations

2C	Two's complement
ASIC	Application specific integrated circuit
ASIP	Application specific instruction set processor
BILP	Binary integer linear programming
BLE	Basic logic element
BMI	Brain machine interface
CFGLUT	Configurable LUT
CLB	Configurable logic block
CPA	Carry propagate adder
CPLD	Complex programmable logic device
CPU	Central processing unit
CSA	Carry save adder
CSD	Canonic signed digit
CSE	Common subexpression elimination
CU	Control unit
DA	Distributed arithmetic
DAG	Directed acyclic adder graphs
DF	Direct form
DFT	Discrete fourier transform
DSP	Digital signal processing

EEPROM Electrically erasable programmable read only memory

EPROM Electrically programmable read only memory

FF Flip flop

FFA Fast FIR

FFT Fast fourier transform

FIFO First in first out

FIR Finite-length impulse response

FPGA Field programmable gate array

FRM Frequency-response masking

FSM Finite state machine

GPC Generalized parallel counter

GPU Graphical processing unit

HDL Hardware description language

HMM Hidden Markov model

HPM High performance multiplier

IC Integrated circuit

ICAP Internal configuration access port

IDFT Inverse DFT

IFIR Interpolated FIR

IIR Infinite impulse response

ILP Integer linear programming

IMHA Independent Metropolis Hastings algorithm

LNS Logarithmic number system

LP Linear programming

LSB Least significant bit

LUT Look-up table

MAC Multiply-accumulate

MC	Monte Carlo
MCM	Multiple constant multiplication
MILP	Mixed integer linear programming
MPGA	Mask programmable gate array
MSB	Most significant bit
MSD	Minimal signed digit
MSE	Mean square error
NP	Non-deterministic polynomial
OPR	Overlapped partial resampling
PAL	Programmable array logic
PAG	Pipelined adder graph
PE	Processing element
PLA	Programmable logic array
PLD	Programmable logic device
PMCM	Pipelined MCM
PROM	Programmable read only memory
RCA	Ripple-carry adder
RNA	Resampling with nonproportional allocation
RNS	Residue number system
RPA	Resampling with proportional allocation
RPAG	Reduced pipelined adder graph
RSG	Reduced slice graph
RSR	Residual systematic resampling
RTL	Register transfer level
S-ASIC	Structured ASIC
SCM	Single constant multiplication
SD	Signed digit

SEU	Single-event upset
SIMD	Single instruction multiple data
SIS	Sequential importance sampling
SM	Signed magnitude
SMC	Sequential Monte Carlo
SPT	Signed power of two
SRAM	Static random access memory
SSF	Single stage FIR
STM	State transition model
TDF	Transposed direct form
TID	Total ionizing dose
ulp	Unit of least significant position
VLSI	Very large scale integrated

Contents

Part I

Background

Chapter 1

Introduction

The topic of this thesis is *techniques for efficient implementation of finite-length impulse response (FIR) and particle filtering.* It encompasses four different contributions towards fulfilling the requirements of this thesis:

- Proposing an architecture for the implementation of time-multiplexed frequency-response masking (FRM) filters and analyzing different memory organization and access schemes involved in this architecture.
- Design of FIR filters by optimizing the filter coefficients in the logarithmic number system (LNS) domain
- Analysis and unified design of different computation sharing multiplication schemes and their applications to complexity reduction in FIR filters
- Proposing a scheme and corresponding architecture for the reduction in the latency of the resampling stage of the particle filter, a generalized division-free architecture and compact memory structure for its implementation

This chapter aims to introduce these research topics and motivate their relevancy in modern day applications. It also presents all the research publications that have resulted as part of the research work done and the thesis organization.

1.1 Motivation

FIR filters are one of the most widely used filters and have played a leading role in frequency selective digital filtering since its inception [1–4]. They are inherently stable and free of limit cycle oscillations caused by using finite word length as long as they are not implemented in a recursive manner. They can be easily designed to be linear phase and hence achieve constant group delay, which helps in preserving the integrity of the information carrying signals and is crucial in communication signals [5].

However, FIR filters suffer from a major disadvantage as they require a higher order to achieve narrow transition bands as compared to infinite impulse response (IIR) filters. This results in more arithmetic operations like multipliers and adders and also an increase in the number of delay elements. Since the filter order of an FIR filter is inversely proportional to the transition band-width, any decrease in band-width increases the computational complexity of the FIR filters significantly [3].

Due to this high computational complexity of FIR filters, research has been on going for decades to reduce it [3, 6]. The proposed techniques to reduce the computational complexity can be broadly divided with respect to the optimization goals; reduction in the number of multipliers [7–27] and reduction in the multiplier complexity [28–42]. The contributions of this thesis are towards both research fronts.

Another topic covered in this thesis the resampling step of particle filtering. The execution of the resampling step is a bottleneck as it cannot be executed in parallel with the other steps in the particle filtering. Furthermore, the multinomial resampling algorithm suffers from high computational cost because its implementation requires a search through two large sequence of numbers and their normalization. A number of resampling algorithms has been proposed that deals with the parallelism problem [43, 44] but none has been proposed that reduces the computational cost of multinomial resampling algorithm. Furthermore, the bottleneck remains in the implementation of traditional resampling algorithms [45]. The work presented in this thesis proposes solutions to reduce the latency which can be used to increase the parallelism and reduce the computational cost of multinomial resampling.

1.1.1 Reduction in Number of Multipliers

A common approach to reduce the number of multipliers is to realize the filter through the cascade of sub-filters. These sub-filters can either be different [46–51] or identical [13, 52–55]. The main premise of these techniques is that by the use of sub-filters, less stringent requirement with regards to transition bandwidth will be placed on these sub-filters, thus reducing the number of distinct multipliers at the cost of an increased order. The non-identical sub-filter technique uses building blocks having different powers of z^{-1}. The identical sub-filter technique using identical building blocks and connects them with the aid of additional adders and multipliers [54]. One of the most popular techniques utilizing different powers of z^{-1} is the FRM technique [48]. These techniques achieve a reduction in the number of multiplications by making use of a combination of wide transition-band filters generally termed as model filters and masking filters. The model filter is first upsampled by L by the insertion of $L-1$ zeros between every coefficient, resulting in a filter called the periodic model filter, which compresses the spectrum of the filter to form the desired transition band of the target filter but produces images. The images are then filtered or masked out by the masking filters. The non-identical sub-filters can

be combined in different ways to either produce arbitrary, narrow or wide-band filters. For narrow-band FIR filters, when the pass-band is less than $\pi/2$, the overall filter structure can be further reduced to just one periodic model and one masking filter [46, 47, 50]. From this narrow-band structure, efficient wide-band structure can also be derived by the use complementary filter while the technique of using identical sub-filters with different up-sampling factors have also been proposed to synthesize narrow and wide-band FRM filters [13].

The design of FRM filters have received considerable attention but only a few attempts have been made towards the dedicated implementation of these filters [14, 16, 22, 24, 56–58]. Furthermore, since contemporary state-of-the-art implementation platforms like application specific integrated circuit (ASIC) or field programmable gate array (FPGA) allow the circuits to be clocked at hundreds of MHz upto a few GHz and only rarely do the sampling rate requirements of DSP systems correspond to these high frequencies, time-multiplexed architectures are crucial. These architecture re-use different resources thereby reducing the number of such resources. Since typically FRM filters have more delay elements than a single-stage implementation of FIR filters, it is necessary to study what affects time-multiplexing has on not only the number of multipliers but also on the mapping of these delay elements to memories. Paper A presents contributions towards the implementation of time-multiplexed FRM filter where different memory organizations, access schemes, affect of pipelining on these schemes and the effect of time-multiplexing on the optimal value of L which gives the minimum number of multipliers are analyzed.

1.1.2 Reduction in Multiplier Complexity

The other method to reduce the complexity in FIR filters is to reduce the complexity of the multipliers. There are different methods to do this and can be broadly divided into three categories

1. Single/multiple constant multiplication
2. Number representation
3. Computation sharing

These three techniques can be combined in different ways to further optimize the multiplications and can also be combined with the techniques outlined in Section 1.1.1 [32, 59–61].

Single constant multiplication refers to the optimization of the filter coefficients in the signed power of two (SPT) space [62, 63] because each coefficient can be represented as a sum of a limited number of SPT terms. The multiplication of each coefficient with the input data can either be implemented as a general multiplier or by using a fixed shift-add network because typically the filter coefficients are constant. The number of adders in a shift-add network is primarily determined by the number of non-zero terms in the representation of a filter coefficient and a reduction of these non-zero terms is what is referred to as reduction or minimization in the number of SPT terms and in-

teger linear programming (ILP) has been a popular technique to achieve this minimization [62, 64–70].

Further reduction can be achieved by combining single constant multiplication (SCM) with across multiple constants, known as multiple constant multiplication (MCM). MCM is applicable to the transposed direct form (TDF) FIR filter whose operation can be modeled as an MCM problem [38]. The reduction in the number of adders is achieved by extracting common subexpressions within a filter coefficient and across multiple coefficients. The techniques proposed for MCM can be broadly divided into two categories, common subexpression elimination (CSE) [71] and the adder graph technique [28].

The CSE technique is based on pattern matching techniques and the result depends on the initial representation of the filter coefficients where typically canonic signed digit (CSD) is used as the number representation [29, 30, 72–74]. This is because CSD number representation has only around 33% non-zero digits relative to the word length as compared to $2's$ complement number representation which has approximately 66%. These techniques have been combined with integer or mixed integer linear programming [6, 60, 62, 70, 71, 75–78] and minimum spanning trees [31, 33, 79] to yield even better results in terms of number of additions required to realize these coefficients.

The adder graph technique is value based and independent of the underlying number representation [71]. Here partial sums are symbolically represented in the nodes of the graph while the edges are used to represent the shift amounts [28]

Furthermore, different number representations have been used to take advantage of the inherent simplification of multiplication in them, like residue number system (RNS) [80–84] and LNS [85–92]. Most efforts towards utilizing LNS for digital filters have focused on either implementing the non-linear conversion to and from LNS, selecting the logarithm basis, or implementing the LNS addition and subtraction efficiently [92–97]. The finite word length filter design has not been considered, but instead relied on rounding the obtained coefficients to the nearest LNS number.

Paper B presents the contribution in this area where an integer linear programming (ILP) approach to design optimal finite word length linear-phase FIR filters in the LNS domain is proposed. Here, instead of optimizing filters in the linear domain and converting them into LNS with rounding in the LNS, the filter is directly optimized in the minmax sense in the LNS domain with finite word length constraints.

Another source of reducing the multiplication complexity is by sharing some parts of the actual computation when Booth [98–101] or high-radix multiplication [102] is used. A number of proposed techniques have used alphabets to pre-compute multiples of the multiplicand to be selected based on multiplier bits and shared this pre-computer for all multiplications in the TDF FIR filter [103–109]. However, this approach is a special case of high-radix multiplication when the radix is 16. Similar sharing can also be performed with Booth

multiplication and in Paper C the attempt is made to analyze both of these multiplications with respect to computation sharing. Furthermore, different design choices have been discussed that are available while designing different parts of the multiplier.

1.1.3 Improved Particle Filter Resampling Architectures

The final contribution of this thesis is the proposal of efficient architectures for the resampling step in particle filtering. In particle filters a weighted set of particles is propagated that approximates the probability density of the unknown state conditioned on the observations. This is achieved by the recursive generation of random measures which are composed of particles drawn from relevant distributions and of importance weights of the particles [110, 111]. Particle filters find application in a wide variety of complex problems including target tracking, computer vision, robotics and channel estimation in digital communication or any application involving large, sequentially evolving datasets [112–115].

Among the three steps that accomplish particle filtering, resampling is the most crucial to obtain an efficient implementation of the estimation. It presents a bottleneck in that this step of resampling cannot be executed in parallel with other steps. A number of research work have focused on different resampling algorithms [43, 44, 116–121]. However, these contributions do not discuss the multinomial resampling which is the most basic form of the resampling step. The current work in Paper D looks into improving different aspects of the resampling stage. The first technique proposed is the proposal of a generalized division free architecture and compact memory structure which helps in complexity of the multinomial resampling algorithm. In addition to this, a technique has been proposed to reduce the latency of the resampling stage along with the required hardware details.

1.2 List of Publications

This thesis contains research work done between March 2010 and January 2016, and has resulted in the following publications.

Paper A

- S. A. Alam, and O. Gustafsson, "On the implementation of time-multiplexed frequency-response masking filters," *IEEE Trans. Signal Process.*, under second review.

Preliminary versions of the above work have been published in

- S. A. Alam and O. Gustafsson, "Implementation of time-multiplexed sparse periodic FIR filters for FRM on FPGAs," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, Rio de Janeiro, Brazil, May 15–18, 2011.

- S. A. Alam and O. Gustafsson, "Implementation of narrow-band frequency-response masking for efficient narrow transition band FIR filters on FPGAs," in *Proc. NORCHIP*, Lund, Sweden, Nov. 14–15, 2011.

Paper B

- S. A. Alam, and O. Gustafsson, "Design of finite word length linear-phase FIR Filters in the logarithmic number system domain," *VLSI Design*, vol. 2014, Article ID 217495, 14 pages, 2014.

Paper C

- S. A. Alam, and O. Gustafsson, "A unified approach to the design and implementation of computation sharing multipliers," *Manuscript.*

Paper D

- S. A. Alam, and O. Gustafsson, "Improved particle filter resampling architectures," *IEEE Trans. Signal Process.*, under review.

A preliminary version of the above work has been published in

- S. A. Alam and O. Gustafsson, "Generalized division-free architecture and compact memory structure for resampling in particle filters," in *Proc. Europ. Conf. Circuit Theory Design (ECCTD)*, Trondheim, Norway, Aug. 24–26, 2015.

1.2.1 Other Publications

Contributions have also been made in the following publication but the contents are not relevant to the topic of this thesis.

- F. Qureshi, S. A. Alam and O. Gustafsson, "4k-point FFT algorithms based on optimized twiddle factor multiplication for FPGAs ," in *Proc. IEEE Asia Pacific Postgraduate Research on Microelectron. Electron.*, Shanghai, China, Sept.22–24, 2010, pp. 225–228.

1.3 Thesis Organization

The thesis is organized in two parts. The first part establishes the background of the work. It outlines and summarizes the previous research work that has been done in the related field and how this work has brought forward the research front. The second part contains the collection of the research publications outlined above.

The first part of the thesis is organized in five chapters. Chapter 2 outlines various areas connected with the implementation of digital signal processing (DSP) algorithms. Specifically it divides these areas into thee distinct fields, (a) implementation platforms, (b) arithmetic operations and (c) number systems. The two main platforms discussed in this chapter are ASIC and FPGA. It has

been shown that the way these platforms are utilized for implementing DSP algorithms, their key features and their differences. It moves on to describing the key arithmetic operations involved in the implementation, i.e., adders, multipliers and number systems used to represent data and their effect on the overall performance of DSP algorithms.

In Chapter 3, a backround of FIR filters is presented. Advantages and challenges involved in the use of FIR filters is highlighted. To meet these challenges, a number of techniques have been proposed to design and optimize FIR filters with respect to its computation complexity and this chapter attempts to highlight key areas important towards the reduction in the implementation cost of FIR filters.

An overview of the particle filter algorithm with special focus on the resampling step is presented in Chapter 4. Different algorithms to implement the resampling step is discussed in this chapter while also highlighting different hardware architectures presented in various works.

Finally, Chapter 5 concludes the background part and presents future challenges in the considered work.

Chapter 2

Implementation Aspects of Digital Signal Processing Algorithms

2.1 Introduction

The design and synthesis of a DSP algorithm, based on a set of specifications, is the first step towards the realization of the complete DSP system. The second step is the mapping of the algorithm to a set of hardware resources like memories, processing elements (PEs), control and interconnection network. The connection between these four fundamental elements is shown in Fig. 2.1. The two most important operations of these PEs, specially for implementation of DSP algorithms, are the adder and multiplier. The memory also has an important role to play in the implementation of these algorithms [122]. The third and the final step is the implementation of these resources on some hardware platform which require the data to be represented using finite word length based on some number system.

The organization of this chapter is as follows: in Section 2.2, various hardware platforms available are described along with their main features, advantages and drawbacks. Furthermore an overview of the two primary arithmetic operators involved in DSP algorithms, the adder and the multiplier is presented in Section 2.3. Number representation plays an important role in the implementation of DSP systems and affects the overall performance and cost of implementing them. A brief overview of different number systems is presented in Section 2.4.

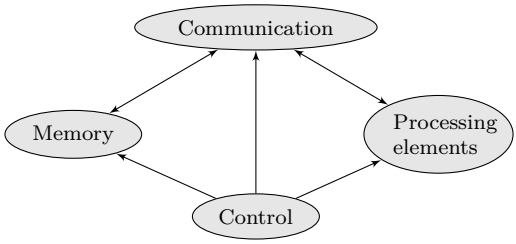

Figure 2.1: Hardware resources. Redrawn with permission from [122].

2.2 Implementation Platforms

For real world applications, all algorithms need to be implemented on some kind of hardware platform. With the growing trend of very large scale integrated (VLSI) designs, a number of platforms are available. Each of these platforms have different levels of programmability. The platform with the highest degree of programmability is the general purpose micro-processors such as Intel® Core® processors. On the other hand, dedicated hardware accelerators, which have a fixed functionality occupy the other end of the spectrum of devices. In general, all implementation platforms can be categorized under ASICs because every platform is associated with a specific application, only the granularity is different. In terms of integrated circuit (IC) fabrication technologies, an IC can be classified as full-custom, semi-custom and programmable ICs [123].

The full custom IC is a layout-based technique where the circuit is drawn manually at the transistor level. Highest layout efficiency and maximum circuit performance is achieved at the cost of high initial design effort. These circuits and layouts are collected in libraries together with automatic generation to form *mega-cells*, for e.g., state-of-the-art micro-processor cores.

However, such an effort is not necessary for majority of applications. To achieve the required performance in these applications, circuits can be composed of pre-designed cells. These cells are made up of elementary logic gates and storage elements. These cells are automatically placed and routed using dedicated layout strategies. These techniques take the form of either standard-cell, gate-array/sea-of-gates and FPGA. The design of circuits targeting this level of abstraction is generally carried out using hardware description languages (HDLs) like Verilog or VHDL. Using these languages, the circuit may be described at the behavioral level, register transfer level (RTL) level or structural level.

Standard-cells are themselves a full-custom design while gate-arrays/sea-of-gates consist of preprocessed wafers with predefined but unconnected transistors. Only the metallization is customized which defines the interconnect between the transistors and is often called an mask programmable gate array (MPGA) [124]. In FPGAs, an array of logic blocks and routing channels are configured or programmed using a configuration stored in a static memory. Generally, in the wider electronics engineering community, standard-cell based layout is referred

to as an ASIC and this terminology will also be used here.

Another implementation platform, called structured ASIC (S-ASIC), marries the benefits of FPGAs and ASICs in terms of cost, capabilities, turn around time and ease of design [125]. S-ASICs typically contain prefabricated elements which either implements generic logic (called a *tile*) or special logic like configurable I/O, microprocessor cores, embedded memories and others [125]. Another key differentiator of S-ASIC is the availability of prefabricated metal layers and the design only needs to specify a few metallization layers to complete the device.

From a functionality point of view, a central processing unit (CPU) is only used for general purpose processing. For computation intensive tasks, the main work load is transferred to hardware accelerators. These hardware accelerators can again be programmable, for e.g., a graphical processing unit (GPU) or fixed, for e.g., ASIC [126]. From a signal processing point of view, instead of using a CPU, there is a need to use a more specific processor which is flexible yet not very generic like a CPU. This need is filled by either a digital signal processors (DSPs) or application specific instruction set processors (ASIPs) [127].

DSPs and ASICs occupy the two ends of the spectrum of platforms used to implement DSP algorithms. DSPs are flexible but slow and power hungry while ASICs are non-flexible but very fast and power efficient. FPGAs fill the gap between these two extreme ends. They provide a flexibility not achievable in an ASIC while being faster and consume less power than a DSP [128, 129].

Here the focus will be on standard-cell based ASICs which is described in some detail, in Section 2.2.1 while FPGAs find their description in Section 2.2.2.

2.2.1 Application Specific Integrated Circuits

A standard-cell based ASIC, referred to as an ASIC here, uses pre-designed standard cells, like logic gates (AND, OR, etc.), multiplexers, flip flop, half and full adders and tri-state buffers, to implement a system.

These cells, arranged as rows, may also be combined with megacells like microcontrollers, microprocessors and memories. These standard cells are placed by the ASIC designer who also defines the interconnect. These standard cells are constructed using full-custom design methods and their use allow the same performance and flexibility as a full-custom ASIC but reduces design time and risk. However, all the mask layers of an ASIC are unique and customized for a particular design.

An ASIC vendor provides all cells in a library called a standard cell library. Each cell in a library contains the following

- A physical layout
- A behavioral model
- A Verilog and/or VHDL model
- A detailed timing model
- A test strategy
- A circuit schematic

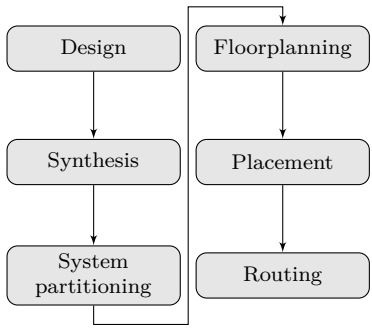

Figure 2.2: ASIC design flow [124].

- A cell icon (symbol)
- A wire-load model
- A routing model

The behavioral model is needed for simulation while the timing model is required to determine the performance of an ASIC. Wire-load models are used to estimate the parasitic capacitance of wires. Circuit schematic and cell icon is used in schematic based design entry.

A ASIC Design Flow

The ASIC design flow is divided into a number of steps, as shown in Fig. 2.2. System partitioning can be done before the design steps and there may be iterations between the different steps. The design entry is typically made using an HDL while synthesis translates the HDL description into gates.

System partitioning is used to divide a system into multiple sub-systems while floorplanning is used to estimate the physical sizes and set the initial relative location of the various blocks. The location of clock- and power networks and that of the input/output pins are decided in the same stage. The location of the logic cells is defined in the placement step while setting aside space for the interconnect and finally routing makes the connections between the logic cells.

B Implementation of Adders and Multipliers on ASICs

Typically, the standard cell library has four main types of cells [124]. These are combinational, sequential, data-path and I/O cells. Each of these cells come with different drive strengths, power specification, capacitance and delay at different temperatures, supply voltages and threshold voltages (V_T).

Combinational cells range from simple inverter, AND, OR, NAND cells to more complex cells which contain a comhination of different cells. Cells for efficiently implementing multiplexers, transmission gates and tri-state buffers are also a part of the combination cell library. Sequential cells typically contain

different types of latches and flip-flops. In addition to the different properties mentioned above, they are also available with different timing constraints like pulse width, hold and setup times. Cells for scan-based flip-flops which are useful for data-path scanning used in testing of VLSI circuits are also available.

As mentioned earlier, the primary arithmetic operators used in DSP algorithms are adders and multipliers. Cells that implement them are part of the data-path cells [124]. Data-path cells also implement operations that use multiple signals across a data bus. Full-adder and half-adder cells are typically part of any standard library. They are available with different delays between inputs and outputs, specially between carry-in and carry-out as it is part of the carry chain. However, it is not necessary that these cells are used in the actual implementation, specially if there are tighter timing constraints or logic surrounding the adder operation, which is typically the case. The synthesizer may well use other cells or optimize the logic to implement it more efficiently.

In multiplication, the partial product generation stage will use different types of combinational cells. Depending on the timing and area constraints and how the design has been entered using any of the HDLs, summation of the partial products may be implemented using either summation trees or array adders. Key components of these summation structures is typically the carry-save adder which avoids carry-propagation of the ripple-carry adder. Other elements that can be synthesized using data-path cells are multi-input NAND gates, registers, multi-bit multiplexers, or incrementers/decrementers [124].

2.2.2 Field Programmable Gate Arrays

An FPGA is an integrated circuit designed to be configured by the customer or a designer after being manufactured making them programmable. The design entry, similar to that of an ASIC, is typically made using HDLs. Its configuration is generally specified using an HDL, which is also used for designing ASICs. It significantly reduces the design time while also reducing the prototyping cost by more than a few decades. FPGAs can be used to implement any logic function, either combinational or sequential. The ability to re-program without going through the whole fabrication cycle of an ASIC provides a significant advantage, especially for low-volume applications. They can also be reconfigured if a problem is identified [130, 131]. A general architecture of an FPGA consisting of blocks implementing digital logic, interconnect resources and I/O blocks is shown in Fig. 2.3. In more advanced FPGAs, some of the logic blocks are replaced by specialized blocks like memories and multipliers.

A History

The emergence of FPGAs is connected to the emergence of early programmable devices which employed regular architecture and flexible functionality. Earlier types of such devices consisted of cellular and “cutpoint” cellular arrays [132] where the functionality of each logic cell was programmed in the field through

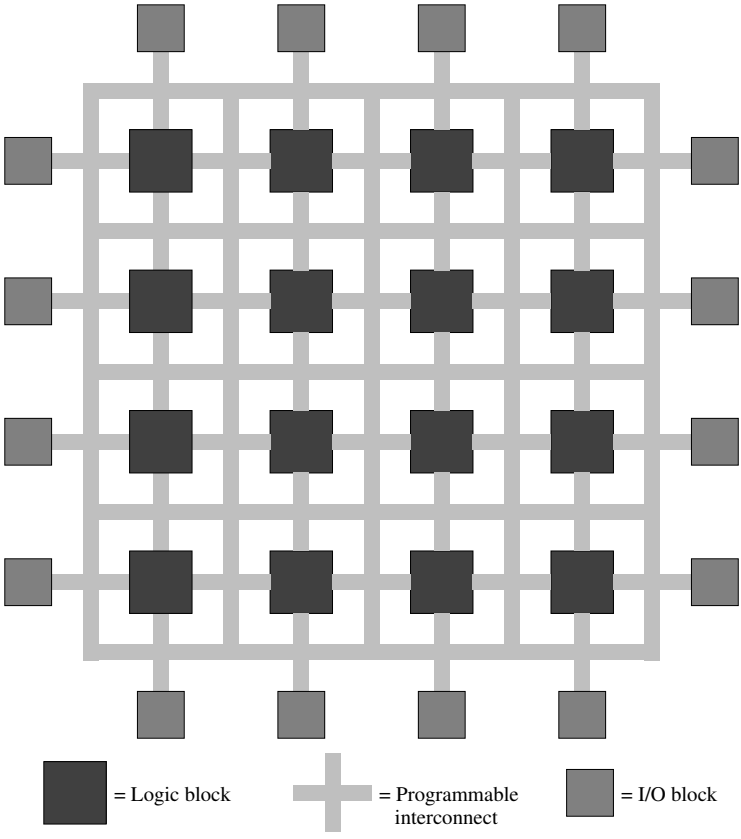

Figure 2.3: A general FPGA architecture.

the use of programming currents or photo-conductive exposure [132]. The next device, made available in the 1970s, was the programmable read only memories (PROMs) with its two variants of mask-programmable and fuse-programmable ROMs. However, the area of a PROM is exponentially dependent on the number of address inputs and thus unfeasible for large number of address inputs.

The first programmable device which had a two-level logic structure was the programmable logic array (PLA) which had a fixed AND plane and a programmable OR plane. Sufficient flexibility is provided by a programmable AND followed by a fixed OR plane, giving rise to the programmable array logic (PAL) device [133]. However, in order to implement sequential circuits, registers (flip flops (FFs)) are added to PALs to form a programmable logic device (PLD). Multiple PLDs are placed on a single chip to form a complex programmable logic device (CPLD) where they are connected using programmable routing devices [128].

From these programmable devices emerged the FPGA which not only has multi-level programmable logic but also has programmable interconnect. Different claims have been made to the origin of the first FPGA [129, 134–137], however, the first modern era FPGA was introduced by Xilinx in 1984 [135] with devices like XC2064, XC4000 and XC6200. These FPGAs consisted of an array of configurable logic blocks (CLBs) and contained around 64–100 such blocks with 3-input look-up tables (LUTs) and 58 inputs and outputs. With time, the complexity of FPGA has grown to include hundreds of thousands of such blocks in addition to large specialized blocks like memories, multipliers and serial interfaces which has greatly expanded the capability of these devices and laid the foundation of a new technology and market [129, 138–140].

Xilinx continued unchallenged and had a quick growth from 1985 to the mid-1990s, when competitors came up, reducing its market-share significantly. The 1990s were an explosive period of time for the growth FPGAs, both in sophistication and the volume of production. In the early 1990s, FPGAs were primarily used in telecommunications and networking. By the end of the decade, FPGAs found their way into consumer, automotive, and industrial applications [140].

In the first decade of the new century, extremely complex FPGAs were marketed, specially by Xilinx and its main competitor Altera. Platforms like Virtex and Kintex series by Xilinx and Cyclone and Stratix series by Altera enabled designers to implement extremely complex applications on FPGAs.

B FPGA Programming

As shown in Fig. 2.3, the architecture of a FPGA is similar to a MPGA. MPGAs are mask programmable which does not give the same flexibility as the field programmability of FPGA [137]. This programming of FPGAs is done through programmable switches of which there are a number of types. Historically, the approaches that have been used to program FPGA include [128, 129] electrically programmable read only memory (EPROM) [141], electrically erasable

programmable read only memory (EEPROM) [142, 143], flash [144], static random access memory (SRAM) [135] and anti-fuses [145]. Of these programming technologies, only the flash (Microsemi, Actel), static memory (Xilinx, Altera, Lattice) and anti-fuse (Actel) are popular in modern FPGAs, depending on the type of applications they are employed for.

Typically, SRAM based FPGA are for more mainstream applications like communication and signal processing while flash is used for low power applications [146, 147]. Anti-fuse FPGAs, meanwhile, find applications in space applications where they have been shown to be immune to single-event upsets (SEUs) and degrading of the characteristics due to total ionizing dose (TID) [148]. However, anti-fuse FPGAs are only one-time-programmable, a significant drawback from FPGA perspective.

C Basic Building Blocks

The basic building blocks of an FPGA, as shown in Fig. 2.3, consists of the following [128, 129, 137]:

- configurable logic block (CLB)
- Programmable interconnect
- I/O block

A single FPGA CLB can be as simple as a transistor [149] or as complex as a microprocessor [129]. However, there are inherent problems with either of these two extremes. The kind of fine-grained programmability provided by using transistor as a CLB will entail large amounts of programmable interconnect resulting in poor area efficiency, low performance and high power consumption. On the other end, think of implementing a small adder or multiplier using a microprocessor. The inherent inefficiency is visible illustrating the problems of architectures that are too coarse-grained.

In between the fine and coarse-grained architectures lies a full spectrum of CLB choices that are based on one or more of the following [137]

- NAND gates [150]
- Interconnection of multiplexers [139]
- LUTs [135]
- Wide-fanin AND-OR gates [151]

The CLB consisting of a pair of transistors in shown in Fig. 2.4 [149]. Similarly, the multiplexer based FPGA from Plessey [137] is shown in Fig. 2.5 [150]. These are the examples of fine-grained CLBs. Coarse-grained blocks include a multiplexer, LUT and wide-fanin gates based CLBs.

The multiplexer based CLB from Actel, shown in Fig. 2.6 [139, 152], is based on a multiplexer's ability to implement various logic functions by connecting its input to either some constant value or to a signal [153]. The functionality of a LUT based FPGA is similar to distributed arithmetic (DA) where a LUT is used to implement a truth table [128]. It requires a memory with 2^n locations to implement a n-input function in a LUT. This arrangement is shown in Fig. 2.7 [128]. Typically, a CLB is also used to implement sequential logic thus it will

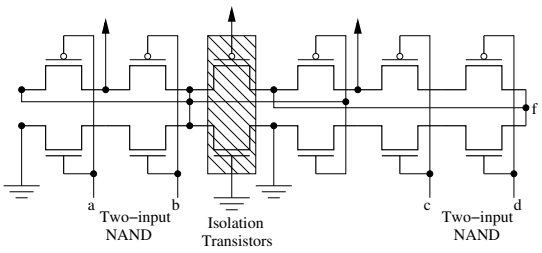

Figure 2.4: Transistor based CLB implementing $f = ab + c'd'$.

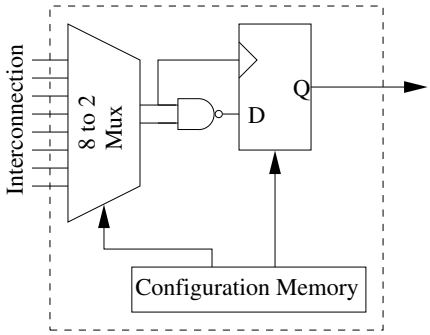

Figure 2.5: The CLB from Plessey [137] ©1993 IEEE.

have clocked circuits such as flip-flops and latches. A very basic arrangement of this is shown in Fig. 2.8 [128, 129].

The architecture of different types of CLBs shown in these figures are very basic. State-of-the-art FPGAs have very advanced and complex CLBs. One of the earliest FPGAs by Xilinx, the XC3000, had a very complex logic block, illustrated in Fig. 2.9. It contains a 5-input LUT which can also be configured as two 4-input LUTs [129].

As the years progressed, the size of the LUT increased. However, it was important that the effect of this increased size on area and speed is explored. It was shown in [155] that as the LUT size increases, the number of LUTs required to implement the circuit decreases. The cost of this decrease is an increase in the area cost of implementing the logic and routing for each block. A product of these two cost metrics shows that initially there is a decrease in the total area before increasing as LUT size is increased [129].

One alternative way to change this level of granularity is to use multiple LUTs in one CLB, referred to as cluster in [129]. A number of basic logic elements, as the one shown in Fig. 2.8, are grouped together and a local interconnect structure is used to connect them programmably. This arrangement is shown in Fig. 2.10, which transformed the increase in the logic and routing area from exponential to quadratic [129].

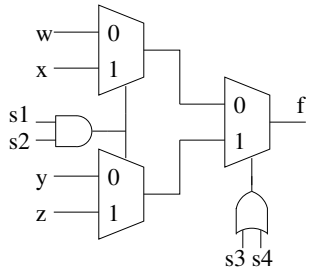

Figure 2.6: Multiplexer based CLB from Actel [137] ©1993 IEEE.

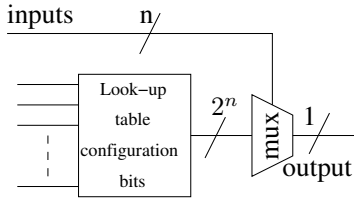

Figure 2.7: LUT based CLB. Redrawn with permission from [128].

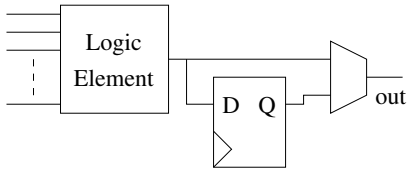

Figure 2.8: LUT based CLB with a flip-flop. Redrawn with permission from [128, 129].

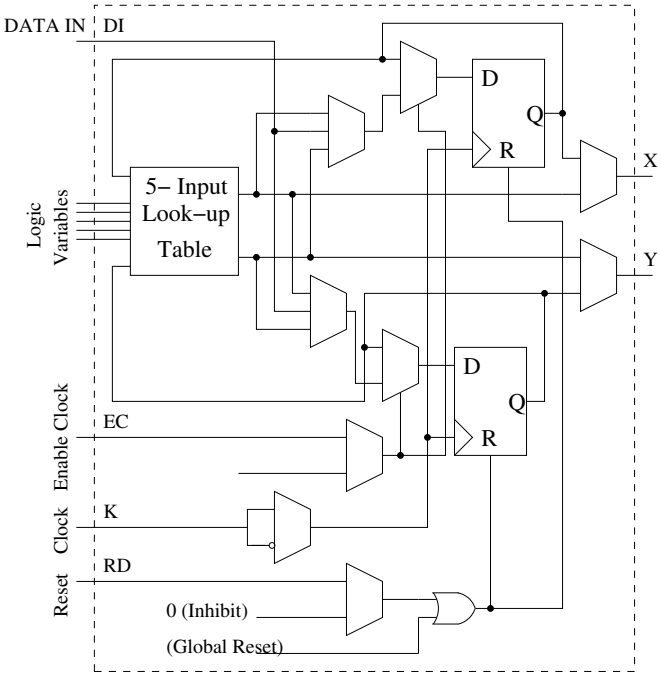

Figure 2.9: CLB of Xilinx XC3000 Series [154].

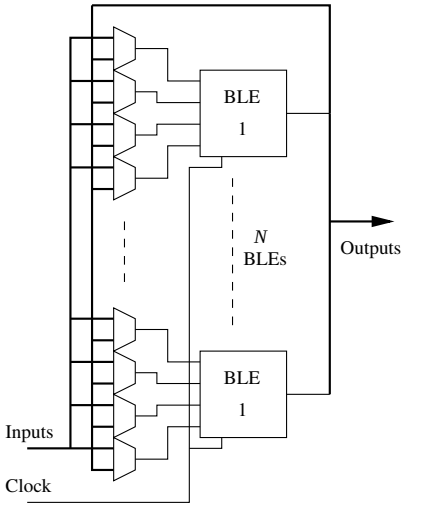

Figure 2.10: BLEs grouped together in a cluster. Redrawn with permission from [129].

Another technique to achieve a better trade-off between larger LUT and cluster sizes is to use clusters of different sized LUTs [156–158]. Different combinations were realized using this technique achieving a 10% reduction in pin count [156] and 25% improvement in performance [157].

All these technological advances have resulted in current state-of-the-art FPGAs employing a variety of the discussed techniques. The Virtex-7 FPGA provided by Xilinx combines eight LUTs in one CLB by packing four of these in one slice [159]. Its 6-input LUT can also be configured as two separate 5-input LUTs, making the architecture both homogeneous and heterogeneous at the same time. Similary, Altera Stratix II architecture employs a 6-input LUT which can be configured as either one 6-input, two 4-input and a combinatin of 5 and 3-input LUT [160]. These CLBs also contain high-speed carry propagation for arithmetic operations and wide multiplexers and LUTs can also be used to implement memories and shift registers. These memories are commonly referred to as distributed memories because the memory function is distributed across a number of LUTs [159].

D FPGAs for DSP Implementation

High parallelism and throughput requirements of DSP algorithms can be realized by specialized ASICs. However, as noted earlier, ASICs do not provide high flexibility in terms of reconfigurability [161]. FPGAs on the other hand are inherently built such that they support highly parallel algorithms and provide a higher degree of flexibility in terms of reconfiguration. The introduction of dedicated multipliers and multiply-accumulate units has enabled designers to implement multiplier and multiply-accumulate (MAC) intensive applications in FPGAs. Among the various DSP algorithms, FIR filters are one of the most important algorithms which are widely used in numerous applications. Due to the high amount of MAC operations inherent in an FIR filter, state-of-the-art FPGAs are often used to implement FIR filters [58, 86, 162–169].

Current FPGAs have numerous specialized blocks which map specifically to multiply and MAC operations. These are commonly referred to as DSP blocks. In fact, implementation of digital filters was one of the key factors to push for the inclusion of these DSP blocks in the FPGA fabric [170] which also helped in reducing the performance gap between ASICs and FPGAs.

The DSP block comes in various flavors. The DSP block by Xilinx in their $5, 6$ and 7 series FPGAs support various functions, shown in Fig. 2.11. It supports a number of operations, namely a 25×18 two's complement multiplier, a 48-bit accumulator, a power saving pre-adder, single instruction multiple data (SIMD) operation, an optional logic unit, pattern detector, optional pipelining and dedicated buses for cascading DSP blocks (beneficial for FIR filters) and support for wide multiply operations up to 35×26 by cascading two DSP blocks [171].

Altera on the other hand has implemented a variable precision DSP block, the basic structure of which is shown in Fig. 2.12. The functionalities supported

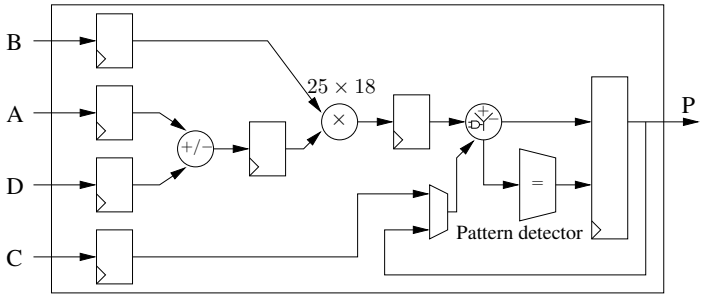

Figure 2.11: Basic structure of a Xilinx DSP slice in series-7 FPGAs [171].

by it are, three 9×9 multipliers, two 18×18 multipliers, one 27×27 multiplier, a 64-bit accumulator and adder, chainout adder for cascading, support for storage of up to eight coefficients storage and a special systolic FIR Mode [172].

The high performance provided by these dedicated DSP blocks have a downside. The number of DSP blocks available is limited and the word lengths they support are limited. Although Altera provides a variable precision DSP block yet it is still coarse and for applications like video and image processing, which require 8 to 10 bits of resolution, there will be wasted resources if the multiplications are mapped to these blocks [173]. Furthermore, if there is a need for large size multiplication, like floating point multiplication with mantissa sizes of 24 and 54 bits, it will require cascading of multiple DSP blocks. This will significantly increase the number of required DSP blocks.

Therefore, there is a need to realize multipliers in the soft logic, i.e., using the programmable CLBs and LUTs for applications where the DSP blocks are not sufficient or the word size does not match the system requirements. For example, a method termed as tiling was introduced in [174, 175]. Here, large multiplications were implemented using several DSP blocks and smaller multiplications were implemented using the softcore multipliers to "fill gaps" where a DSP block is too large.

A number of techniques have been proposed on the efficient use of the logic elements to implement softcore multipliers. The fast carry chain of modern FPGAs was used to implement a multiplier in [176] which generates the partial products using Booth recoding and adds them using ripple carry adder instead of a compressor tree. To take advantage of 5-input LUTs where two of the outputs can be used independently, the technique proposed in [177] showed that it is possible to generate and compress two partial products of a Baugh-Wooley multiplier [178] which reduces the number of partial products by half, similar to the Booth multiplier. However, there is no decoding/encoding required in a Booth multiplier.

The low level logic of FPGAs have been used to efficiently implement the compressor trees required to add all the partial products. Generalized parallel counters (GPCs) are used to replace the full-adders used in compressor trees

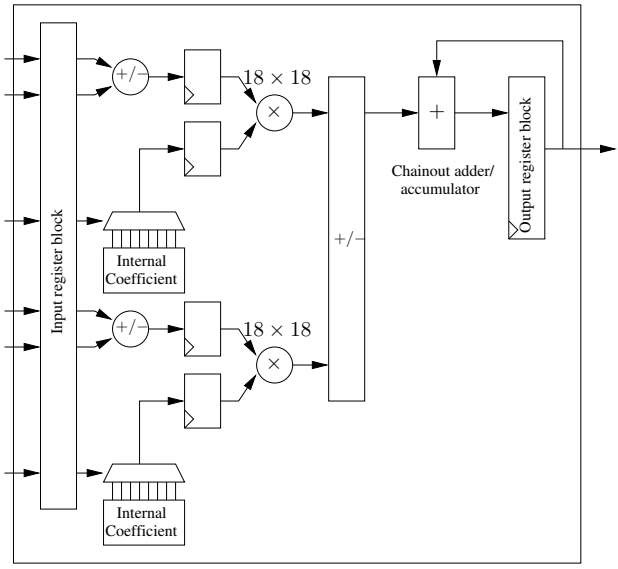

Figure 2.12: Basic structure of an Altera DSP slice in stratix-V FPGAs [172].

as a means to reduce the combinatorial delay. Heuristics [179] and ILP formulations [180] are proposed for optimizing the delay of compressor trees while also considering the FPGA carry chain for the implementation of GPC [181]. ILP formulations are also proposed for reduced power consumption by reducing not only the depth but also the number of GPCs [182, 183] while efforts to optimize the number of resources for high throughput pipelined designs have been proposed in [184]. In [173], authors present a technique to completely avoid the compressor trees by merging the Booth recoding with the ripple-carry summation of the partial products in a single stage of LUTs, fast carry chains and flip flops. Futhermore, the presence of LUTs in FPGAs makes it attractive for distributed arithmetic based DSP systems [169, 185–189].

Details about multiplication, partial products and compressor trees are presented in Section 2.3.1.

E Reconfigurable DSP Implementation on FPGA

The built-in parallelism of resources in FPGA allows massively parallel applications to be easily implemented in an FPGA. It allows for a high throughput even at low MHz clock rates. This has given birth to a new type of processing called reconfigurable processing, where FPGAs perform time intensive tasks instead of software [161, 166, 189, 190].

Reconfigurable computing consists of a device, such as an FPGA, performing computations with spatially programmable architectures [191, 192]. It is rapidly establishing itself as a major discipline that inherits a wide body-of-knowledge

from many disciplines including custom hardware design, digital signal processing, general purpose computing on sequential and multiple processors, and computer aided design.

FPGAs are specially useful for dynamic reconfiguration [192]. It allows for hardware configuration changes during different phases of the computation. Such reconfiguration is important for different systems like communication and networking systems where hardware configurations must change to match protocol needs.

DSP algorithms, specially FIR filters, can also benefit from the reconfigurability provided by FPGAs. For applications like multi-stage filters for decimation and interpolation, polyphase FIR filters [193] or frequency variable filters for telecommunications and digital audio [194], multiplications with constants need to reconfigured from time to time [161].

This reconfiguration can only be achieved in ASICs by low-level multiplexing. However, for current FPGAs, there are standard solutions that can provide internal reconfiguration. One is the internal configuration access port (ICAP) of Xilinx FPGAs which allows the logic function as well as the routing to be completely reconfigured during run time. To change the logic only without reconfiguring the routing, for example changing the FIR filter coefficients, Xilinx FPGAs provide configurable LUT (CFGLUT). These LUTs can be reconfigured in 32 clock cycles by sourcing their contents from block RAM resources resulting in reconfiguration times of the order of 100 ns as compared to the ICAP interface where times are in order of microseconds to milliseconds [161]. These LUTs are similar to standard LUTs but provide a reconfiguration interface of data in, data out, clock enable and clock. In [161] and [188], authors have presented reconfigurable FIR filters using LUT based multiplier and distributed arithmetic.

2.3 Key Arithmetic Operators in DSP Implementations

2.3.1 Adders

Adders perform the most fundamental of all operations in digital signal processing algorithms. They are used as both standalone operators and as part of operations. Furthermore, they are not only used to add two input operands but can also be used to for multi-operand inputs [123]. From a FIR filter point-of-view, addition appears in two places. First they are needed to add all the partial products generated in a multiplication operation or when a multiplier is realized using a shift-add network. Secondly they appear as structural adders used to add all the products of the multiplication of the filter coefficient with the input data.

An overview of different adder structures, the relationships and inter dependencies is given in Fig.2.13 [123]. Brief descriptions about these structures are

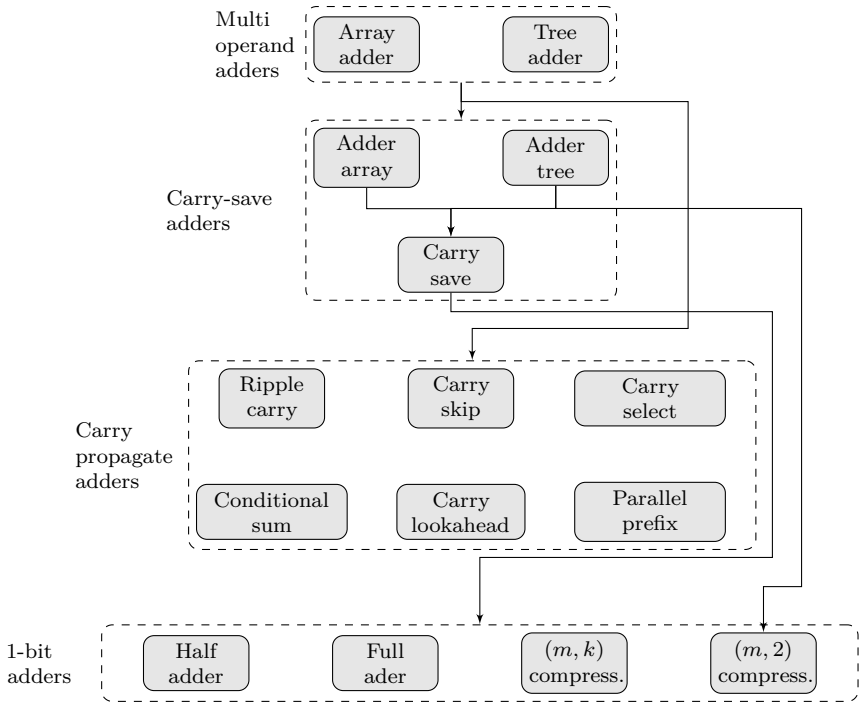

Figure 2.13: Adder structures and their dependencies.

presented later.

A Fundamental Adder Structures

The primary concern with respect to addition is the efficient speed up of the carry propagation. This becomes an even greater concern when adding a number of partial products which typically take the form of multi-operand adders like array-adders or tree-based adders. To improve the carry-propagation a number of adders have been proposed in the literature [123, 195–201]. However, central to all these adders are the 1-bit adder structures which are explained next.

Half-Adder, $(2, 2)$ – Counter The half adder has two inputs and two outputs making it a $(2, 2)$ – counter. It is referred to as a counter because it counts the number of 1's in the input bits. The two outputs are commonly referred to as the sum and carry out bit with the relationship between the inputs and outputs given by:

$$2c_{\text{out}} + s = a + b. \tag{2.1}$$

The resulting truth table resulting from (2.1) is given in Table 2.1. Since a half adder does not have any carry input, its use is rather limited. It finds use in

Table 2.1: Half Adder Truth Table.

a	b	c_{out}	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 2.2: Full Adder Truth Table.

a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

column reduction techniques like those proposed by Wallace [195], Dadda [196] and the reduced area heuristic [197].

Full-Adder, $(3, 2)$ **– Counter** A full adder has three inputs, two data inputs and one carry-in input which can also be a data input and two outputs, one data output and one carry output. It is also referred to as a $(3, 2)$ – counter as it also counts the number of 1's in the input signals. Similar to a half-adder, the outputs are typically referred to as sum and carry out and relationship is given by:

$$2c_{\text{out}} + s = a + b + c_{\text{in}}. \tag{2.2}$$

The resulting truth table resulting from (2.2) is given in Table 2.2. A full-adder can be constructed using a number of structures like half-adders, 2–input gates, multiplexers or complex gates. For details, refer to [123].

(m, k) **– Counter** The $(3, 2)$–counter can be generalized into (m, k)–counter. The key building block of generalized counters is the full adder. The benefit of using large counters arises from the associativity of the adder operation which allows the input bits to be added in any order. This provides a large degree of flexibility to optimize the speed of addition [123].

B Carry Propagate Adder

Large word length adders are constructed using the basic building blocks described earlier. The most basic form of adding n-bit operands is through carry-propagation. Arithmetically, addition of two n-bit operands can be described as

$$2^n c_{\text{out}} + \sum_{i=0}^{n-1} 2^i s_i = \sum_{i=0}^{n-1} 2^i a_i + \sum_{i=0}^{n-1} 2^i b_i + c_{\text{in}}. \tag{2.3}$$

Each stage of the addition adds two bits of the operands and the carry from the previous stage. Each stage either generates (g_i) a new carry, propagates (p_i) the carry from the previous stage or kills (k_i) it. The logic equations for these three signals are given below:

$$\begin{aligned} g_i &= a_i b_i \\ p_i &= a_i \oplus b_i \\ k_i &= \overline{a_i + b_i} \end{aligned} \tag{2.4}$$

The sum and carry out associated with each bit of the input operands can be given in terms of the generate and propagate signals

$$\begin{aligned} s_i &= p_i \oplus c_i \\ c_{i+1} &= g_i + p_i c_i \end{aligned}. \tag{2.5}$$

where $c_0 = c_{\text{in}}$ and $c_{\text{out}} = c_{n-1}$. Carry-propagate adders are typically used as vector merging adders at the end of the multiplication unit [201].

There are different types of addition algorithms which implement basic functionality of carry propagate adder with trade-offs in power, speed and area. Some of these are briefly explained next.

Ripple-Carry Adders Ripple-carry adder (RCA) is the most basic and straightforward way to perform addition by carry propagation. It is implemented using n full-adder cells connected in series. As the name suggest, the input carry is propagated throughout the adder stage by stage, i.e., it ripples through each stage [123, 201].

One of the main disadvantages of an RCA is that the worst case delay is proportional to the input operand's word length. To improve the delay, there is a need of a bit-level pipelining which will increase the area significantly. Without pipelining though, energy per computation of an RCA is relatively small because of its simple design [123].

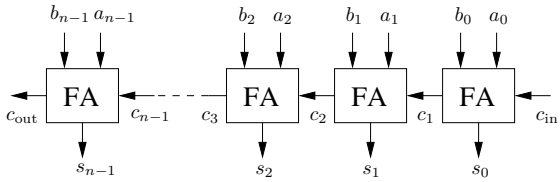

Figure 2.14: Ripple-carry adder.

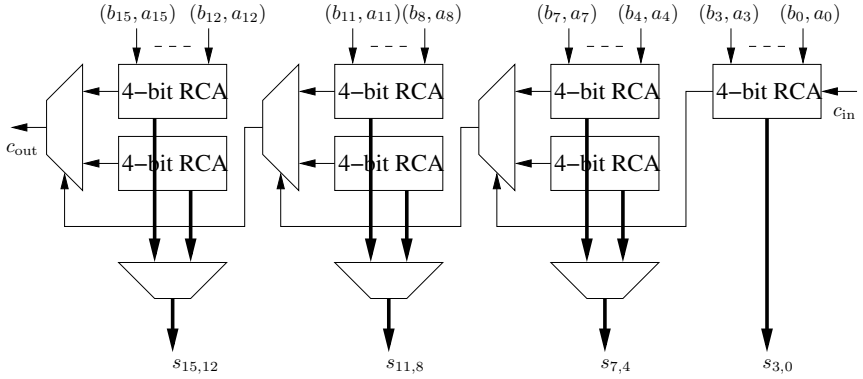

Figure 2.15: Carry-select adder.

Carry-Select and Conditional Sum Adder To reduce the critical path of the carry-chain in an RCA, a long adder can be partitioned into fixed-size adder groups which compute the summation of that group for carry-in values of both zero and one. Selection of the true output is then based on the correct carry-in from the previous group. In other words, carry input is used to select between two groups of addition, hence the name. Using this form of addition, the critical path of the adder is reduced to the critical path of one group and $G - 1$ multiplexers where G is the number of groups the input operands are divided into. As an example, a 16-bit addition is shown in Fig. 2.15 where the group size is four. However, more speed-up is obtained if the group size is irregular. Regular group size has an advantage that the all stages can be computed by multiplexing the same circuitry if the the clock rate is higher than the sample rate. If the group size is reduced to only 1-bit adders, the resulting structure is a conditional sum adder [201].

Carry-Skip Adder A similar concept of dividing the word length into stages is utilized in the carry-skip adder. Here the propagation of a carry in a group is skipped if carry-propagate signal, given in (2.5), of that particular group is true. If it is not, either a carry is generated or killed in that stage. The critical path, if carry is generated in the first block and is propagated all the way to c_{out} will be the carry generation of the first block and carry-skip block of the remaining

groups. However, it can change depending whether a carry is either generated, propagated or killed in a particular group. Similar to carry-select adders, the group size can be variable.

Carry-Lookahead/Parallel-Prefix Adder The carry generate and propagate signals given in (2.5) can be used to compute the carry signal as given by (2.5) for stage $i+1$. For stage $i+2$, the carry expression becomes [201]:

$$\begin{aligned} c_{i+2} &= g_{i+1} + p_{i+1}c_{i+1} \\ &= g_{i+1} + p_{i+1}(g_i + p_i c_i) \\ &= g_{i+1} + p_{i+1}g_i + p_{i+1}p_i c_i \end{aligned} \quad . \tag{2.6}$$

Similarly, the carry for the last, n^{th}, stage can also be computed by same recursive computation of the previous stage carries. This is a typical prefix problem. In a prefix problem, every output depends on all inputs of equal or lower order or every input signal influences all outputs of equal or higher order [123].

As such, addition of two numbers is a prefix problem and the adder using the prefix property is referred to as carry-lookahead or parallel-prefix adder. In (2.6) the terms $g_{i+1} + p_{i+1}g_i$ and $p_{i+1}p_i$ are termed as group generate and group propagate respectively. Since these signals are independent of the carry signal, the carry can be propagated n stages with a delay of an AND and OR gate at the expense of a complex pre-computation network which grows with n.

This approach can be generalized using the dot or prefix-operation: $\bullet$, which is defined as

$$\begin{bmatrix} g_k \\ p_k \end{bmatrix} = \begin{bmatrix} g_i \\ p_i \end{bmatrix} \bullet \begin{bmatrix} g_j \\ p_j \end{bmatrix} \triangleq \begin{bmatrix} g_i + p_i g_j \\ p_i p_j \end{bmatrix}. \tag{2.7}$$

The associativity of this operator allows for the individual operations to be carried out in an arbitrary order. It is this associativity that allows for generating the group generate and group propagate signals by combining smaller, possibly overlapping, group generate and propagate signals [201]. The k^{th} carry signal can thus be written as

$$c_k = G_{(k+1):l} + P_{(k+1):l}c_l, \tag{2.8}$$

where $k \geq l$. Similarly, the sum signal can be expressed as

$$\begin{aligned} s_k &= a_k \oplus b_k \oplus c_k \\ &= p_k \oplus c_k \end{aligned} \quad . \tag{2.9}$$

It is now obvious that to speed up addition, all group generate and propagate signals originating from the least significant bit (LSB) position should

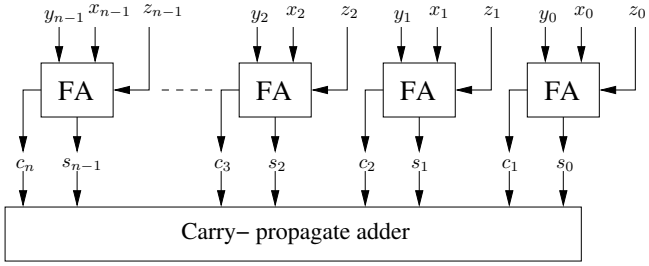

Figure 2.16: Carry-save adder (CSA).

be computed. These can either be obtained using a sequential-prefix algorithm which needs a minimal number of (•) operators at the cost of slow speed. On the other hand, a parallel-prefix algorithm can be used where all outputs can be computed separately and in parallel [123]. It decreases the computation time but increases the number of required (•) operations

Based on the associativity of the (•) operator, the adders can be connected in various ways to reduce the depth and the number of (•) operators. In Ladner-Fischer prefix-adder, intermediate signals are computed by a minimal tree structure and is then distributed in parallel to higher bit positions [198]. This adder has a high-fanout for some nodes which is reduced in the Brent-Kung [199] tree-prefix algorithm at the cost of an increased depth. Another approach is the Kogge-Stone algorithm [200] which has minimal depth and bounded fan-out but a high number of (•) operations. Details about these and other algorithms are available in [123].

C Carry Save and Multi-Operand Adder

One way of removing carry-propagation is to avoid it which is achieved in a carry save adder (CSA) by treating the intermediate carries as outputs instead of propagating them to the next stage [202]. It is realized using full-adders, is a redundant adder and plays an important role in multi-operand adders. This is because since carry is not propagated, it leaves one input of the full-adder cell unused making it possible to add three input vectors [201]. These multi-operand adders find application in the summation of the partial products generated in a multiplication operation.

The output of a CSA is in the form of two vectors: sum and carry. These are generally added together using a fast carry propagate adder (CPA), as shown in Fig. 2.16.

To add m input operands using CSA, $m-2$ CSAs are required followed by a fast CPA. These are referred to as array adders [123, 201] and are explained in more detail, along with tree adders in Section 2.3.2.B. Since carry-save is an example of a $3,2$-compressor, in general, multi-operand adders are implemented either using counters or compressors or a combination of both.

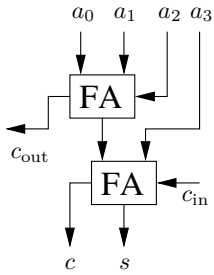

Figure 2.17: $(4, 2)$–compressor using two full-adders.

$(m, 2)$ – **Compressors** In contrast to counters which produce a true binary representation of the number of ones in the input, the compressor does not produce valid binary count [201]. It rather reduces the number of partial products and has a number of incoming and outgoing carries. An $(m, 2)$ compressor is also a one bit adder, as shown in Fig. 2.13, which is combined to form multi-operand adder arrays. Arithmetically, it can be formulated as [123]:

$$2(c + \sum_{l=0}^{m-4} c_{\text{out}}^{l}) + s = \sum_{i=0}^{m-1} a_i + \sum_{l=0}^{m-4} c_{\text{in}}^{l}. \tag{2.10}$$

No horizontal carry-propagation occurs as c_{in}^{l} only influences $c_{\text{out}}^{k>l}$. An $(m, 2)$–compressor is built using $(m - 2)$ full-adders. The most frequently used compressor is the $(4, 2)$–compressor which is built using two full-adders as shown in Fig. 2.17.

2.3.2 Multipliers

Multiplication is one of the most basic and widely used arithmetic operations in various signal processing algorithms. The two operands of the multiplier are termed as *multiplier* and *multiplicand*. A significant amount of research material is available for the reduction in the implementation cost of this operation. This reduction revolves around the two basic operations in a multiplication: generation of partial products and their accumulation. Thus, in order to speed-up the multiplication, one either needs to reduce or the number of partial products, speed-up their generation or reduce the cost of the accumulation. Over the decades, various high-speed multipliers have been proposed [98, 195, 196, 203–208].

To reduce the number of partial products, one of the earliest algorithm was proposed by Booth in 1951 [98]. It forms groups of two consecutive bits with one overlapping bit from the lower order group. In this way, no partial product is generated when the multiplier has a group of two zeros or two ones. For other values, a partial product of X or $-X$ is generated. A number of modifications to this algorithm have been proposed and is discussed later.

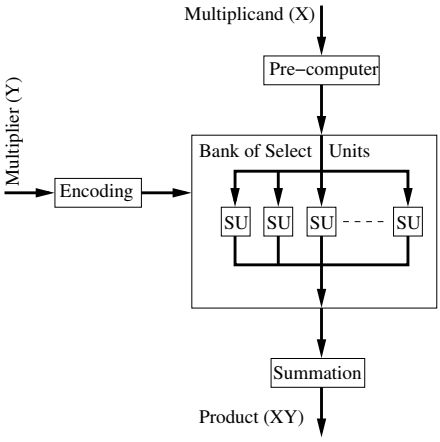

Figure 2.18: General block diagram of Booth/high-radix multiplier.

While the Booth algorithm analyzes overlapping groups of bits, multiplication can also be performed by analyzing non-overlapping groups of bits. This is referred here as high-radix multiplication, which is a misnomer as Booth multiplication is also a high-radix multiplication. But for brevity, multiplication where the multiplier is divided into non-overlapping groups of bits is referred to as high-radix multiplication. An example of high-radix multiplication is the standard binary multiplication which is referred as radix-2 high-radix multiplication.

The main building blocks of Booth and high-radix multiplication are the same. The multiplier bits are grouped and encoded by an encoder. Different integer multiples of the multiplicand is computed by the pre-computer, one of which is selected for each multiplier group based on the encoding in the select unit. The selected multiples of the multiplicand are then summed up for the final product. A unified block diagram representing both Booth and high-radix multiplication is shown in Fig 2.18.

The summation of the partial products, in general, can be referred as a multi-operand addition problem. In the literature, it has been solved in various ways: compressors, array accumulation and tree accumulation [123]. Section 2.3.2.B provides more details.

A Partial Product Generation

Partial product generation is the first step of multiplication process. Each bit/groups-of-bits of the multiplier is multiplied with the whole multiplicand to generate a partial product. Each partial product is shifted by an amount that is equal to the number of bits of the multiplier considered for the generation of that partial product.

The formulation of a bit-wise representation of an unsigned multiplication

is:

$$Z = XY = \sum_{i=0}^{W_X-1} x_i 2^i \sum_{j=0}^{W_Y-1} y_j 2^j = \sum_{i=0}^{W_X-1} \sum_{j=0}^{W_Y-1} x_i y_j 2^{i+j}. \tag{2.11}$$

where W_X and W_Y is the word length of the multiplier and the multiplicand. Typically, the word length of the final product is the sum of W_X and W_Y.

The total number of partial products is equal to the total number of bits or groups-of-bits in the multiplier (W_Y in (2.11)). Two techniques, used to reduce the number of partial products, i.e., the Booth algorithm and high-radix multiplication are discussed next.

Booth Algorithm - Original and Modified One of the earlier attempts to reduce the number of partial products came in the form of Booth algorithm and its variants. The original algorithm divides the multiplier bits into groups of two with one overlapping bit with the adjacent groups and encodes them. A 0 is appended at the LSB side before the encoding. The original Booth algorithm can be referred to as radix-2 or Booth-1 [209], however, it is rarely used. An improvement on this encoding was shown in [99] where multiplier bits were grouped into three and four bits respectively and encoded, thus reducing the number of partial products to $\lceil \frac{W_Y}{2} \rceil$ and $\lceil \frac{W_Y}{3} \rceil$ respectively. Each partial product, except the first one, was then shifted by two or three positions for their summation. These techniques can be referred as radix-4/Booth-2 and radix-8/Booth-3, respectively. The encodings for radix-4 and radix-8 are shown in Table 2.3(a) and (b), where the Enc. column indicates the values of the multiples of X to be generated by the pre-computer.

In both of these encodings, the LSB of each group is most significant bit (MSB) of the previous low-order group. The encoding occurs in parallel as these multiples have no dependency. In case of radix-4 all the multiples can be obtained using simple shifts however, the non-trivial multiplies of 3 in radix-8 requires addition.

The partial products can be both positive and negative. For their summation, the partial products have to be sign extended, a disadvantage as argued in [209, 210]. However, the problem of sign extension can be avoided be using the identity $-p = \bar{p} - 1$ which enables one to replace all negative partial products with an inverted version and adding a compensation vector at the end. Furthermore, because of negative partial products generated for the Booth algorithm, a sign-bit needs to be added to each partial product.

The pre-computer in Fig. 2.18 produces all the required multiples of the multiplicand which is then selected by the select unit based on the encodings produced by the encoder. The select unit consists of a set of multiplexers while the encoder is a mapper from the input multiplier bits to the encoded digits. Typically, the pre-computer is required to only produce odd multiples as even and negative multiples can be obtained shifts and negation. Thus the shift and negation can be performed after the appropriate odd multiple has been selected.

Table 2.3: Booth Encoding with (a) Radix-4 and (b) Radix-8.

(a)

y_{2k}	y_{2k+1}	y_{2k+2}	Enc.
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

(b)

y_{2k}	y_{2k+1}	y_{2k+2}	y_{2k+3}	Enc.
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	2
0	1	0	0	2
0	1	0	1	3
0	1	1	0	3
0	1	1	1	4
1	0	0	0	-4
1	0	0	1	-3
1	0	1	0	-3
1	0	1	1	-2
1	1	0	0	-2
1	1	0	1	-1
1	1	1	0	-1
1	1	1	1	0

It is possible to extend the modified Booth encoding to higher radix which will reduce the partial products roughly by a factor of r for radix-2^r Booth multiplier [211]. But it involves even more computation of non-trivial multiples such as $3, 5$ and 7 for radix-16, increasing the complexity of the pre-computer. It will also increase the complexity of the encoders and there is a possibility that any reduction in the complexity of the partial product summation is offset by the increase in the complexity of the encoder and the pre-computer. Formal proofs of the original and modified Booth multipliers appear in [211, 212].

The implementation cost of the pre-computer can be reduced based on the fact that the multiples of the input multiplicand are known, making it an MCM problem. Efficient techniques have been proposed to implement MCM using directed acyclic adder graphs (DAG) and CSE [28, 29, 38]. More details about MCM techniques are available in Section 2.3.3.

Standard High-Radix Multiplication The multiplier can be divided into groups of disjoint sets, referred to as standard high-radix multiplication where the normal binary multiplication can be termed as a radix-2 multiplication. This can be extended to form groups of r bits, where $r \geq 1$, to form radix-2^r standard high-radix multiplier. This will compare with a higher radix Booth multiplier, where pre-computed multiples can be used to generate the partial products. Therefore, in case of a radix-4 multiplier, one will need multiples

of $0, 1, 2, 3$ times the multiplicand. The encoding will be very simple with the multiplier being divided into disjoint groups of two bits. Each partial product will then need to be shifted by two positions for the summation.

A number of papers have been published utilizing the same encoding, however, the proposed ideas are simply a higher radix multiplication. For example, in [103–108] the authors present a computation sharing multiplier where the bank of pre-computers generate odd multiples between $1X$ and $15X$, which is an example of a radix-16 standard high-radix multiplier with other multiples being non-trivial and generated using just shifts. Also in [213] authors propose a common subexpression elimination where the multiples produced are again a case of radix-8 standard high-radix multiplication.

Standard high-radix multiplier has the same building blocks as the Booth multiplier, as shown in Fig. 2.18. Again, only odd multiples need to be computed by the pre-computer. However, standard high-radix multiplication does not have negative multiples except for the most significant group of bits.

B Partial Product Summation

The next step after partial product generation is the problem of their summation. This summation can be performed either serially using a serial/parallel multiplier using carry-save adders [201] or in a parallel manner. The parallel summation is primarily a problem of multi-operand addition which is solved either using $(m, 2)$ compressors, array accumulation and tree accumulation [123]. Within these broad schemes, a number of different techniques have been proposed like Wallace tree adder array using carry-save adders [103, 104, 107, 214], logarithmic carry-select adder [105] and carry-select adder using dual transition skewed logic (DTSL) [106, 108].

A brief description of the $(m, 2)$ compressor was given earlier while a brief description of the remaining methods is given next.

Serial Accumulation of Partial Products For serial accumulation of partial products, they are also generated sequentially and then accumulated successively as they are generated. Due to this, these multipliers have large latencies but can be clocked at a high clock rate because, typically, the critical path is restricted to just one full adder [201].

Serial accumulation has a further advantage of requiring significantly less area by eliminating wide buses and having simplified routing. Typically, the multiplier used for serial accumulation is a serial/parallel multiplier where the multiplicand (X in Fig. 2.18) arrives serially while the multiplier (Y in Fig. 2.18) arrives in a bit-parallel fashion [201].

Array Accumulation of Partial Products Array accumulation of partial products has the advantage of producing a regular structure. This will help in routing of signals and can decrease routing delays. The accumulators consist of identical cells, typically a full adder. There is no horizontal carry propagation

in the initial stages where the output of each stage consists of an intermediate sum and carry output, carry-save type. Only the final stage requires it which can be replaced by a fast CPA. A typical example of such an accumulator can be seen in a Baugh-Wooley multiplier [178]. The problem of signed numbers requiring sign-extension, similar to the one in Booth-multiplier, is solved by using the property $-p = \bar{p} - 1$.

This approach, however, has its shortcomings. It requires a large area because of the presence of many full adders. Also, the delay is quite high which consists of $(W_Y - 1) \times T_{FA}$ for the initial rows and $W_X \times T_{FA}$ for the final carry-propagate adder, where T_{FA} is the delay of a full adder. However, unlike tree accumulation, this technique does not require any vector merging adder.

Tree Accumulation of Partial Products Tree accumulation is a technique to reduce as many partial products as possible. It utilizes both half- and full-adders to minimize the number of levels and reduces the word length for the final vector merging adder. Popular approaches are Wallace [195], Dadda [196] and reduced area [197]. They use the fact that not all columns in a partial-product array have the same number of bits as in the final product. The full and half adders act as carry save adders because the carry generated is only placed in a higher column to be used in the next level of reduction.

Wallace [195] uses many half adders, which is a drawback. Dadda [196], instead uses an approach which proposes that full and half adders are only to be used if required to obtain a number of partial products equal to a value in the Dadda series. The benefit is the reduction of half adders while at the same time reducing the number of levels. A compromise between the two approaches is proposed in [197] where, like Wallace, as many full adders as possible are introduced in each level while half adders are only introduced to reach the sequence in the Dadda series or when combining two bits towards the right of the partial product array. In this way, a minimum number of stages is obtained with both the length of final adder and number of half adders kept small.

There is, however, a drawback. The reduction trees do not provide any regularity and thus the routing is complicated. This can increase the delays and become a limiting factor in the final implementation. Reduction trees that are more regular include overturned stairs-reduction tree [215] and the high performance multiplier (HPM) tree [216].

Final Addition The final addition step is used when the partial products are accumulated using a tree structure. This is also called a vector merging adder. Its role is to add the outputs of the reduction tree. Generally a carry-propagation adder can be used. The author in [214] has used an accelerated carry adder which has a Manchester carry chain circuit. One can use a carry-select or carry-look-ahead implementation as well.

In a multiplier, the arrival time of different inputs to the final adder will typically be different and this can be used to optimize the adder delay. In [217],

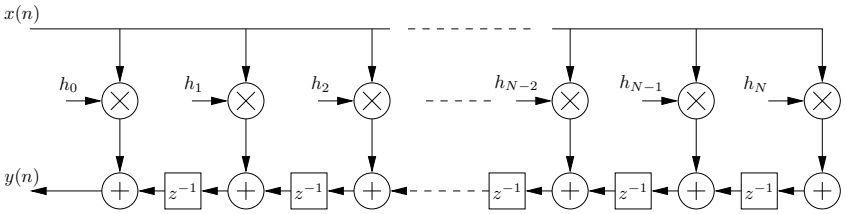

Figure 2.19: Transposed direct form FIR filter.

the authors have shown that fast adder designs based on uniform signal delay profiles can give poor results and that using a hybrid structure for the final adder that may consist of a variety of different designs will give better results. They have shown the design based on blocks of ripple-carry, carry-skip and carry-select adders with an approach that can also be used for multi-level carry-skip and carry-look-ahead adders.

2.3.3 Multiple Constant Multiplication

The implementation complexity of a multiplication circuit can be reduced if the multiplier coefficient is a constant. Generation of partial products is not required and the multiplication is implemented using a shift-add network. This technique, called SCM can be extended to MCM if multiple constants are being multiplied with a single input. This type of multiplication is observed in the TDF FIR filter, as shown in Fig. 2.19.

With MCM, the complexity of FIR filters can be further reduced by utilizing the potential redundancy between filter coefficients. Complexity reduction through MCM can be achieved using two techniques, CSE [29] and DAG [28].

A Sub-expression Elimination (CSE)

The CSE technique is based on pattern matching techniques [29, 30, 72–74] and uses CSD number representation which typically has less number of non-zero digits as compared to two's complement (2C) number representation. CSD number representation is a subset of signed digit (SD) number representation. However, unlike SD which is redundant, CSD is non-redundant. Each digit, x_i, in a CSD coefficient, X, can take three different values, i.e., $x_i \in -1, 0, 1$. The difference between SD and CSD is that CSD imposes a restriction that no consecutive digits can be non-zero.

Even for one filter coefficient value, CSE can be used to reduce the implementation cost. Each coefficient will be represented by a number of digits. Consider for example, the coefficient, 413/512. In CSD, its representation is $1.0\bar{1}0100\bar{1}01$. In this representation, $10\bar{1}$ occurs twice. This is because $10\bar{1}$ and $\bar{1}01$ are considered the same because addition and subtraction are considered to have the same complexity [201]. Similary, 100001 also occurs twice. These are termed as

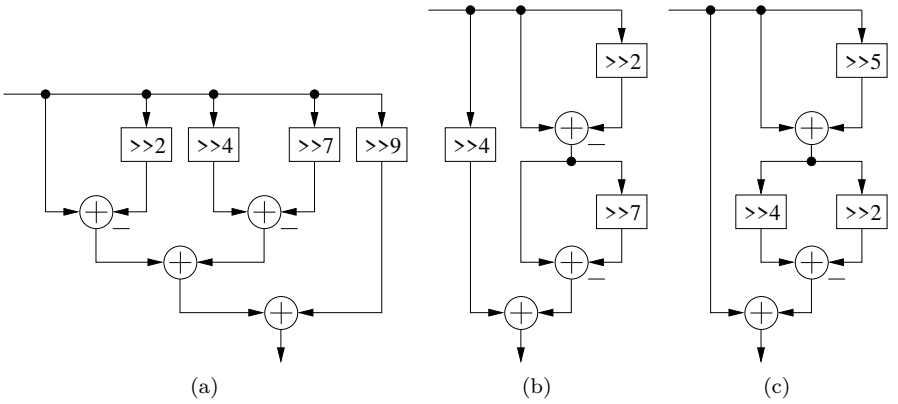

Figure 2.20: Constant multiplication with $413/512 = 1.0\bar{1}0100\bar{1}01$ with (a) no sharing (b) sharing of $10\bar{1}$ and (c) sharing of 100001.

subexpressions and sharing of these reduce the number of adders required for their implementation, which is shown in Fig. 2.20

This technique can be extended to multiple filter coefficients where subexpressions can be shared across them. Consider the coefficient set 3/32, 13/32, 25/32 where $3/32 = 0.00010\bar{1}$, $13 = 0.10\bar{1}01$ and $25 = 1.\bar{1}001$. Here the most common subexpression is $10\bar{1}$. Sharing of this subexpression can reduce the number of adders from five to three, as shown in Fig. 2.21.

This procedure can be performed in a systematic way by finding and counting possible subexpressions, typically picking the one with highest frequency of occurrence and introducing a new symbol in its place. If some subexpressions were replaced, this procedure is repeated. If no subexpression is replaced, the procedure is aborted [29, 73]. However, this greedy approach might not be optimal [201].

The subexpression sharing problem can also be solved using ILP. The objective function to minimize in such problems is the number of additions required to form each subexpression and the total number of subexpressions [71]. Further improvements can be achieved by not restricting the coefficients to CSD as more non-zero digits allow for more common subexpressions to exist [201].

Different techniques and heuristics have been proposed to solve MCM using CSE. In [29], the authors propose a method which uses only two types of subexpressions, i.e. 101 and $10\bar{1}$, as they are the most frequent. A greedy solution, which selects the subexpression with the highest frequency is presented in [73] while in [30], the authors present a technique which extracts all the large subexpressions, i.e. with most non-zero digits, and selects the one with the highest frequency. Then the number of non-zero digits are decreased and the algorithm is repeated. In [74] authors propose an ILP model where subexpressions are restricted to have at most two non-zero bits.

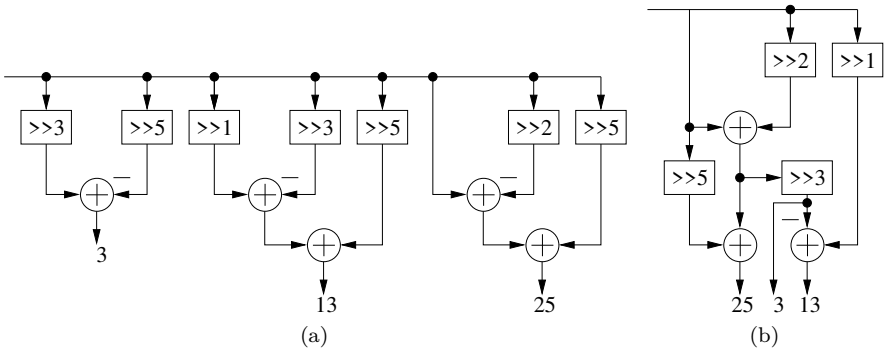

Figure 2.21: Multiple constant multiplication with 3/32, 13/32, 25/32 with (a) no sharing and (b) CSE.

A comparison of a number of MCM techniques was made in [218] and two new algorithms were proposed. One was an extension of the two term approach of [29] where at first the algorithm was applied to each coefficient separately. The technique of [29] was again applied to the result of the first step. It also explored a large number of different combinations of minimal signed digit (MSD) representations and compared the results to the proposed technique of [219] which eliminates subexpressions using MSD number representation. MSD number representation requires the same number of adders as CSD, however the restriction of not having consecutive non-zero digits is removed.

B Directed Acyclic Adder Graphs (DAGs)

Another form of MCM, having the same underlying technique of sharing/ eliminating of common subexpressions, is the adder graph technique [28, 72, 220]. The difference between adder graphs and CSE is that it is value based and independent of the underlying number representation used [71] and usually achieves better solutions than CSE. Here partial sums are symbolically represented in the nodes of the graph while the edges are used to represent the shift amounts [28].

The key concept behind DAGs is the use of graph representation to implement multiplier blocks and was introduced in [72]. Graph-based optimization has been used for single coefficients and shown to be superior to CSD representation [221]. The same concept has been extended for MCM in [28, 220] by using the concept of multiplier blocks for TDF FIR filter.

The graph-based technique to implement multiplication consists of vertices and edges representing adders and multiplication by a power of two, respectively. A comparison of realizing a multiplication by 45 using CSD and the techniques proposed in [221] is shown in Fig. 2.22. The number of adders required is reduced from three to two using the improved technique. In this graph, the values assigned to each vertex is termed as a *fundamental* with 4 and 8 in

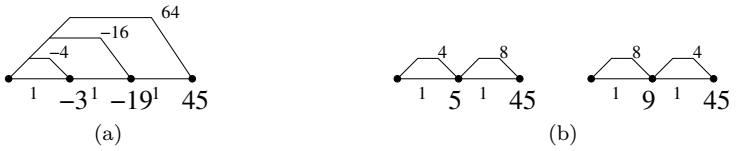

Figure 2.22: Representation of 45 using (a) CSD and (b) technique proposed in [221].

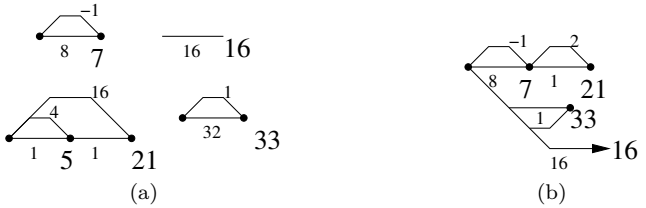

Figure 2.23: Multiple constant multiplication with 7, 16, 21, 33 with (a) no sharing and (b) DAG.

Fig. 2.22(b) being intermediate fundamentals.

The same concept is used when implementing MCM problems, like FIR filters. The multiplier block consists of a network of shifts and adds where sharing of network happens across multiple constants. As an example, Fig. 2.23 shows multiplication by four constants, first implemented as separate multipliers and then as a single multiplier block with a sharing of the shift-add network resulting in a reduction in the number of adders required.

Different heuristics have been proposed in works published during the last few decades. In [72], which is the first known contribution, the authors propose four MCM algorithms: add, add/subtract, add/shift and add/subtract/shift with the last one being the most generic. The fundamentals in the graph are arranged in only ascending order, i.e., intermediate nodes cannot be obtained from other intermediate nodes using right shifts and that as intermediate fundamentals, only those values are allowed that are smaller than the set which contains all constants to be realized (target set). As soon as a constant is realized, it is removed from the target set. Typically, a target set is reduced to only contain positive odd numbers as even and negative numbers can be realized using shifts and subtraction.

This algorithm was improved in [221] by relaxing the constraints described above in [72]. It also synthesizes the target constants in order of increasing complexity in terms of adders which was determined by using a precomputed LUT. This algorithm was further improved in [28] which relaxed some of the constraints and used a table of precomputed optimal SCM decompositions obtained by an exhaustive search of the method described in [221]. A more computationally expensive but a better heuristic was proposed in [220] which does

not use an optimal SCM look-up table. It improves the synthesis of constants which require more than one adder to be synthesized.

C MCM on FPGAs

FPGAs, as mentioned in Section 2.2.2, have dedicated hardware blocks suitable for the implementation of DSP systems. However, there is a disadvantage in using these dedicated blocks. They are limited in number and restricted in terms of word length. Many applications do not require such types of large multipliers leading to a need of soft-core multipliers which use the general fabric of the FPGA.

Apart from different methods of implementing softcore multipliers mentioned earlier, contributions have also been made towards efficient implementation of MCM problems on FPGAs [40, 41, 222–225]. Although the presence of a LUT indicates that a DA based FIR filter will be beneficial, it has been shown that MCM based FIR filters achieve better results as compared to DA based methods [223, 226].

The problem of implementing MCM problems on FPGAs is the efficient use its resources; the LUTs, full-adder logic with fast carry chains and flip-flops. For high-speed implementation, it is necessary to introduce pipelining as the critical path is equal to the maximum number of adders in one path (two in case of Figs. 2.21 and 2.23). In other words, each adder needs to be a registered adder. Typically the resources identified are contained in one unit. Thus, when pipelining an adder, the flip-flop in the unit can be utilized requiring no further addition to the implementation cost. Thus the main problem is the efficient pipelining so that there is a reduced resource usage.

Different heuristics and optimization based solutions have been proposed in various works. In [223], a hand optimization of pipelining was proposed which showed that pipelined adder graphs proposed in [28] is beneficial when compared to pipelined DA based FIR filters. Optimization routines were first used in [227] to propose a reduced slice graph (RSG) algorithm. It was also based on the method proposed in [28] but used a low adder depth resulting in a low usage of FPGA resources. A method based on CSE was proposed in [226] which was called the Add/Shift method. Binary integer linear programming (BILP) was used in [40] to optimize the pipelining by finding the best scheduling as well as adder duplication. Adder duplication means that an adder can be eliminated in the adder graph if it is computed again in a later stage. Gate level BILP optimization is proposed in [228] to further reduce the cost. These methods though are applied on an existing MCM problem. A direct optimization of these pipelined adder graphs (PAGs) is proposed in [41] called reduced pipelined adder graph (RPAG). Here it is shown that pipelined MCM (PMCM) is a generalization of the MCM problem with limited adder depth and the proposed heuristic produces better results as compared to those presented in [40] and [228].

2.4 Number Systems

The choice of number system in any digital system has an impact on its implementation cost and power consumption. The most common ways to represent numbers in DSP systems is to use representations based on radix-two, i.e., where the weight of each is digit is a certain power of two. Typically for DSP systems numbers are fractional rather than integer. A common number system is the unsigned binary number system where each digit takes on two values $0, 1$ and a number of digits combine to form a number as:

$$X = \sum_{i=1}^{F} x_i 2^{-i}, \tag{2.12}$$

where F is the number of fractional bits. The LSB in this case, commonly referred as unit of least significant position (ulp) has a weight of $Q = 2^{-F}$ and X has a range of $0 \leq X \leq 1 - Q$. However, to represent negative numbers there are several different number representations. The two most common among them is the signed magnitude (SM) and 2C number representation.

SM, as the name suggests represents the sign and magnitude of the number separately. The MSB is used to represent the sign, where a one represents a negative sign and zero a positive. The other bits are used to represent the magnitude of a number. For example, $-5 = 1101$ and $5 = 0101$ in SM number representation. The benefit of SM representation is the simplicity with which a transition can be made between negative and positive numbers of the same magnitude. There is only a change in one bit position which has a positive effect on the switching activity and hence the power consumption. One the other hand, addition and multiplication are more involved as there is a need to explicitly determine the sign of the result. The number range of SM is $-(1 - Q) \leq X \leq 1 - Q$ and can be expressed mathematically as

$$X = (-1)^{x_0} \sum_{i=1}^{F} x_i 2^{-i}. \tag{2.13}$$

2C number representation on the other hand uses a weighted MSB to represent signed numbers. This helps in making addition and multiplication simpler as their is no need of explicitly handling the sign of the answer. Furthermore, one can add an arbitrary long sequence of numbers in any order as long as the final result is within the range of the representation because any intermediate overflows/underflows will cancel out. The range of 2C is $-1 \leq X \leq 1 - Q$ and can be represented as

$$X = -x_0 + \sum_{i=1}^{F} x_i 2^{-i}. \tag{2.14}$$

The number representations described so far are non-redundant number representations, i.e., for each number there is only one representation. There is

another category where each number can have multiple representations and is referred to as redundant representation. An example of a redundant representation is the SD number representation which was briefly described in Section 2.3.2 and another is the carry-save representation. The benefit of redundant number representation is the ability to add two numbers without any carry-propagation and in constant time.

In contrast to regular number systems like 2C, SM and SD, there are number systems which have benefits with respect to the operation of multiplication and addition [229–231]. The two most common number systems are LNS and RNS.

The most expensive operation in any DSP system is the multiplication and the LNS takes advantage of the fact that multiplications become additions. It also simplifies other operations like division, roots and powers. This, however, comes at the cost of increased complexity to implement addition. However, the simplification of the multiplication operation to addition is a major benefit which has been utilized to implement DSP systems [85–92, 95, 96, 232–235].

The LNS representation of a number X consists of a triplet $\mathcal{X}$ as follows [93]:

$$\mathcal{X} = (z_x, s_x, m_x)\,, \tag{2.15}$$

where z_x is a one-bit flag to indicate if X is zero, s_x is the sign of X and $m_x = \log_b |X|$ is the base-b logarithm of the absolute value of X [90]. Representation capabilities, computational complexity and conversion to and from LNS numbers greatly depend on the choice of the base [230].

As stated before, the motivation behind using LNS is the simplicity of implementing the multiplication of $\mathcal{X}$ and $\mathcal{Y}$ which is reduced to the computation of the triplet $\mathcal{Z}$ [90]:

$$\mathcal{Z} = (z_z, s_z, m_z)\,, \tag{2.16}$$

where $z_z = z_x$ or z_y, i.e., the zero flag of the output, $s_z = s_x$ xor s_y, i.e., the sign of the output and the output itself, $m_z = m_x + m_y$.

The addition and subtraction are more complex and are given by (2.17) and (2.18).

$$add = \max(m_x, m_y) + \log_b\left(1 + b^{-|m_x - m_y|}\right), \tag{2.17}$$

$$sub = \max(m_x, m_y) + \log_b\left(1 - b^{-|m_x - m_y|}\right). \tag{2.18}$$

RNS simplifies the addition and multiplication operation by eliminating carry propagation and allowing for constant operation time. It achieves this by using the Chinese remainder theorem of modular arithmetic and uses positional bases $m_1, m_2, \ldots, m_P$ which are relatively prime [236] to represent a number. Each number has a unique RNS representation given by

$$X \rightarrow ((x)_{m_1}, (x)_{m_2}, \ldots, (x)_{m_P})\,, \tag{2.19}$$

where $(x)_{m_i} = X \bmod i$. To perform any operation (addition, multiplication or division) on the RNS number, the operation is performed modulo on each

moduli (m_i). So for example, to represent the number 49 in RNS with bases $5, 3, 2$, the resulting RNS number will be 49 mod 5 = 4, 49 mod 3 = 1 and 49 mod 2 = 1, i.e., $[4, 1, 1]$. To add 49 to, for example, 29 ($[4, 2, 1]$), one will perform modulo addition on each digit, i.e.,

$$(1 + 1)\text{mod}2 \rightarrow 0$$
$$(2 + 1)\text{mod}3 \rightarrow 0$$
$$(4 + 4)\text{mod}5 \rightarrow 3$$

i.e., $[3, 0, 0]$ which is the RNS representation of 78. The speed of the RNS depends on the largest moduli in the base and not on the magnitude of the number [237]. This is because although there is no carry propagation between the RNS digits, in order to implement RNS in hardware, each moduli will be implemented as binary numbers and there will be carry propagation within each moduli. Therefore small moduli will result in a cost-effective implementation of arithmetic operations. However, if each moduli is small, then to ensure a sufficient dynamic range, given by $m_1 m_2 \ldots m_P$, a large number of them will be required [238].

However, there are disadvantages as well. Input/output conversion from binary to RNS is non-trivial and exhibits a significant overhead, solutions for which has been proposed in [239, 240]. A direct conversion of the analog signal to residue representation and vice versa has also been proposed in [241].

Determination of the sign of a number is complex. This imposes restrictions on how efficient arithmetic operations can be performed with signed numbers and thus limits its applications. However, despite its limitations, RNS has been used to design and implement DSP systems because they are chiefly composed of multiplication and additions [80–82, 84, 236, 242, 243].

Chapter 3

Finite-length Impulse Response Filters

3.1 Introduction

The filtering of digital data is the most fundamental and oldest technique in the field of digital signal processing. Filtering is the process of changing the signal's original spectral content by processing it in the time-domain. Typically it involves allowing certain frequencies within the signal to pass while attenuating other frequencies, referred to as frequency selective filtering. These digital filters can be categorized in to finite-length impulse response (FIR) and infinite impulse response (IIR) filters. As the name suggests, FIR filters have a finite length impulse response, i.e., an input impulse will produce a response that will eventually become zeros. The fundamental arithmetic operation used by FIR filters to calculate output is multiplication and addition and the most simplest form of an FIR filter is the averaging operation. As compared to IIR filters, FIR filters are guaranteed to be stable, can be designed to have a linear-phase response and to require shorter data word lengths [2]. However, these filters require higher filter orders resulting in higher implementation cost to meet the same specifications as compared to IIR filters.

3.2 Impulse Response of FIR Filters

The impulse response of FIR filters completely defines its behavior and is of finite length. For an order N filter, it lasts for $N + 1$ samples [2]. A typical impulse response for 8^{th} order filter is shown in Fig. 3.1(a).

The poles of a FIR filter are always located at the origin of the z-plane, making it inherently stable. The zeros are typically located on the unit circle or

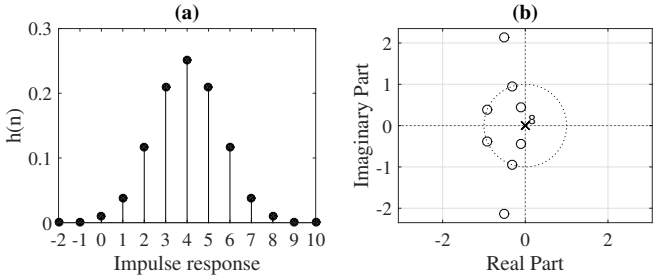

Figure 3.1: Low-pass FIR filter with $N = 8$ (a) Impulse response (b) Pole-zero configuration.

as mirrored pairs as shown in Fig. 3.1(b). The transfer function and frequency response of a FIR filter can be written as

$$H(z) = \sum_{n=0}^{N} h(n) z^{-n}, \tag{3.1}$$

$$H\left(e^{j\omega T}\right) = \sum_{n=0}^{N} h(n) e^{-j\omega T n}. \tag{3.2}$$

3.3 Linear Phase FIR Filters

FIR filters can be designed to achieve linear phase. This leads to that the impulse response either exhibits symmetry or anti-symmetry [5]. The impulse response shown in Fig. 3.1(a) exhibits symmetry, i.e., $h(n) = h(N-n)$, except for the mid-tap. The impulse response can also exhibit antisymmetry if $h(n) = -h(N-n)$.

Based on the filter order and symmetry/anti-symmetry property of the impulse response, four different types of FIR filters can be obtained that have linear phase [2]:

$$\begin{aligned} \text{Type I} &: h(n) = h(N-n), && N \text{ even} \\ \text{Type II} &: h(n) = h(N-n), && N \text{ odd} \\ \text{Type III} &: h(n) = -h(N-n), && N \text{ even} \\ \text{Type IV} &: h(n) = -h(N-n). && N \text{ odd} \end{aligned} \tag{3.3}$$

Figure 3.2 shows typical impulse responses for different types of FIR filters exhibiting symmetry and anti-symmetry.

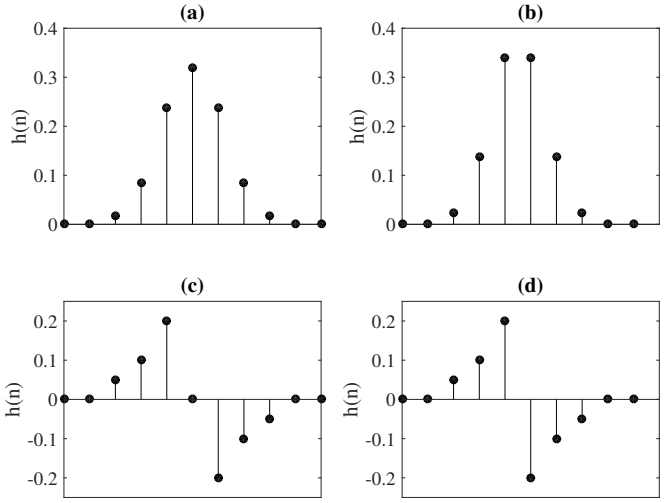

Figure 3.2: FIR filter impulse responses. (a) Type I, (b) type II, (c) type III, and (d) type IV.

3.4 FIR Filters: Input and Output Relationship

An FIR filter applies filtering to an input signal, $x(n)$, to produce an output signal, $y(n)$, by convolving the input signals with the impulse response of the filter, also known as filter coefficients, $h(k)$. Mathematically, this operation can be formulated as the convolution:

$$y(n) = \sum_{k=0}^{N} h(k)x(n-k), \tag{3.4}$$

where N is the filter order.

In a convolution, in order to multiply the filter coefficients with the input data samples, the time order of the input samples is flipped and stepped across the filters coefficients [5].

3.5 FIR Filter Structures

The basic arithmetic operation involved in the computation of the output of a FIR filter is multiplication and addition, as evident in (3.4). Since the input data sequence is to be flipped and stepped across the filter coefficients, a direct, isomorphic mapping of (3.4) will result in the structure, known as direct form (DF) FIR filter structure , shown in Fig. 3.3.

Another form of implementing FIR filter is derived from the DF FIR filter by applying certain transformations [122], known as TDF FIR filter, and is shown

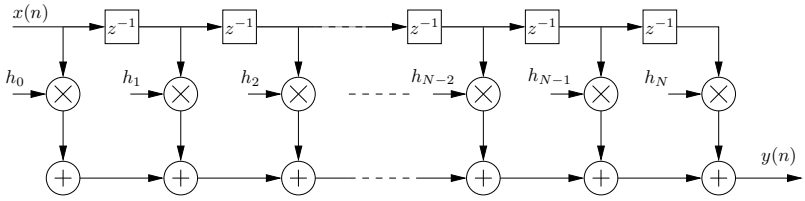

Figure 3.3: Direct form FIR filter.

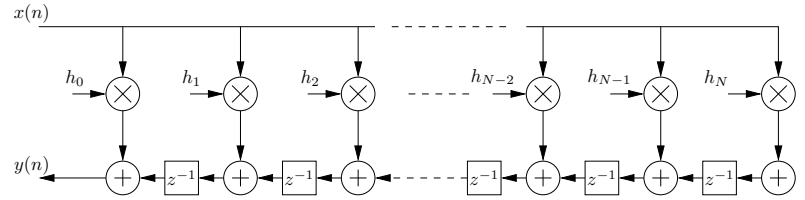

Figure 3.4: Transposed direct form FIR filter.

in Fig. 3.4.

As stated earlier, FIR filters can realize an exact linear phase response resulting in a symmetric/anti-symmetric impulse response. Implementation complexity of FIR filters can be reduced by exploiting the symmetric coefficients by reducing the number of multiplications. Such a linear-phase structure is shown in Fig. 3.5 using DF FIR structure. A TDF linear phase structure can also be derived from the structure shown in Fig. 3.5.

Other structures to realize FIR filters is to cascade several low-order filters, lattice structures and FFT based filter structures [2].

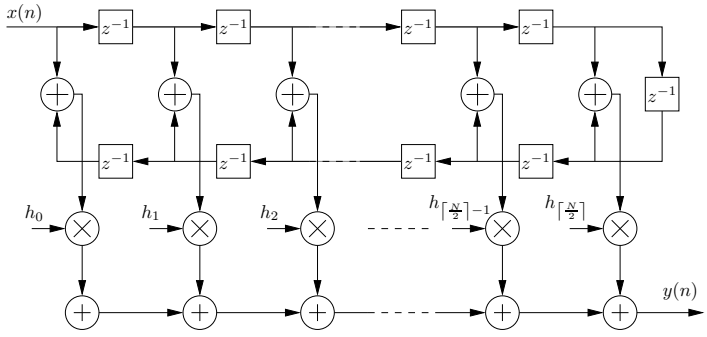

Figure 3.5: Direct form linear phase FIR filter.

3.6 Design of FIR Filters

The design of FIR filter is primarily concerned with obtaining filter coefficients that meet certain design specifications. These design specifications may be a desire magnitude or phase response [3]. A measure of "goodness" is established by which the nearness of the approximating response to the given response is determined. This is generally expressed in the form of an error measure which normally can take two different forms

- Minimax error
- Least-squared error

The specifications against which a filter is designed is typically specified in the frequency domain. For example, in the lowpass case, the desired magnitude response is usually given by:

$$D(\omega T) = \begin{cases} 1 & \omega T \in [0, \omega_c T] \\ 0 & \omega T \in [\omega_s T, \pi] \end{cases}, \tag{3.5}$$

where $\omega_c T$ and $\omega_s T$ are the pass-band and stop-band edges of the desired magnitude response. Since the filter design is an approximation to the required response, there will be deviations. These deviations or errors are termed as ripples, δ_c in the pass-band and δ_s in the stop-band. The amplitudes of these ripples is usually given in decibels in terms of the maximum pass-band variation and the minimum stop-band attenuation and is formulated as [3]:

$$A_{\max} = 20 \log_{10} \left(\frac{1 + \delta_c}{1 - \delta_c} \right) \text{dB}, \tag{3.6}$$

$$A_{\min} = -20 \log_{10} (\delta_s) \text{dB}. \tag{3.7}$$

The specifications of designing a filter can then be stated for the most general case as

$$\begin{aligned} D_c(\omega T) - \delta_c(\omega T) \leq \left| H\left(e^{j\omega T}\right) \right| &\leq D_c(\omega T) + \delta_c(\omega T), \quad \omega T \in X_c \\ \left| H\left(e^{j\omega T}\right) \right| &\leq \delta_s(\omega T), \quad \omega T \in X_s \end{aligned} \tag{3.8}$$

where X_c and X_s denote the pass-band and stop-band regions of the filter respectively, $\delta_c(\omega T)$ is the permissible deviation from the desired pass-band response $D_c(\omega T)$ and $\delta_s(\omega T)$ is the allowed deviation from zero in the stop-band region. These specifications can be reformulated to include a weight function $W(\omega T)$. This weighting function specifies the cost of the deviation from the desired function and allows obtaining different deviations in ripples in different frequency bands. The larger this function, the smaller is the ripple and can be included in the specification as [3]:

$$-\delta_c \leq W_c(\omega T)\left[\left|H\left(e^{j\omega T}\right)\right| - D_c(\omega T)\right] \leq \delta_c, \quad \omega T \in X_c$$
$$W_s(\omega T)\left|H\left(e^{j\omega T}\right)\right| \leq \delta_s. \quad \omega T \in X_s \tag{3.9}$$

These specifications can be combined to give the following form [3]:

$$|E(\omega T)| \leq \bar{\epsilon} \quad \omega T \in X = X_c \cup X_s, \tag{3.10}$$

where

$$E(\omega T) = W(\omega T)\left[\left|H\left(e^{j\omega T}\right)\right| - D(\omega T)\right], \tag{3.11}$$

$$\bar{\epsilon} = \delta_c, \tag{3.12}$$

$$D(\omega T) = \begin{cases} D_c(\omega T) & \omega T \in X_c \\ 0 & \omega T \in X_s \end{cases}, \tag{3.13}$$

$$W(\omega T) = \begin{cases} W_c(\omega T) & \omega T \in X_c \\ \frac{\delta_c}{\delta_s} W_s(\omega T) & \omega T \in X_s \end{cases}. \tag{3.14}$$

Equation (3.11) can also be written in terms of the real zero-phase frequency response $H_R(\omega T)$ because $|H\left(e^{j\omega T}\right)| = |H_R(\omega T)|$ [2].

$$E(\omega T) = W(\omega T)\left[H_R(\omega T) - D(\omega T)\right]. \tag{3.15}$$

3.6.1 Error Approximation

As stated earlier, normally the approximation of the deviation between the desired and achieved response can take two different forms. Each form is briefly described in this section.

A Minimax Error Designs

In minimax error designs, the filter coefficients are optimized to minimize the maximum error between the approximating and the desired response. This is also referred to as *Chebyshev approximation* and can be, in reference to (3.15), stated as [3]:

$$\epsilon = \max_{wT \in X} |E(\omega T)|. \tag{3.16}$$

B Least-Squared Error Designs

In least-squared error designs, the function to minimize is the L_2 norm of the error between the approximating and the desired response, described in (3.15) and can be formulated as [3, 63]

$$E_2 = \int_X W(\omega T)\left[H_R(\omega T) - D(\omega T)\right]. \tag{3.17}$$

3.6.2 FIR Filter Design by Optimization

FIR filters are also designed using the windowing technique where the Fourier series of an ideal filter is truncated and smoothed using a window function. However, this technique does not produce optimal filters in terms of complexity. One of the most common method to design linear phase FIR filter is to use numeric optimization procedures like linear programming [4, 6, 66, 75, 244–255]. Such methods are primarily used for minimax designs where the typical optimization goal is to minimize the maximum error between the approximating and desired response as given by (3.16). One of the most famous algorithms to design FIR filters using optimization is the one proposed by McClellan, Parks and Rabiner [256].

The specifications are the same as given by (3.9), except that $|H\left(e^{j\omega T}\right)|$ can be replaced by $|H_R(\omega T)|$, as mentioned in Section 3.6. The specification in (3.9) is for a general case. For low-pass filter, it can be translated into

$$\begin{aligned} 1-\delta_c \leq H_R(\omega T) &\leq 1+\delta_c, \quad \omega T \in [0, \omega_c T] \\ -\delta_s \leq H_R(\omega T) &\leq \delta_s, \qquad \omega T \in [\omega_s T, \pi] \end{aligned} \tag{3.18}$$

where δ_c, δ_s, $\omega_c T$ and $\omega_s T$ denote the pass-band ripple, stop-band ripple, pass-band edge and stop-band edge respectively [2].

3.6.3 Remez/Park-McClellan FIR Filter Design

The Park-McClellan FIR filter design method, also known as Remez Exchange, requires the specification of the filter order, N, all pass-band and stop-band edges and the ratios between the values of the peak desired pass-band and stop-band errors [256]. It then finds the unique set of filter coefficients that minimizes the weighted error function given in (3.16). In other words, it minimizes the maximum of the (*Chebyshev*) error function given by (3.15), where the desired function, $D(\omega T)$, typically has the value one in the pass-band and zero in the stop-band.

3.6.4 FIR Filter Design by Linear Programming

Design of FIR filters can also be viewed as constrained optimization problems which can be solved using linear programming (LP), integer linear programming (ILP) and mixed integer linear programming (MILP) methods [60, 62, 70, 75, 248, 253, 256]. These methods provide greater flexibility, as compared to Parks-McClellan method, of applying additional constraints on the frequency response and filter coefficients.

Linear programming problems are constrained optimization problems where the goal is to either minimize or maximize a certain objective function subject to a finite number of constraints. Mathematically, for a maximization problem, it can be stated as

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \\ & x, c \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{n \times m} \end{aligned} \tag{3.19}$$

where $c^T x$ is the object function, A is a $n \times m$ matrix with m constraints and n optimization variables. A number of real world problems requires integer variables. If all the variables are integers, then we have an ILP problem which can be stated as (3.19) with the addition constraint of $x \in \mathbb{Z}^n$.

The design of FIR filters with finite word length constraint can be modeled as an ILP problem because even though all filter coefficients are fixed point fractional numbers, they can be modeled as integers. MILP problem is formulated if there is a non-integer variable like scaling or pass-band gain. There are two important algorithms for solving ILPs: branch and bound (BB) and cutting plane.

In ILP, the FIR optimization problem is same as in the linear programming case with the addition of a constraint that all coefficients will now be integers. A general FIR optimization problem can be stated as the following minimization problem [2]:

$$\begin{aligned} \text{minimize} \quad & \delta \\ \text{subject to} \quad & -\delta \leq E(\omega_i T) \leq \delta, \quad i = 1, 2, \ldots, K \end{aligned} \tag{3.20}$$

where δ is the approximation error, K is the number of frequency points and

$$E(\omega_i T) = W(\omega_i T)\left[H_R(\omega_i T) - D(\omega_i T)\right], \quad \omega_i T \in \Omega \tag{3.21}$$

where Ω is the union of the passband and stopband regions and is a dense set of frequency samples taken from the passbands and stopbands, including the band edges. $D(\omega_i T)$ is the desired function to be approximated by $H_R(\omega_i T)$. Formulations for $D(\omega_i T)$ and $W(\omega_i T)$ are given in (3.5) and (3.14).

As stated earlier, additional constraints can be applied to either the frequency response or the filter coefficients to, for e.g., reduce the arithmetic complexity of implementing the FIR filters. In case of additional constraints in the frequency domain, specific values of frequency can have a prescribed value, like zero. This constraint is utilized in designing sparse filters where some of the coefficients are reduced to zero [9–11, 15, 25–27].

An additional constraint on the filter coefficients typically take the form of a finite word length constraint [253–255, 257, 258]. FIR filters realized using Park-McClellans method gives infinite precision filter coefficients which cannot be implemented in hardware. One of the ways is to round the coefficients which is a non-optimal approach. A better way is to use linear programming with the additional finite word length constraint. A comparison of a filter realized

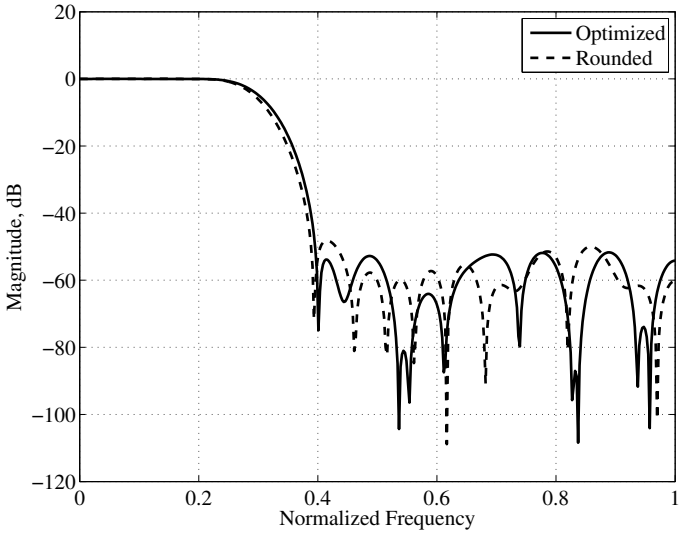

Figure 3.6: Magnitude response of FIR filter. Coefficients obtained by linear programming with finite word length constraints and rounding of coefficients obtained by Park-McClellans method. $N = 40$, $Q = 10$.

using Park-McClellans method where the coefficients are rounded to 10-bits and another one realized using linear programming with finite word length constraint is shown in Fig. 3.6. It can be seen that the optimized filter achieves lower stop-band attenuation as compared to the rounded one.

Apart from constraining the filter design problem to finite word length, it is beneficial from implementation complexity point of view to reduce the number of non-zero digits within a coefficient [62–67, 70], normally referred to as reducing the number of signed power of two (SPT) in a coefficient. Reduction in the number of non-zero digits reduces the number of adders required to realize the multiplication with a coefficient using a shift-add network. Typically CSD number representation is used while solving these type of problems. This is because a B-bit CSD code has, on average, $B/3$ number of non-zero digits as compared to 2's complement number representation which on average has $B/2$ non-zero bits [122].

To optimize the number of SPT terms in a coefficients, the fixed-point coefficient of wordlength B can be represented as a sum of SPT terms in the general form [70]

$$h_m = \sum_{i=1}^{B} s_{m,i} 2^{-i}, \tag{3.22}$$

where $s_i \in -1, 0, 1$. To make the formulation of the optimization linear, it is beneficial to represent the SPT terms using 0/1 variables as

$$h_m = \sum_{i=1}^{B} \left(a_{m,i}^{+} - a_{m,i}^{-}\right) 2^{-i}, \tag{3.23}$$

where $a_{m,i}^{+} \in 0, 1$ and $a_{m,i}^{-} \in 0, 1$. It leads to the following constraint to ensure CSD coefficients

$$a_{m,i}^{+} + a_{m,i}^{-} + a_{m,i+1}^{+} + a_{m,i+1}^{-} \leq 1 \tag{3.24}$$

and the objective function can be formulated as

$$\text{minimize} \quad \sum_{i=1}^{M} \sum_{i=1}^{B} \left(a_{m,i}^{+} + a_{m,i}^{-}\right), \tag{3.25}$$

where $M = \frac{N}{2} + 1$ for a type-I FIR filter.

These techniques of reducing the number of SPT terms has been further extended to combine with sub-expression sharing where the filters are designed such that not only the number of SPT terms are minimized, the resulting shift-add network is also shared optimally [77, 78, 259].

3.6.5 FIR Filter Design by Cascade of Sub-Filters

To reduce the implementation cost of an FIR filter, it can be designed using cascade of sub-filters as building blocks. It not only allows the frequency response of the composite filter to be better than that of the sub-filters but also leads towards the reduction in the implementation complexity [54].

One approach to cascade sub-filters is to use filters with different powers of z^{-1}. [12, 13, 18, 47–51]. Another approach is to design the filter by interconnecting a number of identical sub-filters in the form of a tapped cascaded interconnection with the help of additional adders and multipliers [52–54].

This section briefly describes the periodic sub-filter approach in Section A as it is most relevant to the work presented in this work. For details regarding tapped cascade interconnection of identical sub-filters, refer to [3].

A Periodic Sub-filters

The basic building block to realize an FIR filter using periodic sub-filters is the periodic filter. It is a filter whose transfer function is obtained by replacing $z^{-1}1$ with z^{-L}, i.e., there are L delays between successive filter taps, where L is termed as the upsampling factor. The transfer function is of the form [3]:

$$G\left(z^{L}\right) = \sum_{n=0}^{N_G} g[n] z^{-nL}, \quad g\left[N_G - n\right] = f\left[n\right]. \tag{3.26}$$

The resulting frequency spectrum is a compressed version of $G\left(z\right)$, i.e., the interval $[0, L\pi]$ is squeezed onto $[0, \pi]$. An example of the difference between in the spectrum between $G\left(z\right)$ and $G\left(z^{L}\right)$ is shown in Fig. 3.7.

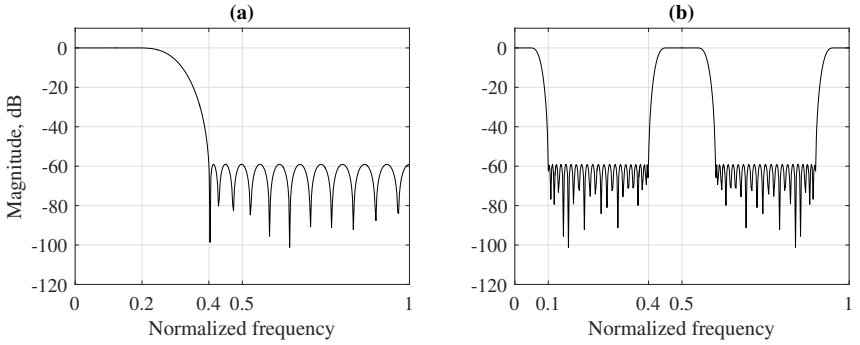

Figure 3.7: (a) Model filter, $G(z)$ with $\omega_c T = 0.2$, $\omega_s T = 0.4$ and (b) periodic model filter, $G(z)^L$ with $L = 4$.

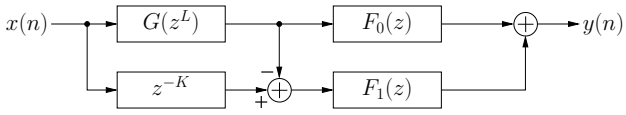

Figure 3.8: Block diagram of arbitrary-band FRM filter.

As the spectrum shows, in addition to a compression of the band edges of the filter, there are additional components, called *images* in the spectrum. This means that the periodic filter cannot be a stand-alone solution and needs to be combined with conventional non-periodic filters to achieve the desired response.

The combining of period filters with non-periodic filters gives rise to the technique of FRM, proposed in [48]. The general FRM structure shown in Fig.3.8 allows the implementation of arbitrary-band FIR filters [48]. It utilizes a periodic model filter, $G(z)$, the complement of the periodic model filter, $G_c\left(z^L\right) = z^{-\frac{N_G L}{2}} - G\left(z^L\right)$, and two masking filters, $F_0(z)$ and $F_1(z)$, with a transfer function

$$H(z) = G\left(z^L\right) F_0(z) + G_c\left(z^L\right) F_1(z), \tag{3.27}$$

where $G(z)$ is termed the model filter and $G\left(z^L\right)$ the periodic model filter. One or several pass-bands of the periodic model filters are extracted by the two masking filters, $F_0(z)$ and $F_1(z)$. The transition band of the overall filter is equal to the transition band of one of the transition bands of either $G\left(z^L\right)$ (type-1) or $G_c\left(z^L\right)$ (type-2) [48]. Note that N_G must to be even. The filter order required for the model and masking filter is considerably lower than a normal single stage implementation. The delay line of the model filter is used to implement the delay needed for the complementary filter. The structure of FRM technique is shown in Fig. 3.8.

Important to note that for a given set of band edges, $\omega_c T$ and $\omega_s T$ and

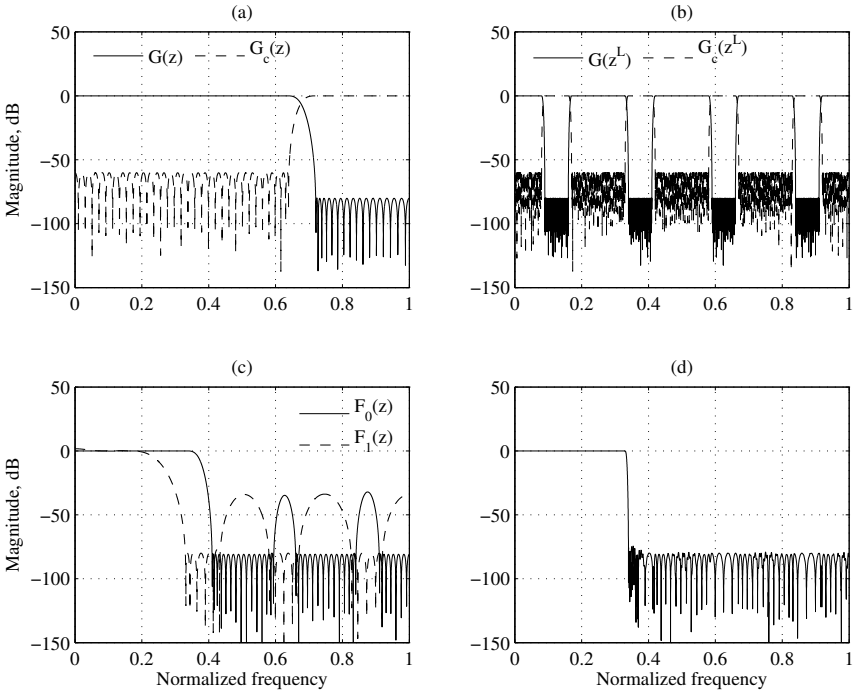

Figure 3.9: Magnitude response of type-1 FRM FIR filter with $w_cT = 0.33\pi$, $w_sT = 0.34\pi$ and $L = 8$. (a) Model, $G(z)$ and complementary filter, $G_c(z)$, (b) periodic model, $G\left(z^L\right)$ and complementary filter, $G_c\left(z^L\right)$, (c) masking filters, $F_0(z)$ and $F_1(z)$ and (d) target filter, $H(z)$.

upsampling factor L, only one of type-1 or type-2 will yield a valid solution. And there is also a possibility that for some choice of L, neither yield a valid solution. The magnitude response of various sub-filters and the target specification for type-1 and type-2 FRM FIR filter is shown in Figs. 3.9 and 3.10.

The general structure shown in Fig. 3.8 can be simplified for narrow-band ($\omega_cT \leq \pi/2$ for low-pass filters) and wide-band ($\omega_cT \geq \pi/2$ for low-pass filters) FIR filters [47, 50]. This simplified structure only consists of a periodic model and masking filter for narrow-band implementation and an additional complementary filter for wide-band implementation. The narrow-band implementation is also termed as interpolated FIR (IFIR) and the transfer function is given by:

$$H(z) = G\left(z^L\right) F(z), \tag{3.28}$$

where $G(z)$, the model filter, has its band edges at L times the band edges of the target filter. This considerably lowers the filter order required to realize $G(z)$. The structure of IFIR is shown in Fig. 3.11 and the magnitude response of the sub-filters and target filter is shown in Fig. 3.12.

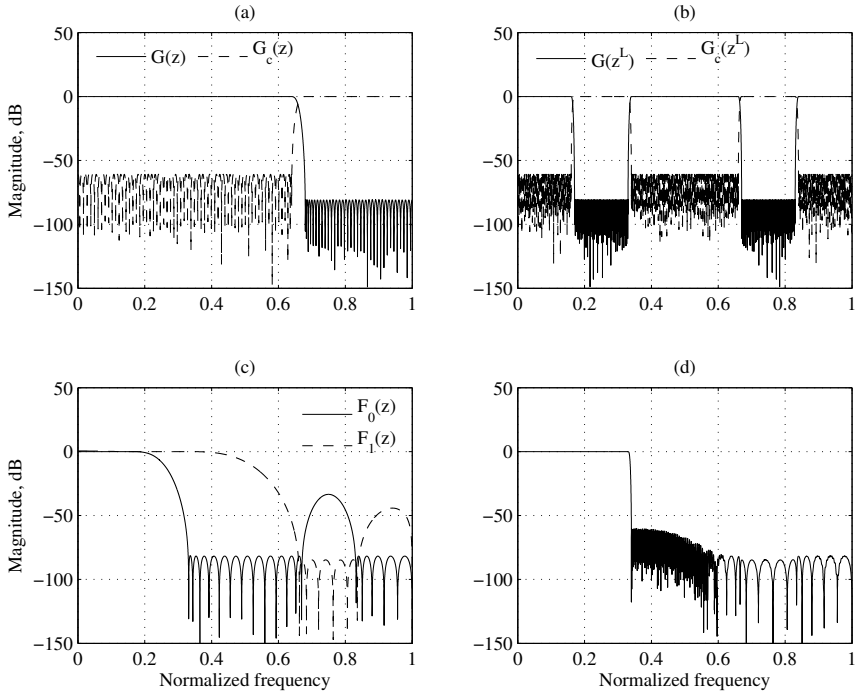

Figure 3.10: Magnitude response of type-2 FRM FIR filter with $w_cT = 0.33\pi$, $w_sT = 0.34\pi$ and $L = 4$. (a) Model, $G(z)$ and complementary filter, $G_c(z)$, (b) periodic model, $G\left(z^L\right)$ and complementary filter, $G_c\left(z^L\right)$, (c) masking filters, $F_0(z)$ and $F_1(z)$ and (d) target filter, $H(z)$.

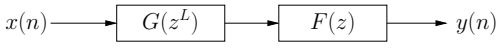

Figure 3.11: Narrow-band FRM filter block diagram.

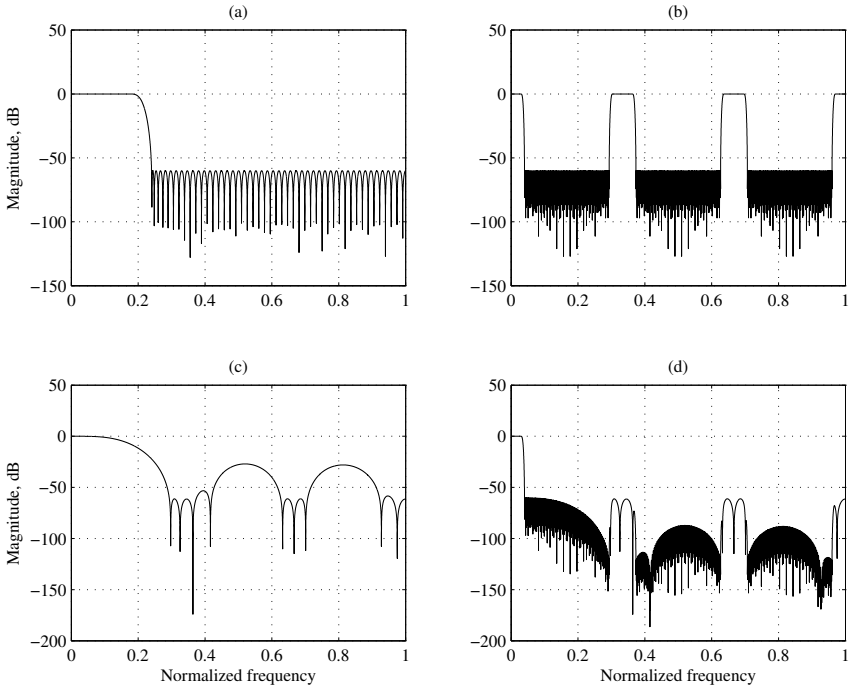

Figure 3.12: Magnitude response of IFIR filter with $w_c T = 0.03\pi$, $w_s T = 0.04\pi$ and $L = 6$. (a) Model filter, $G(z)$, (b) periodic model filter, $G\left(z^L\right)$, (c) masking filter, $F(z)$ and (d) target filter, $H(z)$.

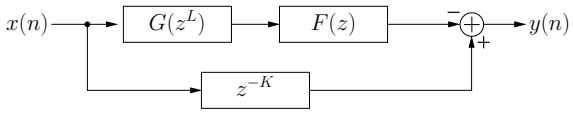

Figure 3.13: Wide-band FRM filter block diagram.

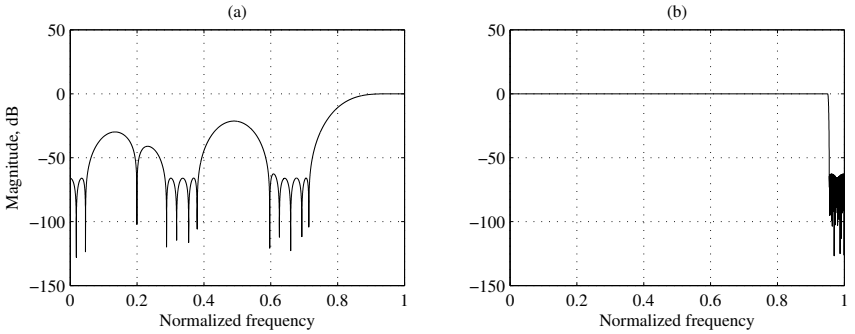

Figure 3.14: Magnitude response of IFIR filter with $w_cT = 0.95\pi$, $w_sT = 0.955\pi$ and $L = 6$. (a) High-pass masking filter, $F(z)$ and (d) target filter, $H(z)$.

Wide-band (WB) filters are obtained by computing the complementary transfer function, as shown in Fig. 3.13, with a transfer function as [2]:

$$H(z) = z^{-K} - G\left(z^L\right) F(z). \tag{3.29}$$

As mentioned earlier, the implementation of z^{-K}, where K is the delay through $G\left(z^L\right)$ and F_z, can be obtained from the delay line implementing $G\left(z^L\right)$. This is because the filter order of $G\left(z^L\right)$ is typically more than twice the filter order required for $F(z)$. The magnitude response of $G(z)$ and $G\left(z^L\right)$ for a wide-band FRM filter is same as that shown in Fig. 3.12. The masking filter is a high-pass filter which when cascaded with $G\left(z^L\right)$ produces a narrow-band high-pass filter. The complementary filter then realizes the wide-band low-pass filter as shown in Fig. 3.14.

At the heart of the reduction in the implementation cost is the upsampling factor, L. The higher the value of L, the lower the implementation cost of the model filter. However, an important question to ask here is whether arbitrarily increasing the value of L will reduce the cost of the overall structure. To answer this question, one needs to look at the relationship between L and complexity of the masking filter(s). As L increases, more images are introduced in $[0, \pi]$. This increases the requirement on the masking filter, i.e., the transition width of the masking filter reduces, thereby increasing its filter order and its implementation cost. Initially, with an increasing L, the decrease in the implementation cost of the model filter will dominate, reducing the overall implementation cost of FRM. However, a further increase in the value of L will make the implemen-

tation cost of the masking filter to dominate, thus increasing the overall cost of implementing FRM FIR filters. This leads to that there is an optimal value of L which gives the least implementation cost. This is shown Fig. 3.15 and also compared with the implementation cost of a single stage FIR (SSF) filter realization.

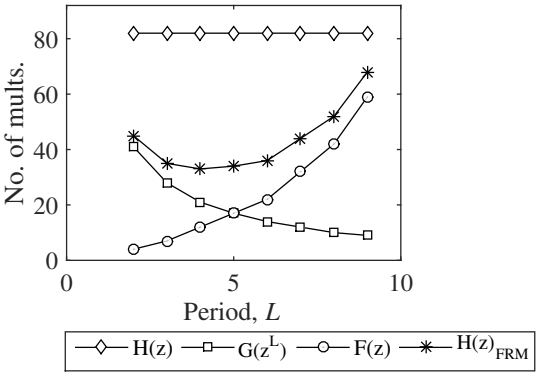

Figure 3.15: Point of minimum complexity for narrow-band FRM for $w_cT = 0.05\pi$ and $w_sT = 0.1\pi$.

One of the dis-advantages of the FRM technique is that the overall filter order required is high, primarily because of upsampling. This leads to a large number of delay elements leading to a higher register usage when implemented either on an ASIC or FPGA. However, high clock rates obtainable using current state-of-the-art hardware platforms typically do not correspond to the required sample rate of the target application. This leads to time-multiplexed architectures where hardware resources are re-used to further reduce the implementation cost. It has been shown in Paper A that even though FRM has a high register cost, when time-multiplexed, it requires less memory than single-stage implementation due to better utilization of memories. Beyond the regular design of FIR filters using the FRM technique, a number of optimizations have been published in the literature to further improve the reduction in the implementation cost. The original idea of FRM was generalized in [260] where additional savings were obtained by interpolating the masking filters and using a common masking filter which removes the images introduced by the interpolation of the masking filters.

A multi-stage FRM technique was used to implement the model and complementary model filter in [7] and criterion was established for selecting the optimum number of stages. It was shown that the optimal value of L approaches e as the number of stages increase. A similar technique to implement the masking filters using a cascade of a common sub-filter and a equalizer which equalizes the transition-band of the masking filters was proposed in [261].

The IFIR approach proposed in [50] also proposes a multi-stage realization

of the masking filter. The IFIR approach has also been incorporated in the design of the periodic model filter, also known as the shaping filter, to further lower the requirement on the model filter. A similar technique, based on the pre-filter equalizer, has been used to to design different sub-filters [56] where either the periodic model filter, masking filters or all sub-filters are replaced by the pre-filter equalizer pair.

IFIR filter is also used in the approach proposed by [262], where the shaping filter is decoupled from the masking filters with the insertion of a decoupling filter between them. The shaping filter in the FRM can also be implemented using a cascade of two or three short filters [58]. This balances the length of each sub-filter and helps improve the throughput in a direct-form implementation. This is because the shaping filter has the largest length of all the sub-filters in the original FRM technique and has the greatest influence on the throughput.

Design of FIR filters using the FRM technique without the use of masking filters has been proposed for narrow-band [12], for both narrow-band and wide-band [13] and for all types of FRM technique [18]. In these papers, the authors, instead of using a cascade of periodic model and masking filters, used a cascade of a number of same periodic model filters with different values of L. The idea behind using the same filter is to use the same arithmetic structure which can be multiplexed to reduce the required number of multipliers and adders. Furthermore, optimization procedures were utilized to minimize the filter order.

Use of optimization and linear programming has also been used to design the sub-filters. Typically, each of the sub-filter is designed by optimization separately. A significant reduction in the implementation cost can be achieved by simultaneously optimizing the sub-filters together [20]. Joint optimization has been applied to both the original FRM [48] and the generalized FRM [260].

Further contributions towards the design of FIR filters using the FRM technique has involved MILP design with few SPT terms [60], development of new optimization procedures with finite word-length constraints [21], non-periodic sub-filters [26, 27] and implementation on hardware platforms like FPGAs [16, 24, 58, 263, 264].

3.6.6 Sparse FIR Filter Design

The implementation complexity of a FIR filter is primarily influenced by the multiplication of the filter coefficient with the input data. The complexity can be reduced if some of the filter coefficients are forced to zero while designing a filter to meet a certain specification. Filters realized through this technique are referred to as sparse FIR filters [265]. This class of filters include the Nyquist or M^{th}-band filters and non-periodic sparse filters. A number of different techniques have been proposed for the design and implementation of such filters [9–11, 15, 22, 265–271]. The periodic model or the shaping filter of the FRM/IFIR technique is also a sparse filter.

The M^{th}-band linear phase FIR filters are filters whose every M^{th} filter coefficient is zero except the middle coefficient. Hence they are typically even

order filters [272]. They find application in intersymbol interference rejection, perfect reconstruction filter banks and in interpolation and decimation by a factor M. However there are some additional constraints imposed by a M^{th}-band filter design. One of them is that the transition width of the filter should be symmetric across π/M and that the pass-band (δ_c) and stop-band (δ_s) ripples should be equal. These additional constraints may lead to an overly constrained solution as compared to the original solution [10]. Thus these filters are not used for more general applications [267]. It has been shown that it is possible to replace a M^{th}-band filter with a general FIR filter by increasing the stop-band attenuation and transition width. A class of M^{th}-band filter is the half-band filter where every other coefficient is zero. By relaxing the constraints on the pass-band ripple and edge, half-band like non-periodic sparse filters have been designed which achieve lower complexity as compared to half-band filters [10].

Non-periodic sparse filters have also been designed by forcing coefficients which make small or no contribution to the spectrum shaping of the filter [265] which can also be used for sub-filters in an FRM design [26, 27]. Approximate methods for obtaining reasonably sparse filters have also been proposed to reduce the computation time of the optimization routines. Included in this technique is to re-optimize a non-sparse filter after forcing small valued coefficients to zero [273], determine those coefficients that can be permitted to be zero [15] and successively thinning the impulse response of a pre-designed filters and re-optimizing the remaining coefficients [268]. A direct-form implementation of these filters have been shown to provide improvement with moderately wide pass-bands and transition widths by providing a significant improvement in the stop-band attenuation given a fixed number of multipliers [267].

Typically, the design of all these filters employ LP techniques and use methods like branch-and-bound [268]. Additional constraints are added to the filter design problem in the form on non-zero binary variables $x_i \in [0, 1]$. The objective is then to minimize the sum of these variables. Additional variable indicating the filter order can be included in the objective function of the optimization routine to jointly optimize the filter order resulting in further reduction in the implementation complexity [10, 271].

3.7 Fast FIR Filters

The process of FIR filtering can be seen as a convolution operation given by the following equation

$$y(n) = \sum_{k=0}^{N} h(k)x(n-k). \tag{3.30}$$

This convolution can also be seen as a polynomial multiplication when $x(n)$ and $h(n)$ are realized as polynomials in the z-domain which transforms the convolution into multiplication as:

$$Y(z) = H(z)X(z). \tag{3.31}$$

To visualize a polynomial multiplication, consider a P- and a Q-term polynomials:

$$p(x) = a_{P-1}x^{P-1} + \cdots + a_1 x + a_0, \tag{3.32}$$
$$q(x) = b_{Q-1}x^{Q-1} + \cdots + b_1 x + b_0, \tag{3.33}$$

and the product is defined as:

$$u(x) = c_{N-1}x^{N-1} + \cdots + c_1 x + c_0. \tag{3.34}$$

Multiplication of a P- and a Q-term polynomial will produce PQ multiplications. However, the multiplications can be performed more efficiently in a three step process; evaluation of the input polynomial at $N = P + Q - 1$ points, point wise multiplication and reconstruction of the product, $u(x)$. The first and the last step can be referred to as *pre-computation* and *post-computation*, respectively. In this way there will be a total of $P + Q - 1$ multiplications with some non-trivial constant multiplications.

Evaluation at $N = P + Q - 1$ points is needed because multiplication of a P- and a Q-term polynomials produces an output polynomial of $N = P + Q - 1$ terms and that will be uniquely defined at N points. Consider the polynomials where $P = Q = 2$ with their product:

$$\begin{aligned} p(x) &= a_1 x + a_0 \\ q(x) &= b_1 x + b_0 \\ u(x) &= c_2 x^2 + c_1 x + c0 = (a_1 x + a_0)(b_1 x + b_0). \end{aligned} \tag{3.35}$$

The product polynomial is a 3-term polynomial and can be uniquely defined by three points. Evaluation of $u(x)$ at $x = 0, 1, \infty$ gives

$$x = 0 \quad \Rightarrow \quad c_0 = a_0 b_0 \tag{3.36}$$
$$x = 1 \quad \Rightarrow \quad c_2 + c_1 + c_0 = (a_1 + a_0)(b_1 + b_0) \tag{3.37}$$
$$x = \infty \quad \Rightarrow \quad c_2 = a_1 b_1. \tag{3.38}$$

This can be represented in matrix form:

$$\begin{bmatrix} a_0 b_0 \\ (a_1 + a_0)(b_1 + b_0) \\ a_1 b_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}, \tag{3.39}$$

where the matrix of ones and zeros can be inverted to determine the output polynomial coefficients

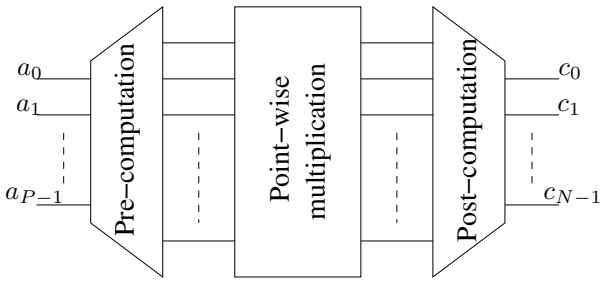

Figure 3.16: Three step polynomial multiplication with pre-computation, point-wise multiplication and post-computation.

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 b_0 \\ (a_1 + a_0)(b_1 + b_0) \\ a_1 b_1 \end{bmatrix}. \tag{3.40}$$

The second matrix on the right hand side can be further transformed into a diagonal matrix consisting only of coefficients $p(x)$, a single column matrix of coefficients of $q(x)$ and a matrix connecting the two.

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \overbrace{\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}}^{\text{post-comp.}} \underbrace{\begin{bmatrix} a_0 & 0 & 0 \\ 0 & a_1 + a_0 & 0 \\ 0 & 0 & a_1 \end{bmatrix}}_{\text{mult.}} \overbrace{\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}}^{\text{pre-comp.}} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}, \tag{3.41}$$

where the three stages are also highlighted. The post-computation is required because the point-wise multiplication will produce extra terms which needs to be removed to achieve the correct output polynomial. A general structure to achieve polynomial multiplication is shown in Fig. 3.16. The length of the number of polynomials can also be different.

Similarly, if the polynomial is interpolated at $X = e^{\frac{-j2\pi k}{G}}$ where G is the discrete fourier transform (DFT) size and $k \in 0 \ldots G-1$, it will result in DFT-based polynomial multiplication. In other words, two compute the multiplication of two polynomials, one can compute the DFT of the two polynomials, perform point-wise multiplication and compute the inverse DFT (IDFT), thus implementing a convolution in time-domain through multiplication in the frequency domain. The inverse matrix shown as post-computation in (3.41) will be an IDFT matrix while the pre-computation matrix will be a DFT matrix. It can be formulated as:

$$c = \mathbf{IDFT}_G \left(\text{diag}\{\mathbf{DFT}_G \mathbf{a}\} \mathbf{DFT}_G \mathbf{b} \right), \tag{3.42}$$

where $\mathbf{a} = [a_0\ a_1\ \ldots\ a_{P-1}]$, $\mathbf{b} = [b_0\ b_1\ \ldots\ b_{Q-1}]$ and **DFT** and **IDFT** are matrices in G points. The vectors **a** and **b** can be appended be zeros to make G a power of two, which will make the **DFT** and **IDFT** matrices simpler. The number of multiplications in the matrix operations can be large but they can be efficiently implemented using an fast fourier transform (FFT) algorithm.

To realize an FIR filter using polynomial multiplication, the z-transform formulation shown in (3.31) can be transformed as a polynomial product where the polynomial weights/coefficients are the polyphase components [274]. It is given as:

$$\sum_{i=0}^{F-1} Y_i\left(z^F\right) z^{-i} = \sum_{k=0}^{F-1} H_k\left(z^F\right) z^{-k} \sum_{j=0}^{F-1} X_j\left(z^F\right) z^{-j}, \tag{3.43}$$

where each of $Y_i(z^F)z^{-i}$, $H_k(z^F)z^{-k}$ and $X_j(z^F)z^{-j}$ is itself a polynomial in z^F [275].

This polynomial can also be evaluated at some arbitrary points or at points given by $Z^{-1} = e^{\frac{-j2\pi k}{G}}$ for a DFT based implementation. Consider, for the case of $F = 2$,

$$\begin{aligned} Y(z) &= Y_0\left(z^2\right) + z - 1Y_1\left(z^2\right) \\ &= \left(H_0\left(z^2\right) + z - 1H_1\left(z^2\right)\right)\left(X_0\left(z^2\right) + z - 1X_1\left(z^2\right)\right) \\ &= c_0 + z^{-1}c_1 + z^{-2}c_2, \end{aligned} \tag{3.44}$$

which leads to:

$$\begin{aligned} Y_0\left(z^2\right) &= c_0 + z^{-2}c_2 \\ Y_1\left(z^2\right) &= c_1. \end{aligned} \tag{3.45}$$

which is known as overlap-ad. In general, for F polyphase branches:

$$\sum_{i=0}^{F-1} Y_i\left(z^F\right) z^{-i} = \sum_{j=0}^{F-1} \left[\left(c_j + c_{F+j}z^{-F}\right) z^{-j}\right] \tag{3.46}$$

where an F parallel implementation entails execution at F times lower sample rate, reducing $z^{\pm F}$ to $z^{\pm 1}$. Evaluation (3.44) at $\pm 1, 0$ will result in the following formulation

$$\begin{aligned} \begin{bmatrix} Y_0(z) \\ Y_1(z) \end{bmatrix} = \overbrace{\begin{bmatrix} 1 & 0 & z^{-1} \\ 0 & 1 & 0 \end{bmatrix}}^{\text{overlap-add}} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & -\frac{1}{2} \\ -1 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \\ \text{diag}\begin{bmatrix} H_0(z) \\ H_0(z) + H_1(z) \\ H_0(z) - H_1(z) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} X_0(z) \\ X_1(z) \end{bmatrix}. \end{aligned} \tag{3.47}$$

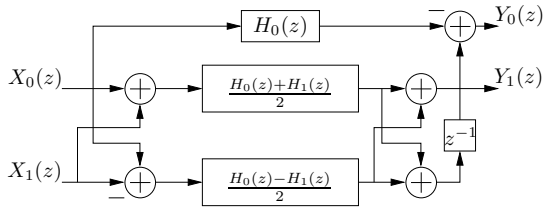

Figure 3.17: FIR filter implementation using polynomial multiplication.

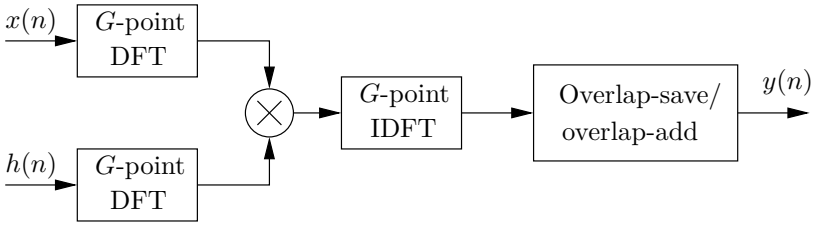

Figure 3.18: FIR implementation using DFT.

The resulting structure is given in Fig. 3.17. If the evaluation points were based on DFT, then the resulting filters will be complex. Thus the FIR filter equation is transformed as a product of two finite degree polynomials where the computation is broken down into three main parts. First is interpolation of the input polynomials at different points, the actual filtering which is performed on the F polyphase branches of the FIR filter and reconstruction of the filter output [276]. Finally, an overlap-add operation is needed to obtain $Y_i(z)$ [5].

The result is the breaking up of the filter into a number of sub-filters which operate in parallel to reduce the computational load. This is known as fast FIR (FFA). Each sub-filter in the resulting design can be further divided into ever shorter length sub-filters which reduces the arithmetic complexity, as argued in [277].

In fact, the typical implementation of FIR filtering using DFT, as shown Fig. 3.18, is the same as discussed above, but with evaluation points based in the complex domain on the unit circle. For fast computation, DFT can be replaced by FFT [276]. The use of transforms for fast processing of convolution was first proposed in [278].

For DFT based implementation, evaluation at $G = 2^g$ points produces simple matrix. The DFT sizes G shown in Fig. 3.18 must be equal and dependent on the filter order and size of the input data. For a FIR filter, $h(n)$ is of a finite duration of $Q = N + 1$. Assuming $x(n)$ to be finite of duration L, the duration of the output sequence $y(n)$ is $L + Q - 1$. Thus $G \geq L + Q - 1$, and $x(n)$ and $h(n)$ must be zero-padded to make their lengths equal to G.

However, for all practical purposes, the size of the input $x(n)$ is so large that it needs to be partitioned into multiple blocks of length L, denoted by $x_m(n)$,

and processed individually. Thus DFT based FIR filtering is also referred to as block convolution.

The overlapping operation is needed because either there will be overlapping output blocks or some part of the block will have corrupted data. For overlapping output blocks, overlap-add method is used and for the later, overlap-save method is used.

For overlap-add method, both $h(n)$ and the input block, $x_m(n)$, are zero-padded to make their length equal to G. The short convolutions between the two sequences will be of length G which will map to the structure shown in Fig. 3.18. However, there will be overlap between the output blocks. This is because the first convolution will be defined for $0 \leq l \leq L+Q-2$ while the second convolution will be defined for $L \leq l \leq 2L+Q-2$ and so on, meaning there is an overlap of $Q-1$ samples. These overlapping samples are added, given the method the name overlap-add.

Instead of performing linear convolutions of size $L+Q-1$, a circular convolution can be performed of length L. However, since the circular convolution length is smaller than $L+Q-1$, there will be aliasing or in simpler words, the first $Q-1$ samples will not correspond to the linear convolution term and will be rejected. To save from this aliasing, the first input block, $x_0(n)$ will have $Q-1$ zeros padded at the beginning while $Q-1$ samples from each input block will be padded to the beginning of the next block, resulting in overlapping blocks of length $L+Q-1$. This is the overlap-save method since a part of each input block is being saved for the next.

3.8 FIR Filter using Alternate Number Systems

Two alternate number systems, the LNS and RNS were introduced in Chapter 2. These number systems reduce the complexity of implementing multiplication which dominates the complexity cost of implementing FIR filters. A brief description of the use of these number systems in implementing FIR filters is given in Sections 3.8.1 and 3.8.2, respectively.

3.8.1 FIR Filter using Logarithmic Number System

The inherent simplification of the multiplication operation to an addition has made LNS an attractive option to implement FIR filters [82, 85, 86, 90–92, 95, 96, 279]. This simplification of the multiplication allows a high speed implementation of FIR filters over a wide dynamic range. Numbers represented using LNS have been shown to have a higher dynamic range [85] and better round-off noise performance [280–282] as compared to floating-point based numbers for a given number of bits.

It has been shown that LNS requires a reduced data word length to achieve the same SNR when compared to fixed point representations [90, 92], with a reported power saving of nearly 60% in some cases. In [283], the authors analyze

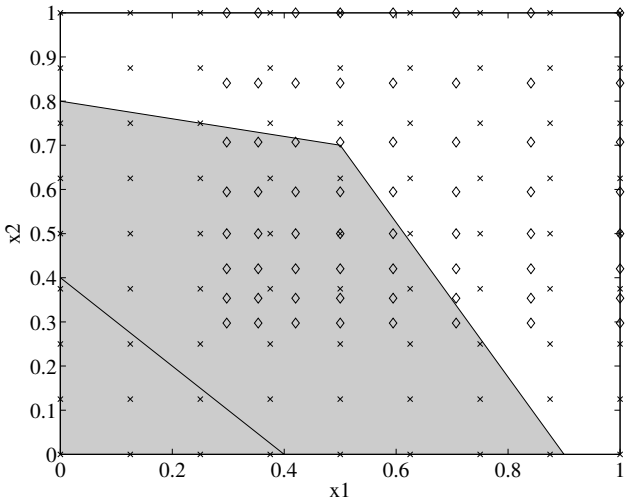

Figure 3.19: Example solution space for ILP with LNS (⋄) and linear (×) integers with three fractional bits with valid region in gray and the line representing the pareto front.

the round-off error accumulation in the direct realization of LNS based recursive digital filters and have shown that LNS gives superior filtering performance as compared to floating-point number system of equivalent word length and range.

One of the drawbacks of using LNS is the complexity of implementing additions/subtractions which typically require LUTs to implement them. In [92], the authors propose techniques for low-power implementation of addition/subtraction for LNS and quantify the impact of partitioning LUTs on complexity, performance and power dissipation. Furthermore, in [92], techniques for low power implementation of LNS MAC units are investigated.

The design of FIR filters directly in the LNS domain, however, is different to designing them using linear numbers like 2C, CSD and other. This is because the search space of for an LNS ILP problem is different from that of a linear representation, as shown in Fig. 3.19. In Paper B, an approach to directly design FIR filter using ILP in the LNS domain with finite word length constraint is presented and is shown to achieve better approximation error as compared to a standard FIR filter designed by optimization in the linear domain.

3.8.2 FIR Filter using Residue Number System

The RNS, introduced in [237], are well known for fast multiplication and addition [80]. This property suits the implementation of FIR filters in applications, like communication systems, which demand speed and low-power consumption [83]. Furthermore, FIR filters implemented using RNS produces out-

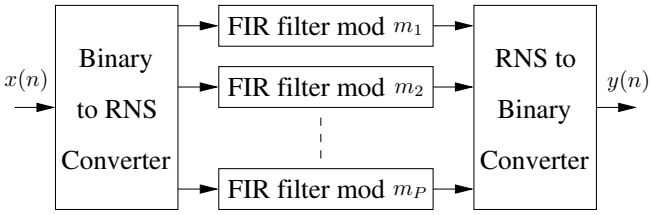

Figure 3.20: RNS FIR filter.

puts with full precision which is attractive for a number of applications radar and image processing [80].

RNS allows the division of the dynamic range into smaller ranges, operations on which can be performed in parallel. This allows for high-speed parallel implementation. The power consumption can also be reduced by taking the advantage of this parallelism [284]. Although RNS has its shortcomings, it is well suited to applications that are primarily composed of multiplications and additions, of which FIR filter is the best example [284].

To implement FIR filter using RNS, the filter is decomposed into P filters working in parallel, where P is the number of moduli used in the RNS representation [83]. This is shown in Fig. 3.20.

One of the earliest attempts to implement FIR filters using RNS is reported in [80]. In it, the authors propose a hardware implementation of the chinese remainder theorem for conversion of RNS coded outputs into binary numbers. In [83], it is shown that FIR filters implemented using RNS is smaller and consumes less power that the traditional FIR filter at the same clock rate when the number of taps is larger than one. Authors in [284] use a voltage reduction scheme operating at a specified sample speed which becomes increasingly favorable as the word-length is increased. In [82], the feasibility of implemented RNS based FIR filters on FPGAs was demonstrated including the conversion to and from binary using 12-bit word length and a base set of $5, 7, 11, 13$. A low power and low leakage implementation of RNS FIR filters is reported in [285] which takes the properties of RNS to reduce static power dissipation when implementing the filter on 90 nm CMOS technology.

Chapter 4

Particle Filters

4.1 Introduction

In problems where the state of a system cannot be determined analytically because it is hidden, filtering refers to estimation of the state of a system, from a set of observations that are corrupted by noise. The system is modeled as a Markov process and the model where the states are unknown or hidden is known as hidden Markov model (HMM). In other words, it refers to determining the distribution at time n, given observations up to n [44, 110, 111, 286].

The specific nature of the estimation depends greatly on the state to be estimated, the evolution of state with time and relationship between the state to the observations and noise sources. The model that captures the evolution of states is called the state transition model (STM). Conceptually, estimation makes a prediction and refines/validates it with an observation over a long period of time. Particle filtering is used when the state transition and the observation models are non linear and noise is non-Gaussian, in contrast to Kalman filters where the state transition is linear and noise Gaussian [287].

At the core of the estimation work, particle filters use sequential Monte Carlo (SMC) methods, of which sequential importance sampling (SIS) is the basis for most SMC filters developed. It is a technique for implementing a recursive Bayesian filter by Monte Carlo (MC) simulations and is based on importance sampling. The main idea is the recursive generation of random measures which are composed of a weighted set of particles drawn from relevant distributions that approximates the posterior distribution, $p\left(x_{1:n}|y_{1:n}\right)$, and particularly the filtering distribution, $p\left(x_n|y_{1:n}\right)$, of the unknown state conditioned on the observations [110, 111, 288]. Different types of estimates of the unknowns, like mean and variance is possible by the use of these measures.

A number of complex problems employ particle filters including target tracking, computer vision, robotics and channel estimation in digital communication or any application involving large, sequentially evolving data-sets [112–115, 289].

4.2 Mathematical Formulation

Particle filtering involves tracking states of dynamic state-space models, given some observation. More formally, given a hidden Markov process and a sequence of observations, $x_0, x_1, x_2, \ldots, x_n$ and $y_1, \ldots, y_n$, respectively, all the information about the process can be obtained from the posterior probability. The evaluation is performed recursively in time as successive observations y_n become available [110, 111].

The evolution of the STM (x_n) and the noisy measurements (y_n) through which it is estimated can be modeled by:

$$\begin{aligned} x_n &= f\left(x_{n-1}, z_{n-1}\right) \\ y_n &= g\left(x_n, v_n\right), \end{aligned} \tag{4.1}$$

where z_n and v_n are independent noise vectors with known distributions and $f(.)$ and $g(.)$ are known functions.

The estimation of the STM is also referred to as the tracking problem [286] and from a Bayesian perspective, the recursive estimation requires a construction of the posterior distribution $p\left(x_n|y_{1:n}\right)$. This can be done in two steps. First step is the prediction step which uses the posterior distribution at $n-1$ to obtain the prior at n. At $n=0$, the prior probability is $p\left(x_0\right)$. When at n, observation y_n becomes available, the prior is updated to produce the posterior based on Bayes' rule and is given by [111]:

$$p\left(x_n|y_{1:n}\right) = \frac{p\left(y_n|x_n\right) p\left(x_n|y_{1:n-1}\right)}{p\left(y_n|y_{1:n-1}\right)}, \tag{4.2}$$

where $p\left(y_n|x_n\right)$ is the likelihood function, $p\left(x_n|y_{1:n-1}\right)$ is the prior obtained in the prediction step and $p\left(y_n|y_{1:n-1}\right)$ is the normalizing constant. This posterior distribution is also referred to as marginal or filtering distribution [286]. The full posterior probability is given by

$$p\left(x_{1:n}|y_{1:n}\right) = p\left(x_{1:n-1}|y_{1:n-1}\right) \frac{p\left(x_n|x_{n-1}\right) p\left(y_n|x_n\right)}{p\left(y_n|y_{1:n-1}\right)}. \tag{4.3}$$

However, this recursive estimation of the posterior is only tractable analytically for a few restrictive cases. In other cases, this is numerically approximated and particle filter is one of the methods of this approximation.

For approximating either (4.3) or (4.2), particle filters updates a random measure $\{x_n^m, w_n^m\}_{m=1}^M$, which consists of M particles x_n^m and their normalized weights w_n^m defined at time n, recursively. The weights are typically normalized

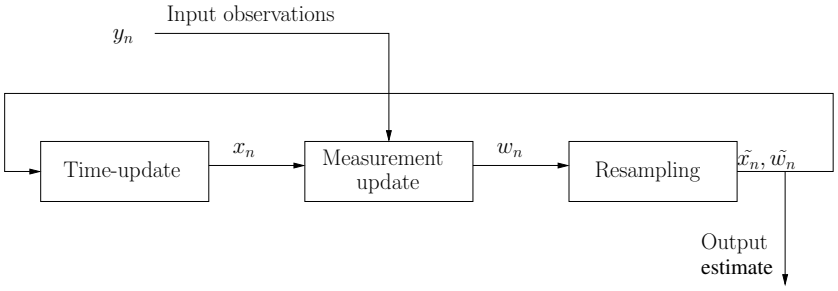

Figure 4.1: Overall structure of particle filter.

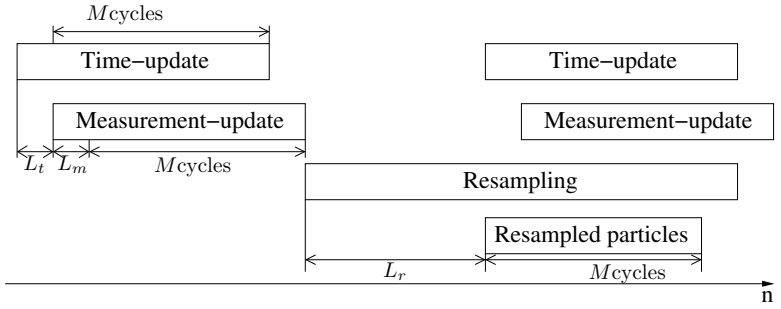

Figure 4.2: Basic scheduling of different steps in particle filter.

such that their sum is one by dividing the non-normalized weights ($\hat{w}_n^m$) with the sum of all the non-normalized weights, given as [111, 112, 290]:

$$w_n^m = \frac{\hat{w}_n^m}{\sum_{i=1}^{M} \hat{w}_i^m}. \tag{4.4}$$

4.3 Particle Filtering Steps

To achieve the required approximation to the posterior using particle filters, three steps are involved, namely:

- Time-update (particle generation/sampling)
- Measurement-update (weight computation)
- Resampling

The order in which each step is executed is shown in Fig. 4.1 and the basic scheduling of the three steps is shown in Fig. 4.2, where L_t, L_m and L_r are the latencies of the time-update, measurement-update and resampling steps.

4.3.1 Time-Update

The time-update step is also referred as particle generation or importance sampling step. Initially, particles x_n^m will be drawn from an importance density. Subsequently, after each iteration, the particles resampled in the resampling step is used to update the new set of particles for the next iteration [286].

4.3.2 Measurement-Update

In this step, weights associated with each particles are generated dependent on the input observations y_n as shown in Fig. 4.1. The weights generated are non-normalized which can be normalized as given in (4.4). The weight computation step involves the computation of trignometric and exponential functions and is the most computationally intensive of all the particle filtering steps [286].

4.3.3 Resampling in Particle Filters

A typical problem with SIS methodology is that with time, the variance of the estimates increases exponentially [286]. In other words, particles will negligible weights will dominate over particles with significant weight. This implies that a large computational effort is required in updating particles whose contribution to the estimation is negligible [110] and will lead to a poorer approximation on the posterior density and leading to inferior estimates. This is referred to as the degeneracy probelm and one of the ways to limit degeneracy of the algorithm is to select the importance density such that the variance is minimized [111]. Another method is the resampling techniques which are a important component of SMC methods and is part of the work presented in this dissertation.

The operation is called resampling because it involves sampling from an already sampled approximation, $\pi_n(x_n)$. In other words, some of the particles generated in the particle generation step are replicated which have large weights and those particles with negligible weights are discarded. This leads to better approximation and estimates and is achieved by modifying the weighted approximate density $\pi_n(x_n)$ to an un-weighted density $\hat{\pi}_n(x_n)$. More formally [291]:

$$\pi_n(x_n) = \sum_{m=1}^{M} w_n^m \delta(x - x_n^m) \tag{4.5}$$

is replaced by

$$\hat{\pi}_n(x_n) = \sum_{k=1}^{M} \frac{1}{M} \delta\left(x - x_n^k\right) = \sum_{m=1}^{M} \frac{c_n^m}{M} \delta(x - x_n^m), \tag{4.6}$$

where c_n^m is the number of copies of the original particle x_n^m in the new set of particles x_k^m. The weight associated with each particle is typically $1/M$. In resampling, c_n^m is proportional to the normalized particle weight, w_n^m, a constraint known as the unbiasedness condition [292]. This can be stated as:

$$E\left(c_n^m | w_n^m\right) = N w_n^m, \tag{4.7}$$

where N is the number of particles after resampling, which for traditional resampling algorithms is equal to M [292].

Although resampling removes degeneracy, it introduces other problems, outlined below [111]:

- Limits the opportunity to parallelize the implementation of resampling with other steps since it requires all particles and weights to be generated
- Particles having large weights are statistically selected many times reducing the diversity and may lead to sample impoverishment
- Because of sample impoverishment and loss of diversity, estimates based on these particles can degenerate

There are techniques to overcome these problems. For parallelism, techniques like overlapped partial resampling (OPR), which uses a threshold based algorithm [43] or independent Metropolis Hastings algorithm (IMHA) based resampling algorithm [44] have been proposed to introduce parallelism. Techniques like *resample-move* [293] and *regularization* [294] have been proposed to deal with the sample impoverishment problem.

A Resampling Algorithms

Different resampling algorithms have been proposed which can be classified based on different criteria. An exhaustive list of resampling algorithms and their classification is available in [292]. Among different types and categories listed in [292], work in this dissertation is concerned with the resampling algorithms categorized as *single distribution sampling methods*. In this category, all particles are resampled by using a single-distribution. Intuitively, particles are resampled by comparing their weights with some values generated from a random function generator. The difference in how the values are generated from the random function generator and how it is used for resampling differentiates between the resampling algorithms listed below:

- Multinomial [290]
- Stratified [112, 295]
- Systematic [111, 295]
- Residual [296]
- Residual systematic [116]
- Branch-kill/branching [297, 298]
- Rounding-copy [299]

Multinomial resampling is the basic approach which uses uniformly distributed random numbers to select x_k^m. These numbers are generated according to [291]:

$$u_r = \tilde{u}_r \text{ with } \tilde{u}_r \sim U\left[0, 1\right). \tag{4.8}$$

The complexity of multinomial resampling is of the order of $M \times N$ where N arises from the search of the index of the particle to be replicated. In other

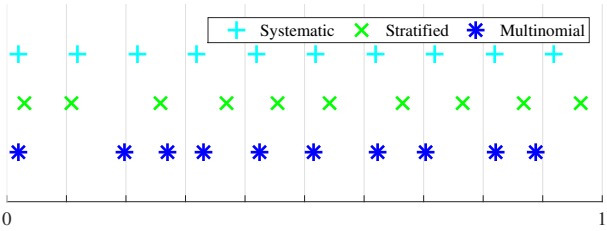

Figure 4.3: Standard uniformly distributed samples for $M = 10$.

words, the random numbers need to traversed multiple times to find all the replicated indices [292]. However, this high complexity can be reduced to at most of $2M$ by ordering these random numbers which will result in only one traversal of the random numbers.

Stratified divides the interval into uniform intervals, also termed as stratification, which helps the samples to be more uniformly distributed. Each value is then picked from each interval with a random offset. This can be mathematically stated as [291]:

$$u_r = \frac{1}{M}(k-1) + \tilde{u}_r, \text{ with } \tilde{u}_r \sim U\left[0, \frac{1}{M}\right) \tag{4.9}$$

Systematic resampling is an extended view of stratified resampling where the the offset is same resulting in the following formulation of generation of random numbers [291]:

$$u_r = \frac{1}{M}(k-1) + \tilde{u}, \text{ with } \tilde{u} \sim U\left[0, \frac{1}{M}\right). \tag{4.10}$$

The different random samples used by multinomial, stratified and systematic methods are shown in Fig. 4.3.

Typically in the resampling process, the weights need to be normalized. This requires M divisions per resampling process which has a high hardware cost. In [118], the authors proposed a modified systematic resampling scheme whereby M divisions are replaced by one division by drawing the uniform random number from a distribution spanning $\left[0, \frac{W_M}{M}\right)$, where W_M is the sum of weights $\left(\sum_{k=1}^{M} \hat{w}_n^k\right)$ and then updating it by $\frac{W_M}{M}$, as given in (4.11). However, there is still one division, making it only attractive when M is a power of two, in which case division is a binary shift.

$$u_r = \frac{W_M}{M}(k-1) + \tilde{u}, \text{ with } \tilde{u} \sim U\left[0, \frac{W_M}{M}\right). \tag{4.11}$$

Multinomial, stratified and systematic resampling algorithms can be represented in a unified way using a flow chart, as shown in Fig. 4.4, where a

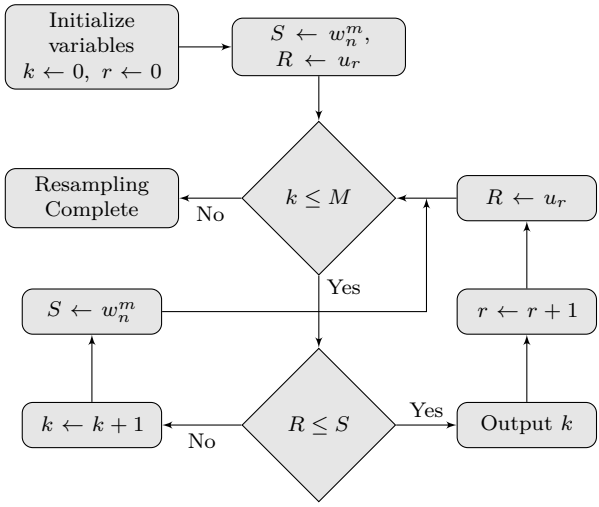

Figure 4.4: Flow chart of resampling.

normalized weight is read into variable S and a random number into R and comparison of them either fetches a new normalized weight or a new random number. In this figure, the random number generator can be based on any one of the three methods.

Residual resampling calculates the replication factor for each particle in two steps [291]. The number of replications of a particle is determined by the truncated product of M and w_k while the remaining partilces are sampled in the second step using one of the earlier described resampling algorithms. A modified version of this, called the residual systematic resampling (RSR), reduces the number of steps from two to one by including the systematic resampling principle to to update the random number [43, 116]. It uses the knowledge of the fixed intervals between two consecutive random function values generated in systematic resampling to calculate the replication factors which reduces the number of random numbers fetched. In this way, RSR reduces the number of iterations in the right side loop of the flow graph shown in Fig. 4.4 by reducing the number of times a value is fetched from the random number generator. A comparison of the four dominant resampling algorithms, i.e., multinomial, stratified, systematic and residual resampling is reported in [291].

Some more variations of the single-distribution sampling methods have been proposed [297–299]. These techniques, known as branch-kill [298] or branching [297] and rounding-copy [299], do not keep the particle size M constant at every time step and allows it to vary. Some threshold based algorithms have also been proposed in [43] which can be mapped to distributed architectures. Four algorithms are proposed and they differ in ways the threshold is defined. In all these schemes, a high threshold T_h and a low threshold T_l is defined. The

replicated particles can either be dominating and negligible particles or only dominating particles.

One of the threshold based methods proposed [43] is named as OPR, which defines a set of threshold for resampling. The particles are classified while their weights are computed allowing the overlap of this step with the measurement-update step while actual resampling is performed after the sum of weights is available, allowing some parallelism. However, it is not known how many cycles after the measurement-update step is needed to complete the resampling. But since the particles are classified into distinct groups, the execution of the resampling can be distributed. However, memory requirements will be high for this algorithm. A variation of the threshold based partial resampling algorithm which uses only one threshold was reported in [121]. The major difference between the proposed algorithms in [43] and [121] is that in [121], the weights of the particles after resampling is equal which is not in [43].

Resampling based on IMHA was proposed in [44] which instead of using normalized weights, works with ratio of importance weights which allows the resampling step to start parsing through the particles as they are generated and not wait for the first two steps to be completed. This allowed for a pipelined implementation which resulted in significant speed-up as compared to systematic resampling. However, no results was shown which can compare the tracking performance of the two resampling schemes.

B Architectural Concerns for Resampling

In terms of architecture, all resampling algorithms mentioned above use a weight memory, a random number generation unit, resampling unit and a control unit as is shown in Fig. 4.6 together with the time-update unit.

Based on the type of resampling algorithm, the random number generation unit and the resampling unit will be modified. Multinomial resampling requires a random number generator and a memory to hold it during the resampling step while the control unit will then generate address for this memory. Stratified and systematic only need a generator and an accumulator. For stratified, as given in (4.9), a new random number is produced every cycle while for systematic, as in (4.10), one random number is produced at the start of the resampling process.

For the residual resampling algorithm the second step will require a random number generator and memory for the weights while for RSR, a similar random number generator is required as used by systematic resampling. However, for these two algorithms, the resampling unit will be different to multinomial, stratified and systematic resampling algorithms.

They key bottleneck in implementing particle filters is the fact that the resampling step cannot be executed in parallel with other steps, the time-update and measurement-update, as mentioned earlier and shown in Fig. 4.2. In the worst case, single-distribution sampling methods take $2M-1$ cycles to complete, meaning that at least after the first M cycles, the first particle will be resampled. this allows a partial parallelism between the resampling step of the current

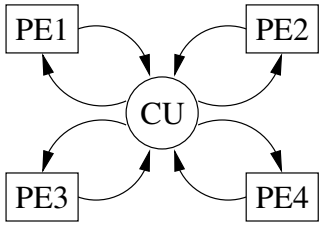

Figure 4.5: Particle filter architecture with parallel PEs [300].

iteration and the start of the next iteration, as shown in Fig. 4.2.

Several algorithm level techniques have been published to introduce or increase parallelism in the execution of the different steps, as mentioned in the preceding section. An implementation level technique is be to reduce the latency of the resampling step such that the parallelism shown in Fig. 4.2 can be increased. A technique to achieve this is presented in Paper D.

Different architecture level techniques have been proposed to improve the implementation of the particle filtering, specifically the resampling step. In a straight forward implementation, there is a requirement of two memories of depth M, one to store the particles after the time-update step and one for storing the resampled particles. One of the techniques proposed in [119] for systematic resampling uses one dual port memory for the two tasks. This means that for resampling, particles are read and written to the same memory and read and write pointers are used to guarantee the correctness of the implementation. However, this scheme requires two smaller memories for storing the indices of the resampled and discarded particles. Indices of the resampled particles is needed for reading the correct particle from the dual-port memory while discarded particles indices are required to write the replicated particles.

In [119], another proposed technique is to split the resampling into two parallel resampling. This is because for systematic resampling, the initial and final value generated by the random number generator is known which can be used to split the resampling. However, this technique requires duplicating the hardware resources required. For RSR, the memory scheme proposed stores the replicated particles along with the replication factors in such a way that the replicated particles are stored in the beginning of the memory while discarded particles, with replication factors of zero, are written at addresses after the replicated particles. However, a complex resampling unit is required for this technique which consists of two counters to control the part of the memory being currently accessed.

A distributed architecture for implementing particle filtering is proposed in [45, 300]. It uses the fact the the time-update and measurement-update states are deterministic and data independent. Thus they can be easily executed using multiple PEs, as shown Fig. 4.5 where the control unit (CU) synchronizes the operation between the PEs.

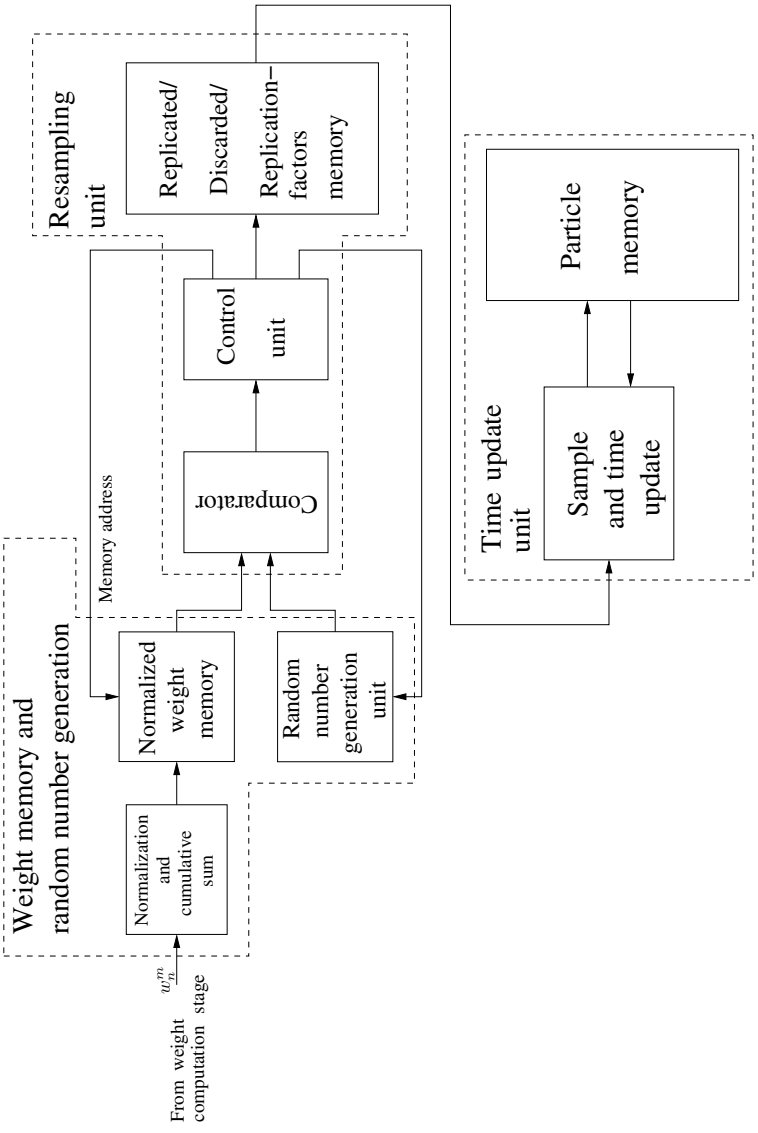

Figure 4.6: Basic architecture to implement the resampling step together with the time-update unit.

The distribution of the work load for the resampling step has been also been proposed in [300]. Two techniques proposed are named as distributed resampling with proportional allocation (RPA) and distributed resampling with nonproportional allocation (RNA). In RPA, the sample space is divided into disjoint set and resampling is carried out in two steps. First, the number of particles each set will replicate is calculated using RSR, referred to as inter-resampling. Second, resampling is carried out within each set, referred to as intra-resampling. However, the resampling is still non-deterministic because the resampling is dependent on the total weight of particles within each group and there is a need of exchange of particles between groups based on the number of particles allocated to each group. RNA, on the other hand, is a much more deterministic technique as the routing of particles is determined prior to the execution. In RNA, particle generation, weight computation, normalization and resampling is done in parallel for each group. However, performance results in terms of lost tracks and mean square error (MSE) for bearings only tracking problems is shown only for RNA and compared to systematic resampling. Although the results are comparable, systematic resampling achieves better results as compared to the proposed techniques. Architectures for implementing these techniques have also been proposed and speed-up and memory requirements for the proposed techniques analyzed.

Memory organization schemes and design of the CU in Fig. 4.5 to support the proposed techniques in [300] is reported in [120]. This proposed technique can support up to four PEs. An FPGA based implementation of auxiliary particle filters for neural signal processing in the implementation of brain machine interface (BMI) has been reported in [301]. Here the parallelism offered by FPGAs is utilized to have parallel particle processors which implement the first two steps of the particle filter. Resampling though is still sequential.

Chapter 5

Summary and Future Work

5.1 Summary

Efforts towards complexity reduction of FIR filters, one of the two focus areas of this thesis, has been going on for decades and various methods to reduce its implementation cost has been reported. In this thesis, the research front has been brought forward by proposing techniques to reduce the cost of implementation of FIR filters.

The techniques to reduce the complexity of implementing FIR filters is roughly divided into two categories; reduction in the multiplier complexity and in the number of multipliers, FRM being an example of the later. However, FRM has high order filters resulting in a significant amount of delay elements. An isomorphic mapping of it onto hardware platforms like ASICs and FPGAs will result in a large register usage. However, seldom is the case when the input sample rate is high enough to match the obtained clock frequency to warrant a fully parallel and isomorphic implementation. Instead, the implementation can be time-multiplexed to enable the system to run at a high clock frequency while supporting a low input sample rate allowing the reuse of hardware resources.

In this time-multiplexed implementation, the large number of delay elements required by the FRM technique will map to memories and in order to analyze this mapping, its effect on the number of multipliers, overall complexity, and power consumption, time-multiplexed architecture for FIR filters realized using the FRM technique is proposed in this technique. All basic types of FRM structures have been realized and analyzed with respect to different types of memory and memory access schemes, pipelining and reduction in the number of multipliers. It is shown that not only FIR filters realized using the FRM technique achieve lower multiplier usage but a better utilization of memory leads to a reduced LUTs usage. The power consumption, as a result, decreased

in the range of 23% and 68% when implemented on FPGAs, 80% of which is due to multiplier reduction.

The LNS can be used to reduce the multiplier complexity because of its inherent nature of transforming multiplications into additions. In this thesis, a filter design technique has been proposed using ILP in the LNS domain which are optimal in the minimax sense under finite word length constraints. The branch and bound algorithm has been implemented based on LNS integers and various node selection schemes have been proposed and analyzed. The effect of changing the word length in the LNS domain is analyzed and it is shown that four integer bits provide the best result, with three being applicable for larger approximation errors. Furthermore, filters realized in the LNS domain provided better approximation error as well as required a lower word length as compared to finite word length filters optimized in the linear domain.

The implementation cost of FIR filters can also be reduced by the use of computation sharing multipliers and an analysis and a unified design of such multipliers based on the Booth and high-radix multiplication schemes has been presented. Cost models for the implementation of different part of the multiplier are proposed while each multiplier is analyzed with respect to changing word length and multiplier radix. It is shown that if for a higher radix, the implementation cost per FIR filter tap is less than lower radix, the overall cost of the higher radix will eventually be lower with increasing filter length and word length for a given ratio between the cost of adders and multiplexers. The analysis has also been extended to the use of tri-state buffers to implement multiplexers and use of computation sharing in complex multipliers.

Particle filters and in particular the resampling step of particle filters is the second focus area of this thesis. Contributions have been made towards efficiently implementing existing resampling algorithms. A new technique, called the pre-fetch technique, has been proposed which significantly reduces the latency of the resampling step by up to 95%, dependent on the number of pre-fetches. Associated hardware architectures required to implement this technique has also been proposed. Furthermore, a division free architecture and a compact memory scheme has also been proposed in this thesis which helps in reducing the complexity of the multinomial resampling algorithm and reduce the number of memories required by up to 50%.

5.2 Future Work

The areas covered in this thesis can be extended in different aspects. Some suggestions are listed below:

- The time-multiplexed architecture proposed for the FRM technique assumes a single stage implementation of the sub-filters. This technique can be extended to cases where the sub-filters are themselves composed of FRM filters. These sub-filters will be shorter in length and hence it will be interesting to analyze the relationship between the length of the

subfilters, total complexity and memory usage.

- FIR filters optimized in the LNS domain have been shown to require lower word length as compared to fixed-point implementation. This can be extended to analyze whether this word length reduction translates to better hardware implementation as compared to fixed point implementation, specially considering that additions in LNS are expensive.
- Cost model for the encoder block in Booth and standard high-radix multiplier can be developed which will help increase the accuracy of the complexity numbers evaluated using these models. Furthermore, the granularity of the pipelining needs to be evaluated and its effect on speed, area and power consumption analyzed.
- The latency reduction achieved due to the pre-fetch technique is stochastic. From an architecture design and implementation perspective, there is a need for a closed form expression for the average latency reduction. It can be further extended to determine the distribution of the latency and formulate closed form expressions of other parameters.

References

[1] S. K. Mitra, *Digital Signal Processing.* University of California, Santa Barbara: TATA McGraw-Hill, 2006.

[2] L. Wanhammar and H. Johansson, *Digital Filters.* Department of Electrical Engineering: Linköping University, 2007.

[3] T. Saramäki, "Finite impulse response filter design," in *Handbook for Digital Signal Processing*, S. K. Mitra and J. F. Kaiser, Eds. Wiley-Interscience, 1993, pp. 155–277.

[4] T. N. Davidson, "Enriching the art of FIR filter design via convex optimization," *IEEE Signal Process. Mag.*, vol. 27, no. 3, pp. 89–101, 2010.

[5] R. G. Lyons, *Understanding Digital Signal Processing*, 3rd ed. Pearson Prentice Hallq, 2011.

[6] T. Saramäki and J. Yli-Kaakinen, "Design of digital filters and filter banks by optimization: Applications," in *Proc. Europ. Signal Process. Conf.*, 2000, pp. 1–2.

[7] Y. C. Lim and Y. Lian, "The optimum design of one- and two-dimensional FIR filters using the frequency response masking technique," *IEEE Trans. Circuits Syst. II*, vol. 40, no. 2, pp. 88–95, 1993.

[8] M. G. Bellanger, "Improved design of long FIR filters using the frequency masking technique," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 3, 1996, pp. 1272–1275.

[9] J. H. Webb and J. Munson, D. C., "Design of sparse FIR filters using linear programming," in *Proc. IEEE Int. Symp. Circuits Syst.*, Chicago, IL, May 1993, pp. 339–342.

[10] O. Gustafsson, L. S. DeBrunner, V. DeBrunner, and H. Johansson, "On the design of sparse half-band like FIR filters," in *Proc. Asilomar Conf. Signals Syst. Comput.*, Nov. 2007, pp. 1098–1102.

[11] Y.-S. Song and Y. H. Lee, "Design of sparse FIR filters based on branch-and-bound algorithm," in *Proc. IEEE Midwest Symp. Circuits Syst.*, vol. 2, Aug. 1997, pp. 1445–1448.

[12] O. Gustafsson, H. Johansson, and L. Wanhammar, "Design and efficient implementation of narrow-band single filter frequency masking FIR filters," in *Proc. Europ. Signal Process. Conf.*, vol. 1, 2000, pp. 4–8.

[13] ——, "Narrow-band and wide-band single filter frequency masking FIR filters," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, 2001, pp. 181–184.

[14] Y. Lian, "A new frequency-response masking structure with reduced complexity for FIR filter design," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, 2001, pp. 609–612.

[15] D. Mattera, F. Palmieri, and S. Haykin, "Efficient sparse FIR filter design," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 2, Orlando, FL, May 1993, pp. 1537–1540.

[16] L. Cen and Y. Lian, "Low-power implementation of frequency response masking based FIR filters," in *Proc. Joint Conf. Int. Conf. Inf. Comm. Signal Process. and Pacific Rim Conf. Multimedia*, vol. 3, 2003, pp. 1898–1902.

[17] O. Gustafsson, H. Johansson, and L. Wanhammar, "Single filter frequency masking high-speed recursive digital filters," *Circuits Syst. Signal Process.*, vol. 22, no. 2, pp. 219–238, 2003.

[18] ——, "Single filter frequency-response masking FIR filters," *J. Circuits Syst. Comput.*, vol. 12, no. 05, pp. 601–630, 2003.

[19] T. Saramäki and Y. C. Lim, "Use of the Remez algorithm for designing FRM based FIR filters," *Circuits Syst. Signal Process.*, vol. 22, no. 2, pp. 77–97, 2003.

[20] T. Saramäki, J. Yli-Kaakinen, and H. Johansson, "Optimization of frequency-response masking based FIR filters," *J. Circuits Syst. Comput.*, vol. 12, no. 05, pp. 563–591, 2003.

[21] Y. C. Lim, Y. J. Yu, K. L. Teo, and T. Saramäki, "FRM-based FIR filters with optimum finite word-length performance," *IEEE Trans. Signal Process.*, vol. 55, no. 6, pp. 2914–2924, Jun. 2007.

[22] S. G. Patronis and L. S. DeBrunner, "Sparse FIR filters and the impact on FPGA area usage," in *Proc. Asilomar Conf. Signals Syst. Comput.*, Pacific Grove, CA, Oct. 2008, pp. 1862–1866.

[23] Z. U. Sheikh and H. Johansson, "Wideband linear-phase FIR differentiators utilizing multirate and frequency-response masking techniques," in *Proc. IEEE Int. Symp. Circuits Syst.*, Taipei, May 2009, pp. 293–296.

[24] S. Li and J. Zhang, "Efficient FPGA implementation of sharp FIR filters using the FRM technique," *IEICE Electron. Express*, vol. 6, no. 23, pp. 1656–1662, Dec. 2009.

[25] W.-S. Lu and T. Hinamoto, "Digital filters with sparse coefficients," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2010, pp. 169–172.

[26] Z. U. Sheikh and O. Gustafsson, "Design of narrow-band and wide-band frequency-response masking filters using sparse non-periodic sub-filters," in *Proc. Europ. Signal Process. Conf.*, Aalborg, Denmark, Aug. 2010, pp. 1704–1707.

[27] Z. U. Sheikh, O. Gustafsson, and L. Wanhammar, "Design of sparse non-periodic narrow-band and wide-band FRM-like FIR filters," in *Proc. IEEE Int. Conf. Green Circuits Syst.*, Shanghai, China, Jun. 2010, pp. 279–282.

[28] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II*, vol. 42, no. 9, pp. 569–577, 1995.

[29] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II*, vol. 43, no. 10, pp. 677–688, 1996.

[30] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova, "A new algorithm for elimination of common subexpressions," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 1, pp. 58–68, 1999.

[31] O. Gustafsson and L. Wanhammar, "A novel approach to multiple constant multiplication using minimum spanning trees," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, vol. 3, 2002.

[32] O. Gustafsson and A. Dempster, "On the use of multiple constant multiplication in polyphase FIR filters and filter banks," in *Proc. IEEE Nordic Signal Process. Symp.*, Espoo, Finland, Jun. 2004, pp. 53–56.

[33] O. Gustafsson, H. Ohlsson, and L. Wanhammar, "Improved multiple constant multiplication using a minimum spanning tree," in *Proc. Asilomar Conf. Signals Syst. Comput.*, vol. 1, 2004, pp. 63–66.

[34] O. Gustafsson, A. G. Dempster, K. Johansson, M. D. Macleod, and L. Wanhammar, "Simplified design of constant coefficient multipliers," *Circuits Syst. Signal Process.*, vol. 25, no. 2, pp. 225–251, Apr. 2006.

[35] R. Guo, L. Wang, and L. S. DeBrunner, "A novel FIR filter implementation using truncated MCM technique," in *Proc. Asilomar Conf. Signals Syst. Comput.*, 2009, pp. 718–722.

[36] R. Guo, L. S. DeBrunner, and K. Johansson, "Truncated MCM using pattern modification for FIR filter implementation," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2010, pp. 3881–3884.

[37] P. K. Meher and Y. Pan, "MCM-based implementation of block FIR filters for high-speed and low-power applications," in *Proc. IEEE/IFIP Int. VLSI System-on-Chip Conf.*, 2011, pp. 118–121.

[38] F. Xu, C.-H. Chang, and C.-C. Jong, "Contention resolution—a new approach to versatile subexpressions sharing in multiple constant multiplications," *IEEE Trans. Circuits Syst. I*, vol. 55, no. 2, pp. 559–571, 2008.

[39] M. Faust and C.-H. Chang, "Minimal logic depth adder tree optimization for multiple constant multiplication," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2010, pp. 457–460.

[40] M. Kumm and P. Zipf, "High speed low complexity FPGA-based FIR filters using pipelined adder graphs," in *Proc. IEEE Int. Conf. Field Programmable Technology.*, 2011, pp. 1–4.

[41] M. Kumm, P. Zipf, M. Faust, and C.-H. Chang, "Pipelined adder graph optimization for high speed multiple constant multiplication," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2012, pp. 49–52.

[42] I. Hatai, I. Chakrabarti, and S. Banerjee, "An efficient constant multiplier architecture based on vertical-horizontal binary common sub-expression elimination algorithm for reconfigurable FIR filter synthesis," *IEEE Trans. Circuits Syst. I*, vol. 62, no. 4, pp. 1071–1080, 2015.

[43] M. Bolić, P. M. Djurić, and S. Hong, "Resampling algorithms for particle filters: A computational complexity perspective," *EURASIP J. Advances Signal Process.*, vol. 2004, no. 15, pp. 2267–2277, 2004.

[44] A. C. Sankaranarayanan, A. Srivastava, and R. Chellappa, "Algorithmic and architectural optimizations for computationally efficient particle filtering," *IEEE Trans. Image Process.*, vol. 17, no. 5, pp. 737–748, May 2008.

[45] M. Bolić, "Architectures for efficient implementation of particle filters," Ph.D. dissertation, The Graduate School of Electrical Engineering, Stony Brook University, 2004.

[46] Z. Jing and A. T. Fam, "A new structure for narrow transition band, lowpass digital filter design," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 32, no. 2, pp. 362–370, 1984.

[47] Y. Neuvo, D. Cheng-Yu, and S. Mitra, "Interpolated finite impulse response filters," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 32, no. 3, pp. 563–570, May 1984.

[48] Y. C. Lim, "Frequency-response masking approach for the synthesis of sharp linear phase digital filters," *IEEE Trans. Circuits Syst.*, vol. 33, no. 4, pp. 357–364, Apr. 1986.

[49] G. Rajan, Y. Neuvo, and S. K. Mitra, "On the design of sharp cutoff wide-band FIR filters with reduced arithmetic complexity," *IEEE Trans. Circuits Syst.*, vol. 35, no. 11, pp. 1447–1454, Nov. 1988.

[50] T. Saramäki, T. Neuvo, and S. K. Mitra, "Design of computationally efficient interpolated FIR filters," *IEEE Trans. Circuits Syst.*, vol. 35, no. 1, pp. 70–88, 1988.

[51] T. Saramäki and A. T. Fam, "Subfilter approach for designing efficient FIR filters," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1988, pp. 2903–2915.

[52] J. Kaiser and R. Hamming, "Sharpening the response of a symmetric nonrecursive filter by multiple use of the same filter," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 25, no. 5, pp. 415–422, 1977.

[53] S. Nakamura and S. Mitra, "Design of FIR digital filters using tapped cascaded FIR subfilters," *Circuits Syst. Signal Process.*, vol. 1, no. 1, pp. 43–56, 1982.

[54] T. Saramäki, "Design of FIR filters as a tapped cascaded interconnection of identical subfilters," *IEEE Trans. Circuits Syst.*, vol. 34, no. 9, pp. 1011–1029, Sep. 1987.

[55] ——, "A systematic technique for designing highly selective multiplier-free FIR filters," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1991, pp. 484–487.

[56] Y. Lian, "Complexity reduction for FRM-based FIR filters using the prefilter-equalizer technique," *Circuits Syst. Signal Process.*, vol. 22, no. 2, pp. 137–155, 2003.

[57] Y. C. Lim, Y. J. Yu, H. Q. Zheng, and S. W. Foo, "FPGA implementation of digital filters synthesized using the FRM technique," *Circuits Syst. Signal Process.*, vol. 22, no. 2, pp. 211–218, 2003.

[58] Y. Lian, "A modified frequency-response masking structure for high-speed FPGA implementation of sharp FIR filters," *J. Circuits Syst. Comput.*, vol. 12, no. 5, pp. 643–654, 2003.

[59] O. Gustafsson, H. Ohlsson, and L. Wanhammar, "Minimum-adder integer multipliers using carry-save adders," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, 2001, pp. 709–712.

[60] O. Gustafsson, H. Johansson, and L. Wanhammar, "MILP design of frequency-response masking FIR filters with few SPT terms," in *Proc. Int. Symp. Control Comm. Signal Process.*, 2004, pp. 405–408.

[61] O. Gustafsson, K. Johansson, H. Johansson, and L. Wanhammar, "Implementation of polyphase decomposed FIR filters for interpolation and decimation using multiple constant multiplication techniques," in *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, 2006, pp. 924–927.

[62] Y. Lim and S. Parker, "FIR filter design over a discrete powers-of-two coefficient space," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 31, no. 3, pp. 583–591, 1983.

[63] Y. C. Lim and S. Parker, "Discrete coefficient FIR digital filter design based upon an LMS criteria," *IEEE Trans. Circuits Syst.*, vol. 30, no. 10, pp. 723–739, 1983.

[64] N. Benvenuto, L. Franks, and F. S. Hill, Jr., "On the design of FIR filters with powers-of-two coefficients," *IEEE Trans. Commun.*, vol. 32, no. 12, pp. 1299–1307, 1984.

[65] Q. Zhao and Y. Tadokoro, "A simple design of FIR filters with powers-of-two coefficients," *IEEE Trans. Circuits Syst.*, vol. 35, no. 5, pp. 566–570, 1988.

[66] H. Samueli, "An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficients," *IEEE Trans. Circuits Syst.*, vol. 36, no. 7, pp. 1044–1047, Jul. 1989.

[67] Z. Jiang, "FIR filter design and implementation with powers-of-two coefficients," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 1989, pp. 1239–1242.

[68] A. Mahmood and J. R. Kunk, "Design of nearly optimum power-of-two coefficient cascaded filters," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, 1990, pp. 1083–1086.

[69] W. J. Oh and Y.-H. Lee, "Implementation of programmable multiplierless FIR filters with powers-of-two coefficients," *IEEE Trans. Circuits Syst. II*, vol. 42, no. 8, pp. 553–556, Aug. 1995.

[70] O. Gustafsson, H. Johansson, and L. Wanhammar, "An MILP approach for the design of linear-phase FIR filters with minimum number of signed-power-of-two terms," in *Proc. Europ. Conf. Circuit Theory Design*, 2001.

[71] O. Gustafsson and L. Wanhammar, "ILP modelling of the common subexpression sharing problem," in *Proc. IEEE Int. Conf. Electron. Circuits Syst.*, vol. 3, 2002, pp. 1171–1174.

[72] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," *IEE Proc. G Circuits Devices Syst.*, vol. 138, no. 3, pp. 401–412, 1991.

[73] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 2, pp. 151–165, 1996.

[74] A. Yurdakul and G. Dündar, "Multiplierless realization of linear DSP transforms by using common two-term expressions," *J. VLSI Signal Process. Syst. Signal Image Video Techn.*, vol. 22, no. 3, pp. 163–172, 1999.

[75] Y. C. Lim, S. Parker, and A. G. Constantinides, "Finite word length FIR filter design using integer programming over a discrete coefficient space," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 30, no. 4, pp. 661–664, 1982.

[76] Y. C. Lim, "Design of discrete-coefficient-value linear phase FIR filters with optimum normalized peak ripple magnitude," *IEEE Trans. Circuits Syst.*, vol. 37, no. 12, pp. 1480–1486, 1990.

[77] O. Gustafsson and L. Wanhammar, "Design of linear-phase FIR filters combining subexpression sharing with MILP," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, vol. 3, Aug. 2002, pp. III–9–III–12.

[78] Y. J. Yu and Y. C. Lim, "Design of linear phase FIR filters in subexpression space using mixed integer linear programming," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 10, pp. 2330–2338, 2007.

[79] H. Ohlsson, O. Gustafsson, and L. Wanhammar, "Implementation of low complexity FIR filters using a minimum spanning tree," in *Proc. IEEE Mediterranean Electrotech. Conf.*, vol. 1, 2004, pp. 261–264.

[80] W. Jenkins and B. Leon, "The use of residue number systems in the design of finite impulse response digital filters," *IEEE Trans. Circuits Syst.*, vol. 24, no. 4, pp. 191–201, 1977.

[81] R. Krishnan, G. A. Jullien, and W. C. Miller, "Computation of generalized FIR filter structure using the modified quadratic residue number system," *IEEE Trans. Circuits Syst. II*, vol. 39, no. 1, pp. 58–62, 1992.

[82] G. Loonawat and R. E. Siferd, "FPGA implementation of a FIR filter using residue arithmetic," in *Proc. IEEE National Aerospace Electron. Conf.*, vol. 1, 1996, pp. 286–290.

[83] G. Cardarilli, A. Nannarelli, and M. Re, "Reducing power dissipation in FIR filters using the residue number system," in *Proc. IEEE Midwest Symp. Circuits Syst.*, vol. 1, 2000, pp. 320–323.

[84] A. Mirshekari and M. Mosleh, "Hardware implementation of a fast FIR filter with residue number system," in *Proc. Int. Conf. Industrial Mechatronics Automation*, vol. 2, Wuhan, China, May 2010, pp. 312–315.

[85] N. Kingsbury and P. Rayner, "Digital filtering using logarithmic arithmetic," *Electron. Lett.*, vol. 7, no. 2, pp. 56–58, Jan. 1971.

[86] P. Lee, "An FPGA prototype for a multiplierless FIR filter built using the logarithmic number system," in *Proc. Int. Conf. Field-Programmable Logic Applicat.*, 1995, vol. 975, pp. 303–310.

[87] V. Paliouras and T. Stouraitis, "Logarithmic number system for low-power arithmetic," in *Proc. Int. Workshop Power Timing Modeling Optimization Simulation*, vol. 1918, 2000, pp. 285–294.

[88] ——, "Low-power properties of the logarithmic number system," in *Proc. IEEE Symp. Comput. Arithmetic*, Jun. 2001, pp. 229–236.

[89] A. Heřmánek, Z. Pohl, and J. Kadlec, "FPGA implementation of the adaptive lattice filter," in *Proc. Int. Conf. Field-Programmable Logic Applicat.*, ser. Lecture Notes in Computer Science, 2003, pp. 1095–1098.

[90] C. Basetas, I. Kouretas, and V. Paliouras, "Low-power digital filtering based on the logarithmic number system," in *Proc. Int. Workshop Power Timing Modeling Optimization Simulation*, 2007, vol. 4644, pp. 546–555.

[91] Y. Sun and M. S. Kim, "A high-performance 8-Tap FIR filter using logarithmic number system," in *Proc. IEEE Int. Conf. Comm.*, 2011, pp. 1–5.

[92] I. Kouretas, C. Basetas, and V. Paliouras, "Low-power logarithmic number system addition/subtraction and their impact on digital filters," *IEEE Trans. Comput.*, vol. 62, no. 11, pp. 2196–2209, 2013.

[93] E. Swartzlander Jr. and A. Alexopoulos, "The sign/logarithm number system," *IEEE Trans. Comput.*, vol. C-24, no. 12, pp. 1238–1242, Dec. 1975.

[94] J. H. Lang, C. A. Zukowski, R. O. Lamaire, and C. H. An, "Integrated-circuit logarithmic arithmetic units," *IEEE Trans. Comput.*, vol. C-34, no. 5, pp. 475–483, May 1985.

[95] O. Vainio and Y. Neuvo, "Logarithmic arithmetic in FIR filters," *IEEE Trans. Circuits Syst.*, vol. 33, no. 8, pp. 826–828, Jan. 2003.

[96] O. Vainio, "Biased logarithmic arithmetic in FIR filters," *Electron. Lett.*, vol. 41, no. 10, pp. 580–581, Jun. 2005.

[97] K. Johansson, O. Gustafsson, and L. Wanhammar, "Implementation of elementary functions for logarithmic number systems," *IET Comput. Digit. Tech.*, vol. 2, no. 4, pp. 295–304, Jun. 2008.

[98] A. D. Booth, "A signed binary multiplication technique," *Quart. Journ. Mech. and Applied Math*, vol. 4, no. 2, pp. 236–240, 1951.

[99] O. L. MacSorley, "High-speed arithmetic in binary computers," *Proc. IRE*, vol. 49, no. 1, pp. 67–91, Jan. 1961.

[100] H. Sam and A. Gupta, "A generalized multibit recoding of two's complement binary numbers and its proof with application in multiplier implementations," *IEEE Trans. Comput.*, vol. 39, no. 8, pp. 1006–1015, 1990.

[101] D. Crookes and R. M. Jiang, "A low-power high-radix serial-parallel multiplier," in *Proc. Europ. Conf. Circuit Theory Design*, 2007, pp. 460–463.

[102] D. Guevorkian, A. Launiainen, V. Lappalainen, P. Liuha, and K. Punkka, "A method for designing high-radix multiplier-based processing units for multimedia applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 5, pp. 716–725, 2005. [Online]. Available: Designofdigitalfiltersandfilterbanksbyoptimization:Applications

[103] H. Choo, K. Muhammad, and K. Roy, "Decision feedback equalizer with two's complement computation sharing multiplication," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 2, 2001, pp. 1245–1248.

[104] ——, "Two's complement computation sharing multiplier and its applications to high performance DFE," *IEEE Trans. Signal Process.*, vol. 51, no. 2, pp. 458–469, 2003.

[105] I. Jongsun Park, H. Choo, K. Muhammad, S. Choi, Y. Im, and K. Roy, "Non-adaptive and adaptive filter implementation based on sharing multiplication," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 1, 2000, pp. 460–463.

[106] I. Jongsun Park, W. Jeong, H. Choo, H. Mahmoodi-Meimand, Y. Wang, and K. Roy, "High performance and low power FIR filter design based on sharing multiplication," in *Proc. Int. Symp. Low Power Electron. Design.*, 2002, pp. 295–300.

[107] I. Jongsun Park, K. Muhammad, and K. Roy, "High-performance FIR filter design based on sharing multiplication," *IEEE Trans. VLSI Syst.*, vol. 11, no. 2, pp. 244–253, 2003.

[108] J. Park, W. Jong, H. Mahmoodi-Meimand, Y. Wang, H. Choo, and K. Roy, "Computation sharing programmable FIR filter for low-power and high-performance applications," *IEEE J. Solid-State Circuits*, vol. 39, no. 2, pp. 348–357, Feb. 2004.

[109] S.-W. Hsu and Y.-H. Huang, "Sd-based computation sharing programmable FIR filter for software radio," in *Proc. Int. Conf. Comm. Circuits Syst.* IEEE, 2009, pp. 435–438.

[110] A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Stat. Comput.*, vol. 10, no. 3, pp. 197–208, 2000.

[111] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 174–188, 2002.

[112] A. Doucet, N. de Freitas, and N. Gordon, Eds., *Sequential Monte Carlo methods in practice*. Springer Verlag, New York, 2001.

[113] B. Ristic, M. S. Arulampalam, and N. Gordon, *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, Norwood, 2004.

[114] A. Doucet and X. Wang, "Monte Carlo methods for signal processing: a review in the statistical signal processing context," *IEEE Signal Process. Mag.*, vol. 22, no. 6, pp. 152–170, 2005.

[115] Z.-G. Shi, S.-H. Hong, J.-M. Chen, K.-S. Chen, and Y.-X. Sun, "Particle filter-based synchronization of chaotic colpitts circuits combating AWGN channel distortion," *Circuits Syst. Signal Process.*, vol. 27, no. 6, pp. 833–845, 2008.

[116] M. Bolić, P. M. Djurić, and S. Hong, "New resampling algorithms for particle filters," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 2, 2003.

[117] A. Athalye, M. Bolić, S. Hong, and P. M. Djurić, "Architectures and memory schemes for sampling and resampling in particle filters," in *Proc. IEEE Digital Signal Process. Workshop and Proc. IEEE Signal Process. Edu. Workshop.*, 2004, pp. 92–96.

[118] M. Bolić, A. Athalye, P. M. Djurić, and S. Hong, "Algorithmic modification of particle filters for hardware implementation," in *Proc. Europ. Signal Process. Conf.*, 2004, pp. 1641–1644.

[119] A. Athalye, M. Bolić, S. Hong, and P. M. Djurić, "Generic hardware architectures for sampling and resampling in particle filters," *EURASIP J. Appl. Signal Process.*, vol. 2005, no. 17, pp. 2888–2902, 2005.

[120] S. Hong, S.-S. Chin, P. Djurić, and M. Bolic, "Design and implementation of flexible resampling mechanism for high-speed parallel particle filters," *J. VLSI Signal Process. Syst.*, vol. 44, no. 1–2, pp. 47–62, 2006.

[121] S.-H. Hong, Z.-G. Shi, J.-M. Chen, and K.-S. Chen, "A low-power memory-efficient resampling architecture for particle filters," *Circuits Syst. Signal Process.*, vol. 28, no. 1, pp. 155–167, 2010.

[122] L. Wanhammar, *DSP Integrated Circuits*. Academic Press, 1999.

[123] R. Zimmermann, "Binary adder architectures for cell-based VLSI and ttheir synthesis," Ph.D. dissertation, Swiss Federal Institute of Technology (ETH), Hartung-Gorre Verlag, Zurich, 1998.

[124] M. J. S. Smith, *Application-specific integrated circuits*. Addison-Wesley, 1997.

[125] B. Zahiri, "Structured ASICs: opportunities and challenges," in *Proc. IEEE Int. Conf. Comput. Design.*, 2003, pp. 404–409.

[126] K. Gulati and S. P. Khatri, *Hardware Acceleration of EDA Algorithms*. Springer, 2010.

[127] D. Liu, *Embedded DSP Processor Design: Application Specific Instruction Set Processors*. Morgan Kaufmann Publishers, 2008.

[128] W. Wolf, *FPGA-Based System Design*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[129] I. Kuon, R. Tessier, and J. Rose, *FPGA Architecture: Survey and Challenges, Foundations and Trends® in Electronic Design Automation*. now Publishers Inc., 2007, vol. 2, no. 2.

[130] J. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: the effect of logic block functionality on area efficiency," *IEEE J. Solid-State Circuits*, vol. 25, no. 5, pp. 1217–1225, 1990.

[131] D. Chinnery and K. Keutzer, *Closing the gap between ASIC and Custom tools and techniques for High-Performance ASIC Design*. Kluwer Academic Publishers, 2002.

[132] R. Minnick, "A survey of microcellular research," *J. ACM*, vol. 14, pp. 203–241, Apr. 1967.

[133] J. M. Birkner and H. T. Chua, "Programmable array logic circuit," United States of America Patent 4 124 899, 1978.

[134] S. E. Wahlstrom, "Programmable logic arrays – cheaper by the millions," *Electronics*, vol. 40, pp. 90–95, Dec. 1967.

[135] W. Carter, K. Duong, R. H. Freeman, H. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo, and S. L. Sze, "A user programmable reconfigurable gate array," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1986.

[136] R. H. Freeman, "Configurable electrical circuit having configurable logic elements and configurable interconnect," United States of America Patent 4 870 302, 1989.

[137] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays," *Proc. IEEE*, vol. 81, no. 7, pp. 1013–1029, 1993.

[138] H. Hsieh, K. Duong, J. Ja, R. Kanazawa, L. Ngo, L. Tinkey, W. Carter, and R. Freeman, "A second generation user programmable gate array," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1987, pp. 515–521.

[139] A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. A. El-Ayat, and A. Mohsen, "An architecture for electrically configurable gate arrays," *IEEE J. Solid-State Circuits*, vol. 24, no. 2, pp. 394–398, 1989.

[140] C. Maxfield, *The Design Warrior's Guide to FPGAs.* Elsevier, 2004.

[141] D. Frohman-Bentchkowsky, "A fully-decoded 2048-bit electrically-programmable MOS ROM," in *Proc. IEEE Int. Solid-State Circuit Conf.*, 1971, pp. 80–81.

[142] R. Cuppens, C. D. Hartgring, J. F. Verwey, H. L. Peek, F. A. H. Vollebragt, E. G. M. Devens, and I. A. Sens, "An EEPROM for microprocessors and custom logic," *IEEE J. Solid-State Circuits*, vol. 20, no. 2, pp. 603–608, 1985.

[143] A. Scheibe and W. Krauss, "A two-transistor SIMOS EAROM cell," *IEEE J. Solid-State Circuits*, vol. 15, no. 3, pp. 353–357, 1980.

[144] D. C. Guterman, I. H. Rimawi, T.-L. Chiu, R. D. Halvorson, and D. J. McElroy, "An electrically alterable nonvolatile memory cell using a floating-gate structure," *IEEE Trans. Electron Devices*, vol. 26, no. 4, pp. 576–586, 1979.

[145] J. a. Birkner, A. Chan, H. Chua, A. Chao, K. Gordon, B. Kleinman, P. Kolze, and R. Wong, "A very-high-speed field-programmable gate array using metal-to-metal antifuse programmable elements," *Microelectron. J.*, vol. 23, no. 7, pp. 561–568, 1992.

[146] "Flash FPGAs in the value-based market," Actel, Tech. Rep., Jan. 2005. [Online]. Available: http://www.microsemi.com/

[147] T. Morin, "Flash FPGAs give designers more flexibility," Microsemi Corp., Tech. Rep., Jan. 2015. [Online]. Available: http://www.embedded.com/electronics-blogs/

[148] K. Neil, "Antifuse FPGA technology: Best option for satellite applications," Actel, Tech. Rep., Dec. 2003. [Online]. Available: http://www.cotsjournalonline.com/articles/view/100087

[149] D. Marple and L. Cooke, "An MPGA compatible FPGA architecture," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1992.

[150] *ERA60100 preliminary data sheet*, Plessey Semiconductor, 1989.

[151] S. C. Wong, H. C. So, J. H. Ou, and J. Costello, "A 5000-gate CMOS epld with multiple logic and interconnect arrays," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1989.

[152] M. Ahrens, A. El Gamal, D. Galbraith, J. Greene, S. Kaptanoglu, K. Dharmarajan, L. Hutchings, S. Ku, P. McGibney, J. McGowan, A. Sanie, K. Shaw, N. Stiawalt, T. Whitney, T. Wong, W. Wong, and B. Wu, "An FPGA family optimized for high densities and reduced routing delay," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1990.

[153] S. S. Yau and C. K. Tang, "Universal logic modules and their applications," *IEEE Trans. Comput.*, vol. C-19, no. 2, pp. 141–149, 1970.

[154] "XC3000 series field programmable gate arrays," Nov. 1998. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/3000.pdf

[155] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," *IEEE Trans. VLSI Syst.*, vol. 12, no. 3, pp. 288–298, Mar. 2004.

[156] J. He and J. Rose, "Advantages of heterogeneous logic block architecture for FPGAs," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1993.

[157] J. Cong and S. Xu, "Delay-optimal technology mapping for FPGAs with heterogeneous LUTs," in *Proc. Design Automation Conf.*, 1998, pp. 704–707.

[158] A. Kaviani and S. Brown, "The hybrid field-programmable architecture," *IEEE Des. Test. Comput.*, vol. 16, no. 2, pp. 74–83, 1999.

[159] "7 Series FPGAs configurable logic block – user guide," Xilinx, Tech. Rep., Nov. 2014. [Online]. Available: http://www.xilinx.com/support/documentation

[160] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, and J. Rose, "The Stratix II logic and routing architecture," in *Proc. ACM/SIGDA Int. Symp. Field-program. Gate Arrays*, ser. FPGA '05. New York, NY, USA: ACM, 2005, pp. 14–20.

[161] M. Kumm, K. Moller, and P. Zipf, "Dynamically reconfigurable FIR filter architectures with fast reconfiguration," in *Proc. Int. Workshop Reconfig. Communication-Centric Systems-on-Chip.*, 2013, pp. 1–8.

[162] J. B. Evans, "Efficient FIR filter architectures suitable for FPGA implementation," *IEEE Trans. Circuits Syst. II*, vol. 41, no. 7, pp. 490–493, 1994.

[163] S. Mohanakrishnan and J. B. Evans, "Automatic implementation of FIR filters on field programmable gate arrays," *IEEE Signal Process. Lett.*, vol. 2, no. 3, pp. 51–53, 1995.

[164] C. H. Dick and F. Harris, "Implementing narrow-band FIR filters using FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, 1996, pp. 289–292.

[165] Y.-Y. Tzou and H.-J. Hsu, "FPGA realization of space-vector PWM control IC for three-phase PWM inverters," *IEEE Trans. Power Electron.*, vol. 12, no. 6, pp. 953–963, 1997.

[166] R. Baines and D. Pulley, "A total cost approach to evaluating different reconfigurable architectures for baseband processing in wireless receivers," *IEEE Commun. Mag.*, vol. 41, no. 1, pp. 105–113, 2003.

[167] S.-S. Jeng, H.-C. Lin, and S.-M. Chang, "FPGA implementation of FIR filter using m-bit parallel distributed arithmetic," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2006.

[168] A. Benkrid and K. Benkrid, "Novel area-efficient FPGA architectures for FIR filtering with symmetric signal extension," *IEEE Trans. VLSI Syst.*, vol. 17, no. 5, pp. 709–722, 2009.

[169] S. Y. Park and P. K. Meher, "Efficient FPGA and ASIC realizations of a DA-based reconfigurable FIR digital filter," *IEEE Trans. Circuits Syst. II*, vol. 61, no. 7, pp. 511–515, 2014.

[170] M. Kumm, M. Hardieck, J. Willkomm, P. Zipf, and U. Meyer-Baese, "Multiple constant multiplication with ternary adders," in *Proc. Int. Conf. Field-Programmable Logic Applicat.*, 2013, pp. 1–8.

[171] "7 Series DSP48E1 slice – user guide," Xilinx, Tech. Rep., Nov. 2014.

[172] *Stratix V device handbook*, Altera, Jun. 2015.

[173] M. Kumm, S. Abbas, and P. Zipf, "An efficient softcore multiplier architecture for Xilinx FPGAs," in *Proc. IEEE Symp. Comput. Arithmetic*, 2015, pp. 18–25.

[174] F. de Dinechin and B. Pasca, "Large multipliers with fewer DSP blocks," in *Proc. Int. Conf. Field-Programmable Logic Applicat.*, 2009, pp. 250–255.

[175] S. Banescu, F. de Dinechin, B. Pasca, and R. Tudoran, "Multipliers for floating-point double precision and beyond on FPGAs," *SIGARCH Comput. Archit. News*, vol. 38, no. 4, pp. 73–79, Jan. 2011.

[176] S. Kumar, K. Forward, and M. Palaniswami, "A fast-multiplier generator for FPGAs," in *Proc. Int. Conf. VLSI Design.*, 1995, pp. 53–56.

[177] H. Parandeh-Afshar and P. Ienne, "Measuring and reducing the performance gap between embedded and soft multipliers on FPGAs," in *Proc. Int. Conf. Field-Programmable Logic Applicat.*, 2011, pp. 225–231.

[178] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. Comput.*, vol. C-22, no. 12, pp. 1045–1047, 1973.

[179] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Efficient synthesis of compressor trees on FPGAs," in *Proc. Asia and South Pacific Design Automation Conf.*, 2008, pp. 138–143.

[180] ——, "Improving synthesis of compressor trees on FPGAs via integer linear programming," in *Proc. Design Automation Test Europe*, 2008, pp. 1256–1261.

[181] H. Parandeh-Afshar, A. Neogy, P. Brisk, and P. Ienne, "Compressor tree synthesis on commercial high-performance FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 4, pp. 39:1–39:19, Dec. 2011.

[182] T. Matsunaga, S. Kimura, and Y. Matsunaga, "Power and delay aware synthesis of multi-operand adders targeting LUT-based FPGAs," in *Proc. Int. Symp. Low Power Electron. Design.*, 2011, pp. 217–222.

[183] ——, "A exact approach for GPC-based compressor tree synthesis," *IEICE Trans. Fundamentals Electron. Comm. Comp. Sci.*., vol. E96-A, no. 12, pp. 2553–2560, Dec. 2013.

[184] M. Kumm and P. Zipf, "Pipelined compressor tree optimization using integer linear programming," in *Proc. Int. Conf. Field-Programmable Logic Applicat.*, 2014, pp. 1–8.

[185] S. Zohar, "New hardware realizations of nonrecursive digital filters," *IEEE Trans. Comput.*, vol. C-22, no. 4, pp. 328–338, 1973.

[186] A. Peled and B. Liu, "A new hardware realization of digital filters," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 22, no. 6, pp. 456–462, 1974.

[187] W. Sen, T. Bin, and Z. Jun, "Distributed arithmetic for FIR filter design on FPGA," in *Proc. Int. Conf. Comm. Circuits Syst.*, 2007, pp. 620–623.

[188] M. Kumm, K. Moller, and P. Zipf, "Partial LUT size analysis in distributed arithmetic FIR filters on FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2013, pp. 2054–2057.

[189] ——, "Reconfigurable FIR filter using distributed arithmetic on FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2013, pp. 2058–2061.

[190] A. Saha and A. Sinha, "An FPGA based architecture of a novel reconfigurable radio processor for software defined radio," in *Proc. Int. Conf. Education Technology Comput.*, 2009, pp. 45–49.

[191] T. Todman, A. G. Constantinides, S. J. E. Wilton, W. Mencer, O. Luk, and P. Y. K. Cheung, "Reconfigurable computing: architectures and design methods," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 2, pp. 193–207, 2005.

[192] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable computing architectures," *Proc. IEEE*, vol. 103, no. 3, pp. 332–354, 2015.

[193] M. Faust, O. Gustafsson, and C.-H. Chang, "Reconfigurable multiple constant multiplication using minimum adder depth," in *Proc. Asilomar Conf. Signals Syst. Comput.*, 2010, pp. 1297–1301.

[194] P. Lowenborg and H. Johansson, "Minimax design of adjustable-bandwidth linear-phase FIR filters," *IEEE Trans. Circuits Syst. I*, vol. 53, no. 2, pp. 431–439, 2006.

[195] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, pp. 14–17, 1964.

[196] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, no. 5, pp. 349–356, 1965.

[197] K. C. Bickerstaff, M. J. Schulte, and E. E. Swartzlander Jr, "Parallel reduced area multipliers," *J. VLSI Signal Process. Syst.*, vol. 9, no. 3, pp. 181–191, 1995.

[198] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *J. ACM*, vol. 27, no. 4, pp. 831–838, 1980.

[199] R. P. Brent and H. Kung, "A regular layout for parallel adders," *IEEE Trans. Comput.*, vol. C-31, no. 3, pp. 260–264, 1982.

[200] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Comput.*, vol. C-22, no. 8, pp. 786–793, 1973.

[201] O. Gustafsson and L. Wanhammar, "Arithmetic," in *Handbook of signal processing systems*, S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, Eds. Springer Science, 2013, pp. 593–637.

[202] T. G. Noll, "Carry-save architectures for high-speed digital signal processing," *J. VLSI Signal Process. Syst. Signal Image Video Techn.*, vol. 3, no. 1-2, pp. 121–140, 1991.

[203] M. Mehta, V. Parmar, and E. Swartzlander Jr, "High-speed multiplier design using multi-input counter and compressor circuits," in *Proc. IEEE Symp. Comput. Arithmetic*. IEEE, 1991, pp. 43–50.

[204] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Trans. Electron. Comput.*, vol. EC-10, no. 3, pp. 389–400, Sep. 1961.

[205] H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara, and K. Mashiko, "An 8.8-ns 54×54-bit multiplier with high speed redundant binary architecture," *IEEE J. Solid-State Circuits*, vol. 31, no. 6, pp. 773–783, 1996.

[206] T. N. Rajashekhara and O. Kal, "Fast multiplier design using redundant signed-digit numbers," *Int. J. Electron.*, vol. 69, no. 3, pp. 359–368, 1990.

[207] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi, and N. Takagi, "Design of high speed MOS multiplier and divider using redundant binary representation," in *Proc. IEEE Symp. Comput. Arithmetic*, 1987, pp. 80–86.

[208] N. Ohkubo, M. Suzuki, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, and Y. Nakagome, "A 4.4-ns CMOS 54×54-b multiplier using pass-transistor multiplexer," *IEEE J. Solid-State Circuits*, vol. 30, no. 3, pp. 251–257, 1995.

[209] N. Besli and R. G. Deshmukh, "A novel redundant binary signed-digit (RBSD) Booth's encoding," in *Proc. IEEE SoutheastCon.* Institute of Electrical & Electronics Engineers (IEEE), 2002, pp. 426–431.

[210] D. Villeger and V. G. Oklobdzija, "Evaluation of Booth encoding techniques for parallel multiplier implementation," *Electron. Lett.*, vol. 29, no. 23, pp. 2016–2017, 1993.

[211] D. R. Noaks and D. P. Burton, "A high-speed, asynchronous, digital multiplier," *Radio and Electronic Engineer*, vol. 36, no. 6, pp. 357–365, 1975.

[212] L. P. Rubinfield, "A proof of the modified Booth's algorithm for multiplication," *IEEE Trans. Comput.*, vol. 24, no. 10, pp. 1014–1015, 1975.

[213] R. Mahesh and A. P. Vinod, "A new common subexpression elimination algorithm for realizing low-complexity higher order digital filters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 2, pp. 217–229, 2008.

[214] A. R. Cooper, "Parallel architecture modified Booth multiplier," *IEE Proceedings G Electronic Circuits and Systems*, vol. 135, no. 3, pp. 125–128, 1988.

[215] Z.-J. Mou and F. Jutand, "'overturned-stairs' adder trees and multiplier design," *IEEE Trans. Comput.*, vol. 41, no. 8, pp. 940–948, 1992.

[216] H. Eriksson, P. Larsson-Edefors, M. Sheeran, M. Själander, D. Johansson, and M. Scholin, "Multiplier reduction tree with logarithmic logic depth and regular connectivity," in *Proc. IEEE Int. Symp. Circuits Syst.* IEEE, 2006, pp. 4–pp.

[217] P. F. Stelling and V. G. Oklobdzija, "Design strategies for optimal hybrid final adders in a parallel multiplier," *J. VLSI Signal Process. Syst.*, vol. 14, no. 3, pp. 321–331, 1996.

[218] M. Macleod and A. Dempster, "Multiplierless FIR filter design algorithms," *IEEE Signal Process. Lett.*, vol. 12, no. 3, pp. 186–189, 2005.

[219] I.-C. Park and H.-J. Kang, "Digital filter synthesis based on an algorithm to generate all minimal signed digit representations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 12, pp. 1525–1529, 2002.

[220] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algorithms*, vol. 3, no. 2, May 2007.

[221] A. G. Dempster and M. D. Macleod, "Constant integer multiplication using minimum adders," *IEE Proc. Circuits Devices Syst.*, vol. 141, no. 5, pp. 407–413, 1994.

[222] M. J. Wirthlin, "Constant coefficient multiplication using look-up tables," *J. VLSI Signal Process. Syst. Signal Image Video Techn.*, vol. 36, no. 1, pp. 7–15, 2004.

[223] U. Meyer-Baese, J. Chen, C. H. Chang, and A. Dempster, "A comparison of pipelined RAG-n and DA FPGA-based multiplierless filters," in *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, 2006, pp. 1555–1558.

[224] M. Faust and C.-H. Chang, "Bit-parallel multiple constant multiplication using look-up tables on FPGA," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2011, pp. 657–660.

[225] M. Kumm and P. Zipf, "Hybrid multiple constant multiplication for FPGAs," in *Proc. IEEE Int. Conf. Electron. Circuits Syst.*, 2012, pp. 556–559.

[226] S. Mirzaei, R. Kastner, and A. Hosangadi, "Layout aware optimization of high speed fixed coefficient fir filters for fpgas," *Int. J. Reconfigurable Comp.*, vol. 2010, p. 1, 2010.

[227] K. N. Macpherson and R. W. Stewart, "Rapid prototyping - area efficient FIR filters for high speed FPGA implementation," *IEE Proc. Vision Image Signal Process.*, vol. 153, no. 6, pp. 711–720, 2006.

[228] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Optimization of gate-level area in high throughput multiple constant multiplications," in *Proc. Europ. Conf. Circuit Theory Design*, Aug. 2011, pp. 588–591.

[229] P. E. Landman and J. M. Rabaey, "Architectural power analysis: The dual bit type method," *IEEE Trans. VLSI Syst.*, vol. 3, no. 2, pp. 173–187, Jun. 1995.

[230] T. Stouraitis and V. Paliouras, "Considering the alternatives in low-power design," *IEEE Circuits Syst. Mag.*, vol. 17, no. 4, pp. 22–29, Aug. 2002.

[231] K.-H. Chen and C. T.-D., "A low-power digit-based reconfigurable FIR filter," *IEEE Trans. Circuits Syst. II*, vol. 53, no. 8, pp. 617–621, Aug. 2006.

[232] V. Paliouras, J. Karagiannis, C. Aggouras, and T. Stouraitis, "A very-long instruction word digital signal processor based on the logarithmic number system," in *Proc. IEEE Int. Conf. Electron. Circuits Syst.*, vol. 3, 1998, pp. 59–62.

[233] C. Litchfield, R. J. Langley, P. Lee, and J. Batchelor, "The use of hybrid logarithmic arithmetic for root raised cosine matched filters in WCDMA downlink receivers," in *Proc. IEEE Wireless Comm. Networking Conf.*, vol. 1, 2005, pp. 596–600.

[234] I. Kouretas, C. Basetas, and V. Paliouras, "Low-power logarithmic number system addition/subtraction and their impact on digital filters," in *Proc. IEEE Int. Symp. Circuits Syst.*, Washington, USA, Jun. 2008, pp. 692–695.

[235] M. Azarmehr, M. Ahmadi, and G. A. Jullien, "A two-dimensional logarithmic number system (2DLNS)-based finite impulse response (FIR) filter design," in *Proc. IEEE Northeast Workshop Circuits Syst.*, 2011, pp. 37–40.

[236] G. Cardarilli, A. Nannarelli, and M. Re, "Residue number system for low-power DSP applications," in *Proc. Asilomar Conf. Signals Syst. Comput.*, 2007, pp. 1412–1416.

[237] N. S. Szabo and R. I. Tanaka, *Residue arithmetic and its applications to computer technology.* McGraw-Hill, 1967.

[238] A. Omondi and B. Premkumar, *Residue number systems: theory and implementation.* Imperial College Press, 2007.

[239] G. Cardarilli, M. Re, and R. Lojacono, "A residue to binary conversion algorithm for signed numbers," in *Proc. Europ. Conf. Circuit Theory Design*, vol. 3, 1997, pp. 1456–1459.

[240] G. Cardarilli, M. Re, R. Lojacono, and G. Ferri, "A new efficient architecture for binary to RNS conversion," in *Proc. Europ. Conf. Circuit Theory Design*, vol. 2, 1999, pp. 1151–1154.

[241] A. Preethy and D. Radhakrishnan, "A VLSI architecture for analog-to-residue conversion," in *Proc. Int. Conf. on Advanced A/D and D/A Conversion Techniques and Their Applications*, 1999, pp. 83–85.

[242] K. Kaluri, W. F. Leong, K.-H. Tan, L. Johnson, and M. Soderstrand, "FPGA hardware implementation of an RNS FIR digital filter," in *Proc. Asilomar Conf. Signals Syst. Comput.*, vol. 2, 2001, pp. 1340–1344.

[243] Y. Kong, A. Safari, and C. V. Niras, "A low-cost architecture for DWT filter banks in RNS applications," in *Proc. Int. Symp. Integrated Circuits.*, 2014, pp. 448–451.

[244] R. K. Cavin, C. H. Ray, and V. T. Rhyne, "The design of optimal convolutional filters via linear programming," *IEEE Trans. Geosci. Electron.*, vol. 7, no. 3, pp. 142–145, 1969.

[245] O. Herrmann, "Design of nonrecursive digital filters with linear phase," *Electron. Lett.*, vol. 6, no. 11, pp. 328–329, 1970.

[246] L. Rabiner, B. Gold, and C. McGonegal, "An approach to the approximation problem for nonrecursive digital filters," *IEEE Trans. Audio Electroacoust.*, vol. 18, no. 2, pp. 83–106, 1970.

[247] H. D. Helms, "Digital filters with equiripple or minimax responses," *IEEE Trans. Audio Electroacoust.*, vol. 19, no. 1, pp. 87–93, 1971.

[248] L. R. Rabiner, "Linear program design of finite impulse response FIR digital filters," *IEEE Trans. Audio Electroacoust.*, vol. 20, no. 4, pp. 280–288, Oct. 1972.

[249] J. Lewis, "Interactive minimax design of linear-phase nonrecursive digital filters subject to upper and lower function constraints," *IEEE Trans. Audio Electroacoust.*, vol. 20, no. 2, pp. 171–173, 1972.

[250] T. Parks and J. McClellan, "Chebyshev approximation for nonrecursive digital filters with linear phase," *IEEE Trans. Circuit Theory*, vol. 19, no. 2, pp. 189–194, 1972.

[251] J. McClellan and T. Parks, "A unified approach to the design of optimum FIR linear-phase digital filters," *IEEE Trans. Circuit Theory*, vol. 20, no. 6, pp. 697–701, 1973.

[252] L. Rabiner, J. H. McClellan, and T. W. Parks, "FIR digital filter design techniques using weighted Chebyshev approximation," *Proc. IEEE*, vol. 63, no. 4, pp. 595–610, 1975.

[253] D. M. Kodek, "Design of optimal finite wordlength FIR digital filters using integer programming techniques," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 28, no. 3, pp. 304–308, Jun. 1980.

[254] D. Kodek, "An algorithm for the design of optimal finite word-length FIR digital filters," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 5, 1980, pp. 73–76.

[255] D. Kodek and K. Steiglitz, "Comparison of optimal and local search methods for designing finite wordlength FIR digital filters," *IEEE Trans. Circuits Syst.*, vol. 28, no. 1, pp. 28–32, 1981.

[256] J. H. McClellan, T. W. Parks, and L. Rabiner, "A computer program for designing optimum FIR linear phase digital filters," *IEEE Trans. Audio Electroacoust.*, vol. 21, no. 6, pp. 506–526, 1973.

[257] V. B. Lawrence and A. Salazar, "Effects of finite coefficient precision on FIR filter spectra," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 4, 1979, pp. 378–379.

[258] D. M. Kodek, "Performance limit of finite wordlength FIR digital filters," *IEEE Trans. Signal Process.*, vol. 53, no. 7, pp. 2462–2469, Jul. 2005.

[259] J. Yli-Kaakinen and T. Saramaki, "A systematic algorithm for the design of multiplierless FIR filters," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, 2001, pp. 185–188.

[260] R. Yang, B. Liu, and Y. C. Lim, "A new structure of sharp transition FIR filters using frequency-response masking," *IEEE Trans. Circuits Syst.*, vol. 35, no. 8, pp. 955–966, 1988.

[261] Y. C. Lim and Y. Lian, "Frequency-response masking approach for digital filter design: complexity reduction via masking filter factorization," *IEEE Trans. Circuits Syst. II*, vol. 41, no. 8, pp. 518–525, 1994.

[262] Y. Lian and C. Z. Yang, "Complexity reduction by decoupling the masking filters from the bandedge shaping filter in the FRM technique," *Circuits Syst. Signal Process.*, vol. 22, no. 2, pp. 115–135, 2003.

[263] Y. Lian, "FPGA implementation of high speed multiplierless frequency response masking FIR filters," in *Proc. IEEE Workshop Signal Process. Syst.*, Lafayette, LA, 2000, pp. 317–325.

[264] Y. C. Lim, Y. J. Yu, H. Q. Zheng, and S. W. Foo, "FPGA implementation of digital filters synthesized using the frequency-response masking technique," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, Sydney, NSW, May 2001, pp. 173–176.

[265] J.-K. Liang, R. de Figueiredo, and F. Lu, "Design of optimal Nyquist, partial response, nth band, and nonuniform tap spacing FIR digital filters using linear programming techniques," *IEEE Trans. Circuits Syst.*, vol. 32, no. 4, pp. 386–392, 1985.

[266] J. Webb and J. Munson, D.C., "Chebyshev optimization of beamformers and FIR filters having failed elements," in *Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on*, vol. 2, 1992, pp. 557–560.

[267] ——, "Chebyshev optimization of sparse FIR filters using linear programming with an application to beamforming," *IEEE Trans. Signal Process.*, vol. 44, no. 8, pp. 1912–1922, 1996.

[268] T. Baran, D. Wei, and A. Oppenheim, "Linear programming algorithms for sparse filter design," *IEEE Trans. Signal Process.*, vol. 58, no. 3, pp. 1605–1617, 2010.

[269] D. Wei and A. Oppenheim, "Sparsity maximization under a quadratic constraint with applications in filter design," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2010, pp. 3686–3689.

[270] A. Jiang, H. K. Kwan, Y. Tang, and Y. Zhu, "Efficient design of sparse FIR filters with optimized filter length," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2014, pp. 966–969.

[271] A. Jiang, H. K. Kwan, Y. Zhu, X. Liu, N. Xu, and Y. Tang, "Design of sparse FIR filters with joint optimization of sparsity and filter order," *IEEE Trans. Circuits Syst. I*, vol. 62, no. 1, pp. 195–204, 2015.

[272] O. Gustafsson and H. Johansson, "Complexity comparison of linear-phase mth-band and general FIR filters," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2007, pp. 2335–2338.

[273] M. Smith and D. Farden, "Thinning the impulse response of FIR digital filters," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 6, 1981, pp. 240–242.

[274] Z.-J. Mou and P. Duhamel, "Fast fir filtering: Algorithms and implementations," *Signal Process.*, vol. 13, no. 4, pp. 377–384, 1987.

[275] Z. Mou and P. Duhamel, "A unified approach to the fast FIR filtering algorithms," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 1988, pp. 1914–1917.

[276] A. Zergaïnoh, P. Duhamel, and J. Vidal, "DSP implementation of fast FIR filtering algorithms using short FFT's," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 1, 1995, pp. 219–222.

[277] A. Zergaïnoh and P. Duhamel, "Implementation and performance of composite fast FIR filtering algorithms," in *Proc. IEEE Signal Process. Society Workshop VLSI Signal Process.*, 1995, pp. 267–276.

[278] T. G. Stockham Jr, "High-speed convolution and correlation," in *Proc. Spring joint computer conference*, ser. AFIPS '66 (Spring). ACM, 1966, pp. 229–233.

[279] W. Wang, M. N. S. Swamy, and M. O. Ahmad, "Low power FIR filter FPGA implementation based on distributed arithmetic and residue number system," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, vol. 1, 2001, pp. 102–105.

[280] M. Arnold, T. Bailey, J. Cowles, and M. Winkel, "Applying features of the IEEE 754 to sign/logarithm arithmetic," *IEEE Trans. Comput.*, vol. 41, no. 8, pp. 1040–1050, Aug. 1992.

[281] D. V. S. Chandra, "Error analysis of FIR filters implemented using logarithmic arithmetic," *IEEE Trans. Circuits Syst. II*, vol. 45, no. 6, pp. 744–747, Jun. 1998.

[282] M. Haselman, M. Beauchamp, A. Wood, S. Hauck, K. Underwood, and K. S. Hemmert, "A comparison of floating point and logarithmic number systems for FPGAs," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines.*, Napa, California, USA, Oct. 2005.

[283] T. Kurokawa, J. Payne, and S. C. Lee, "Error analysis of recursive digital filters implemented with logarithmic number systems," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 28, no. 6, pp. 706–715, 1980.

[284] W. Freking and K. Parhi, "Low-power FIR digital filters using residue arithmetic," in *Proc. Asilomar Conf. Signals Syst. Comput.*, vol. 1, 1997, pp. 739–743.

[285] G. Cardarilli, A. Del Re, A. Nannarelli, and M. Re, "Low power and low leakage implementation of RNS FIR filters," in *Proc. Asilomar Conf. Signals Syst. Comput.*, 2005, pp. 1620–1624.

[286] A. Doucet and A. M. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of Nonlinear Filtering*, vol. 12, pp. 656–704, 2009.

[287] F. Daum, "Nonlinear filters: beyond the Kalman filter," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 20, no. 8, pp. 57–69, 2005.

[288] S. Hong and O. Seong-Jun, "Architectures for particle filtering," in *Handbook of signal processing systems*, S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, Eds. Springer Science, 2013, pp. 639–670.

[289] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund, "Particle filters for positioning, navigation, and tracking," *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 425–437, 2002.

[290] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proc. Radar Signal Process.*, vol. 140, no. 2, pp. 107–113, 1993.

[291] J. D. Hol, T. B. Schön, and F. Gustafsson, "On resampling algorithms for particle filters," in *Proc. IEEE Nonlinear Stat. Signal Process. Workshop*, 2006, pp. 79–82.

[292] T. Li, M. Bolić, and P. M. Djurić, "Resampling methods for particle filtering: Classification, implementation and strategies," *IEEE Signal Process. Mag.*, vol. 32, no. 3, pp. 70–86, May 2015.

[293] W. R. Gilks and C. Berzuini, "Following a moving target-monte carlo inference for dynamic bayesian models," *Journal of the Royal Statistical Society. Soc. B*, vol. 63, pp. 127–146, 2001.

[294] C. Musso, N. Oudjane, and F. Le Gland, "Improving reguregular particle filters," in *Sequential Monte Carlo Methods in Practice*, A. Doucet, N. de Freitas, and N. Gordon, Eds. New York: Springer, 2001, pp. 247–271.

[295] G. Kitagawa, "Monte Carlo filter and smoother for non-Gaussian nonlinear state space models," *J. Comput. Graph. Stat.*, vol. 5, no. 1, pp. 1–25, Mar. 1996.

[296] J. S. Liu and R. Chen, "Sequential Monte Carlo methods for dynamic systems," *J. Am. Stat. Assoc.*, vol. 93, pp. 1032–1044, 1998.

[297] D. Crisan and T. Lyons, "A particle approximation of the solution of the kushner–stratonovitch equation," *Probability Theory and Related Fields*, vol. 115, no. 4, pp. 549–578, 1999.

[298] A. Budhiraja, L. Chen, and C. Lee, "A survey of numerical methods for nonlinear filtering problems," *Physica D*, vol. 230, no. 1, pp. 27–36, 2007.

[299] T. Li, T. Sattar, and D. Tang, "A fast resampling scheme for particle filters," in *Proc. Constantinides Int. Workshop on Signal Process.*, 2013, pp. 1–4.

[300] M. Bolic, P. Djuric, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *IEEE Trans. Signal Process.*, vol. 53, no. 7, pp. 2442–2450, 2005.

[301] J. Mountney, I. Obeid, and D. Silage, "Modular particle filtering FPGA hardware architecture for brain machine interfaces," in *Proc. IEEE Int. Conf. Eng. in Med. Bio. Society.*, 2011, pp. 4617–4620.

[302] J.-H. Lee and C.-K. Chen, "Design of sharp FIR filters with prescribed group delay," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1993, pp. 92–95.

[303] S. A. Alam and O. Gustafsson, "Implementation of time-multiplexed sparse periodic FIR filters for FRM on FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst.*, Rio de Janeiro, Brazil, May 2011.

[304] ——, "Implementation of narrow-band frequency-response masking for efficient narrow transition band FIR filters on FPGAs," in *Proc. Norchip.*, Lund, Sweden, Nov. 2011.

[305] T. Ahmed, M. Garrido, and O. Gustafsson, "A 512-point 8-parallel pipelined feedforward FFT for WPAN," in *Proc. Asilomar Conf. Signals Syst. Comput.*, 2011, pp. 981–984.

[306] R. E. Morley Jr., G. L. Engel, T. J. Sullivan, and S. M. Natarajan, "VLSI based design of a battery-operated digital hearing aid," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 1988, pp. 2512–2515.

[307] J. R. Sacha and M. J. Irwin, "Number representation for reducing switched capacitance in subband coding," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 1998, pp. 3125–3128.

[308] M. G. Arnold, "Reduced power consumption for MPEG decoding with LNS," in *Proc. IEEE Int. Applicat.-Specific Syst. Arch. Processors Conf.*, 2002, pp. 65–67.

[309] H. A. Taha, *Operations Research - An Introduction*, 8th ed. Pearson Prentice Hall, 2007.

[310] K. Muhammad, "Algorithmic and architectural techniques for low-power digital signal processing," Ph.D. dissertation, 1999.

[311] R. Mahesh and A. Vinod, "New reconfigurable architectures for implementing FIR filters with low complexity," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 2, pp. 275–288, 2010.

[312] O. Gustafsson, "Lower bounds for constant multiplication problems," *IEEE Trans. Circuits Syst. II*, vol. 54, no. 11, pp. 974–978, Nov. 2007.

[313] G. A. Ruiz and M. Granda, "Efficient implementation of 3x for radix-8 encoding," *Microelectron. J.*, vol. 39, no. 1, pp. 152–159, 2008.

[314] O. Gustafsson, "A difference based adder graph heuristic for multiple constant multiplication problems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 1097–1100.

[315] M.-B. Lin, "On the design of fast large fan-in CMOS multiplexers," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 8, pp. 963–967, 2000.

[316] M. Alioto, G. Di Cataldo, and G. Palumbo, "Optimized design of high fan-in multiplexers using tri-state buffers," *IEEE Trans. Circuits Syst. I*, vol. 49, no. 10, pp. 1500–1505, 2002.

[317] G. Hendeby, R. Karlsson, and F. Gustafsson, "Particle filtering: the need for speed," *EURASIP J. Advances Signal Process.*, vol. 2010, p. 22, 2010.

[318] S. A. Alam and O. Gustafsson, "Generalized resampling architecture and memory structure for particle filters," in *Proc. Europ. Conf. Circuit Theory Design*, Trondheim, Norway, Aug. 2015.

[319] Y. Joo and N. McKeown, "Doubling memory bandwidth for network buffers," in *Proc. IEEE Joint Conf. IEEE Comput. Comm. Soc.*, vol. 2, 1998, pp. 808–815.

[320] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines-a survey," *Proc. IEEE*, vol. 84, no. 8, pp. 1090–1123, 1996.

Part II

Publications

Papers

The articles associated with this thesis have been removed for copyright reasons. For more details about these see:
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-124195

Linköping Studies in Science and Technology, Dissertations
Division of Electronics Systems
Department of Electrical Engineering
Linköping University

J. Carlsson, *Contributions to Asynchronous Communication Ports for GALS Systems*, Linköping Studies in Science and Technology, Diss., No. 1062, Dec. 2006.

E. Säll, *Implementation of Flash Analog-to-Digital Converters in Silicon-on-Insulator CMOS Technology*, Linköping Studies in Science and Technology, Diss., No. 1093, May 2007.

E. Backenius, *Reduction of Substrate Noise in Mixed-Signal Circuits*, Linköping Studies in Science and Technology, Diss., No. 1094, May 2007.

L. Rosenbaum, *On Low-Complexity Frequency Selective Digital Filters and Filter Banks*, Linköping Studies in Science and Technology, Diss., No. 1097, June 2007.

M. Olsson, *Contributions to Delay, Gain, and Offset Estimation*, Linköping Studies in Science and Technology, Diss., No. 1198, June 2008.

K. Johansson, *Low Power and Low Complexity Shift-and-Add Based Computations*, Linköping Studies in Science and Technology, Diss., No. 1201, Oct. 2008.

A. Eghbali, *Contributions to Reconfigurable Filter Banks and Transmultiplexers*, Linköping Studies in Science and Technology, Diss., No. 1344, Dec. 2010.

A. Blad, *Low Complexity Techniques for Low Density Parity Check Code Decoders and Parallel Sigma-Delta ADC Structures*, Linköping Studies in Science and Technology, Diss., No. 1385, Sep. 2011.

M. Abbas, *On the Implementation of Integer and Non-Integer Sampling Rate Conversion*, Linköping Studies in Science and Technology, Diss., No. 1420, Feb. 2012.

F. Qureshi, *Optimization of Rotations in FFTs*, Linköping Studies in Science and Technology, Diss., No. 1423, Mar. 2012.

Z. U. Sheikh, *Efficient Realizations of Wide-Band and Reconfigurable FIR Systems*, Linköping Studies in Science and Technology, Diss., No. 1424, Mar. 2012.

Linköping Studies in Science and Technology, Dissertations
Division of Computer Engineering
Department of Electrical Engineering
Linköping University

U. Nordqvist, *Protocol Processing in Network Terminals*, Linköping Studies in Science and Technology, Diss., No. 865, Apr. 2004.

D. Wiklund, *Development and Performance Evaluation of Networks on Chip*, Linköping Studies in Science and Technology, Diss., No. 932, Apr. 2005.

E. Tell, *Design of Programmable Baseband Processors*, Linköping Studies in Science and Technology, Diss., No. 969, Oct. 2005.

A. Nilsson, *Design of Programmable Multi-Standard Baseband Processors*, Linköping Studies in Science and Technology, Diss., No. 1084, May 2007.

A. Ehliar, *Performance driven FPGA design with an ASIC perspective*, Linköping Studies in Science and Technology, Diss., No. 1237, Feb. 2009.

D. Wu, *Scalable Multi-Standard Radio Baseband for Modern Wireless Communications*, Linköping Studies in Science and Technology, Diss., No. 1279, Nov. 2009.

J. Eilert, *ASIP for Wireless Communication and Media*, Linköping Studies in Science and Technology, Diss., No. 1298, Feb. 2010.

R. Asghar, *Flexible Interleaving Sub-Systems for FEC in Baseband Processors*, Linköping Studies in Science and Technology, Diss., No. 1312, May 2010.

P. A. Karlström, *NoGAP: Novel Generator of Accelerators and Processors*, Linköping Studies in Science and Technology, Diss., No. 1347, Nov. 2010.

J. Wang, *Low Overhead Memory Subsystem Design for a Multicore Parallel DSP Processor*, Linköping Studies in Science and Technology, Diss., No. 1532, May 2014.

J. Sohl, *Efficient Compilation for Application Specific Instruction set DSP Processors with Multi-bank Memories*, Linköping Studies in Science and Technology, Diss., No. 1641, Feb. 2015.