Design Methodologies for Complexity Reduction of FIR Filters

Mathias Faust

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2014

Acknowledgments

This thesis was only possible with the great support and supervision that has been given to me very generously, for which I am thankful for.

First of all, I wish to thank my supervisor, Associate Professor Chip Hong Chang, for giving me the chance to do my PhD under his supervision, his intellectual guidance, his attention, his encouragement and the very fruitful discussions. In particular, I wish to thank him for teaching me some of the philosophical aspects of research and some intangible skills as well as all the advice for life he gave me. Besides the technical aspects there was and still is much invaluable knowledge and different perspectives of live I had the chance to learn.

I would like to thank Oscar Grustafsson at Linköping University (LiU) for all the very interesting and focused discussions and for providing me the opportunity to do research in Linköping for two months. Many thanks also go to Mr Li Fei, Mrs. Ramya Muralidharan, Mr. Qian Hanhua, Dr. Chen Jiajia for their friendship, help, discussions and suggestions pertaining to this research work. Many further thanks go to Dr. Andreas Kieble at NTU and Dr. Kent Palmkvist at LiU for all the interesting discussions over lunch.

To the staff and many other students at the Center for High Performance Embedded Systems, I wish to convey my appreciation to all of them for their kind and friendly assistance.

I would not want to miss thanking the University of Applied Sciences, Rapperswil, Switzerland (HSR) and Professor Erwin Brändle for giving me the opportunity to get to know NTU and thereby initiating the process of starting my thesis. Further I would like to thank HSR for the very good foundation in Electrical Engineering education that made it possible for me to pursue my PhD degree.

I would like to express my gratitude to my parents, in particular my dad, Hans Faust, and my now wife Wei Faust-Ran for their consistent support, encouragement and love throughout the period of my study. My gratitude also goes to all friends in Switzerland and all the new friends found in Singapore.

I would like to acknowledge the School of Electrical & Electronic Engineering, Nanyang Technological University for providing the great research facility and the research scholarship. Finally I would like express my gratitude to the country of Singapore for being a great place to live and study.

Summary

Digital signal processing is ubiquitous and many new applications have been developed for portable wireless communication devices due to the demand for connectivity. Versatile applications running on smaller, faster and energy efficient digital signal processors impose tremendous challenges in the design of the channelizers and signal conditioning circuits after the analog-to-digital conversion. With the advance in VLSI technology, FIR digital filter and other convolution like calculations can be implemented by dedicated hardware to address the needs for high throughput applications such as those in the communication system frontend where sharp transition bands for high selectivity are required. Over the past two decades of research, the transposed direct form structure has gained a strong foothold for high-speed application specific integrated circuit (ASIC) implementation of FIR filters. In this architecture, the coefficient multipliers can be grouped into a Multiple Constant Multiplication (MCM) block and implemented multiplication free. The optimization of the number of adders required in the MCM block has been a main subject of research in many leading publications on circuits and systems, and design automation. The design methodologies have advanced from simple minimization algorithms to exact, guaranteed minimal adder algorithms and powerful heuristic algorithms that can tailor the solutions to specific area and delay constraints. The time delayed accumulation of the outputs from the MCM block to produce the final output of the FIR filter can be grouped into a Structural Adder (SA) block. Although this block occupies a substantial part of the overall area of the filter, it has never been a target of complexity reduction in existing optimization algorithms.

In this thesis, the design methodologies for optimizing the MCM block are reviewed and discussed. New methods for the optimization of the MCM as well as SA blocks are proposed. First of all, a platform for design automation of FIR filters is introduced with enhanced features added onto this tool for result generation and functional verification. For research convenience and fair evaluation of MCM and SA optimization algorithms, a new benchmark suite has been established and introduced in this thesis. A novel graph based MCM optimization method aimed at minimizing the critical path delay is proposed and extended to deal with the reconfigurable MCM problem. Based on the analysis of the bit-level signals, two new methods for optimizing the MCM block at the bit-level are proposed. One of which targets Field Programmable Gate Array (FPGA) implementation and the other addresses the demand of very high-speed applications in ASIC implementation. A key discovery of this thesis is the optimization opportunity of SA block. Two SA block optimization methods have been proposed, one for the SAs implemented in two's complement representation and one for the offset representation. In essence, the methods strive for a positive tradeoff between combinational logic and registers to reduce the implementation cost. Pipelining and bit width reduction techniques for

SA block optimization are also proposed. By making use of the existing registers in the tap-delay line of SA block, only a small percentage of additional registers need to be introduced to allows for pipelining. A new method has also be introduced to gradually reduce the bit width of the operators in the SA block towards the output with a very small sacrifice of the output precision. Finally, very high-speed circuit architectures for the conversions from Binary to Canonical Signed Digit (CSD) and from CSD to Binary representations are proposed. These components can be used to speed up and simplify the implementation of adaptive filters.

Contents

List of Tables

List of Figures

Acronyms

AG	Adder Graph
ASIC	Application Specific Integrated Circuit
BCS	Binary Common Subexpression
BSE	Binary Subexpression Elimination
CLA	Carry Look-ahead Adder
CLT	Central Limit Theorem
CMOS	Complementary Metal-Oxide-Semiconductor
CPA	Carry Propagate Adder
CRA	Contention Resolution Algorithm
CS	Common Subexpression
CSA	Carry Save Adder
CSD	Canonical Signed Digit
CSE	Common Subexpression Elimination
CV	Correction Vector
DA	Distributed Arithemtic
DAG	Directed Acyclic Graph
DCT	Discrete Cosine Transform
DOF	Digits of Freedom
DSP	Digital Signal Processor
DWT	Discrete Wavelet Transform
FA	Full Adder
FF	Flip Flop
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
GPC	Glitch Path Count
GPS	Glitch Path Score
HA	Half Adder
Hcub	Cumulative Benefit Heuristic
HDL	Hardware Description Language
HND	Hidden Nonzero Digit
IF	Intermediate Frequency
ILP	Integer Linear Programming
IIR	Infinity Impulse Response
LO	Logic Operator
LD	Logic Depth
LUT	Look up Table
LSB	Least Significant Bit
MAG	Minimum Adder Graph
MBPG	Multi-root Binary Partition Graph
MCM	Multiple Constant Multiplication
MSB	Most Significant Bit
MSD	Minimal Signed Digit
NR-SCSE	Non-Recursive Signed Common Subexpression Elimination
NTU	Nanyang Technological University
NOF	Non-Output Fundamental
ODP	Overlapping Digit Pattern
RCA	Ripple Carry Adder
ReMB	Reconfigurable Multiplier Block
ReMCM	Reconfigurable Multiple Constant Multiplication
SA	Structural Adder
SCM	Single Constant Multiplication
SD	Signed Digit
SDR	Software Defined Radio
SIMO	Single Input Multiple Output
sinc	sinus cardinalis $\left(\frac{sin(x)}{x}\right)$
SISO	Single Input Single Output
SQNR	Signal-to-Quantization-Noise Ratio
TDFG	Time-multiplexed Data-Flow Graph
VCS	Vertical Common Subexpression
VCSE	Vertical Common Subexpression Elimination
VHDL	VHSIC Hardware Description Language
VLSI	Very Large Scale Integration

Chapter 1

Introduction

1.1 Motivation

Nowadays, we are surrounded daily by electronic gadgets and increasingly more of them are mobile. The quest for mobility presents new challenges like limited power sources and wireless communication to the designer of electronic equipments. With the rapid advance of Very Large Scale Integration (VLSI) technology, paired with the economy of scale of CMOS fabrication, it is possible today to migrate more signal processing functions for the wireless communication from the analog to the digital domain. High-speed clocked Application Specific Integrated Circuit (ASIC) chips are used to meet the demand of high throughput for the filtering of Intermediate Frequency (IF) signals. The high clock rate for these filter circuits incurs high power consumption which is in conflict with the low power requirement for battery powered devices. To meet the low power requirement, stringent area and power constraints are imposed during the design phase.

The downsizing of cell geometry and high integration density of advanced CMOS semiconductor fabrication processes lead to a productively gap between the tools available for the design of VLSI circuits and the complexity of these circuits. To reduce the design turnaround time and lower the non-recurrent engineering cost, new design methodologies are needed. The use of specific optimization methods dedicated to a well-defined class of application specific problems can help to achieve higher optimality in area, delay and power consumption. The Multiple Constant Multiplication (MCM) [1] problem represents such a generic class of applications. Many kernel functions in communication signal processing and multimedia signal processing can be modeled as MCM problems. As the multiplications are carried out with constants, it can be decomposed into shifters and simpler additions or subtractions. The most commonly studied MCM problems are Finite Impulse Response (FIR) filters as filters and filter-like functions are abundantly found in Digital Signal Processor (DSP) and Software Defined Radio (SDR).

The design and optimization of low-complexity FIR filters have been studied extensively over the past three decades [1]–[24]. In the research, two main approaches for the optimization can be categorized. One approach decomposes numerical values of the coefficients into a minimal set of partial sums by sharing the summands, the other approach represents the coefficient values in binary or Signed Digit (SD) representations and search for common bit patterns called Common Subexpressions (CSs) to eliminate. Both approaches produce an optimized

shift-and-add network. However, the numerical approach produces results with less number of operators but with a higher number of operators in the input-output path. This is a desired outcome in terms of area minimization but the power consumption is higher due to a higher glitch probability. From design automation perspective, although several power modeling techniques have been proposed, they cannot be efficiently or effectively integrated into FIR filter design methodologies for power driven synthesis at high-level. Therefore, reducing the logic complexity of shift-and-add network without extending the length of the input-output path remains the mainstream and an effective design philosophy to reduce the overall area and power consumption simultaneously, for which the numerical value based approach is advantageous. Until a few years ago, the measure of quality of the shift-and-add network optimization was based on the number of Logic Operator (LO) and the Logic Depth (LD). Using the recently introduced Full Adder (FA) cost provides new insights into the quality assessment which can be used for new and more advanced algorithms.

Looking at the optimization of FIR filter, there are two prominent problems. First, there algorithms are often tested and compared on different filters, which makes it hard to do a fair and comprehensive comparison among several MCM methods. Secondly, the MCM methods as well as many filter design methods focus solely on the optimization of the constant multipliers within the filter and have completely ignored the complexity of the Structural Adder (SA) block. Existing MCM optimization algorithms and logic synthesis tools are so advanced that the size of the constant multiplier block has become a fraction of the overall filter area and the unoptimized Structural Adder (SA) block is actually the dominant throughput and complexity bottleneck. The SA block is in many ways different from the MCM block. The SA block has many inputs, that have to be summed up in successive clock cycles to obtain a single output sample. Its internal signals stem from different samples are uncorrelated, which makes it impossible to use the same optimization methodologies as for the MCM block to reduce its complexity. The sinus cardinalis $\left(\frac{sin(x)}{x}\right)$ (sinc) like envelope of the constant multipliers results in a bit width of the MCM outputs that are smaller towards the outer taps. This leads to large differences in the word length of the inputs to the SAs towards the output of the SA block. This incurs a long sign extension for the arithmetic operators implemented in two's complement representation, and a longer carry propagation through many half adders for the offset representation.

There is not much evidence of studies on the addition of highly correlated signals, although this exists in the MCM block. At the bit-level, there exist some interesting properties in the input-output relation which are significant different from the word level properties considered in most current approach to MCM block optimization. This has also inspired the research into further optimization opportunity of MCM by exploring different bit-level optimization approaches.

1.2 Objective

The objective of this research is to investigate the optimization VLSI or Field Programmable Gate Array (FPGA) implementations of FIR filters or topics related to this optimization. Insights gained from low level VLSI synthesis of previous research results are translated into rules and optimizations at a higher design level in order to reduce the area and the delay cost of FIR filters. For FIR filters, this research targets the transposed direct form structure due to its immense possibilities for complexity reduction in the Multiple Constant Multiplication (MCM) block and the new challenge of Structural Adder (SA) block optimization for very high-throughput operation. The algorithms are to be developed should be at the behavioral and algorithmic level, but it should not exclude the novel application of some low level optimization techniques for the formulation of new MCM optimization methods. Beside the fully parallel MCM implementations, time- multiplexed problems will also be investigated as well.

The MCM problem appears in many DSP operations, such as digital filters and kernels of other transformations but its minimization is an NP-complete problem. This restricts the optimization algorithm to heuristic methods for large problem. Nevertheless, elegant exhaustive searches for smaller problem by some researchers provide a good insight from which good heuristic involving partially or conditional exhaustive search are to be explored.

With the importance of power consumption in VLSI design, contributors to dynamic power consumption of the FIR filter implementation will be taken into consideration at algorithmic level. In previous research it was shown that lower logic depth reduces the possibility of glitches and therefore reduces the dynamic power. CSE optimization tends to produce solutions with lower logic depth while Adder Graph (AG) optimization algorithms tend to produce solutions with less number of adders. Thus, one of the goals of this research is to analyze and leverage the best of CSE and AG based algorithms by considering them as points on a Pareto curve for area and delay minimization in order to find the middle ground to optimize the implementation cost in the area-delay sense.

The SA block contributes a significant amount of area in the filter implementation if the filter contains many taps. The optimization poses a radically different challenge from that of the MCM block because the inputs of the adders in the SA block are not correlated. A probabilistic analysis of the signal properties along the tap-delay line is to be carried out. Bit- or subword-level optimizations will be considered. As there is only one carry propagation adder in the critical path of the SA and it is most probably the throughput stumbling block, new pipelining technique that has never been considered for the SA block will be sought.

CSD coefficients are used frequently during the design process of fixed coefficient FIR filters. In reconfigurable or programmable filter implementation, CSD representation is equally beneficial except that an efficient way to store or convert the external two's complement or binary signals to CSD representation is required. This research also aims to solve this problem by investigating the properties of CSD and how they could be exploited for efficient storage or on-the-fly conversion. For converter, the emphasis is high-speed conversion whereas for storage the overhead is to be minimized.

1.3 Major Contributions

The work reported in this thesis aims to provide new algorithm for the optimization of VLSI or FPGA implementation of digital filters with constant or reconfigurable constant multipliers and the optimization of the adder-delay chain of finite impulse response digital filters. A further aim is to find hardware implementation converters to and from the CSD representation to facilitate efficient implementation of reconfigurable multiple constant multiplication or programmable filters. The attempts have led to following contributions.

(1) sysFIR - A Platform for Design Automation of FIR Filters & FIRsuite

With the latest feature improvements added onto this research tool sysFIR developed by the author, the process of development and comparison of different FIR filter optimization algorithms has been greatly simplified by a standardized input and output formats that allow full automation of synthesizable VHSIC Hardware Description Language (VHDL) code generation. The new option facilitates integrated testing in the sysFIR environment to ensure that the final VHDL code is not only syntactically correct but also functionally correct.

(2) Minimal Logic Depth Adder Tree Optimization & Reconfigurable Multiple Constant Multiplication using Minimum Adder Depth

The switching power consumption of MCM block in operation is mainly contributed by the number of adders and the glitches that they generate. The dilemma is that algorithms that produce the least number of adders have increased logic depth and algorithms reducing the logic depth have less than optimal number of adders. Based on a detailed analysis of many different ways to build an adder tree for a constant multiplication from the CSD coefficient space, an efficient minimal logic depth and near optimal adder optimization algorithm has been proposed. Comparing to the AG algorithms, its number of adders increases only marginally but it has the minimum logic depth for all the coefficient multipliers. The small logic operator coupled with the minimal logic depth reduces the dynamic power of the proposed method by a factor of up to 3.4 times according to two existing algorithmic level shift-and-add network power estimators. The proposed algorithm has been extended to optimize polyphase FIR filters and other multiple output MCM problems.

(3) LUT-based and Partial Product Reduction-based Multiple Constant Multiplication

Two methods for bit-parallel MCM optimization have been proposed. A Look up Table (LUT)-based method targets FPGA implementation with low input bit width. It was shown that there exists an upper limit for the number of LUTs required for each input bit width, which is far below the theoretical limit due to the internal symmetry of the LUT outputs. The proposed algorithm optimizes the use of LUTs of the FPGA for multiplication to outperform FPGA implemented with a regular MCM solution. By splitting an 8-bit input signal into two 4-bit signals, simulation results for several FIR filters showed that the proposed method is comparable with the current-art adder-based MCM optimization techniques in terms of the usage of logic slices and outperforms them by about 33% in terms of delay. The second approach has been derived from a close examination of the internal signals of constant multipliers of a general purpose multiplier. The analysis reveals that it is possible to remove redundancies at

bit-level due to the high repetition of some bit-level information of the signals. Based on the number of nonzero digits in the CSD representation of the constant, an upper bound for the partial product matrix has been derived and proven. A set of rules to achieve the upper bound has been derived to reduce the parital product height and reuse the adders. Using a FIR filter example with an 8-bit input, it is shown that the number of FAs and Half Adders (HAs) required in the first stage of the Carry Save Adder (CSA) tree can be reduced by 41% and 56%, respectively. Both approaches are promising for the problems where the input bit width is only single decimal digit and the constant multipliers have larger bit width.

(4) Structural Adder Optimization

A technique has been proposed to provide an optimized tradeoff between combinational logic and registers for SA implemented in two's complement representation and offset representation. For two's complement representation, the proposed method reduces the sign extension by bisecting the saved sum at the cost of a few additional registers. The tradeoff reduces the area by up to 13.68% as observed from the synthesis results of a set of benchmark filters. For offset representation, the method reduces the carry propagation in the higher order bit position of the adder by saving and forwarding the carry to the next adder. Synthesis results of a set of benchmark filters showed an area reduction of up to 6.23% for ASIC implementation and up to 8.52% for FPGA implementation. When the optimization of the SA block is combined with the MCM optimization, up to 14.48% of area reduction can be obtained without changing the output value of the filter. Pipelining of SA block is also proposed to increase the throughput of very high-speed filter by saving the intermediate carry signal. Delay minimization synthesis results showed that a delay reduction of 30% with an area increment of less than 4% can be achieved. Finally, a new method using truncation and round-half-up was proposed to gradually reduce the fractional part of the signal in the structural adder block to the same number of factional bits as the input signal. The results showed that the proposed method reduced the circuit area by 12.42% with the error of limited to the least significant bit only. The absolute area reduction and the error of this method are both independent of the filter length and the input bit width, but dependent only on the precision of the filter coefficients. Based on statistical probabilities, it was shown that the normal distribution can be used as a good prediction of the probable ranges of signal values within the SA block.

(5) Binary-to-CSD converter and reverse converter

Using a bypass signal, a Binary-to-CSD converter is proposed which outperforms other conversion methods. Only a single gate is used in the carry propagation path of the proposed converter circuit. Using simple look-ahead circuits, the critical path can further be shortened. Beside the Binary-to-CSD converter, a reverse converter with comparable speed is also proposed to provide the communication interface between the peripherals of two's complement and CSD processing units.

The above contributions have led to the publications listed in the author's publications at the end of the thesis.

1.4 Awards

PrimeASIA 2010 Silver Leaf Certificate
Awarded for the article "Reduction of Partial Product Matrix for High-Speed Single or Multiple Constant Multiplication" presented at the *2010 Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)* held in Shanghai, China.

2009 IEEE Circuits and Systems Singapore Chapter Graduate Student Award
Awarded for outstanding academic results[1] during the PhD studies at Nanyang Technological University (NTU). The Award was presented during the *IEEE Circuits and Systems Singapore Chapter Graduate Workshop in Circuits, Systems and Signal Processing* on 29. September 2009.

[1] Two 'A+', three 'A' and a pass in a MAS special topic course (CGPA = 5.0).

1.5 Organization of the Thesis

The thesis is organized in eight chapters. Chapter 1 presents the motivation and objective of this research work. The achievements resulted from the works presented in this thesis are summarized under the main contributions of this chapter.

Chapter 2 presents the fundamentals of FIR filters, their design and implementation followed by a review of hardware multiplication and discussion on the addition of correlated signals. Number representations commonly considered in the implementation of digital circuits are introduced and their respective limitations are highlighted. A concise introduction to the bit width requirements is provided for the understanding of the optimization of structural adders in a FIR filter. This is followed by an extensive review of single and multiple constant multiplication algorithms. Two generic algorithms are described to illustrate the fundamental operations of Adder Graph and Common Subexpression Elimination algorithms to multiple constant multiplication.

Chapter 3 presents the work done on "sysFIR". A framework of design automation algorithms for the ASIC implementation of FIR filters. This tool was originally developed before the PhD candidature and has been improved and heavily used for the PhD research. This chapter describes its design automation features such as its ability to optimize a given coefficient set with different MCM optimization methods as well as the automatic generation of their VHDL codes and associated scripts to run Synopsys Design Compiler or Xilinx ISE. During the development process of optimization methods, the automated functional test provided by the tool is also very useful to ensure that the VHDL code produces the expected outputs.

Chapter 4 presents two optimization methods at word level. First, a close scrutiny of Binary Common Subexpression Elimination is made and it is concluded that there were some severe oversights in the publication of this method. Then a method for the optimization of the MCM block is presented. It uses a graph based approach but limits the logic depth for each constant to its minima. The results obtained by the proposed method are compared with that of previous MCM optimization methods. The power consumption of its solutions are also estimated based on the glitch path count and glitch path score and compared with the graph based method without logic depth constraint. Finally, the extension of this method to reconfigurable multiple constant multiplication for use in polyphase FIR filters is presented. The advantages and disadvantages of different slicing possibilities for polyphase filters are discussed and compared with previously known methods.

Chapter 5 takes the optimization of MCM problems down to the granularity of bit-level. Two different optimizations at bit-level are presented. First, the number of LUTs required to generate a bit of the filter output is analyzed. Based on the finding that the number of LUTs required are relatively low and can be met by the sizes of LUT commonly found in FPGAs,

an optimization method targets mainly FPGAs implementation is presented. The results in terms of used logic slices and timing are analyzed and discussed. Next, the problem of constant multiplication is analyzed from the perspective of a general purpose multiplier, where one input is a constant. An analysis was carried out on the upper bound of the height of the partial product matrix for different constants and input bit widths. A method is then proposed to reduce the partial product matrix and reuse the internal signals. The numbers of half adders and full adders that can be reduced by the proposed method is also determined by experiments.

Chapter 6 analyzes the signal properties and their bit probability distribution in the structural adder block. It is shown that the value distribution of the range of possible values can be approximated by a normal distribution and a closed form expression for the estimation of its mean and variance is presented. Possibilities of area reductions of the structural adder block are studied for the arithmetic operations implemented in two's complement representation and offset representation. For both representations, the general rules to trade logic circuits for registers that will lead to an overall cost reduction are established. A pipelining method with small impact on the area is also proposed to speed up the calculation within the structural adder block. Finally a method to reduce the output bit width of the structural adder is presented. The method reduces the area cost by sacrificing a very small output precision.

Chapter 7 introduces new forward and reverse converters circuit for binary to CSD representations. A high speed converter for the conversion from binary to CSD is introduced first. The inevitable ripple propagation path has reduced to only a single two-input gate. The circuit for the reverse conversion from CSD to binary representation is also introduced. It is slightly slower due to the presence of two cascaded two-input gates in the critical path. For both converters, techniques to speed up the computation are also suggested to limit the delay to the order of only $\mathrm{O}(\log n)$.

Chapter 8 concludes this thesis by highlighting the features and merits of the presented methods, discussing the remaining problems and suggesting potentially relevant topics worthy of further investigation.

Chapter 2

Background and Literature Review

Digital Finite Impulse Response (FIR) finds ubiquitous applications in Digital Signal Processor (DSP), communication and control. With increasing portability and mobility of communication and computing devices, there is a need for better design methodologies to reduce the complexity and cost of implementation of application-specific digital FIR filters.

Many researchers have attempted and are still attempting the challenge of optimizing the implementation of digital FIR filters. Over the past couples of decades, a great number of algorithms and techniques were proposed, including several highly elaborative and exhaustive approaches. These attempts have helped to advance the understanding of the fundamental optimization problem of the Multiple Constant Multiplication (MCM) block underpinning the FIR filter design. Owing to its nature and practicality, the problem can only be solved optimally for small problem size, which opens the door for more efficient new heuristic approaches to thrive. Over time, the figures of merit have changed and derivatives of optimization goals emerge which change the cost measures and pose new design challenges.

The first part of this chapter provides the elementary theory of digital FIR filters, their different implementation topologies and the fundamentals of hardware implementation of addition and multiplication. Different possibilities of representing an integer number for binary logic circuit implementation and their implications will also reviewed and discussed. While there exists a broad design space exploration for the implementation of digital multiplication in hardware, the special case where one input of the multiplier is constant has been heavily exploited in the optimization of digital FIR filters. Another important fundamental building block is the adder for which a multitude of different implementation methods exist, but they all assume that the addends are independent. The special case where the addends are correlated has not been investigated or at least not well-exploited in most research publications although such addends appear frequently in the multiplierless implementation of FIR filters.

The second part of this chapter provides a comprehensive literature review of important design algorithms. Generic algorithms are derived to simplify the understanding of the key design concepts behind two main classes of design algorithms surveyed. A large portion of the literature review was prepared by the author in response to the call for collaboration of a survey paper jointly authored by Pramod Kumar Meher, my thesis advisor Chip Hong Chang, Oscar Gustafsson, A. P. Vinod and me. For some reason, mainly journal page length constraints, this study will be rewritten as a chapter for a book edited by Pramod Kumar Meher.

This chapter lays the foundation for the understanding of later chapters.

2.1 Background

This section presents the background of digital FIR filter, design thereof, constant multiplication, hardware for addition and an introduction to different number representations. A basic understanding of digital signal processing, binary signal representation and arithmetic circuit is assumed. For further details on the topics, [25] is recommended. A good reference for VLSI implementation of digital signal processing algorithms can be found in [26] and [27].

2.1.1 FIR Filters and their Hardware Implementations

A filter is specified by its transfer function. The general transfer function of a filter is given by:

$$H(\omega) = \frac{\sum_{k=0}^{N-1} b_k e^{-j\omega k}}{1 - \sum_{k=1}^{M} a_k e^{-j\omega k}} \tag{2.1}$$

where a_k and b_k are the filter coefficients of the recursive and non-recursive parts of the filter, respectively. If all coefficients a_k are zero, the feedback path of the filter is removed and the output is reduced to a weighted sum of the sampled inputs. The z-transform of the impulse response of a FIR filter can be written as:

$$H(z) = \sum_{k=0}^{N-1} h[k] z^{-k} \tag{2.2}$$

where the values of the impulse response $h[k]$ can also be seen as the weights by which the sampled input signal is modulated and are identical to the filter coefficients b_k in Equation (2.1).

Due to the fact that FIR filters do not contain a feedback or recursive path, they are unconditionally stable. This comes at the cost that FIR filters require a longer impulse responses to achieve the same attenuation compared to Infinity Impulse Response (IIR) filters. The longer the impulse response, the more arithmetic operations and delay elements are required for the implementation of the filter.

The output sequence of a FIR filter is calculated as

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k] = h(n) * x[n] = \mathcal{Z}^{-1}\left\{H(z) \cdot X(z)\right\} \tag{2.3}$$

where $x[n]$ are the input samples and $y[n]$ are the output samples. The values of the impulse response $h[k]$ are often called coefficients and from here onwards denoted as c_k or $c[k]$. How these filter coefficients are obtained for a desired transfer function is not the focus of this thesis.

The multiplication in the frequency domain corresponds to convolution in the time domain. Therefore a FIR filter performs a time convolution of its impulse response with the sequence of input samples. This convolution can be implemented in various ways, where the direct form and

the transposed direct form structures are the most frequently used hardware implementation of FIR filters.

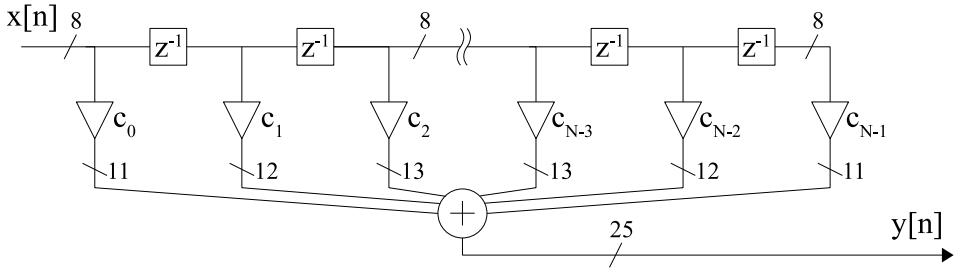


Figure 2.1: FIR Filter in Direct From

The direct form structure is a direct implementation of Equation (2.3) and is shown in Figure 2.1. The output is calculated as a weighted sum of the current and delayed input samples. The critical path for an unoptimized structure is $t_{direct} = t_{mult} + (N-1) \cdot t_{add}$. By implementing the accumulator as a balanced adder tree, the delay of the critical path can be reduced to $t_{direct,tree} = t_{mult} + \log_2(N) \cdot t_{add}$. Implementations of direct form structure using balanced adder trees or CSA trees are limited to filters with smaller N, as the adder tree grows very rapidly with increasing N and to achieve low latency many pipelining registers would need to be inserted. CSA trees speed up the partial sum accumulation, but they have an irregular structure which poses problems to the layout and routing phases of VLSI implementation. Furthermore, CSA is not able to exploit the symmetry of linear phase filters for hardware reduction and CSA tree are expensive in terms of pipelining as both the sum and carry bits of an intermediate sum need to be registered.

To achieve shorter critical path without pipelining, the signal flow graph is reversed which leads to the transposed direct form structure. Figure 2.2 shows the structure of a transposed direct form FIR filter. In this form the input is concurrently multiplied by several coefficients and the results are added and delayed. The critical path is reduced to $t_{transpose} = t_{mult} + t_{add}$ as the coefficient multiplication is calculated in parallel and the accumulation of coefficient multiplier outputs is pipelined. The transposed direct form equivalent of Equation (2.3), with $c(k) = h(k)$ is given by:

$$y[n] = c(0)x[n] + z^{-1}\left(c(1)x[n] + z^{-1}\left(c(2)x[n] + z^{-1}\left(c(3)x[n] + \ldots\right)\right)\right) \tag{2.4}$$

Unless the FIR filter is adaptive, the coefficients $c[k]$ are fixed and can be quantized into finite precision while preserving the filter frequency response specifications for fixed-point hardware implementation. In Figure 2.2, a_i denotes the adder that adds the result q_i from the constant multiplier c_i and the tapped-delay line signal p_i from the previous adder a_{i+1}. It is important to note that the coefficient c_0 is closest to the output $y[n]$.

A transposed direct form FIR filter can be divided into two main parts, the MCM block and the Structural Adder (SA) block, as shown in Figure 2.2. From the literature review (see Section 2.2), it is noted that existing research works on hardware optimization focuses solely

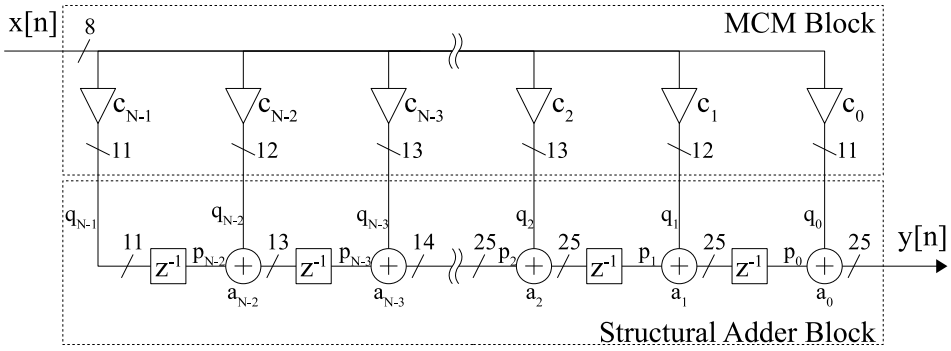


Figure 2.2: FIR Filter in Transposed Direct Form

on the MCM block. No attention is paid on minimizing the cost of the SA block even though it is a major share of the total chip area. The area needed for the SA block is smaller for short filters and grows with the number of taps. The absolute increase in area is almost linear, as each additional tap adds one delay unit and one structural adder into the SA block. By comparing Figure 2.1 and Figure 2.2, and analyzing the bit widths[1] of the delay elements, it can be seen that the delay elements are several times larger for the transposed direct form. This is because the range of the output signal increases in the accumulators along the tap delay line in the transposed direct form structure as opposed to the small constant input signal range to each delay element in the direct form structure. Due to the nature of the signals in the SA block, there is some potential to optimize it. Techniques to optimize the SA block will be proposed and presented in Chapter 6.

The term Multiple Constant Multiplication (MCM) was first used in [1]. The hardware implementation of the MCM block can be simplified by the fact that a multiplication with a constant can be decomposed into multiple shift-and-add operations and the shift amounts can be predetermined from the value of the constant. Each fixed shift corresponds to a multiplication by a power of two factor and is considered free as it can be realized by routing and does not require logic. Therefore, all even multipliers can be divided by two until they are odd. The sign for negative coefficients is changed to positive, as the inversion can be implemented as subtraction in the SA block. Then only this reduced set of odd positive fundamentals, as introduced in [29], needs to be considered for the implementation of MCM block. The optimal implementation of MCM block using this set of odd fundamentals is a hot topic for many research publications. More discussion on this will be detailed in Section 2.2.

2.1.2 Filter Design

Given the filter specification of an application by, for example the passband ripple, stopband attenuation, and the pass- and stopband edges, the filter coefficients can be generated straightforwardly by the Parks-McClellan algorithm, a function that is available in the signal processing toolbox of MatLab. The real valued coefficients are then quantized into finite word length numbers while ensuring that the required passband ripple and stopband attenuation

[1]The bit widths given in Figures 2.1 and 2.2 are from Filter Example 1 of [28] when using 8-bit input.

specifications are still met. This quantization process is simple, but the implication on hardware cost is not considered. There are several design processes that generate finite-precision coefficient values directly from the filter specification with different degrees of awareness of the hardware cost. The idea was introduced by Lim *et al.* in [28], [30] and improved in [31]. These techniques meet the filter response specification with a reduced number of positive and negative power-of-two terms compared with quantization of infinite precision filter coefficients. In recent years several algorithm [32]–[34] were introduced where the coefficient synthesis approaches take into consideration the sharing of subexpressions in MCM optimization. These algorithms optimize the quantized filter response in the subexpression space instead of the signed powers-of-two space. A filter designed by these techniques results in a smaller MCM block, as the coefficients are synthesized with preference to more frequently occurred common subexpressions. Other approaches are the use of the self similarity in [35], where the similarity of the side lobes is used to reduce the complexity. In [36], a branch-and-bound algorithm for the design of FIR filters in signed power-of-two terms was introduced and it was shown to be superior to older algorithms like [28].

Due to the vast interest in MCM optimization, other algorithms that require a filter structure which cannot be transformed into the transposed direct form without a cost implication, will not be reviewed in detail, although they may hold some desirable properties like low coefficient sensitivity. A special case is the work [37], where a filter is decomposed into two subfilters. A detailed bit width analysis is presented in [37]. Although the output of the first filter is larger than the input, and hence the bit-level cost for implementation is larger for the second filter due to the extended input, the results in [37] show that there can be an advantage in FA count. If the signal between the two subfilters is truncated or rounded, the FA count could be even lower but the truncation or rounding has an apparent influence on the filter response. Its impact needs to be addressed.

This thesis focuses on the optimization of any fixed (as opposed to programmable) filter by either optimizing the MCM or the SA block. The only requirement for the application of the presented algorithms is that the filter can be transformed into the transposed direct form with finite precision coefficients.

2.1.3 Constant Multiplication in Hardware

The multiplication of two signals, $y = c \cdot x$, in hardware is generally executed in a way similar to the multiplication by hand. An array of intermediate signals called the partial products is generated first. Each row of the partial product array is a shifted version of one signal multiplied by the corresponding digit from the other signal. In binary representation, the partial product row is generated by a simple array of AND gates of the respective signals. The addition of these partial products requires $n-1$ additions, where n is the bit width of the multiplier[1]. For faster addition, the partial products can be summed up parallelly with a CSA tree to only two final rows, which will be added by a Carry Propagate Adder (CPA).

[1] the multiplier here refers to one of the two operands of a multiplication that determines the number of partial products. The other operand is called the multiplicand.

If one operand of the multiplication is a constant c, as in the coefficient multiplier of the MCM block, the AND array for the partial products can be eliminated. No partial product will be generated when the corresponding bit of the constant is 0, and the multiplicand is wired though when it is 1. The number of additions is dependent on the number of ones in the constant (n = number of ones in c). This reduces the area considerably and shortens the delay as well. As all partial product rows are simply a shifted version of the input signal, the multiplierless network is referred to as the shift-and-add network in the literature [4], [38]. For example, if $c = 79$, c can be written as $[01001111]_2$ to obtain

$$79x = (x \ll 6) + (x \ll 3) + (x \ll 2) + (x \ll 1) + x \tag{2.5}$$

where $x \ll l$ represents the left-shift of x by l bit positions.

By introducing subtractors, the digit -1, also written as $\bar{1}$, can be included in the weighted number representation of the constant. The resulting representation is called the Signed Digit (SD) representation, which will be discussed in Section 2.1.6. The use of SD representation reduces the number of nonzero digits and hence the number of additions and subtractions required as a whole. For the previous example, 79 can be represented as $[0101000\bar{1}]_{SD}$ and the multiplication can be written as

$$79x = (x \ll 6) + (x \ll 4) - x \tag{2.6}$$

For this example, the use of SD reduces the number of additions from four for Equation (2.5) to only one addition and subtraction for Equation (2.6).

As the MCM block contains several parallel constant multiplications, there are possibilities to share intermediate results among the constant multiplications. This is the fundamental idea behind all MCM optimization algorithms. Different approaches and algorithms will be reviewed in Section 2.2.

Several special properties are observed in the multiplication of an input signal with a constant. If the constant is positive, the sign bit of the output has to be the same as that of the input.

$$y = c \cdot x \Rightarrow \text{sign}(y) = \text{sign}(c) \cdot \text{sign}(x),\ |y| = |c| \cdot |x| \tag{2.7}$$

If the constant is negative, it is necessary to have a zero detector, as the sign is inverted for all numbers except Zero[1].

Another property is related to the range of the output. If the input signal has a bit width of $n = 8$, it can have 256 consecutive values. As the input signal is variable and if all input values are probable, the entire range of the input signal space is fully utilized. When this input signal is multiplied by a constant, which is for example 59, then the maximum amplitude of the output signal will be enlarged to $59 \cdot 256 = 15104$ for this example. This output variable requires at least 14 bits to be represented. Two important observations can be made. First, a 14-bit word allows 16384 values to be represented, but only 15104 values can be used maximally.

[1]Given that two's complement or any other representation with one unique representation of zero is used.

Therefore not all the binary combinations of 14 bits are used. Secondly, and more importantly, although 16384 values can be represented, only 256 values are possibly used. The reason for this is that multiplying a variable by a constant can only produce as many different multiples of the constant as the full dynamic range of the variable. In contrast, the range of the product of two variables spans the entire codomain. Mathematically speaking, a general multiplication is a surjection, $\mathbf{R} \times \mathbf{R} \mapsto \mathbf{R}$ while a constant multiplication is a bijection, $\mathbf{R} \mapsto \mathbf{R}$. These observations is used later in Section 6.1 to derive the signal probabilities in the SA block.

By analyzing several constant multiplications, it was discovered that the complexity of the Boolean covering problem[1] is very different for variable and constant multiplications. It was also recognized that a constant multiplication can be seen as an accumulation of fully correlated signals whereas a variable multiplication is an accumulation of uncorrelated signals.

2.1.4 Addition of Correlated Signals

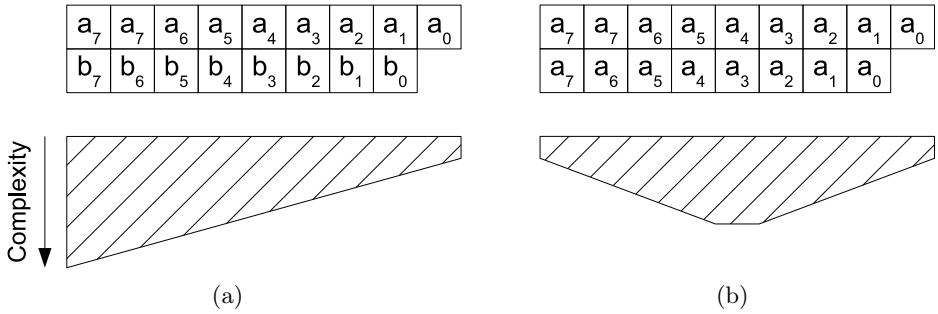


Figure 2.3: Complexity of the Boolean function of the addition of two (a) uncorrelated and (b) correlated signals

Figure 2.3(a) shows that the complexity of the Boolean function for a given output bit for the addition of two uncorrelated signals increases from the Least Significant Bit (LSB) towards the Most Significant Bit (MSB). This originates from the fact that the MSB output of an addition of two uncorrelated signals depends on all input bits of both signals. Figure 2.3(b) shows that for the addition of correlated signals, the complexity of the Boolean function for a given output bit increases from the LSB towards the center bits and then reduces towards the MSB. The lower complexity for the MSB of the correlated signals can be easily explained as it was shown in Equation (2.7) that the sign bit depends only on the sign of the input signal for positive constants.

2.1.5 Adder Types

It is expected that the reader has knowledge about different types of adders for binary signals. Throughout the thesis mainly Carry Propagate Adder (CPA) and Carry Save Adder (CSA), also known as 3-2 counter, are used. Unless otherwise specified, Ripple Carry Adder (RCA) is

[1]The problem of covering the truth table of a function by minimal sum-of-products. The complexity is measured by the number of disjunctive products and literals.

usually assumed for the CPA. For all word level optimizations, delay or power optimized CPA like Carry Look-ahead Adder (CLA) or parallel prefix adders can be used.

2.1.6 Number Representations

In digital signal processing and computer arithmetic, the values of the signals are represented in a digital and therefore discrete format. Several different formats of representation with different properties are available. Two major factors are considered important for the MCM problem. Firstly, based on the required dynamic range and tolerable relative error, either a floating-point (exponent and mantissa) or a fixed-point representation (integer with offset) has to be chosen. Floating-point systems allow a very large dynamic range, but are far more complex to implement than fixed-point systems. As fixed-point satisfies the requirements for many dedicated DSP applications and requires less area, delay and power, fixed-point representation is chosen in this thesis. Secondly, the radix or the unit of least (significant) position of a fixed-point representation has to be decided. Most digital integrated circuits, ASIC and FPGA are implemented in standard CMOS technology. The outputs of their logic gates switch between two voltage levels, which favors radix-2 or binary representation. As signed values are to be represented, the sign and the magnitude information of a number need to be encoded in the selected binary representation. It can be of advantage for the purpose of design automation to consider signed digit representation to allow a direct interpretation of addition and subtraction without the need for sign detection, encoding and decoding.

In the following subsections, the different number representations used throughout the thesis are reviewed.

Fixed Point Representation

As mentioned in Section 2.1.2, the coefficients for the implementation of a fixed coefficient FIR filter have to be quantized to a finite precision. To reduce the circuit size, fixed point representation is chosen with an appropriate dynamic range. Typically, the passband gain of the digital filter is normalized to unity in the design prototype, which leads to all coefficients being smaller than 1 in magnitude. For the fixed point representation, the position of the decimal point is only relevant at the point of interpretation and is immaterial in the calculation as long as its placement in the operands is consistent. For convenience, the decimal point is frequently set at the digit of the smallest precision among all constant coefficients of the filter so that all coefficients can be treated as integers with a constant scaling factor [39]–[41]. For the normally used binary representation the scaling factor is a power-of-2 and can be implemented by shifts. For example:

$$z = a_9a_8a_7a_6a_5.a_4a_3a_2a_1a_0 = a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0 \cdot b^{-5} \tag{2.8}$$

where $a \in 0, 1, \ldots, b-1$ are the digits and b is the base, $b = 2$ for binary positional representation.

Nevertheless, the position of the decimal point was not standardized for the coefficient set representation in different filter design algorithms in the literature. Some authors always present their coefficients as fractions, while others treat them as integers with or without offset. A special case is when the dynamic range is smaller than the offset as shown in the following example:

$$z = 0.0000a_7a_6a_5a_4a_3a_2a_1a_0 = a_7a_6a_5a_4a_3a_2a_1a_0 \cdot b^{-12} \tag{2.9}$$

The first representation may appear to require 12 or even 13 digits to represent the number, but actually only 8 digits are required, as shown in the last expression of the equation.

For simplicity, this thesis treats the coefficients as integers in fixed point representation and the offset will be taken into consideration when necessary.

Binary Representation

Binary representation is straightforward. A number z is represented as an n binary digits as follows:

$$z = \sum_{i=0}^{n-1} a_i \cdot 2^i \tag{2.10}$$

where $a_i \in \{0, 1\}$ and $z \in \{0, 1, \ldots, 2^n - 1\}$.

This basic representation is useful when the arithmetic operations involve only unsigned constants and variables.

Two's Complement

Two's Complement representation is the most commonly used number representation in computer arithmetic and DSP due to the simple implementation of addition and subtraction. For addition, the addends can be added by a CPA while for subtraction, the subtrahend is inverted and a constant 1 is added at its LSB. The constant 1 can be fed to the carry-in of the LSB of the CPA. The same CPA can be configured as either an adder or a subtractor by feeding the control signal to XORs in the input path and to the carry-in port. One drawback of two's complement operations is the sign extension. If two addends of unequal length are to be added, the MSB of the shorter addend has to be signed extended up to the length of the result. This sign extension leads to higher fan-out and loading on the MSB bit and requires stronger drivers at the input. In the simplified MCM block, the MSB of the input has to drive all sign bits of the outputs of positive constant multiplications and many inputs of the internal adders. This leads to an excessive fan-out as described in [42].

The mathematical representation of an integer number z is described as an n-bit two's complement number by:

$$z = -a_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i, \text{ where } a_i \in \{0, 1\}. \tag{2.11}$$

Given a bit width n, the possible values for z range from the minima, -2^{n-1}, to the maxima,

$2^{n-1} - 1$. At the minima, only $a_{n-1} = 1$ and at the maxima, $a_i = 1\ \forall i \in \{0, 1, \ldots, n-2\}$. If one is subtracted from the minima, an underflow occurs and the result is the maxima. For an addition of one to the maxima, the reverse is true. It should be noted that the range of possible values is not symmetrical, as the magnitude of 2^{n-1} occurs only as a negative number. This is due to the fact that an even number of values are available but a symmetrical range from $-m$ to m always has an odd number of values due to 0 which exists only once.

Sign Magnitude and One's Complement

The representation in sign magnitude is, as the name implies, split into a sign (usually the MSB) bit and the magnitude bits. This makes the interpretation of the value rather simple but computation is more complicated than for two's complement. Furthermore, the representation of the value 0 is not unique as the sign bit is irrelevant when the magnitude is 0. The dual representation of zero requires special handling in calculations. Another representation is one's complement, where a negative number is equal to the bitwise inversion of its positive number. One's complement also suffers from the same problem of dual representation of 0 and for additions, the MSB carry has to be added to the sum, which results in additional delay and circuitry.

Offset Representation

To overcome the problem of high MSB loading in addition, the two's complement representation for a given bit width[1] can be modified as in the following 8-bit example:

$$x_4x_4x_4x_4x_3x_2x_1x_0 = 11110000 + 000\bar{x_4}x_3x_2x_1x_0 \tag{2.12}$$

By inverting the MSB, the sign extension is moved from the variable to an error constant. All the error constants can be accumulated into a single error correction constant for multioperand addition. This technique is commonly used in the design of digital multiplier, and was applied to FIR filters in [42]. In multiplication, the accumulated constant is called Correction Vector (CV) while in [42], it is called MSB-Fix.

When this was experimented with sysFIR (see Chapter 3), it was found that the use of the CV leads to erroneous outputs during the first $N - 1$ clock cycles, where N is the number of taps in the filter if no special precautions are taken. The reason as well as the precautions are simple. In parallel multiplier design, the errors introduced by the CV, occur in the same clock cycle as their correction, since the design is fully combinational. However, in the structural adders of the transposed direct form FIR filter, the errors will be propagated though the addition and delay chain before the CV is added at the last tap. The error for c_{N-1} reaches the output only after $N - 1$ clock cycles but the CV is added right from the first clock cycle after reset [43]. As mentioned in [42], this might be tolerable if the FIR filter is employed within a system that has a start up time longer than $N - 1$. If it is not tolerable, precautions

[1]The overflow carry at the leftmost bit position is to be discarded on the right hand side of the equation.

have to be taken during the filter implementation. During the implementation phase, the constant error at each delay element can be calculated by summing up the errors at each SA. If these values are then used to initialize the delay elements at startup, the error vectors are added in place for each time sample during the first $N-1$ samples and all output values are correct [44].

Besides the special handling for the first $N-1$ samples after start or reset, the CV technique also has some implications on the range of values required to be stored within the SA block. The inversion of the MSB of the variable shifts its range from $[-2^{n-1}, \ldots, 2^{n-1}-1]$ to $[0, \ldots, 2^n-1]$, while the correction constant provides the necessary negative offset to shift the range back. The outputs of the MCM block happens to be an exact power of two only in a few occasions, therefore the possible output values do not cover the entire range of values allocated by its bit width. Due to the aforementioned range shift, the range of the possible output values may span from a value higher than zero. In some cases, this result in a situation where the range of all possible output values can fit into a bit width smaller than that suggested by the largest value of the given bit width. This is because the values larger than the highest value of the bit width will be wrapped around zero. The wrap-around is tolerable for the final output bit width, but not for the intermediate sums where the bit width will be expanded in successive adders. The reason for this is that when the range is expanded for positive numbers, the range is extended from the highest value 2^N-1, which shifts up the segment of the wrapped around values and separates it from the contiguous segment of values before the range is expanded. This is illustrated in Figure 2.4.

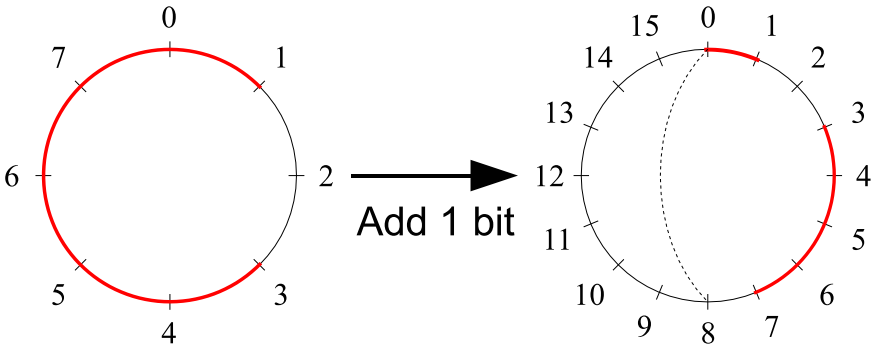


Figure 2.4: Visualization of the problem when a fourth bit is added to a 3-bit number holding the values 3 to 9, where $8 \equiv 0$ and $9 \equiv 1$ are the wrapped around values. By adding the fourth bit, the wrapped around values of 0 and 1 are mapped to $0 \equiv 16$ and $1 \equiv 17$ instead of 8 and 9 of the expanded range

To overcome the problem of the wrap around for subsequent bit width extension, the required bit width has to be calculated from the largest value instead of the width of the range or the offset can be compensated at the cost of an extra adder [44]. For the bit width of the output, two's complement calculation is used because the CV shifts the range back. As no further increment is required, the continuity of range is not broken.

Signed Digit

The Signed Digit (SD) representation, introduced for efficient realization of arithmetic operations in digital circuits in [45], is a positional number system with a symmetrical digit set, i.e., $a = (a_{n-1}, \ldots, a_2, a_1, a_0)$ with $a_i \in \{0, \pm 1, \pm 2, \ldots, \pm(r-1)\}$. Other use of SD to improve the efficiency of constant multiplications can be found in [46]–[48]. SD representation with $r = 2$ is widely used for digital arithmetic and is also referred to as binary SD[1]. A coefficient c can be written in SD form as:

$$c = \sum_i b_i \cdot 2^i \tag{2.13}$$

where $b_i \in \{\bar{1}, 0, 1\}$ with $\bar{1}$ representing -1.

SD representation is not unique, as $[1] = [1\bar{1}]$ and $[\bar{1}] = [\bar{1}1]$. Any $[1]$ or $[\bar{1}]$ in a SD number can be replaced by its equivalent digit pair repeatedly as long as there is an adjacent zero to its left. For example, 1 can be represented as $1\bar{1}\bar{1}\bar{1}\bar{1} = 16 - 15$. If there is no bit width restriction, the leading non-zero digit can be replaced indefinitely [48]. The Minimal Signed Digit (MSD) [46] representation is derived by restricting the SD representations to those that possess the minimum hamming weight, i.e., the minimum number of nonzero digits. Since the bit patterns $[10\bar{1}]$ and $[\bar{1}01]$ have the same nonzero digit count as $[11]$ and $[\bar{1}\bar{1}]$, and correspond to the same integers 3 and -3, respectively, MSD representation may not be unique. By introducing the requirement that $b_i \cdot b_{i-1} = 0,\ \forall\ i \in \{1, 2, \ldots, N\}$, where b_i is the i-th digit of the MSD representation, the only way to represent 3 is $[10\bar{1}]$ and -3 is $[\bar{1}01]$. This subset of the SD representations is unique and is called the Canonical Signed Digit (CSD) [39]. Both CSD and MSD use the same minimum number of non-zero digits and they both require $n + 1$ digits to represent the range of values representable by the binary representation of n bits, as can be seen from the example of 15, which is $[1111]$ in binary and $[1000\bar{1}]$ in CSD and MSD. The reason for this difference is that CSD has a different value range from binary for a given bit width. The range of an n-bit binary number is $[v_x^- = -2^n, v_x^+ = 2^n - 1]$ whereas for an n digit CSD, it is:

$$\left[v_x^- = -\sum_{i=1}^{\lceil n/2 \rceil} 2^{n-2i+1}, v_x^+ = \sum_{i=1}^{\lceil n/2 \rceil} 2^{n-2i+1} \right] \tag{2.14}$$

Basically, maximally every other digit can be nonzero for CSD. This is the reason why the variable u is different for even and odd n. For $n = 1$, the maximal value is 1. The range can be calculated by noting that for every increment of n by one, the number of integers that can be represented in CSD doubles if n is even and is one more than twice if n is odd.

The binary, all distinct MSD and CSD representations of an integer 815 are shown in Table 2.1.

Based on the binary, MSD and CSD representations of 815, the product $y = 815 \cdot x$ can be

[1] Binary SD is referred to as SD in this thesis, unless otherwise specified.

TABLE 2.1: Binary, MSD and CSD Representations of 815

Binary	MSD	CSD
1100101111	$1100110001\bar{1}$	$10\bar{1}010\bar{1}000\bar{1}$
	$11010\bar{1}000\bar{1}$	
	$10\bar{1}00110001\bar{1}$	
	$10\bar{1}010\bar{1}000\bar{1}$	
	$100\bar{1}\bar{1}0\bar{1}000\bar{1}$	

calculated as

$$(x \ll 9) + (x \ll 8) + (x \ll 5) + (x \ll 3) + (x \ll 2) + (x \ll 1) + x,$$
$$(x \ll 9) + (x \ll 8) + (x \ll 6) - (x \ll 4) - x, \text{ and}$$
$$(x \ll 10) - (x \ll 8) + (x \ll 6) - (x \ll 4) - x.$$

It can be observed from these expressions that the implementation based on the unsigned binary representation of constant 815 requires 6 additions whereas those based on MSD and CSD representations require 4 addition/subtraction operations.

Important statistical properties for CSD are presented in many papers, e.g. [39]. In connection with the MCM optimization, they are reviewed in [5]. Any n-bit binary number C can be represented in CSD or MSD form with no more than $S(C) = \lceil \frac{n+1}{2} \rceil$ nonzero digits. The maximum number of nonzero digits is limited to about halve the word length of the binary representation as a consequence of the rule that no two adjacent digits can be nonzero in CSD. Using the limit of nonzero digits, the asymptotical bound of $n/3 + 1/9$ for the average number of nonzero digits for an n-bit CSD number has been derived in [5], [39] and [8]. Therefore CSD, as well as MSD, use on average 33% fewer nonzero digits than the two's complement binary representation. Algorithms for finding the CSD representation for any number can be found in [41], [49], [50]. Efficient hardware implementations for binary to CSD conversion and its reverse conversion will also be proposed and presented in Chapter 7.

Relation of Representation Spaces

CSD and binary are the two most restricted number representations, as both representations are unique. For many integers, like 5 = [101], they are identical. At the other end of the spectrum is SD which possesses indefinitely many forms for the same number. Special representations like Overlapping Digit Pattern (ODP) [51] are not strictly binary, as they have hidden nonzero digits. Between binary and SD are MSD and CSD where MSD is less restrictive as it subsumes CSD and may assume additional forms for certain integers. Figure 2.5 shows the relation of the subspaces for the above signed integer number representations.

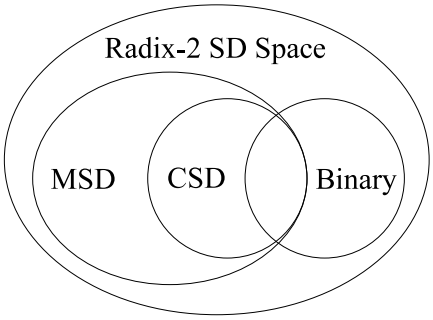


Figure 2.5: Subspaces of Signed Integer Representations

2.1.7 Internal Signal Width Calculation

To minimize the size of the adders and delay elements, which are registers realized by Flip Flops (FFs) in ASIC, knowledge about the internal signals of the FIR filter is required. To use standard adders and subtractors, two's complement representation is chosen. One problem with two's complement representation is that the range of possible values is not symmetrical around zero as mentioned before. If the input signal or any shifted version of it is inverted, the output requires one additional bit to account for the asymmetrical range.

A simplistic approach is to calculate the maximal possible range of the output and uses this to determine the sizes of the adders and registers, but the resources required are overkill.

Let $[v_x^-, v_x^+]$ be the range of the input signal x. For a bit width n_{input} of x and a fully utilized range in two's complement representation, the range is $\left[v_x^- = -2^{n_{input}-1}, v_x^+ = 2^{n_{input}-1} - 1\right]$.

Using the range $[v_x^-, v_x^+]$, the output bit width of the adder is calculated as follows:

$$\begin{gathered} n_{out} = \left\lceil \log_2 \left(\max \left(\sum_{i=0}^{N-1} s_i, \sum_{i=0}^{N-1} t_i \right) \right) \right\rceil + 1, \\ \text{where } s_i = \begin{cases} c_i \cdot v_{input}^+ & \text{if } c_i \geq 0 \\ c_i \cdot v_{input}^- & \text{if } c_i < 0 \end{cases}, \; t_i = \begin{cases} -c_i \cdot v_{input}^- & \text{if } c_i \geq 0 \\ -c_i \cdot v_{input}^+ & \text{if } c_i < 0 \end{cases} \end{gathered} \tag{2.15}$$

If the range $[v_x^-, v_x^+]$ of the input signal x is symmetrical, i.e., $v_x^- = v_x^+$, the output bit width of Equation (2.15) can be simplified to:

$$n_{out} = \left\lceil \log_2 \left(\sum_{i=0}^{N-1} |c_i| \right) \right\rceil + n_{input} \tag{2.16}$$

As the MCM block has only one input signal, all internal signals are highly correlated. Based on the insights of Section 2.1.3, the range of each coefficient multiplier output can be calculated and used for the determination of its bit width. The range of each signal within the MCM block can be calculated by considering the multiplication of the input signal and an odd integer, as multiplication with an even integer will produce at least one trailing zero in the product. Not all integer values represented by the bit width so determined are covered by the range between the minimal and maximal possible values of the output. As a result, the

ranges of all outputs of the MCM block, except those that are directly fed through the input, are under-utilized. The aforementioned calculation is not valid if vertical or oblique common subexpression elimination[1] is used in the MCM block. Information on how to handle such exception is provided in [52].

To calculate the bit widths of the internal signals of the SA block, a different approach is required. The SA block accumulates and delays the signals from the MCM block. As the sequence of input signals needs not be correlated, the values stored in the FFs and those arriving from the output of the MCM block have to be treated as independent. Therefore the output of an adder must have sufficient number of bits to hold the sum of the maxima and the sum of the minima of both inputs. Let c_0 be the coefficient that is closest to the output in the transposed direct form structure and the add-and-delay chain of the SA block starts with c_{N-1}. To calculate the bit width at the m-th coefficient, Equation (2.15) has to modified to

$$\begin{aligned} n_m &= \left\lceil \log_2 \left(\max \left(\sum_{i=m}^{N-1} s_i, \sum_{i=m}^{N-1} t_i \right) \right) \right\rceil + 1, \\ \text{where } s_i &= \begin{cases} c_i \cdot v_{input}^{+} & \text{if } c_i \geq 0 \\ c_i \cdot v_{input}^{-} & \text{if } c_i < 0 \end{cases}, t_i = \begin{cases} -c_i \cdot v_{input}^{-} & \text{if } c_i \geq 0 \\ -c_i \cdot v_{input}^{+} & \text{if } c_i < 0 \end{cases} \end{aligned} \tag{2.17}$$

For an input signal with symmetrical range, i.e., $v_x^- = v_x^+$, Equation (2.17) can be simplified to:

$$n_m = \left\lceil \log_2 \left(\sum_{i=m}^{N-1} |c_i| \right) \right\rceil + n_{input} \tag{2.18}$$

Equations (2.15) and (2.17) are only valid for two's complement representation. If the MSB-Fix technique is used, (2.17) has to be modified according to Section 2.1.6 and the bit width is calculated by:

$$\begin{aligned} n_m &= \min \left(\left\lceil \log_2 \left(\sum_{i=m}^{N-1} s_i + 2^{\lceil \log_2(|c_i|) \rceil + n_{input}} \right) \right\rceil + 1, n_{out} \right), \\ \text{where } s_i &= \begin{cases} c_i \cdot v_{input}^{+} & \text{if } c_i \geq 0 \\ c_i \cdot v_{input}^{-} & \text{if } c_i < 0 \end{cases} \end{aligned} \tag{2.19}$$

Also, Equation (2.19) can be simplified for an input signal with symmetrical range, $v_x^- = v_x^+$.

$$n_m = \min \left(\left\lceil \log_2 \left(\sum_{i=m}^{N-1} |c_i| + 2^{\lceil \log_2(|c_i|) \rceil + n_{input}} \right) \right\rceil + 1, n_{out} \right) \tag{2.20}$$

By using the above formulae, it is possible to calculate the minimal bit width requirement for the signals within the whole FIR filter.

The effect of the asymmetrical range, e.g. -128 to 127 for an 8-bit two's complement number,

[1] See Section 2.2 for details

of the input signal is more significant for a small input signal. Since the filter coefficients can have different signs, the ranges of reversed asymmetry[1] are established within the SA block. The asymmetrical range and the reversed asymmetrical ranges cancel and reduce the required bit width. There is no known case by which the output word length was affected by the cancellation, but the positions of range increments are deferred notably. Figure 2.6 shows the calculated bit width for the first half (taps 121 to 56) of the structural adders of Filter Example 1 [28] for different input bit widths. The increments in the bit width are aligned for the bit widths 5, 6, 7 and 1, as 1 has symmetrical bit width. For the input bit widths 2, 3 and 4 the increment happens at later taps (highlighted in red). This is due to the fact that the range asymmetry is relatively large for smaller bit width and therefore the required range is reduced as mentioned. To date the effect was only investigated for two's complement scheme.

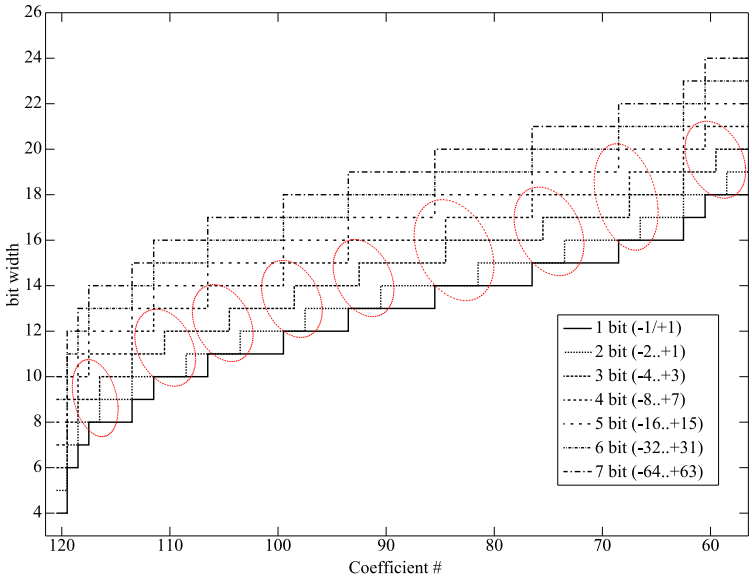


Figure 2.6: Required structural adder bit width for different input bit width. Due to the input range asymmetry the bit width increments are delayed (red circles) for smaller bit width with asymmetrical value range.

2.1.8 Internal Signal Width Calculation for CSA

If the SA block or even the whole filter is implemented using CSA, the range calculation and the handling of the range extension require special attention. If only regular CSAs are used, the partial sum within the SA block has to grow in each step until the maximal signal length is attained. Otherwise errors will be introduced and incorrect samples will be output occasionally [52]. While the sum of the CSA output signals could fit into the bit width, its individual sum and carry signals may not. This problem is also true for the scaling of CSA

[1] the asymmetrical input range is reversed if the coefficient is negative

signals [53].

In [53], two possible overflow scenarios for CSA are described. The first scenario also happens in CPA implementation and is accommodated by the correct calculation of the required bit width. The second scenario of overflow is called the carry overflow, which is characteristic for CSA. It occurs when the carry and sum have opposite signs and requires more bit width than the actual result. The details of carry overflow is given in [53] and its detection condition is given as $c_n \neq c_{n-1}$, where the signals and inputs for the two most significant CSAs are shown in Figure 2.7.

$$\begin{array}{cccc} & & p & u \\ & & q & v \\ & & r & w \\ \hline c_n & c_{n-1} & & \\ & & s_n & s_{n-1} \end{array}$$

Figure 2.7: CSA Overflow Symbols

The outputs c_{n-1} and s_n are corrected by [53]:

$$\tilde{c}_{n-1} = c_n \tag{2.21}$$

and

$$\tilde{s}_n = s_n \oplus (c_n \oplus c_{n-1}) \tag{2.22}$$

Using these extra circuits, the bit width can be calculated as in Section 2.1.7. However, the implementation of Equations (2.21) and (2.22) incurs an additional FA delay, which is not desirable in high speed CSA implementation. This is overcome in [53] by a customized FA, which may not be possible when if the CSA is implemented with standard cell libraries.

During the investigation of the CSA based SA block, it was noted that error occurs on c_n at the positions where there is no bit width increment. As mentioned earlier, the error can be corrected by the suggested implementations of [53], but the solutions are not ideal due to the extra delay or the requirement of special cells not available in typical standard cell libraries. For the SA block the bit width can be designed such that the signal can fit within, based on this Equation (2.22) can be simplified. This is because Equation (2.22) covers two error cases of that only occur when there is a bit width overflow (the first overflow scenario of [53]).

Figure 2.8 shows the cases where a value overflow occurs. In Figure 2.8(a) three positive numbers are added while two have already attained at least half the maximal output value and hence an incorrect output will be produced. In Figure 2.8(b), three negative numbers are added while two have at least half the maximal negative output value and an incorrect result will again be produced.

Equation (2.22) contains two XORs, which are associative and therefore can be reordered. Both c_n and s_n originated in the same input variables p, q, r and occur both in Equation (2.22) and can be simplified. Further the incorrect values shown in Figure 2.8 can be used as don't

	0	1
	0	1
	0	d
0	1	
	0	d

(a)

	1	0
	1	0
	1	d
1	0	
	1	d

(b)

Figure 2.8: Examples of adding (a) positive and (b) negative numbers by CSA that result in a value overflow. d = don't care

cares leading to the simplification of Equation (2.22) to:

$$\tilde{s}_n = p \cdot q \cdot r + \overline{c_{n-1}} \cdot (p + q + r) \tag{2.23}$$

while the modified carry can be expressed with the same variables:

$$\tilde{c}_{n-1} = p \cdot q + q \cdot r + r \cdot p \tag{2.24}$$

Using Equations (2.23) and (2.24), the overflow error can be corrected with an extra AND and OR gate in the critical path, which has a shorter delay than an additional XOR for the sum output. Simulation results showed that the function of the proposed processing element is correct and does not have significant impact on the minimal achievable delay.

2.2 Literature Review

For the multiplication of a single input variable by more than one constant, a set of CSs or intermediate results (also called fundamentals[1]) can be identified and shared among all constants to reduce its cost of implementation. The objective of the MCM scheme is to replace all the multipliers by an optimized shift-and-add/subtract network, where the digital circuit for each subexpression produces an intermediate result to be shared by different multiplications to minimize the overall implementation cost. Two cost metrics frequently used to evaluate the quality of a MCM solution are the number of Logic Operators (LOs) and the Logic Depth (LD). Since a subtractor could be implemented with the similar architecture and cost as an adder, each of the adders and subtractors is counted as one LO, while fixed shifts can be implemented without cost as they are just a matter of wiring. The maximum number of LOs in the critical path is called LD. The problem of minimizing the number of additions/subtractions for the implementation of fixed FIR filter has attracted the attention of researchers since the early eighties [54]. Since deciding if the shift-and-add network has the minimum number of adders for a set of constant multiplications is NP-complete [38], [55], one cannot guarantee the optimality of the solution found in reasonable time for large multiplication blocks. Nevertheless, some efforts have been made to develop optimization schemes and optimal algorithms, which could be used for shift-and-add implementation for a small set of constants in MCM blocks. These algorithms will be reviewed in a subsection separately from the many heuristic algorithms suggested during the three decades of research for the near optimal solutions to this problem for different applications implemented on programmable machines as well as in dedicated hardware. All these approaches can be grouped in many different ways, and are better generalized into two main categories: (i) Adder-Graph (AG) or Graph Dependent (GD) and (ii) Common Subexpression Elimination (CSE).

Based on the analysis of 200 industrial examples taken mainly from DSP, communication, graphics and control applications, it is shown in [1], [56] that more than 60% of them have more than 20% of operations that are multiplications with constants. Constant multiplication optimization can also be applied readily to more generalized applications involving matrix multiplication and inner-product computation, where one of the matrices or vectors is constant.

The focus of the literature review is mainly on the dedicated hardware implementation of MCM, but similar techniques are also used in software compilers, which is known as (weak) strength reduction [57], where expensive operations (multiplications) are replaced with less expensive (shifts, additions, and subtractions) operations.

[1]The term 'fundamental' is widely used in adder-graph approaches. It refers to a multiple of the input value that is used as an intermediate result in the shift-and-add network of the multiplication block. Note that a fundamental is not necessarily a subexpression. But the term subexpression, which is used in the CSE algorithms can be said to be a fundamental. These terms, however, could be used synonymously in the context of the algorithms.

2.2.1 Representation of Constants

The optimization of single or multiple constant multiplication depends not only on the constants itself, but also on how they are represented and whether or not to fix their representation at design time. An important parameter of optimization is the critical path in order to operate at a high clock frequency. As a CSD/MSD number contains the minimal number of nonzero digits, no further reduction is possible for the straightforward implementation of constant multiplier based on a shift-and-add/subtract network. The optimization of the resulting shift-and-add network can only reduce the implementation cost, but not the nonzero digit count. Therefore the LD of the resulting balanced adder tree will be minimal, as other representations have more or hidden nonzero digits. The minimum possible LD can be derived as

$$LD_{min} = \max_i \left(\lceil \log_2 \left(S(C_i) \right) \rceil \right) \tag{2.25}$$

where $S(C_i)$ denotes the minimum number of nonzero digits for the constant C_i in CSD or MSD representation.

Using the minimal number of nonzero digits of CSD for constant representation is advantageous, but in some cases searching among all MSD representations [47] or among all SD representations bounded by a maximum number of additional nonzero digits [48] can produce better solutions. Thong *et al.* introduced the Overlapping Digit Pattern (ODP) in [51]. The ODP are patterns which represent possible additions arising from the overlapping patterns that are not possible to be represented in SD. This concept was extended in [58] to cover more possible hidden patterns. The use of the overlapping and hidden patterns leads to more optimal solutions in several cases.

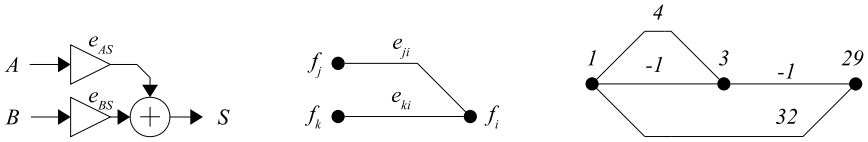


Figure 2.9: (a) Arithmetic implementation of a shift-and-add operation, (b) its corresponding graph representation and (c) a numerical example

Another possibility, which is suggested by several algorithms in the literature, is to reduce the number of adders for constant multiplication without fixing the number representation for the constant. The resulting shift-and-add network for a constant is represented as a directed acyclic graph. A single shift-and-add operation is shown in Figure 2.9(a) where e_{AS} and e_{BS} denote the power-of-two multipliers (left shifts) of the variables A and B, respectively. The graph can be simplified to Figure 2.9(b) by assuming a left-to-right signal flow direction. Each vertex, f, represents an integer multiple of the input variable. The common input variable is implicitly defined and the output of the vertex is annotated only by a constant integer. Each edge from vertex f_a to vertex f_b, e_{ab}, of the graph is assigned a value of $\pm 2^m$, where m is

a non-negative integer indicating the number of bits to be left shifted. The sign of the edge value determines if the output of the left vertex is to be added or subtracted at the vertex to its right. This operation can be expressed as:

$$f_i = e_{ji} \cdot f_j + e_{ki} \cdot f_k \tag{2.26}$$

For a complete shift-and-add network representation, the leftmost vertex with an in-degree of zero represents the input variable and is always annotated by a fixed integer value of 1. Other vertices have an in-degree of two and each can be viewed as a CPA. The value of any vertex can be taken as an external output, irrespective of its number of output edges, which denote the fan-out of the adder it represented.

An example of a simple shift-and-add network is shown in Figure 2.9(c). Let x be the input variable, the leftmost vertex has an output of $1x$. It has three outgoing edges. The two output edges with values 4 and -1 feed the middle vertex. Therefore, the middle vertex has an output value of $1x \cdot 4 + 1x \cdot (-1) = 3x$, which is equivalent to $x \ll 2 - x$. The middle vertex has an outgoing edge to the last vertex with edge value -1. The other input edge to the last vertex comes from the output edge of the first vertex with value 32. Therefore, the output value of the right vertex is $29x$ as $3x \cdot (-1) + 1x \cdot 32 = 29x$.

In hardware implementation, fixed shift is considered free since it can be realized by routing of signals during circuit layout. Furthermore all even numbers have at least one 0 at the LSB and can therefore be divided (right shifted) until they are odd. By reducing the output value of each vertex until it is odd, a minimized graph of a shift-and-add network contains only vertices with unique odd values. Such odd values are commonly referred to as odd fundamentals, as it is a fundamental value in the composition of a minimal shift-and-add network. The graph in Figure 2.9(c) only contains vertices with odd fundamentals and it represents the shift-and-add network for the multiplication of a variable by 29, 58 or any even constant given by $29 \cdot 2^n$, where the integer $n \geq 0$. The LO cost to implement an adder graph is equal to the number of vertices excluding the input vertex. Therefore the graph in Figure 2.9(c) has a cost of 2. Based on the LO cost, a fundamental that requires only one vertex to implement is called a cost-1 fundamental. Examples for such fundamentals are all constants with values $C = 2^n \pm 1$.

There exist many different graphs for a single or multiple constant multiplication. The manipulations and approaches to find the graph with the lowest cost will be discussed further along with the respective algorithms and the search for the set of odd fundamentals for a FIR filter is shown in Table 2.2 on page 34.

In [19], the operation of Equation (2.26) is called '$\mathcal{A}$-operation' and is expressed differently with all parameters included in the formula. The parameters of the '$\mathcal{A}$-operation' are determined by the optimization algorithm.

$$\mathcal{A}_p(u, v) = \left| 2^{l_1} u + (-1)^s \cdot 2^{l_2} v \right| \cdot 2^{-r} \tag{2.27}$$

where $p = |f_i|$ is the output value, $u = f_j$ and $v = f_k$ are the input vertices, $l_1 = \log_2(|e_{ji}|)$ and $l_2 = \log_2(|e_{ki}|)$ are the left shifts of the two inputs, s decides if the operation is an addition or

subtratcion, and r is the number of right shifts.

2.2.2 Single Constant Multiplication

Several techniques have been suggested in the literature to simplify the shift-and-add network in Single Constant Multiplication (SCM) circuits. The main emphasis has been on minimizing the LO count, and in a few cases, however, LD is also given some priority consideration. Although it is difficult to categorize these techniques into exclusively distinct classes, there are three different design approaches, such as (i) direct simplification from a given number representation (ii) simplification by redundant SD representation and (iii) simplification by adder-graph.

Direct Simplification from a Given Number Representation

Bernstein [59] has some earliest contributions on single constant multiplication. He studied SCM from the programming perspective and implemented constant multiplication using addition, subtraction, and shift-operation. Apart from unsigned binary implementations, Bernstein has used the fact that factorization of a number can reduce the number of additions and subtractions involved in a constant multiplication. A simple example of that is the multiplication of a variable with $119 = [1110111]_{BIN}$. Having six nonzero digits in binary, five additions are required, but only three adders are required if it is factored into $7 \times 17 = [111]_{BIN} \times [10001]_{BIN}$. If subtraction is allowable, the cost is further reduced to only one adder and one subtractor as $7 = [100\bar{1}]$. Factoring is advantageous if the factor is of the form $2^c \pm 1$, as such factor can be realized with only an adder or a subtractor. Other forms of factoring could lead to more adders than the direct binary implementation. Lefèvre [60] analyzed the algorithm of Bernstein and proposed an algorithm for simplifying the constant multiplication in a general purpose DSP. His algorithm searches for identical bit patterns called Common Subexpressions (CSs) in the radix-4 Booth encoded constant, which is often equal to its CSD representation. To improve the chance of finding CSs, $[10\bar{1}]$ can be replaced with $[011]$, and similarly $[\bar{1}01]$ can be replaced with $[0\bar{1}\bar{1}]$ without increasing the number of nonzero digits [60], which is equal to using MSD instead of CSD. An example is $105 = [10\bar{1}01001]_{CSD}$, which does not contain a CS. Replacing the single $[\bar{1}01]$ with $[0\bar{1}\bar{1}]$ results in a MSD representation, $[100\bar{1}\bar{1}001]$ with the same minimum number of nonzero digits but contains the subexpression $[100\bar{1}]$ and its negated form $[\bar{1}001]$. This replacement results in a reduction of the additions from 3 to 2. Similarly, the subexpression $[1000\bar{1}]$ and its negative form can also be found. The results obtained by Lefèvre showed that both his original and improved algorithms outperform Bernstein's algorithm [59].

Simplification by Redundant Signed Digit Representation

A new method was introduced in [48] to search for more CSs for elimination from all SD representations with up to a predefined maximum number of nonzero digits above the number of nonzero digits of CSD. However, it is observed that the search for CS in the SD representations does not help in further adder reduction after a certain number of nonzero digits above CSD.

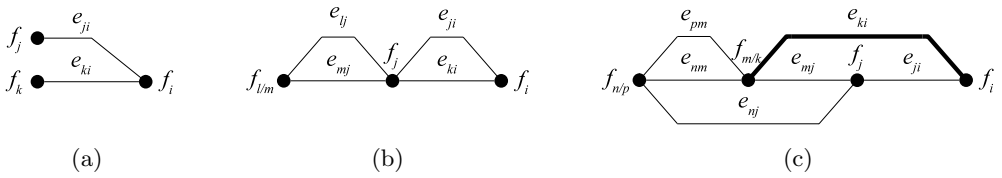
There are no advantages of considering SD representations with more than two extra nonzero digits above CSD for constants of up to 16 bits. An example where it is advantageous to use an additional nonzero digit is $363 = [\bar{1}0100\bar{1}0\bar{1}0\bar{1}]_{CSD}$, which has 5 nonzero digits and no recurrent subexpression and requires four LOs. With one additional non-zero digit, the SD representation $[10\bar{1}0\bar{1}10\bar{1}0\bar{1}]$ contains the subexpression $[10\bar{1}0\bar{1}]$ twice and the LO count is reduced to three, two for the subexpression and one for adding the subexpression to its shifted version.

[61] introduces the $H(k)$ algorithm. This algorithm applies the Hartley algorithm [5] to all SD representations of a constant with up to k more nonzero digits than the CSD representation, and selects the one that requires the fewest adders. The Hartley algorithm was originally targeted for multiple constant multiplication, which will be described in Section 2.2.3 on page 33. Besides the visible redundant digit patterns, as described before, there are redundant patterns that remain obscure in any representation of a constant. In [62], one such obscure pattern was noted with no further pursuit and it is called clashing in [62]. An example of clashing is $39 = [10100\bar{1}]_{CSD}$. It can be decomposed into $[100100]_{CSD} + [00010\bar{1}]_{CSD}$ and constructed by $[1001] \ll 2 + [10\bar{1}]$. In this addition, the two 1 digits at the third bit position 'clash' and form a 1 digit at the fourth bit position of 39. In [51], more such obscure patterns are characterized and referred to as Overlapping Digit Patterns (ODPs). The $H(k)$ algorithm was extended to handle ODPs and shown to further reduce the LO count for SCM.

Simplification by Adder-Graph Approach

Even by including several SD representations or ODPs in the search, the CSs that can be found are limited by the representation. Such a limitation can be overcome by a numerical search that is independent of the representation of constant. Under this premise, Dempster and Macleod [29] have applied the primitive operator graph representation introduced by [3] for MCM to formulate a Minimum Adder Graph (MAG) approach, which will be described in Section 2.2.3 on page 33. MAG finds the minimal set of adders to perform SCM with constants of magnitude up to 2^{12}. This gives an improvement of nearly 16% of LO count over the direct CSD implementation. Furthermore, 32 possible graph topologies for cost-1 to cost-4 fundamentals are shown. In order to find the minimum LO for a constant, all these graph topologies have to be considered. [63] extends the optimal method for all constants up to a precision of 19 bits and reduces the number of graph topologies to only 19 by exploiting graph equivalence to directly reduce the computational effort needed to find a SCM solution with the minimum LO. After a classification of graph representations (based on Equation (2.26) ($f_i = e_{ji} \cdot f_j + e_{ki} \cdot f_k$)) into additive ($(f_j \vee f_k = 1) \cap (f_j \neq f_k)$, see Figure 2.10(a)), multiplicative ($f_j = f_k$, see Figure 2.10(b)) and leapfrog graphs, all 34 possible graph topologies for cost-5 fundamentals are presented. A leapfrog graph contains at least one internal vertex which has a child that is also a child of an other distinct child of the internal vertex. In other words, in a leapfrog graph, there exists a vertex f_i, which is a child of two different internal vertices, $f_j \neq 1$ and $f_k \neq 1$, where f_k is equal to one of the parents of f_j, which both are not the input

vertex $((f_j = e_{mj} \cdot f_m + e_{nj} \cdot f_n) \cap (f_k = \{f_m \vee f_n\} \neq 1))$. Figure 2.10(c) shows a leapfrog graph with the leapfrog edge highlighted.



(a) (b) (c)

Figure 2.10: Classification of Adder Graph Topologies, (a) Additive, (b) Multiplicative, (c) Leapfrog Graphs

In [62], previous algorithms [4], [29], [48], [60], [61], [63]–[65] are reviewed, a detailed analysis of all possible graphs is given and an optimal and a heuristic algorithm are proposed. These algorithms performed better than those published before them. The graph topologies are reviewed and informative observations are given. They are used to reduce the number of topologies to be considered when searching for the solution with minimal number of adders. One of these observations is that the internal vertices (additions/subtractions) can be reordered without changing the value of the output vertex.

State of the Art in SCM

The graph-based algorithm of [62] and the subexpression-based algorithm of [51], to the best knowledge of the author, are the best algorithms for single constant multiplication to date. For all integers up to a maximum word length of 19 bits, the exact minimum number of additions for SCM is known [29], [63] and for higher word length, the lower bound of LOs for SCM is shown in [18] to be

$$LO_{min} = \lceil \log_2 (S(C)) \rceil \tag{2.28}$$

This minimal bound can be used to compare the results of heuristic algorithms. For C with $S(C) = 2^i$, the minimal bound is only achievable if the adder structure shown in Figure 2.10(b) is used, which requires C to be a product of only weight-1 fundamentals. A trivial upper bound is $LO_{max} = S(C) - 1$, which is equal to the cost of a straightforward implementation using CSD/MSD representation of C. [66] presents an asymptotic upper bound of the number of adders in a shift-and-add implementation as $\mathrm{O}\left(n/(\log n)^{\alpha}\right)$, where n is the bit width of the constant multiplier and $\alpha < 1$.

In [67] a method for finding optimal SCM solutions was proposed. A heuristically found solution is either proven to be optimal or improved by a Directed Acyclic Graph (DAG)-based exhaustive search of all graphs with one less adder. Based on the experimentation of a large number of random constants, it is concluded that a SCM solution for any integer up to 32 bits can be realized using at most six adders.

2.2.3 Algorithms for Multiple Constant Multiplications

There are many different application scenarios where a single input is multiplied by several constants. Examples are constant matrix multiplication [1], [13], [68], [69], Discrete Cosine Transform (DCT) and Fast Fourier Transform (FFT) [70] and polyphase FIR filters and filter banks [71]. Since FIR filters are very often encountered in various DSP applications, MCM for their implementation is discussed.

MCM for FIR Digital Filter and Basic Considerations

The input-output equations for FIR filters, Equation (2.3), the transposed direct form equation, Equation (2.4), and the respective figure, Figure 2.2 presented in Section 2.2 on page 12 are revisited. Since the input sample $x[n]$ is multiplied by several constants simultaneously, the hardware minimization of the multiplier block can be directly modeled as a MCM problem [1], [3]. The two-operand adder that adds the product of a filter coefficient and its corresponding input sample with the accumulated sum of such products pertaining to the preceding filter coefficients, is referred to as the Structural Adder (SA), and the complete tapped-delay adder unit is called the SA block.

In all AG algorithms and in some CSE algorithms, a solving set S consisting of all output fundamentals and if required, one or more Non-Output Fundamentals (NOFs) is generated by an algorithm. The set of constant multiplicands G are first mapped to a target set of positive and odd fundamentals T. If the multiplier is followed by an adder (a structural adder in case of FIR filters), the sign-inversion pertaining to a negative constant is moved to the adder, such that the adder is changed to a subtractor and the constant is treated as positive. Since every even number has at least one constant '0' at the LSB, they can be divided by two until the result is odd, and shifted back by appropriate wiring. Table 2.2 illustrates the different steps involved in the generation of target set T consisting of only odd and positive fundamentals for a 25 tap linear phase FIR filter taken from Example 1 of [31].

The resulting reduced, odd and positive set of fundamentals represents the minimal number of constant multiplications required to implement the filter. The amount of redundancy across the filter coefficients depends on many factors like the transfer function of the filter, design algorithm, word length and rounding considerations in fixed-point approximation, etc. It appears that the filter-length is not a good indicator of the complexity of the MCM block, but the size of the target fundamental set T and the complexity of its elements are more closely related to the implementation cost.

2.2.4 MCM - The Adder-Graph Approach

The Early Developments

The first AG algorithm introduced by Bull and Horrocks [3] is called Bull-Horrocks (BH) algorithm. It introduced the graphical solution to the MCM problem. To improve the limited search capability of BH algorithm, an improved version called Bull-Horrocks-Modified (BHM)

TABLE 2.2: Illustration of the Generation of Odd and Positive Fundamentals for a 25 Tap Linear Phase FIR filter (Example 1 of [31])

Steps	Description of functions
1.	Place all coefficients into the set G_1. In case of symmetrical filter, place only the first $\lceil \frac{N-1}{2} \rceil$ into the set to obtain: $G_1 = \{1, 3, -1, -8, -7, 10, 20, -1, -40, -34, 56, 184, 246\}$
2.	Take the absolute values of all elements in G_1 to obtain: $G_2 = \{1, 3, 1, 8, 7, 10, 20, 1, 40, 34, 56, 184, 246\}$
3.	Remove the repetitions and all zeros in G_2 to obtain: $G_3 = \{1, 3, 8, 7, 10, 20, 40, 34, 56, 184, 246\}$
4.	Divide all even elements in G_3 by 2 until the results are odd to obtain: $G_4 = \{1, 3, 1, 7, 5, 5, 5, 17, 7, 23, 123\}$
5.	Remove the repetitions in G_4 and sort in ascending order of implementation cost or magnitude to obtain: $T = \{1, 3, 5, 7, 17, 23, 123\}$

algorithm was suggested in [4]. The AG algorithm begins with an initial solved set S (sources) consisting of a single value '1' and a target set T containing all remaining odd fundamentals (sinks). In the next step, it selects some specific elements of T and moves them to S. A generic pseudocode for populating the solving set S is shown in Algorithm 1. The main difference between these algorithms lies in how the heuristic $FindNOF$ subroutine selects additional elements required to implement the shift-and-add network. The use of Equation (2.26) for the adder-graph construction makes these algorithms independent of number representation, as they manipulate directly with the numerical values of the fundamentals. The resulting graph shows the arithmetic relationship between the operators and the timing delay in terms of LO count on the path to output fundamentals.

n-Dimensional Reduced Adder Graph (RAG-n) Algorithm

Dempster *et al.* introduced a systematic approach called n-Dimensional Reduced Adder Graph (RAG-n) algorithm [4], which consists of an optimal stage and a heuristic stage. At first, all possible fundamentals are synthesized in the optimal stage, and whenever necessary, the heuristic stage is invoked thereafter. The optimal stage guarantees the minimal adder cost for fundamentals in the range from 1 to 4096. The heuristic stage uses a new measure called the 'adder distance' to estimate the number of extra adders needed to generate the requested vertex from the existing vertices of the incomplete graph. It selects the vertex with the minimum adder distance and adds them progressively until all the fundamentals are generated. The adder distances are generated by an exhaustive search using the Minimum Adder Graph (MAG) algorithm [29], over all possible graph topologies and stored in a LUT. The utilization rate of the optimal part of RAG-n algorithm has been improved in [72] to achieve lower adder cost and shorter runtime. It is shown in [73] that the BHM algorithm can

Algorithm 1 Pseudocode for a Generic Adder-Graph Algorithm

Inputs: Target set T, Solved set $S = \{1\}$

AdderGraphAlgorithm(S,T)

```
1: while true
2:    (S, T) = Synthesize(S, T)
3:    if T = ∅ then
4:       return (S)
5:    else
6:       r = FindNOF(S, T)
7:       S ← S ∪ {r}
```

Synthesize(R,T) {optimal part}

```
1: if t_k ∈ T can be calculated by elements from R then
2:    R ← R ∪ {t_k}
3:    T ← T \ {t_k}
4:    (R, T) = Synthesize(R, T)
5: return (R, T)
```

FindNOF(R,T) {heuristic part}

```
1: find non-output fundamental r based on heuristic
2: return r
```

be used here as a subroutine to find the maximally sharable graph for a fundamental in an easier way. The major limitation of RAG-n algorithm arises due to the bit width limit of the adder distances stored in the LUT.

Maximum/Cumulative Benefit Heuristic Algorithm

The RAG-n algorithm was considered as a reference algorithm for a long time until [19] introduced the Maximum Benefit Heuristic ($\mathrm{H_{maxb}}$) and the Cumulative Benefit Heuristic ($\mathrm{H_{cub}}$), which are free from the bit width limitation. Besides, to streamline the search process, Voronenko and Püschel have introduced a new notion called $\mathcal{A}$-operation as an operation on fundamentals, given in Equation (2.27) on page 29. An $\mathcal{A}$-operation basically performs a single addition or subtraction, and an arbitrary number of shifts which do not truncate nonzero bits of the fundamental.

Based on the $\mathcal{A}$-operation, the $\mathcal{A}$-distance is defined, which is analogous to the adder distance used in RAG-n. It gives either the exact or a heuristic estimate for the distance used in their algorithms, and produces significantly improved results than the previous heuristics. The $\mathrm{H_{cub}}$ algorithm finds consistently better results than $\mathrm{H_{maxb}}$ and is the best reference algorithm to be competed against. Besides, it is shown that $\mathrm{H_{cub}}$ provides solutions very close to the minimal solution [21]. The Hardware Description Language (HDL) code as well as the soft code of this algorithm are available at the website [74], which could be used to verify the results easily.

Minimization of Logic Depth

The AG algorithms synthesize the fundamentals sequentially and greedily reduce the LO count. As new fundamentals are generated from the existing ones, these algorithms tend to design solutions with high LD and longer critical path. On the other hand, CSE algorithms either guarantee minimal or near minimal LD solutions but they have in general a higher LO count. High LD increases the likelihood to produce and propagate glitches [8], [75]. The Step-Limiting BHM and Step-Limiting RAG-n are thus proposed to reduce the LD [8]. LD is reduced in the C1 algorithm of [75] at the expense of a few additional adders without guaranteeing minimum LD. The latest release of H_{cub} [74] provides the possibility to restrict the maximal allowable LD, but to the best of the author's knowledge, this option in the updated version has not been described or reported in any archived publication so far. Another possibility to speed up the additions of multiple operands for each constant multiplication is by using Carry Save Adders (CSAs) instead of Carry Propagate Adders (CPAs) [76], [77]. Each CSA has only one full adder delay irrespective of the word length of the CSA, which reduces the critical path delay through these adders. However, this speedup is offset by the need for a final CPA to merge the stored carry and sum vectors. Otherwise, two sets of registers are required to store the sum and carry vectors for the transposed direct form filter before they are accumulated to the sum and carry vectors of the next coefficient multiplier in the tap-delay line. More discussion on the use of CPA and CSA can be found in [78].

The Minimum Adder Depth (MAD) algorithms [79], [80], which include an exact and two heuristic methods, guarantee the minimum LD for all fundamentals. The MAD algorithms use LUTs to enumerate all possible combinations to form fundamentals with minimal LD. The results show a higher LO count and lower power dissipation than the unrestricted AG algorithms. The exact algorithm is, however, applicable only up to 40 fundamentals and the LUTs restrict the maximum size for the fundamentals. In [58], the Minimal Logic Depth (MinLD) algorithm is introduced to generate the fundamentals without increasing the LD. The MinLD algorithm does not require LUTs, since it uses CSD representation to check and ensure that the LD is not increased. The LD results are similar to those of the MAD algorithm but MinLD provides a further reduction of power-dissipation.

Illustration and Comparison of AG Approaches

The fundamentals optimized by different AG algorithms, BHM ([4]), C1 ([75]), H_{cub} ([19]) and MinLD ([58]) are shown in Figure 2.11, where the constants 556, 815 and 1132 are reduced to their odd fundamentals 139, 815 and 283, respectively. Since this small set of odd fundamentals does not contain any fundamentals of the form $(2^n \pm 1) \in \mathrm{N}$, the solution cannot be obtained directly from the initial solved set and suitable NOFs need to be sought. The BHM algorithm uses the NOFs $\{3, 17, 51, 57\}$ (Figure 2.11(a)). Algorithm C1 puts all fundamentals $(2^k \pm 1) \in \mathrm{N}$ into the solved set S, and eliminates unused ones before generating the final solution. It uses the NOFs $\{3, 13, 17, 257\}$ (Figure 2.11(b)), where there is one more NOF of the form $(2^k \pm 1) \in \mathrm{N}$. The H_{cub} algorithm requires only three NOFs $\{3, 35, 47\}$ (Figure 2.11(c))

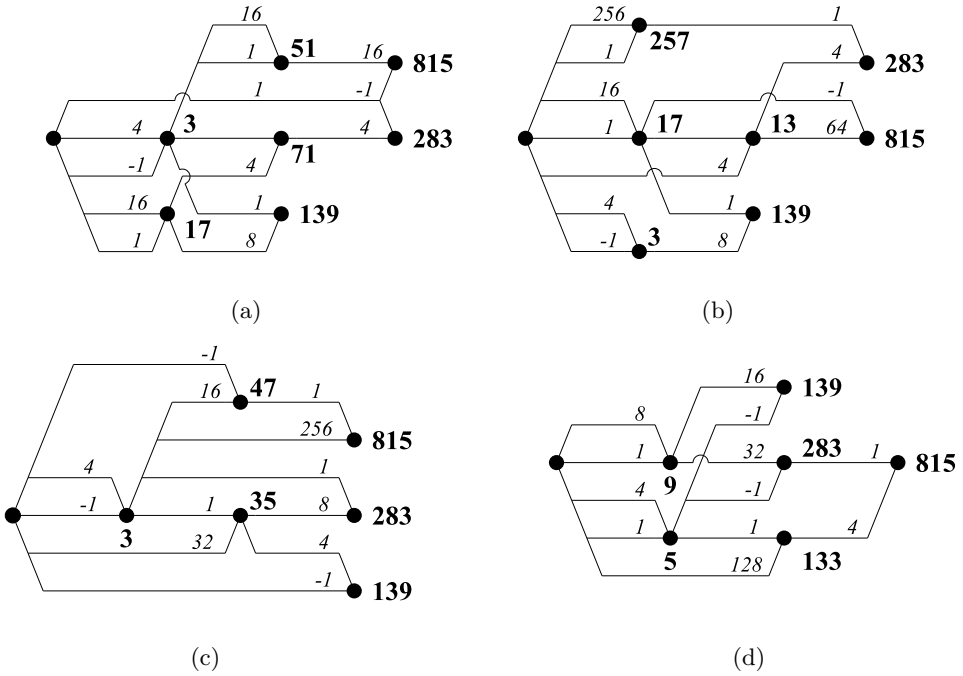


Figure 2.11: The fundamentals 139, 283 and 815 optimized by different adder-graph techniques with highlighted subexpressions. (a) BHM [4] uses 7 LOs and has LDs of {2,3,3}, (b) C1 [75] uses 7 LOs and has LDs of {2,3,3}, (c) H_{cub} [19] uses 6 LOs and has LDs of {3,3,3} and (d) MinLD [58] uses 6 LOs and has the minimal LDs of {2,2,3}

at the cost of a higher LD. The MinLD algorithm also uses only three NOFs $\{5, 9, 133\}$ (Figure 2.11(d)) and constrains all the three output fundamentals to their minimal LD. Figure 2.11 also shows that the output fundamentals can be reused. This is more commonly encountered in larger fundamental sets.

2.2.5 MCM - Common Subexpression Elimination Algorithms

The Common Subexpression Elimination (CSE) algorithms require the constants to be represented in a certain format where common patterns can be sought and eliminated. This is in stark contrast with the graph based algorithms which operate independent of number representation. The challenge for a CSE approach is the search for the most beneficial patterns/subexpressions within the MCM block. The pseudocode for a generic CSE algorithm is shown in Algorithm 2. Before a CSE algorithm can start, a number representation for all elements in target set T is selected and stored in M, which could be a matrix. The algorithm then searches for CS and select one to eliminate from M until no further elimination is possible.

The fundamental idea of CSE can be illustrated by a simple example with $T = \{556, 815\}$. In the CSD representation of $\{556, 815\} = \{[0100101\bar{0}1\bar{0}0], [10\bar{1}010\bar{1}00\bar{1}]\}_{CSD}$ the subexpression $s_1 = [10\bar{1}] \leftrightarrow p_i$ occurs three times. Hence $556 = [010000p_1 0\bar{1}00]_{CSE}$ and $815 = [00p_1 000p_1 00\bar{1}]_{CSE}$, the shift-and-add implementation can be expressed by $p_1 = (1 \ll 2) - 1$,

Algorithm 2 Pseudocode for a Generic CSE Algorithm

Inputs: Target set T, Subexpression set $CS = \emptyset$
CSEAlgorithm(T, CS)
1: $M \leftarrow T$ in a specific number representation {M could be a matrix}
2: **while true**
3: generate subexpression statistic
4: **if** no subexpression exists more than once **then**
5: **return** (M, CS)
6: **else**
7: heuristically select subexpression s_i based on statistic
8: replace suitable occurrences of s_i in M with placeholder p_i
9: update CS with relation $s_i \leftrightarrow p_i$
10: (optional) insert p_i into M

$556 = (1 \ll 8) + (p_1 \ll 5) - (1 \ll 2)$ and $815 = (p_1 \ll 7) + (p_1 \ll 3) - 1$. The selection of the most beneficial subexpression is not simple, as selecting one subexpression for elimination can inhibit the reuse of several other subexpressions. For example, the contention arises when a single constant, e.g., [1 0 $\bar{1}$ 0 0 1], contains more than one subexpression, e.g., [1 0 $\bar{1}$], [$\bar{1}$ 0 0 1] and [1 0 0 0 0 1], but the selection of one subexpression prohibits other available subexpressions from being selected since one or more of their bits overlap with the selected subexpression. Higher order subexpressions or multiple representations of subexpressions add to the intricacy of contention.

Similar to the single constant case, the number representation influences the number of different types of subexpressions that can be found by a CSE algorithm. The canonical representations like CSD or binary representation result in an unique bit matrix for the search for CSs. This is good for the runtime reduction, but limits the possibilities of finding quality solutions with the minimum number of LOs. In the search for better solutions, CSE algorithms have moved from canonical CSD representation to the non-canonical or inconsistent MSD representation. Doing so requires more complicated pattern search and matching. Besides the exploration of non uniqueness in constant representations, [81] generalizes the use of Overlapping Digit Pattern (ODP) in SCM to MCM problems to allow more CSs to be found. Figure 2.12 shows different possibilities to represent numbers for optimizing constant multiplications ordered with increasing size of search space in the direction of S, whereas the unrestricted search space contains all possible '$\mathcal{A}$-operations' that form one or several fundamentals. The CSD and binary number representations are the most restricted form, since both representations are unique and the number of nonzero digits is fixed. The MSD representation is less restrictive. Both subpatterns $3 = [011]$ and $3 = [10\bar{1}]$ are allowable in MSD, which results in additional forms of representation for certain integers. Without bit width restriction, SD representation can have infinitely many forms for a number. When the sign of all non-zero digits are positive for a SD representation, it becomes identical to the binary representation. Therefore the binary representation is a subset of SD representation, but only partially intersects with CSD/MSD space. The intersection contains all the binary representations that have the minimum number of non-zero digits. The ODP space includes the subspace of a

number representation where all bit or digital patterns are visible plus the obscured patterns defined as ODPs. The concept of ODP was extended in [58] to include the hidden non-zero digit patterns. The extension covers additional possibilities of digit patterns that cannot be directly detected from a number representation. These two search spaces can be seen as the expansion of search space from a fixed number representation toward an unrestricted search space that is independent of number representation. Along with the increased search space, the search complexity increases together with the runtime and the chance to find a better solution. Additional constraints like minimal logic depth [58], [79], [80] or the search for only selected patterns, e.g. [23], [82], can reduce the runtime significantly at the expense of compromising the optimality of the solutions.

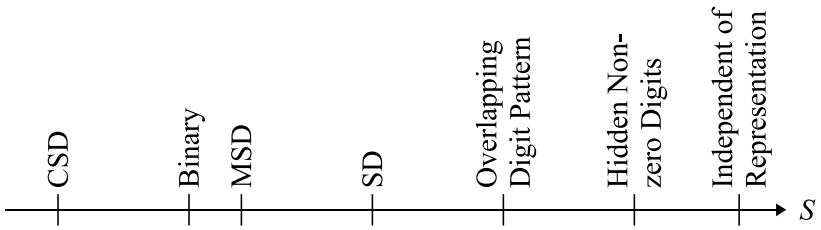


Figure 2.12: Representations in Increasing Order of Complexity and Search Space

Basic CSE Techniques in CSD Representation

The concept of subexpression sharing can be dated back to its use in software context for optimal usage of arithmetic unit in programmable machines [83]. The work of Richard Hartley is one of the first to use the CSE algorithms for constant multiplications in dedicated hardware realization of FIR filters [5], [84]. The algorithm simply records the frequency of all subexpressions, eliminates the CS which occurs most and then recursively eliminates the next most frequently occurred CS. This process also eliminates higher weight (refers to the hamming weight) CSs, which are also called super-subexpressions. A significant advancement on the transformation using CSE was achieved by Potkonjak *et al.* [1]. They have formulated the MCM problem in high level synthesis by considering the multiplications of one variable with several constants at a time and also reduced the number of shifts and additions based on an iterative pairwise matching procedure. Their ITerative Matching (ITM) algorithm finds the maximum number of coincidences between two fundamentals represented in a specific SD representation. It looks for the same nonzero digits within the fundamentals with the least number of shifts from the LSB position. A simple algorithm has been proposed in [85] to iteratively extract CSs with two nonzero digits, which was modified and extended by Pasko *et al.* [7], [86]. Pasko's algorithm differs from Hartley's algorithm in that the higher weight CSs are eliminated first. The higher weight CSs are treated as additional constants such that the CS of lower weights can be eliminated subsequently. The CS selection is based on an exhaustive search for pattern frequencies, and a steepest descent approach is used to select the most frequent patterns. Both global and local binary trees are used to store the frequency

statistics, where a local tree exists for each constant and a global tree is used for the entire set of constants. During pattern elimination, the trees are updated and used for the selection. This technique provides considerable reduction of LOs while maintaining the shortest LDs, although it does not guarantee the minimum LD. The algorithm of [13] is an extension and enhancement of [7], where previous results are generalized by dealing with the optimization problem of the multiplication of a vector with a constant matrix. The Non-Recursive Signed Common Subexpression Elimination (NR-SCSE) algorithm [10] does not search for the CS recursively but allows the CS to have more than two nonzero digits. It guarantees the minimum LD provided that only CSs of weight-two are used. The algorithm of [12] starts with the CSD representation of the fundamentals but uses Equation (2.26) to synthesize the fundamentals and ensures that the LD does not exceed a preset maximum.

Multidirectional Pattern Search and Elimination

Besides eliminating CSs that are contained in a single constant, [5] proposes to search for CSs with their nonzero digits spanning across different coefficients. An array of constants can be stored in a bit matrix, where each row of the matrix is a constant in some positional representation (binary, SD, etc.). A horizontal subexpression has all its bits confined within the same row of the bit matrix, a vertical subexpression that has all its bits confined in the same column of the bit matrix and an oblique subexpression has its bits spanned in different rows and columns of the bit matrix. Figure 2.13 shows three different types of CS that can be found in an array of two integers $556, 815$.

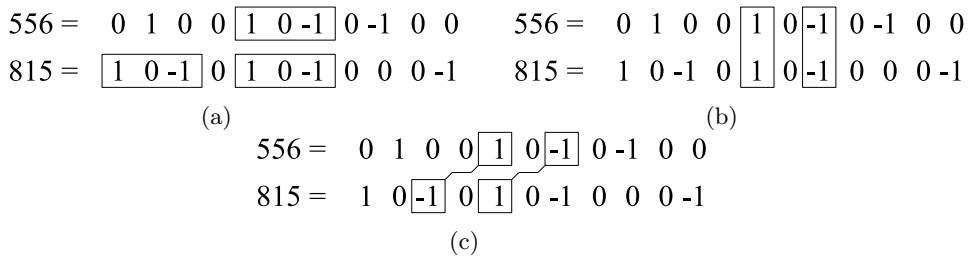


Figure 2.13: The three different types of CS. (a) horizontal, (b) vertical and (c) oblique

Some algorithms that use vertical or oblique CS for CSE are [9], [13], [15], [24], [69]. Vinod *et al.* [11] shows by example that the vertical CSE transcends the horizontal CS if adjacent constants have similar patterns in the MSB portion. However, the vertical CSE alone does not guarantee greater hardware savings over the conventional horizontal CSE methods in some cases, e.g., linear phase filters, where the vertical CSs destroy the symmetry of linear phase filters [87]. Furthermore, the cost of delay elements required for vertical and oblique CSs is often neglected. The use of vertical or oblique CSs also prohibits the reduction to odd fundamentals, as the relative shift of CS bit patterns between constants must be preserved. A more generalized multidirectional search has been suggested in [15] which combines (i) horizontal CSE (ii) vertical CSE (iii) horizontal super-subexpression elimination, and (iv) vertical

super-subexpression elimination. Horizontal and vertical super-subexpressions eliminate the redundant horizontal and vertical CSs, respectively, with identical shifts between them. The design examples of channelizers of multi-standards communication demonstrate the potential of this approach for LO reduction.

CSE in Generalized Radix-2 SD Representation

In [88] the use of CSE with CSD, MSD and binary representations of constants has been compared and it was shown that MSD is superior to CSD, and binary can be advantageous over CSD. These results were refined further in [20]. The CSE algorithms presented in [23] and [89] operate solely on binary representation and are called Binary Subexpression Elimination (BSE) algorithms. The use of binary offers some statistical advantage over CSD [20] at the expense of a somewhat higher LD. Detail comments on it have been discussed in a recent paper [82] and Section 4.1.

Using all possible MSD representations for the constants, the algorithm of [47] synthesizes one fundamental at a time, based on the previously generated fundamentals. The Multi-root Binary Partition Graph (MBPG) algorithm [22] is the first algorithm to use an information theoretic approach to enhance the optimality of CSE. In this algorithm, instead of composing the output fundamentals from non-output fundamentals as in AG algorithms, the constants in SD representation are decomposed into the elementary digits $\{\bar{1}, 0, 1\}$. The entropy and the conditional entropy are calculated from the probability of occurrence of signed digit pairs, and the entropy information is then used to decide which nonzero digits should be assigned first to the children nodes when the parent nodes are decomposed.

The Contention Resolution Algorithm

The Contention Resolution Algorithm (CRA) uses a data structure called 'the admissibility graph' to allow the greedily selected poor CSs to be replaced when there is a contention with a good CS for synthesizing the optimal solution [17], [24]. This allows the algorithm to move out of local minima. The latest version of CRA allows options to set the maximum hamming weight of the CSs and to include different configurations of horizontal, vertical and oblique subexpressions.

556 =	0	1	0	0	1	0	-1	0	-1	0	0
815 =	1	0	-1	0	1	0	-1	0	0	0	-1
1132 =	1	0	0	1	0	0	-1	0	-1	0	0

(a)

2x					1	0	1	
3x					1	0	-1	
2x				1	0	0	1	
2x			1	0	0	0	1	
2x		1	0	0	0	0	-1	
3x	1	0	0	0	0	0	-1	

(b)

Figure 2.14: (a) The constants 556, 815 and 1132 in CSD representation and (b) the subexpressions that occur more than once

Examples and Comparison of MCM Approaches

The CSD representations for the numbers 556, 815 and 1132 are shown Figure 2.14(a). The subexpressions that exist more than once are evaluated and are shown in Figure 2.14(b), where the subexpression statistic is obtained by counting the maximum number of each type of subexpressions without contention. For example, the subexpression $[10\bar{1}]$ appears three times in 815, but at most two of these three subexpressions can be implemented due to the overlap. A super-subexpression[1] $[10\bar{1}01]$ can also be found twice in the same constant, but as the two occurrences overlap in two non-zero digits, it cannot be considered as a CS. There is no super-subexpression that occurs more than once in this example. Super-subexpressions are more commonly found in large array of constants.

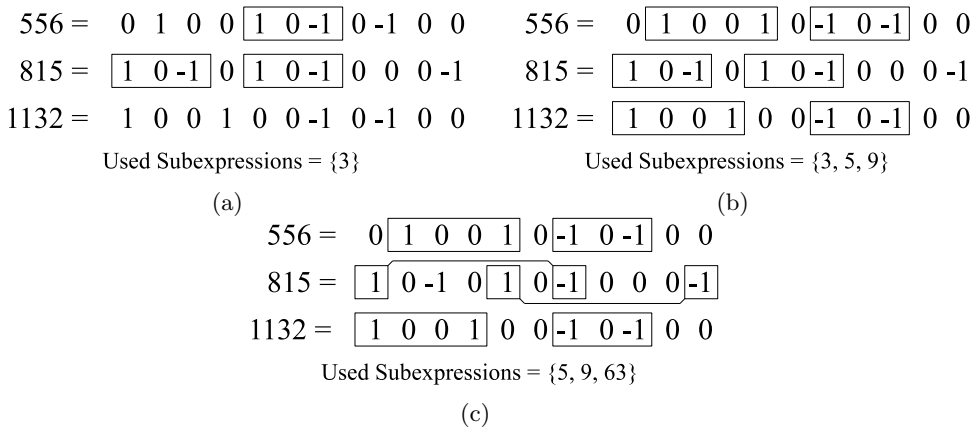


Figure 2.15: The constants 556, 815 and 1132 optimized by different CSE algorithms with highlighted subexpressions. (a) Hartley's [5], Pasko's [7] and NRSCSE [10] only find the subexpression 3 while (b) CRAH-2 [17] and (c) MBPG [22] find two additional subexpressions, but use 3 only twice

The algorithms [5], [7], [10] that make use of such subexpression statistic produce the same result, as they all select the most frequent CS, $[10\bar{1}] = 3$ for elimination first, as shown in Figure 2.15(a). After this subexpression is eliminated from all constants, no subexpression appears more than once. This is a good example of the contention problem mentioned previously. Still the LO count is reduced by two over the direct CSD implementation which requires 10 adders. The newer, and contention aware, algorithm [17] also eliminates the subexpression, $[10\bar{1}]$ in 556 and 815, but later revokes the elimination of $[10\bar{1}]$ in 556 as it finds the contending CSs $[101] = 5$ and $[1001] = 9$ in 556 and 1132. The final subexpression selection is shown in Figure 2.15(b) and it reduces the LO count by three over the direct CSD implementation. The algorithm [22] selects the subexpression $[100000\bar{1}] = 63$ instead of $[10\bar{1}]$. The saving, in terms of LO, is identical and the result is shown in Figure 2.15(c). The solutions in Figure 2.15 clearly show, albeit with an example of only three constants, that the solutions can deviate depending on how an algorithm evaluates the subexpressions, how the candidate CS are selected and their order of elimination.

[1] $[10\bar{1}01]$ has a hamming weight larger than two and is a super-subexpression.

Comparing the results of AG and CSE algorithms in Figure 2.11 and Figure 2.15, it can be seen that the best AG algorithms require one adder less than the best CSE algorithm. By comparing the results of this single example, no conclusion regarding the algorithms can be drawn. In the literature, a brief comparison of CSE and AG algorithms can be found in [90] and [14], but the conclusions are not consistent. While advantages of CSE algorithms for complex problems is highlighted in [90], the advantages for AG algorithms for larger word lengths is inferred in [14]. An important conclusion of [14] is that "even where one algorithm gives the lowest average adder cost, other algorithms may give a lower cost for particular filters."

2.2.6 MCM - Difference Algorithms

The algorithms in [6], [16], [91]–[95] do not attempt to implement the fundamentals directly. Instead, they examine the differences between the fundamentals and order them such that the differences are minimized in order to reduce the implementation cost. The minimization of the adder cost to produce the differences can again be viewed as a MCM problem and the earlier techniques can be applied repeatedly. For the difference algorithm using CSE, the search space changes significantly from the original set of fundamentals, whereas for the graph based difference algorithms, the search space does not change but the direction of search changes.

2.2.7 MCM - Reconfigurable and Time-Multiplexed Multiple Constant Multiplications

The notion of Reconfigurable Multiplier Block (ReMB) for FPGA implementation is introduced in [96] for the time-sharing of resources of a MCM computing block. It is shown that by suitable control of the multiplexers, the macro cells of a FPGA device can be reconfigured to generate more than one partial sum, resulting in a better area utilization of the device [97]. It is further demonstrated that the efficient implementation of a FIR filter using ReMB can reduce the area requirement on FPGA by nearly 20% over the implementation with conventional multipliers. The design methodologies in [96], [97] are developed for Single Input Single Output (SISO) ReMB, where all possible ways of constructing a constant by grouping of the existing partial sums are searched and listed. The partial sums that have common input value and shift amount are grouped into a basic cell of the FPGA. Finally, the outputs of the newly constructed basic cells are combined with the existing partial sums to construct the remaining constants. A systematic synthesis method [98] has also been proposed for both SISO and Single Input Multiple Output (SIMO) ReMB using a different approach.

A Directed Acyclic Graph (DAG) fusion approach has been proposed in [99] for multiplexed multiple constant multiplications. It does not depend on the specialized cells of FPGA, and has been optimized for ASIC mapping. The multiplexed multiple constant multiplication is modeled as a time-multiplexed multiplication of an input variable and one of several selective integer constants. Each integer constant is expressed as a fixed DAG from an existing algorithm [62]. The core idea is to find the best fusion of the DAGs of all constant multipliers. The best solution from all possible DAGs for the individual constant multipliers with the least estimated

area cost is selected as the final DAG for that constant. The design is completed by inserting all the required multiplexers and transforming all nodes in DAG into adders/subtractors. With no constraint on the basic cells of FPGA, the DAG fusion is capable of achieving significant saving in area cost. However, there are two shortcomings of DAG fusion. First of all, the DAG representations for constants are determined and fixed by another algorithm. These DAGs are generated sequentially and individually without a holistic consideration of subexpression sharing in the complete set of constants. Secondly, the search complexity to cover all possible fusion solutions is huge, so that it becomes intractable when the problem size is large. On the other hand, limiting the number of iterations in this search will compromise the quality of the solution since the probability of finding a non-optimal fusion becomes higher in that case.

A Time-multiplexed Data-Flow Graph (TDFG) has been proposed in [100] to model this design problem. The operators and edges in this TDFG have a direct correlation with the adders and shifters of the reconfigurable constant multipliers so that the minimization problem is transformed into a high-level synthesis problem of optimal scheduling and allocation of operators in the TDFG. The mobile operators are scheduled into optimal control steps to maximally time-share as many operators in the same control-step as possible. New aggregate distance and disparity measures based on the distribution of signed digits in the addends were defined to evaluate the opportunity cost of a scheduling decision. Experimental results on application benchmarks and random constant sets showed that the algorithm of [100] is capable of producing more area-time efficient solution over the DAG fusion without exhaustive enumeration and search for independently optimized AG of each constant.

A reconfigurable FIR filter architecture based on BSE method [23] has been proposed in [101]. The proposed FIR filter architecture is capable of operating for different word lengths and filter coefficients without any overhead in the hardware circuitry. It has been shown that the dynamically reconfigurable filters can be efficiently implemented by using CSE algorithms. Design examples showed that the architectures proposed in [101] offer good area and power reduction and speed improvement compared to the existing reconfigurable FIR filter implementations.

Other time-multiplexed and reconfigurable implementations of MCM include the use of carry-save adders on FPGA devices in [102], efficient FPGA realization of multiplications of a variable with a small possible set of constants in [103] and programmable adder graph (PAG) approach for FPGA implementation of shift-and-add network in [104]. The latter does not require prior knowledge of the multiplier value and can be used for adaptive filtering applications.

Most methods reviewed here focus on merging all constants into one single reconfigurable multiplier. Applications like polyphase FIR filter pose another problem, where several MCM blocks are required at different time instances. In [105] and Section 4.3 on page 77, an algorithm for this design problem is presented.

2.2.8 Optimization Schemes and Optimal Algorithms

The methods reviewed for SCM and MCM are mostly heuristic and optimality cannot be guaranteed. An exception is the first part of the adder graph algorithms, as this part of the optimization is optimal. In [106] the first optimal approach for subexpression sharing was proposed by modelling the first-level two-term subexpressions as integer and solving an Integer Linear Programming (ILP) problem. The ILP approach was generalized in [107] to higher-order subexpression. This was achieved by introducing subexpressions with more than two terms. [20] proposes another method. By only allowing subexpressions with fewer nonzero digits in the derivation, the search space is significantly reduced while guaranteeing a valid solution. In [108], the ILP problem is modeled as a hypergraph and it is shown that the solution is equivalent to finding a Steiner tree in the hypergraph. An efficient way to exhaustively search all possible extra subexpressions was proposed in [109]. All these algorithms are optimal in terms of LO but are likely to result in a LD larger than the minimal LD. If the trade-off between LD and LO is acceptable and low power realization is incorporated, approaches like [79], [80] are interesting. The point of optimal trade-off between LD and LO has yet been determined.

2.2.9 Applications

In the literature applications of constant multiplication methods have been discussed. Most of the applications are related to DSP building blocks or calculations in general.

FIR Filter and Filter Bank Implementation

For comparison of MCM algorithms a defacto standard is the implementation of FIR filters in transposed direct from. Low-complexity design of broadband systems was the goal of an early MCM paper [7] and later in [15]. Polyphase FIR filter [71], [105] and filter banks [15], [71] are also possible application fields. The topic of Software Defined Radio (SDR) receiver was presented in [110]. Constant multipliers and multi-standard communication or SDR have incompatible filtering demands, but they can be achieved universally by reconfigurable FIR filters. Example filters were also taken from Digital Advanced Mobile Phone System (D-AMPS) and the Personal Digital Cellular (PDC) channelizers and mainly used by the research group of Vinod [23], [89], [101].

Implementation of Sinusoidal and Other Linear Transforms

A MCM algorithm was presented in [1], which is also applicable to discrete orthogonal transforms like FFT, DCT or Hadamard transform. In [111], [112], the elements of the DCT matix are represented in binary and CSE was used for their optimization. Different constant multiplication schemes were used for linear transforms and matrix multiplication [7], [69], [113]–[115]. Matrix multiplication, or MCM with several concurrent inputs proves to be a complex problem. The search space increases tremendously and CSE were mainly applied to the problem

as the runtime of AG approaches is prohibitive. The Discrete Wavelet Transform (DWT) can also be targeted as its coefficients are constant. In [116], a 2D-DWT is optimized by a CSE method.

2.2.10 Other Applications

In [117], a system-level study of constant multiplications for power and hardware optimization was performed on linear systems like controller in a small guidance system, elliptic wave filter and motor control systems. The use of CSE was shown to reduce the length of the schedule and the switching capacitance, which allowed to save energy by more efficient voltage scaling. The calculation of multiple polynomials is another application of MCM and was illustrated in [1]. General methods for pattern search of constant-vector and matrix-vector multiplications are discussed in [60] and [13]. Several application scenarios for reconfigurable constant multiplication were also identified by [100].

2.2.11 Other Approaches

MCM optimization in general aims to find an optimal solution for the problem in one clock cycle. If several clock cycles are available for the calculation or FPGA is the target platform, other approaches like Distributed Arithemtic (DA) can be used. In DA, the combinational logic elements are traded for memory. In FPGA, every logic slice contains a FF. Hence, distributed arithmetic is often used for the applications that are implemented on FPGA. This thesis focuses on the MCM optimization for low latency and single-cycle high-speed applications. Reader who are interested in DA may refer to [118]–[120] for more details on its applications in DSP and FIR filters.

Chapter 3

sysFIR - A Platform for Design Automation of FIR Filters

The research presented in this thesis and some other FIR filter design works published earlier by the thesis advisor's research group have at large benefited from a tool called "sysFIR" co-developed, revised and maintained by the author. sysFIR has been proven to be very useful in the development of the word level algorithms presented in Chapter 4, the bit-level algorithms presented in Chapter 5 and the structural adder optimizations presented in Chapter 6. This is an appropriate point to highlight the evolution and enhanced features of sysFIR since the inception of the idea in 1993 by the author's thesis advisor.

During the three month internship before his PhD candidature, the author and his project partner, Markus Rähle reconstructed and enhanced the preliminary version of "sysFIR" - a framework of design automation algorithms for ASIC implementation of FIR Filter [121]. The revised tool has greater flexibility due to the modularization and the incorporation of more CSE and AG algorithms for the MCM optimization. A detailed report of the work done during that period can be found in the author's internship report [52].

In the first year of the author's PhD candidature, sysFIR was revisited and several new enhancements were introduced. The fact that the automatic VHDL code generator of sysFIR was structured and written in a modular manner made its further enhancements easy by allowing individual modules to be changed flexibly and new modules to be developed and tested independently before plugging into the main sysFIR framework.

The following sections describe the enhancements and updates to sysFIR, and the benchmark suite [122] that the author and his thesis advisor founded to provide a reference for this research community to benchmark their digital filter design algorithms. This is followed by a list of publications that used the sysFIR design framework to generate VHDL codes for the experiments. The chapter is concluded by the insights gained from the synthesis results of the VHDL files generated by sysFIR.

3.1 Enhancements and Updates for sysFIR

To improve the quality of the results generated by sysFIR, the tool was reviewed and the following enhancements and updates were implemented:

Balanced Adder Tree

The input to the sysFIR is a sysTextFIR file that contains an unified description of the adder topology of FIR filter. In the 2006 implementation of sysFIR, it was intended to arrange the intermediate values presented in sysTextFIR in an order such that a balanced adder tree will be generated when the texts are parsed by sysFIR. Through extensive evaluation, it was found that the implementation was imperfect due to the sysTextFIR file generation process. The creation of a sysTextFIR file was unnecessarily complex. Therefore, an analysis step was introduced. The addends are now sorted in such a way that a balanced adder tree can be constructed easily. The sorting is based on the adder step, but could be enhanced to consider the FA delays.

Updated DC Scripts

To be able to use the current version of Synopsys Design Compiler, it was necessary to adapt the scripts from Solaris to RedHat Linux. During this adaptation, Design Compiler scripts were rebuilt according to recommendations documented by Synopsys User Reference. To ease the compilation using different area and/or delay constraints, the scripts have been revised to allow the constraints to be passed as command line switches. By using a simple hierarchy, different delay optimizations can be generated easily at the expense of some runtime penalty.

Xilinx ISE Scripts

To target for Xilinx FPGA, it was previously necessary to import the VHDL files into a project and then manually compile them in the Xilinx ISE. Now the sysFIR framework can generate a batch script to streamline the entire process from compilation, mapping to placement and routing in a fully automated fashion. This has greatly eased the generation of experimental results for solutions of FIR filters implemented on Xilinx FPGAs.

General Adder Type

The Adder types RCA, CLA and CSA were all described structurally at the logic gate level. A new behavioral adder type “ADD” was added to the adder bank. It generates VHDL code that executes the signed addition without specifying its logic structure. This gives the compilation tools, Design Compiler or Xilinx ISE, the freedom to use specialized adder or adder cells from any cell library chosen for the technology mapping.

MCM Block using CSA

It was made possible for the multipliers in the MCM block to be implemented using simplified CSA trees. Currently it is not possible to pass the Carry and Save signals to the SA block, as this does not give an advantage when compared with methods like [123]. Nevertheless, working on this enhancement initiated the idea that the MCM does

not necessarily have to be implemented as an adder tree and led to the contribution described in Section 5.1.

MCM Block using Multiplier

For comparison, a module was written which implements the constant multiplications within the MCM block using the '*' VHDL operator. This is the simplest way to describe the multiplications in VHDL but it provides no means to optimize the MCM block by CSE or AG algorithms. The behavioral description gives full implementation freedom to the compilation tool such that any available resources or multiplier IP cores can be used. Simulation results show that this leads to the fastest possible implementation when the codes are compiled by Design Compiler. This option also enables Xilinx ISE to use the general purpose multiplier hard IP cores if they are available in the target FPGA. The high-speed implementation by Design Compiler is attributed to the fact that specialized multiplier macros are used to implement the multiplication by a CSA tree and a fast CPA.

MSB-Fix in SA Block

To understand the impact of the use of MSB-Fix technique [42] on area and delay, the SA module was enhanced to generate the code involving the correction vector. As sign extension can be avoided in the MCM block by other simpler means, this was not implemented for the MCM block. Besides the code for the SA block, the verification process was also modified accordingly because the first $N-1$ samples are invalid due to the correction vectors unless the registers are initialized with the correct error vectors. The option to initialize the registers within tha SA block was added in the latest stage of the PhD candidature. The MSB-Fix technique is implemented only for RCA and the behavioral adders, but not for CLA or CSA.

Bugfixes

Since sysFIR was used extensively in this research, numerous tests were carried out to check its robustness. Minor errors were detected in all three phases: the generation of the sysTextFIR files, the generation of the VHDL code and the verification of the VHDL code. They were mainly caused by very special conditions that occur rarely. Todate, all known bugs were fixed.

In addition to the mentioned enhancements and updates, the existing modules and options of sysFIR were used as a basis for capability expansion when new algorithms were developed, which are described later in this thesis. The ability of sysFIR to check the correctness of the input-output transfer function of any FIR filter was proven to be extremely helpful to ensure the implementations of the solutions from the newly developed MCM or SA optimization algorithms function correctly.

3.2 Benchmark Filters

In practice, the coefficient sets of digital filters are not randomly distributed, but MCM optimization algorithms are frequently evaluated with randomly generated integers by assuming an uniform random distribution of coefficients. The non uniformity of the coefficient values was mentioned in [31] and demonstrated for 200 filters with 9-bit quantized coefficients in [124]. In general, it can be verified by the fact that all filters have a $\max(1, 1/x)$ envelope due to the base $\sin(x)/x$ function[1], which means that the coefficients near zero occur more frequently.

In order to provide a set of benchmark filters for a fair evaluation of different CSE algorithms on real world MCM problem, a laborious literature search was performed to find FIR filters with which the coefficient sets are specified in full details. This is because the amplitude response or transfer function specification alone always leaves some room for inconsistent interpretation. As the quantized coefficients are easily subjected to variation depending on the coefficient synthesis algorithm, accurate reproduction of the same experimental results is not guaranteed. Often the specification provided is incomplete or the algorithm used to generate the coefficients is not given. In the literature, 65 filters were found with the fully specified quantized coefficient sets either directly from the publication or by request to the authors of the papers. Long filters are rarely published in full details due to publication page limit. The most prominent filter is a 121 high pass filter, which was originally published in [28]. It was used in many publication on MCM optimization and several improved variants of this filter were published [7], [32], [36]. Table 3.1 lists the data of the filters in this benchmark suite with the references where the data were published. The same data as well as a growing list of filters that were provided by the article authors via email communication can be found on www.firsuite.net [122].

TABLE 3.1: FIR Filter Data Found in the Literature

Filter Name	**Ref.**	**Taps**			**w.l.**	**Binary Nonzeros**			**CSD Nonzeros**		
		#	**Eff.**	**Uq.**	***c***	**#**	**Max**	**Avrg.**	**#**	**Max**	**Avrg.**
CHANG06_3	[125]	3	3	3	11	16	7	5.33	13	5	4.33
DEMPSTER04_A3	[126]	3	3	3	8	11	5	3.67	10	4	3.33
DEMPSTER04_B3	[126]	3	3	3	8	13	6	4.33	11	4	3.67
DEMPSTER04_C3	[126]	3	3	3	12	21	7	7.00	15	5	5.00
DEMPSTER04_D3	[126]	3	3	3	12	19	8	6.33	12	5	4.00
DEMPSTER04_E3	[126]	3	3	3	14	27	10	9.00	18	6	6.00
DEMPSTER04_F3	[126]	3	3	3	14	21	8	7.00	17	6	5.67
GOODMAN77_A3	[127]	3	3	0	0	1	1	n/a	1	1	n/a
GOODMAN77_B3	[127]	3	3	0	1	2	1	n/a	2	1	n/a
DEMPSTER04_G4	[126]	4	4	4	14	29	9	7.25	21	6	5.25

[1] This is true for all filters which approximate an ideal rectangular frequency response, as the ideal rectangular frequency response is a modulated sinc function, $\sin(x)/x$. The modulation is achieved by multiplying it by a sinusoid in time domain, which does not change the envelope.

TABLE 3.1: (continued)

Filter Name	Ref.	Taps			w.l.	Binary Nonzeros			CSD Nonzeros		
		#	Eff.	Uq.	c	#	Max	Avrg.	#	Max	Avrg.
DEMPSTER04_H4	[126]	4	4	4	14	40	12	10.00	22	7	5.50
MARTINEZ02_4	[10]	4	4	4	10	25	8	6.25	18	5	4.50
POTKONJAK96_4	[1]	4	4	4	10	25	8	6.25	18	5	4.50
DEMPSTER04_I5	[126]	5	5	5	12	37	9	7.40	21	5	4.20
DEMPSTER04_J5	[126]	5	5	5	12	27	7	5.40	22	5	4.40
DEMPSTER04_K5	[126]	5	5	5	12	31	8	6.20	24	6	4.80
DEMPSTER04_L5	[126]	5	5	5	12	35	8	7.00	24	5	4.80
JOHANSSON08_A5	[79]	5	5	4	10	18	6	4.50	17	5	4.25
JOHANSSON08_B5	[79]	5	5	5	10	27	7	5.40	20	5	4.00
GOODMAN77_C7	[127]	7	7	1	4	4	4	4.00	4	2	4.00
GOODMAN77_D7	[127]	7	7	2	5	6	3	3.00	6	3	3.00
GOODMAN77_E11	[127]	11	11	3	8	10	4	3.33	10	4	3.33
GOODMAN77_F11	[127]	11	11	4	9	13	5	3.25	13	5	3.25
GOODMAN77_G11	[127]	11	11	3	9	13	5	4.33	11	4	3.67
JAIN91_11	[128]	11	11	4	9	9	2	2.25	9	2	2.25
JANG02_11	[9]	11	11	4	8	10	3	2.50	9	2	2.25
KWENTUS97_11	[129]	11	11	4	8	10	4	2.50	10	4	2.50
CHEN99_15	[130]	15	15	5	13	22	7	4.40	13	3	2.60
GOODMAN77_H15	[127]	15	15	5	10	18	6	3.60	15	4	3.00
KWENTUS97_15	[129]	15	15	4	13	17	6	4.25	16	5	4.00
SHI11_X1	[34]	15	15	4	10	19	8	4.75	12	4	3.00
VINOD03_15	[11]	15	15	5	11	17	5	3.40	16	4	3.20
XU07_15	[33]	15	15	5	12	17	7	3.40	16	6	3.20
SHI11_G1	[34]	16	16	2	7	5	3	2.50	5	3	2.50
GOODMAN77_I19	[127]	19	19	5	13	28	8	5.60	20	5	4.00
SHI11_S1_A	[34]	24	24	4	7	9	3	2.25	9	3	2.25
SHI11_S1_B	[34]	24	24	5	7	13	4	2.60	10	2	2.00
DEMPSTER02_25	[75]	25	25	13	12	73	8	5.62	57	6	4.38
ZAHOSAM89_25	[31]	25	25	7	8	28	6	4.00	22	3	3.14
VINOD03_26A	[11]	26	26	8	6	28	5	3.50	21	3	2.63
VINOD03_26B	[11]	26	26	13	14	86	10	6.62	60	6	4.62
CHEN99_28A	[130]	28	28	10	12	31	4	3.10	29	4	2.90
CHEN99_28B	[130]	28	28	10	11	40	8	4.00	30	3	3.00
CHENYAO01_28A	[131]	28	28	9	11	31	5	3.44	27	4	3.00
CHENYAO01_28B	[131]	28	28	10	11	41	10	4.10	28	3	2.80

TABLE 3.1: (continued)

Filter Name	Ref.	Taps			w.l.	Binary Nonzeros			CSD Nonzeros		
		#	Eff.	Uq.	*c*	#	Max	Avrg.	#	Max	Avrg.
JOHANSSON08_28	[79]	28	28	12	12	49	6	4.08	42	5	3.50
XU07_28	[33]	28	28	9	11	35	6	3.89	31	5	3.44
JHENG04_29A	[132]	29	29	10	10	32	5	3.20	27	4	2.70
JHENG04_29B	[132]	29	29	10	10	33	5	3.30	28	3	2.80
JHENG04_30	[132]	30	30	9	9	32	6	3.56	26	4	2.89
JOHANSSON08_30	[79]	30	30	28	10	151	8	5.39	106	5	3.79
SHI11_Y1	[34]	30	30	6	10	25	6	4.17	19	5	3.17
YLI01_30	[133]	30	30	8	10	31	6	3.88	26	3	3.25
BULL91_32	[3]	32	32	11	7	45	7	4.09	34	4	3.09
SHI11_Y2	[34]	34	34	10	11	40	8	4.00	31	5	3.10
LIM83_36	[28]	36	36	6	8	33	4	5.50	28	2	4.67
LIM83A_36	[30]	36	36	6	8	33	5	5.50	27	2	4.50
LIM83_37	[28]	37	37	6	8	35	7	5.83	28	2	4.67
YEUNG04_40	[110]	40	40	20	19	121	13	6.05	89	7	4.45
KWENTUS97_47	[129]	47	47	13	15	78	12	6.00	48	4	3.69
ROSA04_49	[134]	49	49	12	9	47	6	3.92	42	4	3.50
SHI11_A	[34]	59	59	14	10	58	8	4.14	47	5	3.36
SAMUELI89_60	[31]	60	60	27	13	118	8	4.37	87	4	3.22
SHAHEIN11_S2	[135]	60	60	19	11	68	6	3.58	57	4	3.00
LIM83_63	[28]	63	63	23	10	104	7	4.52	79	4	3.43
SHI11_L2	[34]	63	63	17	10	70	7	4.12	56	5	3.29
YOSHINO90_64	[136]	64	64	26	13	109	7	4.19	86	4	3.31
NIELSEN89_67A	[137]	67	63	27	15	147	14	5.44	106	7	3.93
NIELSEN89_67B	[137]	67	65	29	15	153	14	5.28	107	7	3.69
SHAHEIN11_B	[135]	105	105	10	9	33	6	3.30	26	4	2.60
LIM83_121	[28]	121	121	52	14	240	9	4.62	197	7	3.79
LIMAKT08_121	[36]	121	121	43	14	212	9	4.93	180	6	4.19
LIMPASKO99_121	[7]	121	121	52	14	236	10	4.54	192	6	3.69
LIMYU07_121	[32]	121	121	44	14	252	11	5.73	186	6	4.23

The data from Table 3.1 and [122] were frequently but selectively used by researchers in this field. The filter data was compiled in a MatLab m-file for the convenience of running the coefficient sets in batch by sysFIR. The MatLab m-file was frequently used after it has been established. Based on the data from the m-file a collaboration website [122] was set up and researchers in this research area were encouraged to submit their filter data to the website[1].

[1]The permission to use and publish the data from the filters listed in the Table 3.1 had been granted by the

Since the setup and launch of this website, a few filters were submitted to it and a few papers cite [122] as the reference for the filters used in their experimentation.

Table 3.1 and [122] provide not only the number of filter taps, but also the effective taps (Eff.). During the filter design process, the coefficients are often specified as real numbers and then quantized to a given word length (w.l. in Table 3.1). If the coefficient word length is not chosen carefully, the outermost coefficients can be quantized to 0 and the effective length of the filter can be reduced. Most filters in Table 3.1 are symmetrical and therefore the maximum number of unique (Uq.) taps is limited by $\lceil N/2 \rceil$. The number of unique coefficients can be computed by taking the absolute values of the coefficients, repetitively dividing the even coefficients by two till they are odd and then counting the number of distinct fundamentals remaining in the final set of odd coefficients. These odd fundamentals were employed first in [4] and later in many AG algorithms. They are in essence the number of distinct multipliers that have to be physically implemented since a multiplication by any power of 2 is realized by a fixed shift which can be hardwired directly at no logic cost. The number of unique coefficients is a more indicative measure of the complexity of MCM block than the overall filter length. The exception is when vertical and/or oblique CSs [9], [11], [13], [24], [138] are considered. For example, the filters AKSOY07_D200 and MASKELL07_441 can be considered as having comparable complexity for the MCM block but the complexity of realizing the SA block of the latter is more than double.

The nonzero digit counts of Binary and CSD can indicate to some extent the complexity of the multipliers. The total count (#) can be used to derive the minimal number of adders required to implement the filter, including the structural adders. On the other hand, the maximum nonzero count (Max) can be used to derive the minimal possible logic depth as $LD_{min} = \max_i \lceil \log_2 (S(C_i)) \rceil$. Lastly the average nonzero count per unique tap gives an indication on the average complexity of a single multiplier. It can be easily used to analyze the binary and CSD based implementations. In some cases, e.g., CHEN99_28A, CSD shows only little or no reduction in complexity, but significant reduction was reported in DEMPSTER04_H4. When comparing the binary and CSD data, it has to be done with caution as some filter coefficients were originally designed in the CSD space, which might have very high number of nonzero digits for binary, for example, $[100000\bar{1}]_{CSD} = [0111111]_{BIN}$.

IEEE Intellectual Property Rights Office.

3.3 sysFIR Utilization in Publications

The research group led by my thesis advisor, for which sysFIR was initiated and developed, used the tool for a number of projects and the results are reported in the following list of publications:

[139] J. Chen and C. H. Chang, "A tool for automated synthesis and optimization of integrated FIR filters," in *IET Workshop Microelectron. and Embedded Syst.*, Singapore, Jan. 2007.

[22] C. H. Chang, J. Chen, and A. P. Vinod, "Information theoretic approach to complexity reduction of FIR filter design," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 8, pp. 2310–2321, Sep. 2008.

[140] J. Chen and C. H. Chang, "High-level synthesis algorithm for the design of reconfigurable constant multiplier," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, submitted in Jan. 2009. Revised manuscript re-submitted in May 2009.

[141] J. Chen, C. H. Chang, and H. Qian, "New power index model for switching power analysis from adder graph of FIR filter," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Taipei, Taiwan, 24-27 May 2009, pp. 2197–2200.

[100] J. Chen, C. H. Chang, and C. C. Jong, "Time-multiplexed data flow graph for the design of configurable multiplier block," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Taipei, Taiwan, 24-27 May 2009, pp. 1145–1148.

[142]: M. Faust and C. H. Chang, "Optimization of structural adders in fixed coefficient transposed direct form FIR filters," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Taipei, Taiwan, 24-27 May 2009, pp. 2185–2188.

[58]: M. Faust and C. H. Chang, "Minimal logic depth adder tree optimization for multiple constant multiplication," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Paris, France, 30 May - 2 Jun. 2010, pp. 457–460.

[143]: M. Faust and C. H. Chang, "Reduction of partial product matrix for high-speed single or multiple constant multiplication," in *Proc. Asia Pacific Conf. on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, Shanghai, China, 22-24 Sep. 2010, pp. 416–420.

[144]: M. Faust and C. H. Chang, "Bit-parallel multiple constant multiplication using look-up tables on FPGA," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Rio de Janeiro, Brasil, 15-18 May 2011, pp. 657–660.

[145]: M. Faust and C. H. Chang, "Low error bit width reduction for structural adders of FIR filters," in *Proc. European Conf. on Circuit Theory and Design (ECCTD)*, Linköping, Sweden, 29-31 Aug. 2011, pp. 713–716.

3.4 Insights from the Compilation of sysFIR VHDL Code

The use of sysFIR to generate VHDL code for synthesis using either Synopsys Design Compiler or Xilinx ISE gave some insights into the complexity of implementing FIR filters in ASIC or FPGA platform. These insights have led to the work on structural adder optimization which is described in Chapter 6 on page 107.

sysFIR made large scale analysis of the implementation[1] of different solutions feasible. It can be used to testify an unanswered question that had been posed on whether the reduction in the number of logic operators (LO) in the MCM block reported in the literature will be translated into commensurable area savings in the synthesized ASIC or FPGA. A simple and bold answer is "NO". The anticipated complexity reduction of up to 50% over a direct CSD based implementation as reported in the literature does not translate into physical area savings upon synthesis. Only some trends like slightly reduced area with increased critical path delay were observed in the solutions from AG algorithms, which generally have lower LO count and higher LD. The cause of the discrepancies cannot be easily explained due to the intricacy of other optimizations introduced by the logic synthesis tool and the cell-library dependent technology mapping process. A conclusive interpretation of this interesting phenomenon cannot be established at this juncture and the synthesis results will be revisited in the near future.

Nonetheless, an important and more concrete remark can be made from the synthesis results. The MCM block contributes only a small portion of the overall silicon area of the filter. The area occupied by the MCM block reduces as the filter length grows. Using 8-bit input bit width, the relative MCM area (relative to the total area) is below 20% for filters of length between 25 and 40 taps and drops to just slightly above 3% for a 441 tap filter optimized merely with area constraint. Analysis showed that this is mainly due to the large bit widths of the structural adders. For each nonzero tap, an adder and a delay unit are required and their bit widths are generally about 2 to 4 times the bit width of the input sample, whereas the adders in the MCM block are usually fewer and necessarily smaller. Under tight delay constraints, the share of the area occupancy by the MCM block increases. This is because the data paths of the MCM block are more closely coupled and the critical path can be more readily optimized by the synthesis tools by trading off the area.

The relatively smaller area occupancy of the MCM block triggered a preliminary and ongoing analysis on the complexity of the accumulation of highly correlated data. The findings to date are presented in Section 2.1.3 on page 13.

Although the calculations presented in Section 2.1.7 on page 22 are relatively straightforward for two's complement representation, they need to be modeled and coded carefully. The mentioned difference on the offset representation was only detected because sysFIR flow contains a functional verification by ModelSim. As sysFIR produces only syntactically correct VHDL description, it is important to have automated functional verification of the VHDL description. In [52] it was mentioned that the use of carry save addition requires larger area than that of

[1] Pre-layout synthesis for the case of ASIC.

two's complement addition. Through discussion with another researcher, special treatment for the MSB for carry and save signal as presented in [53] was suggested. Theoretical analysis showed that this solves the problem mentioned in [52].

3.5 Chapter Summary

In this chapter, the development on the research tool sysFIR and new lessons gained from using sysFIR were described. The process of developing this tool gave many good insights and the modularity of the tool proved to be very useful for the development of new algorithms for the optimization of MCM and SA. As the debugging of VHDL and manual synthesis are both very time consuming tasks, this tool that always creates syntactically correct VHDL code and the scripts to run the synthesis is a big time saver. Lastly, the automatic independent validation of the correct functionality of the VHDL code by ModelSim helped ensuring that the VHDL code for the optimized MCM or SA block is functionally correct.

sysFIR was used throughout the research whenever applicable. The results presented in the next few chapters, more specifically in Section 4.2, Section 5.1, Section 5.2, Section 6.2, Section 6.3 and Section 6.4, were obtained with the aid of sysFIR.

Chapter 4

Word Level MCM Optimization

The literature review and the implementation and use of algorithms in sysFIR described in the two preceding chapters had triggered some ideas and questions. The advantage of Binary as claimed in [23] was dubious. An in-depth analysis of the work reported in [23] was triggered by a review of a conference paper that showed an error in LD for CRA data, which was inherited from [23]. The LD data for CRA presented in [23] were higher than the actual results, making the comparison favorable to the presented BSE algorithm. In the search for the origin that led to this discrepancy, other inconsistencies were also detected in the paper. A comment [82] on the controversy and errors of [23] was published in the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* and is presented in the first section of this chapter as a background discussion in the interest of the development of new algorithms for MCM optimization at the word level.

AG based MCM algorithms produce results with fewer adders than CSE algorithms, but exhibit higher logic depth, which is not desirable for high-speed operation. At the same time, it has a high propensity to induce glitches and results in high dynamic power consumption. The second and third sections of this chapter present two algorithms which restrict the logic depth of each individual constant multiplier to its minimal logic depth before the problem is solved by a AG algorithm. The possibilities of forming a number under a minimal logic depth constraint are first analyzed. The analysis provides a cue to improving the quality of the results of the first algorithm while keeping the optimization time within a reasonable bound. The second algorithm is an adaption to reconfigurable MCM problems, where several MCM blocks are merged into one configurable block to produce different sets of multiplications. This problem poses additional challenges in the search for an optimal combination of multipliers.

4.1 Demystification of Binary Common Subexpression Elimination

The logic depth constrained AG based MCM algorithm for word level optimization is a brain child of the work on sysFIR and the comments [82] on the inconclusive evidences of [23]. The paper [23] presents a Common Subexpression Elimination (CSE) method which used binary number representation as opposed to the commonly used Canonical Signed Digit (CSD) representation. Judgmental arguments were given to justify the merit of binary representation for CSE, but careful scrutiny exposes several weaknesses and inconsistency in the chain of arguments. The following subsections highlights the inexact arguments and corrects the oversight of the results presented in [23].

4.1.1 On Nonzero Bits of Binary and CSD Representations

[5] was cited in [23] to argue that the number of nonzero bits is reduced by 50% for CSD representation compared to normal two's complement form. This is an incorrect interpretation of the results presented in [5]. According to [5], an n-bit two's complement number can be uniquely expressed in CSD format with $(N_{nz})_{CSD}$ nonzero bits, where

$$(N_{nz})_{CSD} \leq \left\lceil \frac{n+1}{2} \right\rceil \tag{4.1}$$

When a CSD number possesses the maximum number of nonzero bits, its binary equivalent may have less than n nonzero bits. Therefore, Equation (4.1) should not be construed as the number of nonzero bits in CSD representation is reduced by 50% compared to the two's complement form. On average, binary has $n/2$ nonzero bits while the expected number of nonzero bits of CSD tends asymptotically to $n/3 + 1/9$ according to (17) of [5]. As $n \to \infty$, the number of nonzero bits of CSD is reduced by 33% over that of binary. For finite n, the reduction is smaller. This was also mentioned in [8].

As zero is the only number with no nonzero bit, the average number of adders required for a single constant multiplier can be approximated by the average number of nonzero bits minus 1. Using this, the adder ratio for CSD/binary is given by:

$$\frac{2}{3} \cdot \frac{n - 8/3}{n - 2} \xrightarrow[n\to\infty]{} \frac{2}{3} \tag{4.2}$$

Although started with an overestimate, the potential to find Common Subexpressions (CSs) for elimination is higher in binary than in CSD form is justifiable from the higher number of nonzero bits. What was omitted by [23] is the binary-based CSE algorithm has to eliminate more subexpressions to account for its significantly higher adder count from Equation (4.2). After the analysis of the frequency of occurrences of Binary Common Subexpressions (BCSs) and CSD-CSs, [23] argued that "the number of nonzero bits in binary representation in the worst case is only two times more than in CSD representation". This statement is, as just shown, true for the average case, but not in the worst case. A simple counter example is $2^n - 1$ which is expressed as "$011\ldots11$" in binary and "$100\ldots0\bar{1}$" in CSD.

An exact and a heuristic algorithms to maximize the sharing of partial terms for binary, CSD and MSD representations were studied in [88]. To support the argument that binary representation is better for CSE in the introduction, [23] claimed that "binary closely follows CSD and MSD representations in most cases, and in some cases, the binary representation offers better results than MSD" from the simulations of [88]. However, careful examination of Table III of [88] reveals that exact binary was in par with exact CSD/MSD in three cases and worse in four cases. Only in a biased comparison between exact binary and heuristic MSD, binary was better in two cases and still worse in two cases. An important conclusion of the study [88] "The other observation is that the solution obtained for the binary representation has similar number of adders/subtractors for small examples, although the depth may increase" was omitted in [23].

4.1.2 On Number of Unpaired Bits in CSE

A linear approximation for N_{LO} was assumed in (5) of [23] as follows:

$$N_{LO} = (\alpha \times N_{nz} - \beta \times N_{cs} + \gamma \times N_{up}) \tag{4.3}$$

and α, β and γ were empirically determined to be 0.2345, 0.6643 and 4.0487, respectively from only four different filter response specifications.

Besides the validity of the assumption of linearly independent weighted sum and the small sample size for the analysis, the definitions of N_{nz} and N_{cs} were ambiguous. In [23], N_{nz} was loosely defined as the number of nonzero bits before the application of CSE techniques. Some CSE methods like [4], considered only the set of odd fundamentals while others considered the entire set of coefficients. N_{cs} was defined as the number of CSs in [23] but literally this can mean either the number of different CSs or the frequency of occurrences of one or more CSs. How N_{cs} was accounted is very important due to the nature of the problem. The frequency of one CS can be affected by the frequency of other CSs in the coefficient set due to their overlapping nonzero bits. N_{cs} is also strongly dependent on the CSE algorithm used. As N_{nz} depends on the number representation and N_{cs} and N_{nz} depend on the CSE algorithm, it is important to know what were the number representation and CSE algorithm used for the determination of α, β and γ. Unfortunately, these crucial information were missing, making one of the key conclusions drawn by this analysis, i.e., 67% average reduction of N_{up} for binary over CSD filter coefficients unconvincing.

The problem of estimating N_{LO} by the independent linear decomposition of Equation (4.3) was conspicuous from the theoretical bounds of N_{LO} studied in [18]. Let $S(C)$ be the number of nonzero bits in a coefficient C and M be the size of the set of odd fundamentals of a filter. Without CSE, the number of adders needed in a direct implementation is given by [18]:

$$N_{LO} = \sum_{i=0}^{M-1} \left(S\left(C_i\right) - 1\right) \tag{4.4}$$

and the lower bound with CSE is given by [18]:

$$N_{LO} = \min_i \{\lceil \log_2 (S(C_i)) \rceil\} + M - 1 \tag{4.5}$$

Equation (4.4) exhibits a linear relation between N_{LO} and N_{nz} but its dependency on M cannot be modeled in Equation (4.3). On the other hand, Equation (4.5) is nonlinear and is also dependent on M. The two corner cases of Equation (4.3) corresponding to (i) no CSE was applied ($N_{cs} = 0$, $N_{up} = N_{nz}$) and (ii) all bits are covered by some CS ($N_{cs} = N_{nz}/2$[1], $N_{up} = 0$) were examined. From Equation (4.3) (or (6) of [23]), $N_{LO} = 4.2832 \times N_{nz}$ for case (i) and $N_{LO} = -0.09765 \times N_{nz}$ for case (ii). These are infeasible as the legitimate range of N_{LO} is $(0, 1] \times N_{nz}$.

The high value of γ was used to show the significance of the unpaired bits and argue that N_{up} is significantly lower for binary to outperform CSD in CSE despite its higher N_{nz}. However, the experimental results of Fig. 4 [23] showed that "the number of CSs occurring is almost the same for binary and CSD". Since binary has higher N_{nz} than and similar N_{cs} as CSD, a higher N_{up} is expected for binary which contradicts their original argument.

The Degree of Sparseness (DoS) was introduced in [23] but the definition given was not precise. No formula was presented for this measure. The simplistic example did not help to clarify the computation of DoS for the general case. The theoretical underpinning to the claimed correlation (or whether it exists) between N_{up} and DoS, remains unclear.

It was concluded from Fig. 5 and Fig. 6 in [23] that "the possibility of finding a BCS is respectively 2.95 and 2.73 times more than a CSD-CS for 16-bit coefficients and for filters with the same wordlength". The conclusion was made based on the frequencies of occurrences of different types of subexpressions, [1 0 0 $\bar{1}$], [1 0 $\bar{1}$], [1 0 0 1], [1 0 1], [1 1 1] and [1 1], in CSD and binary coefficient sets. This comparison was inaccurate as the number of occurrences of each type of subexpressions was accounted independently without considering their contention in CSE. The actual number of adders saved by CSE is far less than that estimated by counting the frequency of each type of subexpressions in isolation. The experimentation did not reflect the true picture of the actual number of eliminable CSs achieved by CSE algorithms. In addition, [1 1] and [1 0 1] are subsumed in [1 1 1] for BCS while none of the above subexpressions can be subsumed in another subexpression for CSD. Independent enumeration of these CSs that cannot be eliminated simultaneously results in an overestimate of the average number of CSs in favor of the binary representation. This explains the discrepancy in the conclusions of [23] and [88] on binary and CSD for CSE where the latter had considered the 'true' CSs in CSE.

4.1.3 On Vertical Common Subexpression Elimination

[23] pointed out two constraints for Vertical Common Subexpression Elimination (VCSE). The first constraint is encountered only in CSD due to the sign difference of the selected Vertical Common Subexpressions (VCSs), but it can be easily alleviated by not selecting such VCSs in CSD. Moreover, VCS in SD representation can be easily utilized to the same extent

[1] Assuming a CS contains 2 nonzero bits

as binary. The sign difference can be simply handled by replacing the VCS with the negative version of it. The second constraint that annihilates the symmetry is caused by the delay difference in VCSE. It affects both binary and MSD/CSD equally. Therefore, there is no conceivable advantage of binary over CSD in VCSE. The binary VCSE algorithm proposed in [23] considered only [1 $1(z^{-1})$]. In general, the use of VCSs breaks the symmetry and both symmetrical coefficients must be implemented. If only one VCS is used in a coefficient, the reduction is directly offset by the additional adder that is required to implement the otherwise symmetrical counterpart. Therefore, at least two VCSs have to be found for every symmetrical coefficient to reduce the LO count. After the formation of VCSs, the summation of the remnant nonzero bits can be made equivalent to that of another coefficient to reduce the LO count, but this is not mentioned in [23]. Otherwise, if all nonzero bits were exhausted for the VCSs, the structural adder can be saved. These potential savings as well as the costs for the delay elements were not considered in [23].

4.1.4 On Logic Depth and Filter Length

In the introduction of [23], it was stated that [8] and [17] keep a minimal value of LD while producing almost identical results as [4]. From Table I of [17], the results of CRA [17] and [4] were at best similar in logic operator (LO) count but deviated in logic depth (LD). The authors of [23] might have overlooked that LD presented in [17] included the structural adder, while most other papers excluded the structural adder from LD.

In the results section of [23], CRA [17], NR-SCSE [10] and Cumulative Benefit Heuristic (Hcub) [19] among others were used for the comparison. The algorithms CRA [17] and NR-SCSE eliminated only CSs of hamming weight two. BSE and a generalized version of CRA in [24] use super-subexpressions (subexpressions with higher hamming weight) to enlarge the search space and increase the potential to reduce the LO count. The formation of super-subexpressions has a potential to increase the LD in both binary- and CSD-based CSE. As the subexpression [1 1 1] can only be produced by an asymmetrical adder tree of LD = 2. Its use in the BSE algorithm for the formation of super-subexpressions is likely to increase the LD beyond the minimal achievable LD. For CSE using only weight-two CS, a balanced adder tree can always be formed to achieve the minimal LD. The minimal LD for a CSD coefficient C_i can be calculated by $\max_i(\lceil \log_2(S(C_i)) \rceil)$, where $S(C_i)$ is the number of nonzero bits of C_i. Therefore, CRA [17] in Tables V, VII and IX of [23] should have the same minimum LD as those of NR-SCSE. The original data from [23] and the correct data for CRA are given in Table 4.1.

The incorrect LD data for CRA from Table VI to Table IX of [23] were also directly excerpted into [89] for comparison with BCSE-4, which is an improved binary subexpression elimination algorithm that uses all possible 4-bit patterns. Futhermore, there are eight cases of BCSE-4 having smaller LD than NR-SCSE in Table 8 of [89]. These are impossible as binary has at least as many nonzero bits as CSD.

From Equation (4.1), when $n = 12$, the maximum number of nonzero bits is 7. Therefore it

TABLE 4.1: Reported [23] and Corrected (in columns marked by '*') CRA Results

Table V of [23]		**Table VII of [23]**								**Table IX of [23]**							
Filter	**LD of CRA**	**Filter Length (N)**	**LD of CRA**							**Filter Length (N)**	**LD of CRA**						
			12 bit		**16 bit**		**20 bit**		**24 bit**		**12 bit**		**16 bit**		**20 bit**		**24 bit**
	[23]	*****		**[23]**		**[23]**		**[23]**				**[23]**		**[23]**		**[23]**		**[23]**	
FIR1	3	2	20	3	2	3	3	4	4	5	4	200	3	2	4	3	4	4	5	4
FIR2	3	2	50	3	2	3	3	4	4	5	4	460	3	2	4	3	4	4	5	4
FIR3	4	3	80	3	2	3	3	4	4	5	4	610	4	2	4	3	4	4	5	4
FIR4	3	3	120	3	2	3	3	5	4	5	4	940	4	2	4	3	4	4	5	4
FIR5	4	3	200	3	2	3	3	5	3	5	4	1180	4	2	4	3	4	4	5	4
			400	3	2	3	3	5	4	5	4									
			800	3	2	3	3	5	4	5	4									

is impossible to have LD of CRA [17] larger than 3 in Table IX for 12-bit coefficients and filter length, $N = 610$, 940 and 1180. Similarly, for all cases of 24-bit coefficients, the LD of CRA is at most 4. With binary representation, BSE algorithm [23] has at least as many nonzero bits as CSD. Therefore, it can at best have the same LD as CRA and NR-SCSE. Thus, the LDs of BSE for the 20-bit coefficients in Table VII for $N = 800$ and in Table IX for $N = 460$ reported in [23] were invalid.

According to [23], the lengths of the FIR filters in Example 3 of [23] were determined by

$$N = \frac{-10 \log_{10} \delta_1 \delta_2 - 13}{14.6 \Delta f} + 1 \tag{4.6}$$

where $\Delta f = (f_s - f_p)/F_{sample} = (\omega_s - \omega_p)/2\pi$, $\delta_1 = 1 - 10^{-r_{passband}/20}$, $\delta_2 = 10^{r_{stopband}/20}$ and $r_{passband}$ and $r_{stopband}$ are expressed in dB.

Table 4.2 shows the discrepancies in the filter lengths excerpted from [23], and those calculated using Equation (4.6) and by MatLab's `firpmord` function [146] for the same D-AMPS specification of sampling frequency $F_{sample} = 34.02$ MHz, passband edge $f_p = 30$ kHz and stopband edge $f_s = 30.5$ kHz (after down sampling by a factor of 350) and peak passband ripple of $r_{passband} = 0.1$ dB. The command used in MatLab was: `[n,fo,ao,w] = firpmord([30 30.5], [1 0], [`δ_1 δ_2`], 34.02e3/350)`, with $\delta_1 \approx 0.01145$ and δ_2 calculated by the above formula. This returns n as the required filter length minus one, fo as the normalized bandedge frequencies, ao as the attenuation and w as the weights for the pass- and stopbands of the filter designed by `firpm`.

TABLE 4.2: D-AMPS Filter Lengths

Stopband attenuation	**[23]**	**From eq. (4.6)**	**[146] (MatLab)**
−24dB	200	247	279
−48dB	460	406	418
−65dB	610	520	516
−85dB	940	653	631
−96dB	1180	726	695

The cause of the discrepancies is unknown, but the length of [23] for the first filter is not able to satisfy the passband ripple and stopband attenuation simultaneously, while the other filters had been overdesigned. This issue was also noted in [15], [22], [24]. The relative merits of different CSE algorithms will not be affected as long as the same coefficient sets were used consistently for their comparison. The concern is that different coefficient sets for the same filter specification will result in different implementation complexity from the same CSE algorithm and different number of structural adders.

As the relative merits are not affected by the filter data, in the absence of the original filter data from [23], the corrected filter lengths for the D-AMPS specification were used to generate the filter coefficient sets with 12, 16, 20 and 24-bit word lengths for five different stopband attenuations. Table 4.3 shows the LO and Table 4.4 the LD costs obtained by the solutions

TABLE 4.3: LO Costs of NR-SCSE, CRA and Hcub for D-AMPS Filters with Corrected Filter Length

PSR	Filter	Lower Bound [18]				NR-SCSE [10]				CRA [17]				Hcub [19]			
(dB)	Length (N)	12b	16b	20b	24b	12b	16b	20b	24b	12b	16b	20b	24b	12b	16b	20b	24b
−24	279	30	96	135	141	33	119	242	355	32	116	236	346	30	96	137	158
−48	418	29	102	187	208	34	130	322	474	31	125	312	466	29	102	188	212
−65	516	27	105	234	256	28	131	371	575	28	124	361	562	27	106	235	259
−85	631	28	102	243	313	31	129	366	647	30	123	358	632	28	102	243	315
−96	695	32	109	271	345	34	133	396	706	33	129	387	693	32	109	272	348

TABLE 4.4: LDs of NR-SCSE, CRA and Hcub for D-AMPS Filters with Corrected Filter Length

PSR	Filter	Minimal LD				NR-SCSE [10]				CRA [17]				Hcub [19]			
(dB)	Length (N)	12b	16b	20b	24b	12b	16b	20b	24b	12b	16b	20b	24b	12b	16b	20b	24b
−24	279	3	3	3	4	3	3	3	4	3	3	3	4	3	3	7	26
−48	418	3	3	3	4	3	3	3	4	3	3	3	4	3	3	4	9
−65	516	3	3	3	4	3	3	3	4	3	3	3	4	3	3	4	9
−85	631	3	3	4	4	3	3	4	4	3	3	4	4	3	3	4	6
−96	695	3	3	3	4	3	3	3	4	3	3	3	4	3	3	4	6

of NR-SCSE [10], CRA [17] and Hcub [19] (Version 14. Jan. 2009 with '-nofs' parameter). The lower LO bound defined in (6) of [18] and the minimal LD for each filter example are also provided for comparison. Although [23] claimed that the Hcub algorithm cannot be applied beyond 200 coefficients, the readily available source code of Hcub [19] can actually handle more than 200 coefficients. Only the online version of Hcub [74] is restricted to 20 coefficients.

Hcub [19] achieves the optimal solution defined by the lower bound of [18] for all 12-bit examples, four 16-bit examples and one 20-bit example. For the remaining cases, the maximum difference between the Hcub result and the lower bound is 4 except for the 24-bit coefficient of the -24 dB stopband attenuation. This exceptional case has a difference of 17. The results indicate that there is very little room for other algorithms to achieve lower LO than Hcub. In Table VI of [23], BSE is better than Hcub in 10 out of 20 cases by 2 to 12 LO including four cases in 12-bit and 16-bit examples. Even with the benefit of doubt given to the difference in filter coefficients due to the abovementioned filter length discrepancies, the winning margin of BSE over Hcub for the LO data reported in [23] is unbelievable, especially when most of these cases are found from the higher order filters. The LDs for NR-SCSE and CRA in Table 4.4 are equal to the lower bound as expected while the data for Hcub is comparable to the CSD based algorithms for 12- and 16-bit coefficients and higher for 20- and 24-bit coefficients. Furthermore, the LD of Hcub decreases with increasing filter length. However, in Table VII of [23], Hcub was shown to have higher LD for all coefficient word lengths and increasing LD with increasing filter length. The coefficient sets of the filters in [23] and the ones used in Tables 4.3 and 4.4 are available in [122] for verification.

4.2 Minimal Logic Depth Adder Tree Optimization

While it is important to minimize the LO, aggressive minimization using super-subexpression as discussed in the previous section tends to compromise LD. LD has a direct impact on the critical path delay and an indirect influence on the power consumption. Glitch Path Count (GPC) [147] and Glitch Path Score (GPS) [148] were introduced to estimate the power consumption of shift-add networks. It was argued that higher LD leads to more glitches and a low LD is desired. In [75], 'C1' algorithm was proposed to lower the LD, but it does not guarantee minimal LD. Some AG algorithms [19] allow an optional constraint of a global maximum LD. However, to achieve the lowest average LD, all coefficients have to be independently constrained to their minimal LD.

An analysis of MSD reveals that potential good solutions are omitted by Common Subexpression Elimination (CSE) algorithms as they are hidden in the MSD representations. Some CSE algorithms ensure that all coefficients are implemented at minimal Logic Depth (LD) which is advantageous from power saving perspective. Imposing this requirement on a Adder Graph (AG) algorithm reduces the search space as well as the runtime. It also eliminates the long critical path of AG algorithm. A 'Minimum Adder Depth Algorithm' (MAD) is proposed in [79], [80], which include an exact and two heuristic methods. All three methods use lookup tables to solve a coverage problem. The lookup tables enumerate all possible combinations of $\{u, v\}$ of Equation (2.27) on page 29 to form a coefficient by an $\mathcal{A}$-operation. Good results are obtained, but for filters with more than 40 odd fundamentals, the runtime of the exact solver exceeds one hour and the lookup tables restrict the maximal bit width. On-the-fly generation of only the required entries in the lookup table is theoretically possible, but it is too time consuming and impractical for large coefficients.

This section presents an analysis of the properties and limitations of $\mathcal{A}$-operations (See Equation (2.27) on page 29) under the constraint of a guaranteed minimum LD and different types of hidden nonzero digits compositions. A heuristic logic depth constrained $\mathcal{A}$-operation algorithm is proposed. As the proposed algorithm is not limited by lookup tables, problems containing a large number of fundamentals can be solved efficiently. Simulation results show that the proposed algorithm has lower number of adders than CSE algorithms while having the minimal logic depth. For all filters tested, it consumes less switching power than the AG methods, has fewer LO than the CSE algorithms and produces similar or better results than the algorithms of [79].

4.2.1 Minimal Logic Depth Adder Trees

A single input shift-add network is considered where the addition operations can involve both adders and subtractors. Under this premise, the coefficients of the coefficient multipliers are succinctly expressed by strings of signed digits from the set $\{-1, 0, 1\}$. Again, the digit -1 is denoted by $\bar{1}$. Irrespective of how the digits in the coefficients are eliminated or transformed when the coefficient multipliers are decomposed into shift-and-add operations, the minimal LD of each coefficient multiplier is still confined by the minimum number of nonzero digits.

Table 4.5 shows the minimal LD, the smallest CSD number and the required binary bit width to store this number for $S(C)$ from 2 to 16. The last column shows the number of possible solutions of $\mathcal{A}$-operations with minimal LD for the smallest $C \geq 11\ 184\ 811$.

TABLE 4.5: CSD and Binary Lengths for Different $S(C)$

$S(C)$	$LD_{min}(C)$	CSD_{min}	bw_{BIN}	# $\mathcal{A}$-op sets
2	1	3	2	1
3	2	11	4	12
4	2	43	6	7
5	3	171	8	2900
6	3	683	10	1564
7	3	2731	12	539
8	3	10 923	14	155
9	4	43 691	16	4 090 299
10	4	174 763	18	3 723 852
11	4	699 051	20	3 353 288
12	4	2796 203	22	2 978 142
13	4	11 184 811	24	2 627 549
14	4	44 739 243	26	2 032 494[1]
15	4	178 956 971	28	429 228[1]
16	4	715 827 883	30	25 938[1]

The smallest positive CSD value with a given number of nonzero digits can be calculated by the following recursion:

$$\begin{aligned} CSD_{min}(1) &= 1 \\ CSD_{min}(S(C)) &= 4 \cdot CSD_{min}(S(C)-1) - 1 \end{aligned} \tag{4.7}$$

This can be proven by the fact that with one nonzero digit, the only possible CSD number is 1. By definition of CSD, the next digit must be 0. Therefore, a left shift by 2, equivalent to a multiplication by 4 and a bit width increment of 2, is necessary to obtain the CSD with two nonzero digits. To make up the minimum positive CSD value, the smallest possible nonzero digit from the set $\{-1, 0, 1\}$ that can be added is $\bar{1}$. This process can be repeated to find the value of the smallest positive CSD value for the next higher number of nonzero digits.

To achieve minimal LD, a balanced adder tree is always constructed. A perfectly balanced adder tree is attained only if $S(C) = 2^n$. Otherwise, there will be at least one void leaf in the balanced adder tree. Let the Digits of Freedom (DOF) be defined as the number of void leaves of a balanced adder tree for a number, C. Then,

$$DOF(C) = 2^{\lceil \log_2(S(C)) \rceil} - S(C) \tag{4.8}$$

[1] with a more restricted search space

A number C has the biggest DOF when $S(C) = 2^n + 1$. This implies that more adder trees with the same LD constraint can be constructed for a number with $S(C) = 5$ than for a number with $S(C) = 8$ as their DOF are 3 and 0, respectively, as shown in the last column of Table 4.5 and Figure 4.2.

One integer with $S(C) = 1$ exists in each range $[2^n, 2^{n+1})$ and there are at least one integer with $S(C) = 2$ in the same range for $n \geq 2$. As n increases, more integers with higher $S(C)$ are possible. The frequency of occurrence of different $S(C)$ for the MSD representations of 1 to 2^{18} are shown in Figure 4.1, where the x-axis is plotted on a binary logarithmic scale. Figure 4.1 shows that every $S(C)$ occurs most frequently for approximately 2.5 powers of 2 and then the relative frequency of occurrences declines slowly towards zero. Although the relative frequency of occurrences of a given nonzero digit count declines, the absolute count increases monotonically.

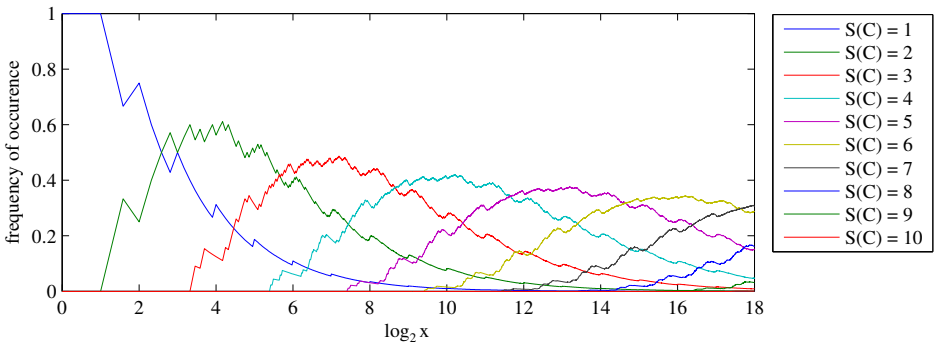


Figure 4.1: Frequency of Occurrence for Different $S(C)$

Not only is the frequency of occurrence of interest for optimization, but also the number of possible pairs that forms a given value. The idea is to find all $\{u, l_1, v, l_2\}$ quartets which satisfy Equation (2.27) for a given number C. It is impractical to generate all possible combinations of $\{u, l_1, v, l_2\}$ to select only the ones that are equal to C. Therefore, r and s in Equation (2.27) are set to 0 and the equation is rearranged to:

$$C - 2^{l_1}u = 2^{l_2}v \tag{4.9}$$

To ensure that LD is not increased, the condition

$$\{S(u), S(v)\} \leq 2^{\lceil \log_2(S(C)) \rceil - 1} \tag{4.10}$$

must hold. If $S(C) > 4$, Equation (4.9) has infinite number of solutions that satisfy Equation (4.10). To avoid this problem and the use of unnecessarily large adders, a bound for $\{u, v\}$ is required. The bound $\{|2^{l_1}u|, |2^{l_2}v|\} < 2 \cdot 2^{\lceil \log_2(C) \rceil}$ allows the bit width of the adder to be incremented by one maximally compared to the minimal bit width. Furthermore, $\{u, v\}$ must both be odd as the even numbers are taken care by the left shifts $\{l_1, l_2\}$.

All possible pairs of $S(u)+S(v)$ that will not increase the LD of $S(C)$ are shown in Table 4.6. The pairs where no additional digit is introduced are italicized. For all numbers with $S(C) \neq$

2^n, several additional combinations exist due to the hidden digit phenomenon elaborated in the next subsection. This is even more prominent for cases beyond 16 bits.

TABLE 4.6: Possible $S(x)$ Pairs Which do not Increase the LD

$S(x)$	possible nonzero counts of the inputs to the additions
2	*1+1*
3	*2+1* 2+2
4	*2+2*
5	*4+1* 4+2 4+3 4+4 *3+2* 3+3
6	*4+2* 4+3 4+4 *3+3*
7	*4+3* 4+4
8	*4+4*
9	*8+1* 8+2 8+3 8+4 8+5 8+6 8+7 8+8 *7+2* 7+3 7+4 7+5 7+6 7+7 *6+3* 6+4 6+5 6+6 *5+4* 5+5
10	*8+2* 8+3 8+4 8+5 8+6 8+7 8+8 *7+3* 7+4 7+5 7+6 7+7 *6+4* 6+5 6+6 *5+5*
11	*8+3* 8+4 8+5 8+6 8+7 8+8 *7+4* 7+5 7+6 7+7 *6+5* 6+6
12	*8+4* 8+5 8+6 8+7 8+8 *7+5* 7+6 7+7 *6+6*
13	*8+5* 8+6 8+7 8+8 *7+6* 7+7
14	*8+6* 8+7 8+8 *7+7*
15	*8+7* 8+8
16	*8+8*
17	*16+1* 16+2 ... 16+16 *15+2* 15+3 ... 15+15 *14+3* 14+4 ... 14+14 *13+4* 13+5 ... 13+13 *12+5* 12+6 ... 12+12 *11+6* 11+7 ... 11+11 *10+7* 10+8 10+9 10+10 *9+8* 9+9
18	*16+2* 16+3 ... 16+16 *15+3* 15+4 ... 15+15 *14+4* 14+5 ... 14+14 *13+5* 13+6 ... 13+13 *12+6* 12+7 ... 12+12 *11+7* 11+8 ... 11+11 *10+8* 10+9 10+10 *9+9*
...	...

Figure 4.2 shows the number of possible additions ($\{u, l_1, v, l_2\}$ quartets) for $S(C) = 1$ to $S(C) = 10$ and 1 to 2^{18} on a binary logarithmic scale. The search assumes that all satisfying $\{u, v\}$ are available. The number of possible additions is not only influenced by the LD but also the DOF. For a LD of 2, as shown in the bottom plot of Figure 4.2, the numbers with a DOF show more possible additions than the ones with no DOF. The same is true for the middle plot of Figure 4.2 with a LD of 3. Here the difference is more prominent as the DOF is bigger. Lastly the top plot shows of Figure 4.2 the values for $S(C) = 9$ and $S(C) = 10$. The latter curve lies slightly below the former. As the absolute values are above 10^5, the difference is indiscernible. The large number of possible additions for all $S(C)$ makes it impossible to store them in a lookup table and a coverage problem might not be solvable due to the immense possibilities. The abrupt jumps between logarithmic steps are caused by the expansion of the search range. The differences in the number of possible additions can be explained by the

hidden nonzero digits.

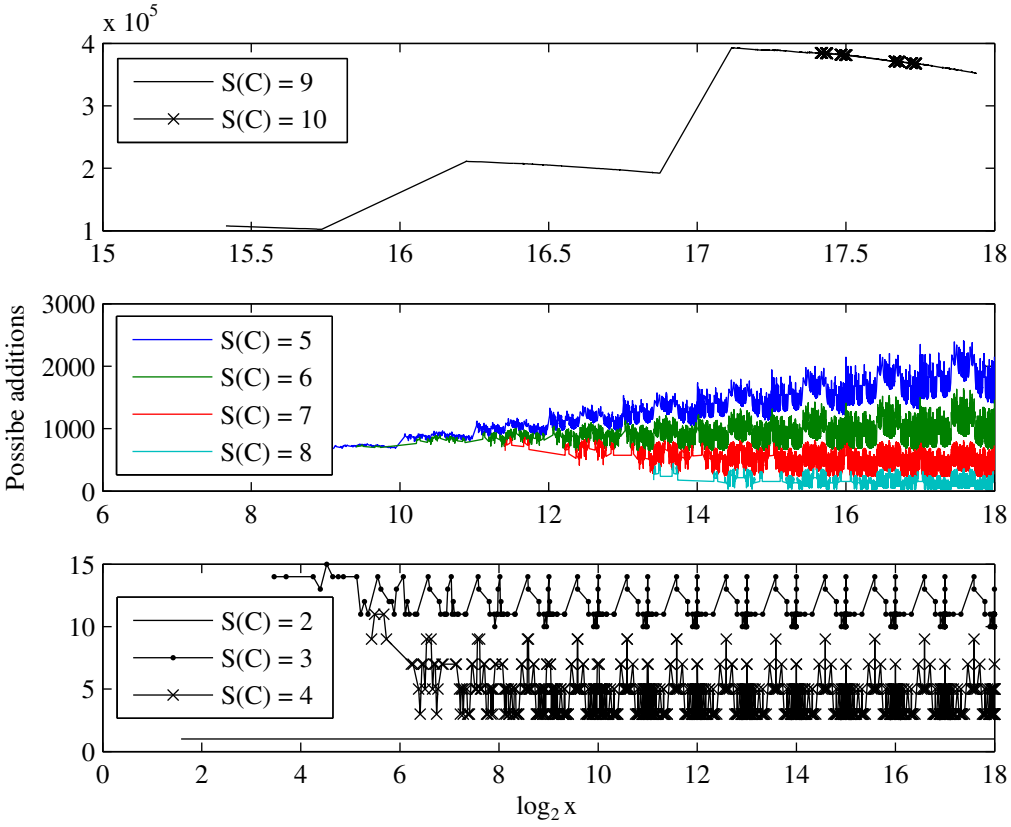


Figure 4.2: Possible Additions that Generate a Number with Minimal LD

4.2.2 Hidden Nonzero Digits

Three general and three nongeneral classes of Overlapping Digit Pattern (ODP) are recently introduced in [51]. One example is $25_{10} = 11001_2$. It contains the hidden pattern 101 as $(101 \ll 2) + 101 = 11001$, where the two colliding 1 bits at the third position of the addends form the 1 bit at the fourth position of the sum. The classification used here is different as more than two digits can be involved in a simplification. As any MSD representation can satisfy Equation (2.27), all MSD representations have to be found and additional adder pairs are generated from each of the Hidden Nonzero Digits (HNDs). The occurrence of the pattern $10\bar{1}$ or $\bar{1}01$ in CSD indicates that other MSD representations exist, as ± 3 is the only value with two MSD representations. For a number with $DOF(C) = 0$, only different MSD representations can generate additional adder pairs, whereas for numbers with higher DOF, more possibilities exist which will be introduced here.

Colliding Digits

If two SD numbers are added, two digits of the same sign could coincide in the same position. These digits form a single digit at the next higher position in the sum and the original digit

position is hidden. This is equivalent to the first general ODP class. Several collisions can occur when $DOF(C) > 2$, but only two digits are involved in each collision.

Eliminating Digits

If $DOF(C) \geq 2$, two digits with the same weight could have opposite signs and therefore eliminated in the sum. This can only happen for numbers with $2^n < S(C) < 2^{n+1} - 1$.

Reducing Digits

If two numbers are added, contiguous digits can form a reduced SD pattern of ± 1. For example, $1\bar{1}$ or $\bar{1}1$ can be reduced to 01 or $0\bar{1}$. If $DOF(C) > 1$, more reduction is possible, as exemplified by $100\bar{1}0\bar{1} + 0010\bar{1}0 = 101\bar{1}\bar{1}\bar{1} = 100001$. This is an extension to the second and third general ODP classes.

Binary to CSD Reduction

It can happen that a CSD pattern like 101 is added to another CSD pattern 010 that results in a binary to CSD reduction. For example, $101 + 010 = 111 = 100\bar{1}$.

For any number C, the HND count is equal to $DOF(C)$. With $DOF(C) \geq 2$, more than one class can be applied simultaneously and different HNDs can exist within each MSD representation. This makes it extremely difficult to find the number of possible adder pairs analytically. For the simple example of $73 = 1001001_2$, where no other MSD representation exists, there are three possibilities to generate this pattern without involving a HND. Colliding digits contribute two, reducing digits contribute six and Binary to CSD reduction contributes one unique possibility to split the pattern into two.

4.2.3 Minimal Logic Depth AG Algorithm

With the aforementioned insights and from other algorithms, a minimal LD AG algorithm is proposed. The design ideas behind the algorithm are that minimal logic depth is attained for all coefficients and that the algorithm is able to handle large coefficient sets. When looking at adder trees of different algorithms it can be noted that the odd fundamentals with small $S(C)$ are reused most frequently. Therefore the choice of these fundamentals has the impact on the result. It is suggested that these fundamentals should be fixed as late as possible. However, this is prohibitive in terms of the runtime and memory as too many intermediate results need to be stored for larger coefficient sets. With the insights gained from the analysis of the minimal logic depth adder trees, the fundamentals with $S(C) = 2$ are required for the outputs and can be reused. Therefore, they are put into the Output Set, OS, in the first step. Then the adder tree is built from the input towards the fundamentals based on the idea that the fundamentals with the smallest number of possible $\mathcal{A}$-op sets are fixed first. This is because the choice is most restricted there and the probability that adding two new fundamentals to the OS, which are Non-Output Fundamental (NOF), is very small as can be derived from the

results of the MAD algorithm. In many cases, there are several possible solutions for a step and fixing a fundamental too early does not lead to an optimal solution. Hence, a number of possible results are carried forward to the next step. Using the large set of benchmark filters (See Section 3.2 on page 50), it was evaluated that taking the 16 best results for $S(C) > 8$ into the next step is the best tradeoff between the optimality of the result and the runtime of the algorithm. If the number is increased further the results do not improve except in terms of LO, but the runtime grows rapidly. Theoretically, if all results would be carried forward to the next step, the algorithm would be able to find the optimum solution, but the runtime is prohibitive and the approach of MADbb is more suitable for delivering these optimal results. The cost function to select the results to be carried forward to the next step focuses primarily on the number of LO, and secondarily on NOFs with small values. This is because the GPC suggests to favor additive graphs over multiplicative graphs and the GPS suggests to favor additions or subtractions with larger shifts over those with smaller shifts.

The algorithm starts with putting all $S(C) = 2^n$, $n = 2$ into the Target Set, TS. Then it proceeds with $S(C) = 3$, 8, $5 \leq S(C) \leq 7$ and $9 \leq S(C) \leq 15$. Basically it starts with $S(C) = 2^n$ and then solves for $2^{n-1} < S(C) < 2^n$, where $n > 1$, as this the order of complexity based on the number of $\mathcal{A}$-op sets. Before each solving process, a Ready Set, $RS = \{1\} \cup OS$ is formed.

The solving process solves Equation (4.9) with $C \in TS$ by finding $v \in RS$ for all $u \in RS$, where RS is restricted to the values that satisfy condition Equation (4.10). Although each group of fundamentals are solved separately, after every step a look ahead is executed without affecting the results. The separation not only simplifies the code, but also speeds up the computation.

In solving for Equation (4.9), the value of $2^{l_2}v$ is converted to CSD and the nonzero digit count is used to check if Equation (4.10) is satisfied. $v \notin RS$ with $S(v) < 2^{\lceil \log_2(S(C)) \rceil - 1}$ are stored in the Potential Set, PS. If a solution is found, $OS = OS \cup C$ and $TS = TS \setminus C$ and all entries for C are removed from PS. After the first iteration, every $D \in PS$ is combined and all the TS is tested with all possible $RS \cup D$. The 16 best solutions are taken for the next iteration. This process continues until $TS = \varnothing$. $D \in PS$ with low additional adder cost is preferred if available. If the RS is too small and no entry is generated for PS, the algorithm puts all possible patterns generated from the MSD representations into PS. If after solving TS, several OS with the same number of NOFs exist, the one with the smallest value is chosen as it has the smallest FA cost. The look ahead might select another more promising set in the later steps.

No implementation is fixed until all fundamentals are solved. This is because a NOF that allows a simpler implementation might be introduced any time in this process. NOFs that are not required are removed and all NOFs are examined to see if they can be replaced with smaller valued NOFs to reduce the FA cost. When several implementations are possible for a given $D \in OS$, the one with the smallest $S(u) + S(v)$ is chosen. If there are still more than one possible implementation, the one with the smaller FA cost is picked. The former reduces glitches while the latter reduces area.

Values with $S(C)$ bigger than 15 are currently not considered as these numbers would be bigger than 715 827 883. If necessary, the algorithm could cater for larger $S(C)$ but the run time can be prohibitively long as the search space is very large for such numbers.

4.2.4 Simulation Results

The algorithm was coded in MatLab and tested on a set of filters from [21], which produces the minimal LO for these filters. In Table 4.7, the column UQ denotes the number of positive and odd unrepeated fundamentals, 'LD_{min}' shows the minimal LD. The algorithms 'BFS' from [21], 'Hcub' and 'Hcub LDmin' from [19], 'MADbb' and 'MADcr' from [79] are compared with our proposed method 'MinLD'. The 'Hcub LDmin' algorithm limits all coefficients to the same LD, which can lead to a smaller LO, but it also increases the GPC and GPS as some coefficients might not have the minimal LD. 'MADbb' denotes the optimal 'MAD' algorithm with a limited runtime of one hour while 'MADcr' selects the best result between the two heuristic 'MAD' algorithms.

Although the minimal LD algorithms 'MAD' and the proposed 'MinLD' do not achieve the smallest LO, the power estimate is better than the near minimal heuristic Hcub method by a factor of up to 3.4 time. The proposed method has the same LO as the optimal 'MADbb' and outperforms 'MADcr' in two cases for the test filters.

The proposed algorithm was also compared to the CSE algorithms 'NRSCSE' [10] and 'MBPG' [22] using a set of 65 FIR filters from the literature, which can be found at [122] (www.firsuite.net). For all filters with one or more than one coefficient with $S(C) > 2$, the results of the proposed method have fewer LOs and better GPC and GPS.

TABLE 4.7: LD Results for Filters from [21]

Filter	UQ	LD_{min}	BFS		Hcub			Hcub LDmin			MADbb[1]			MADcr[1]			minLD	
			LD	LO	LD	LO	GPC	GPS	LO	GPC	GPS	LO	GPC	GPS	LO	GPC	GPS	GPC	GPS
1	19	3	8	**22**	7	23	191	2192	27	90	1218	26	76	931	26	76	931	**73**	**917**
2	39	3	9	**41**	8	42	647	6124	56	218	2933	n/a	n/a	n/a	53	195	2438	51	**188**	**2237**
3	14	3	5	**16**	5	**16**	57	761	18	53	689	18	53	621	18	52	619	18	**51**	**591**
4	33	3	5	**34**	5	**34**	141	1767	38	134	1709	n/a	n/a	n/a	38	123	1490	38	**121**	**1453**
5	18	3	5	**20**	5	**20**	103	1237	23	76	973	25	74	898	25	72	870	25	**69**	**798**
6	36	3	6	**37**	5	38	175	2215	44	153	2015	44	139	1596	44	138	1582	44	**133**	**1504**
7	19	3	5	**21**	7	**21**	104	1104	23	73	845	24	66	793	24	66	793	24	**65**	**761**
8	29	3	7	**31**	7	**31**	204	2628	39	149	2054	n/a	n/a	n/a	38	119	1493	38	**115**	**1476**
9	14	3	7	**21**	6	**21**	126	1558	25	83	999	25	**67**	771	25	**67**	771	25	**67**	**770**
10	16	3	7	**31**	7	**31**	235	2631	39	153	1865	n/a	n/a	n/a	38	122	1506	38	**116**	**1427**
11	12	3	8	**16**	7	**16**	119	1240	18	62	725	n/a	n/a	n/a	20	**57**	**633**	19	60	713

[1]Many thanks to Kenny Johansson for sharing the results of his MAD algorithm

4.3 Reconfigurable Multiple Constant Multiplication with Minimum Adder Depth

The problem of Reconfigurable Multiple Constant Multiplication (ReMCM) is about finding an cost-effective network of shifters, adders, subtractors and multiplexers to implement the multiplication of a single input variable with one out of several sets of coefficients. Most publications [96], [98]–[100], [149] only focus on the problem with a single output, whereas the algorithm proposed here solves a multiple output ReMCM problem using a minimal logic depth adder-graph based approach. The use of minimal logic depth restricts the length of critical path and it was shown earlier that minimum depth MCM is advantageous in terms of power consumption. Adder-graph heuristic offers more possibilities of adder formation to reduce the total number of adders and multiplexers. For the polyphase decimation filters, the filter length and decimation factor are related and have an influence on the implementation cost.

4.3.1 Reconfigurable MCM

In recent years, the interest in reconfigurable constant multiplication [96], [98]–[100], [149] has grown as the computation of all constants in parallel is not always necessary or possible. Applications like polyphase FIR filters [71] or DCT can be modeled such that the distinct inputs are each multiplied by a different MCM block. In the case of a polyphase FIR filter with a decimation factor of four, as many MCM blocks as the decimation factor are required. Some reduction is possible if the initial FIR filter has linear phase and therefore is symmetrical. Figure 4.3 shows a polyphase FIR filter with a decimation factor of two. The filter in Figure 4.3 is operated at the lower output clock frequency, which is half of the input frequency in this example, due to the decimation factor of two. If the delay constraints allow for operating the filter at higher clock frequency, the MCM block can be implemented as a reconfigurable MCM block, where the MCM block is reconfigured in each input cycle and the result is summed in a modified SA block. Generally, the length of the critical path has an impact on the switching activity. Constraining the logic depth of all individual constants to their minimum is important not only for operating the reconfigurable MCM block at the higher clock rate, but also for reducing the power consumption of the MCM block [58], [79], [80], [148].

In the transposed direct form polyphase filters, adders are required to sum the outputs of different subfilters. If the subfilters are merged into one single reconfigurable FIR filter running at a higher clock frequency, these summation adders are not required, as the summation takes place sequentially over time within the loop-back path shown in Figure 4.4.

The two main approaches for the solutions to MCM problems discussed in Chapter 2 can both be adapted to ReMCM. The potentially high LD of AG methods makes them less suitable for application to ReMCM. On the other hand, the need for a fixed number representation in CSE algorithms restricts the search space for common subexpressions and makes sharing of additions obscure. The former problem associated with adder-graph algorithms has been

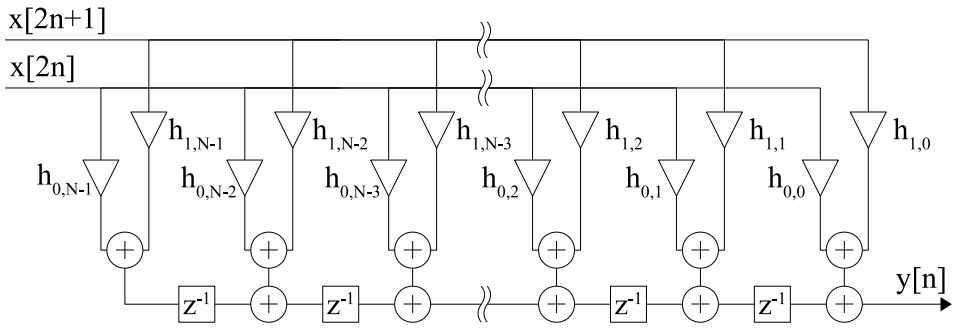


Figure 4.3: Schematic of a Polyphase FIR Filter with a Decimation Factor of $M = 2$

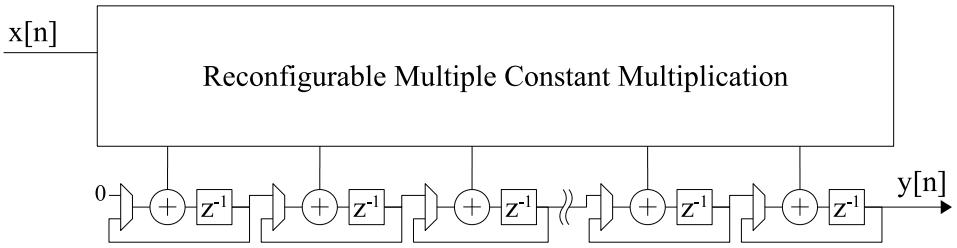


Figure 4.4: Schematic of a Polyphase FIR Decimation Filter with Reconfigurable MCM Block

circumvented by restricting the maximal logic depth for each constant multiplier, as in [79], [80] and Section 4.2. With a larger subexpression space provided by AG methodology, there is more flexibility to merge two multipliers. Hence, the algorithm from Section 4.2 ([58]) is adopted as a basis for the proposed ReMCM algorithm.

4.3.2 Subfilter Allocation

When a FIR filter has linear phase, the coefficients of the filter are symmetrical. When the filter of length N is split into M subfilters, where M is the decimation factor, the symmetric property translates into the pairwise symmetry of the subfilters. This pairwise symmetry can be, and is used to reduce the complexity of the ReMCM block, at the cost of more complicated multiplexing. The number of symmetric subfilters is influenced by both the filter length N as well as the decimation M. Two cases can be formed depending on whether M is even or odd. If M is even, the parity of N is a determinant of the number of unique subfilters. Figure 4.5(a) shows an example for N with odd parity, where the number of unique subfilters is $M/2 + 1$. The two non-repeated subfilters exhibit internal symmetry, which simplifies the subfilters itself, but the output assignment to the MUX is rather complicated. Figure 4.5(b) and Figure 4.5(c) show two examples for N with even parity, where the number of unique subfilters is $M/2$ as all subfilters occur twice with reverse-ordered coefficients. For odd M, the parity of N is irrelevant as there is always one subfilter without a symmetrical counterpart, as shown in Figure 4.5(d), 4.5(e) and 4.5(f). In this case, no optimization is possible.

Besides the number of unique subfilters, the number and fan-in of the MUXs required to select the reverse-ordered coefficients are influenced by the difference in the subfilter lengths,

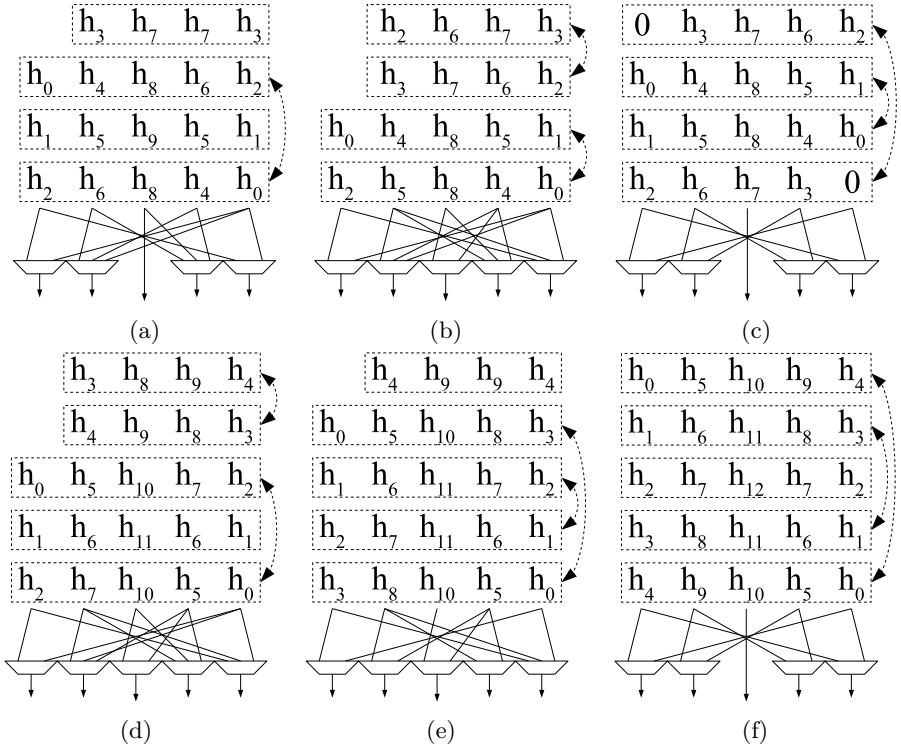


Figure 4.5: Examples of polyphase filters with even parity (a)-(c) and odd parity (d)-(f) for the decimation. In example (a), the alignment is not symmetric and has two unpaired subfilters. (b) has full symmetry, but requires more MUX which are also more complex than the zero padded version shown in (c). The difference between (d) and (e),(f) is that (d) requires more complex MUX, but could be brought to the same complexity as (f) by padding the coefficient with a single zero

which are in turn influenced by N and M. In general, no more than $\lceil N/M \rceil$ 3-to-1 MUXs are required. For even M, if the parity of N is also chosen to be even and either $N = M \cdot c$, $c \in \mathbb{N}$, or zero padding on both ends of the coefficients is allowed, the number of MUXs is reduced to $\lfloor (N+2 \cdot p)/M \rfloor$, where p is the number of zeros used for padding on each side of the coefficients, and the fan-in of the MUX is reduced from 3 to 2. This reduction translates directly into area savings, but the zero padding comes at the cost of a higher delay for the filter function. A non-optimal implementation for the case of odd M is shown in Figure 4.5(d). For odd M, the reduction requires an intermediate step, where the subfilter without a symmetric counterpart is either one bit longer or shorter than the other subfilters. In these cases, a few 3-to-1 MUXs are required, as shown in Figure 4.5(e). If all subfilters are of the same length, the number of MUXs can be reduced to $\lfloor (N + 2 \cdot p)/M \rfloor$ 2-to-1 MUXs as shown in Figure 4.5(f). The filter in Figure 4.5(d) can be padded by one zero on each side of the coefficients to reduce the MUX complexity to the same as that of 4.5(f).

The output MUXs shown in Figure 4.5 are also called crossover or symmetry MUX, as the establish the pairwise symmetry between the subfilters and crossover the signals if not all

subfilters are of the same length.

4.3.3 Proposed Algorithm

Based on the minimal logic depth adder graph based algorithm minLD [58], a new algorithm for the optimization of reconfigurable MCM was developed. The main difference between the proposed algorithm and the algorithms of Tummeltshammer *et al.* [99] and Chen *et al.* [100] is that it optimizes the MCM with several concurrent outputs as a complete reconfigurable set, whereas the latter algorithms [99], [100] only target MCM with a single output.

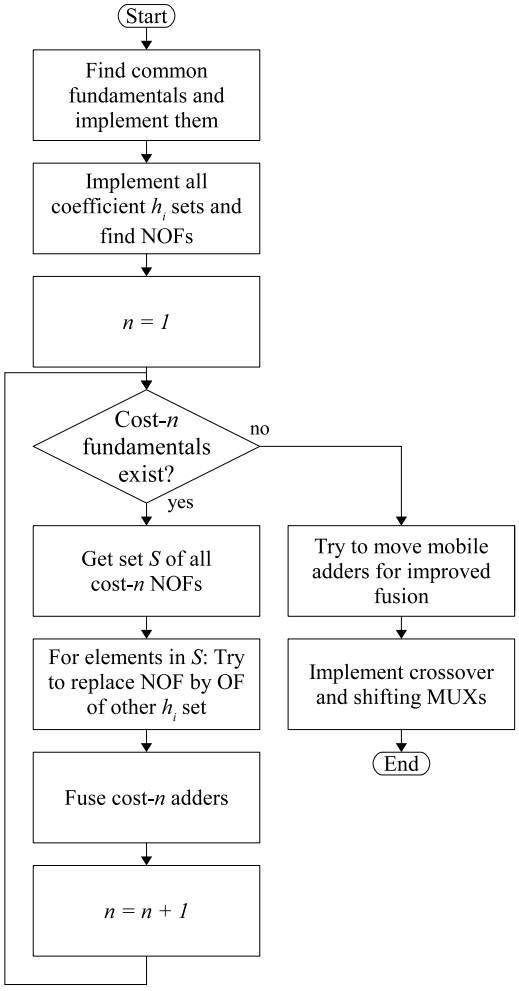


Figure 4.6: Flow Chart of the Proposed Reconfigurable MCM Design Algorithm

The filter design phase is not part of the algorithm, therefore the design of the filter length and filter length parity is left to the filter designer, but the algorithm will, when allowed, use zero padding to optimize the complexity of the output MUX as described in the previous section. The proposed algorithm solves the MCM problems for each unique subfilter as well as the MCM problem for the overall filter. Using the information from the adder graphs for the subfilter problems, the maximum number of adders required per LD is calculated. The NOFs with $LD = 1$ from the overall problem are analyzed to see if there is a set of NOF

that is more beneficial to all subfilters than the ones found in the separate subfilters. Doing so can reduce the number of MUXs required to generate the $LD = 1$ part. Next all adders with $LD = 1$ are fused. Before effectively fusing a NOF, the algorithm tests if the NOF could be replaced by other fundamentals required by another subfilter. If this is not possible or the fundamental is an output fundamental, a fusion routine is started. This routine calculates the cost for all possible fusions. The bit width calculation from [150] is used to obtain the width of the required arithmetic operators and MUXs and then the cost measure from [99] is applied to determine the cost of fusion. When the costs of all possible fusions have been calculated, the fusion with the smallest incremental cost is chosen and the fundamentals are fused. This is repeated until the minimal number of arithmetic operators is achieved. Optionally, the routine allows additional arithmetic operators if further fusion does not reduce the required area, but this might result in higher power consumption. This issue is currently being investigated. When all $LD = 1$ adders are implemented and fused, the next higher LD is tried. When the fusion for all levels of adder depth is completed, the algorithm will attempt to improve the reconfigurable adder graph by moving mobile adders, but only an intuitive and simple implementation of this concept is carried out at the moment. After this, the required MUX at the output is implemented. A flow chart of the algorithm is shown in Figure 4.6.

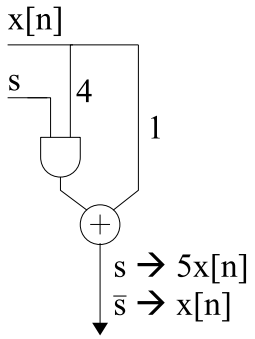


Figure 4.7: Usage of an AND to Bypass Adder

Some additional optimization ideas were experimented but the implementation was not completed before the code was used to generate the results shown in the next section. One of these optimizations was to align the coefficients to optimize the fusion process. For polyphase filters with only two unique subfilters, the search is rather easy, as it is only a pairwise matching. For a filter with a higher number of subfilters, it is fairly complicated as for each unique subfilter, two or three choices are possible due to the symmetry MUXs shown in Figures 4.5 (a)-(f). Also the calculation of the savings often depends on the parts of the filter that has not yet been processed. It was also observed that if an output fundamental is located at a LD which is not the maximal LD, an output MUX might be required to select this output fundamental. This could be optimized by using a simple array of AND gate (bitwise AND) to switch between the signal input and the zero of the other input of the arithmetic operator at the next LD level. This would reduce an adder to a single feed-through and a subtractor to an inverter or a feed-through, depending on which side of the input is used. The schematic of

this operation is shown in Figure 4.7.

4.3.4 Experimental Results

To evaluate the proposed algorithms, both for single and multiple output cases are considered. For the purpose of comparison, the unitless cost function from [99] is adopted, which was also used in [100]. The unitless cost estimate assigns a fixed but relative cost to each bit of an operation. The cost of an operator is $a \cdot k$, where k is the bit width and a is the relative cost which depends on the ASIC technology and the library used. In [99], a $0.18\mu m$ ASIC technology with a commercial $0.18\mu m$ standard cell library was considered for the assessment of the relative cost. To simplify the comparison with the previous methods, the same values for a are employed:

$$\begin{aligned} v\text{-to-1 Multiplexer: } & a_{mux} = 14v \\ \text{Adder: } & a_{add} = 67 \\ \text{Subtractor: } & a_{sub} = 75 \\ \text{Adder/Subtractor: } & a_{addsub} = 98 \end{aligned}$$

The total area cost estimate A is then computed by Equation (3) of [99]:

$$A = \sum_{i=1}^{n_{mux}} a_{mux} \cdot bw_{mux_i} + \sum_{i=1}^{n_{add}} a_{add} \cdot bw_{add_i} + \sum_{i=1}^{n_{sub}} a_{sub} \cdot bw_{sub_i} + \sum_{i=1}^{n_{addsub}} a_{addsub} \cdot bw_{addsub_i} \quad (4.11)$$

where n is the number of operators and bw_i is the respective bit width of an operator. For all results reported, an input bit width of 8 bits is used.

Single Output Reconfigurable Multiplication

For comparison, the degenerated case with only one output was examined. In this case, the decimation is set to be equal to the number of coefficients. The first example was taken from Figure 5 of [100], where the coefficients are $\{815, 621, 831, 105\}$, and hence $M = 4$. The optimized circuit for the example from [100] is shown in Figure 4.8. The proposed method reduces the bit-level cost from 9192 in [100] to 7804 (15.1%), based on the cost estimates from [99] and an input bit width of 8. When allowing one more adder at $LD = 2$, the overall cost is further reduced to 7569 (17.6%) although the adder/subtractor is split into one adder and one subtractor. The corresponding circuit is shown in Figure 4.9. This illustrates that the minimal number of arithmetic operators is not always optimal because two independent arithmetic operations might be of shorter length than that of a combined, time-multiplexed operation. The proposed algorithm chooses the set $\{3, 5, 7, 63\}$ instead of $\{3, 7, 9, 63\}$ for $LD = 1$. This does not result in reduction in terms of the number of operators, but its fusion of the fundamentals is made slightly more area efficient by keeping the multiplexing overheads low with shorter reconfigurable arithmetic operators. In this case, the longer operator is a fixed subtractor. The use of an adder graph algorithm enables the proposed algorithm to have more options

for fusion and hence a simpler fusion of arithmetic operators at $LD = 3$ is found in the given example, where only a subtractor is needed, whereas in [100] a reconfigurable adder/subtractor is required.

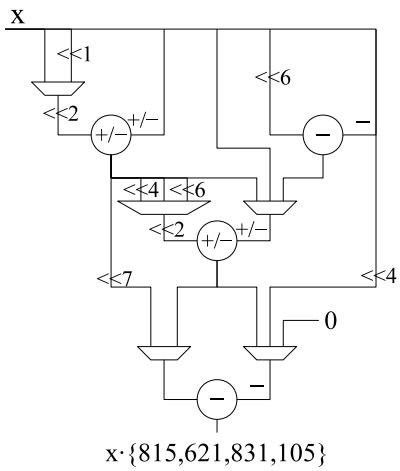


Figure 4.8: Reconfigurable Constant Multiplier for the Constants 815, 621, 831 and 105

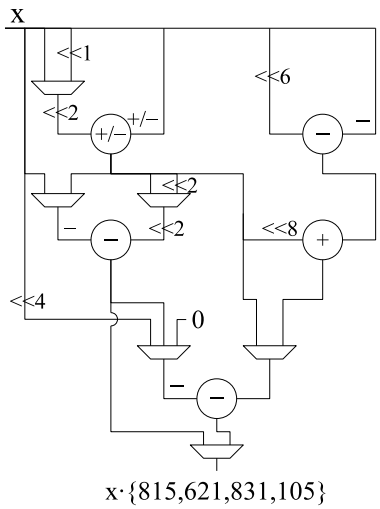


Figure 4.9: Same constants as Figure 4.8, but allowing one more operator. This results in two arithmetic operators at $LD = 2$ and a smaller overall area cost

Two more examples were taken from Figures 7 and 8 of [96], where the coefficients are $\{362, 392, 473\}$ and $\{39, 150, 196\}$, respectively. The optimization by the proposed algorithm results in a cost estimate of 5639 and 3473. For the first set, the results are better than the algorithms in [99], [100] by 11.4% and 5.8% respectively, but requires 11.1% more area than [96] due to the restriction of minimal logic depth. For the second set, there is a reduction of 3.8% over [100] due to different MUX assignments and the replacement of the fused operation for the generation of the fundamentals, 3 and 5.

TABLE 4.8: Components for Polyphase FIR Filters with Different Decimation Factors

Decimation	**4**	**5**	**6**	**7**	**8**
Filter Length	**44**	**55**	**66**	**77**	**88**
Subfilter Length	**11**	**11**	**11**	**11**	**11**
Arithmetic Operators	17	17	18	16	17
Adders	3x(8 bit) 1x(9 bit) 2x(10 bit) 1x(11 bit) 1x(12 bit) 1x(13 bit)	2x(8 bit) 2x(10 bit) 1x(12 bit) 1x(13 bit) 2x(17 bit)	3x(8 bit) 3x(11 bit) 1x(12 bit) 1x(13 bit) 2x(15 bit) 1x(18 bit)	2x(8 bit) 1x(9 bit) 1x(11 bit) 2x(12 bit) 1x(13 bit) 1x(16 bit) 1x(18 bit)	3x(8 bit) 2x(11 bit) 1x(12 bit) 1x(13 bit) 2x(14 bit) 2x(17 bit)
Subtractors	1x(11 bit) 1x(12 bit) 1x(15 bit) 2x(18 bit) 1x(19 bit) 1x(20 bit)	1x(11 bit) 1x(12 bit) 3x(16 bit) 2x(17 bit) 1x(19 bit)	1x(11 bit) 1x(17 bit) 1x(19 bit)	2x(16 bit) 1x(17 bit) 1x(20 bit)	1x(11 bit) 2x(17 bit) 1x(20 bit)
Adder/Subtractors	1x(14 bit)	1x(12 bit)	1x(12 bit) 1x(15 bit) 2x(16 bit)	1x(11 bit) 1x(14 bit) 1x(19 bit)	1x(14 bit) 1x(16 bit)
Multiplexers	1x(9 bit_{2-1}) 3x(11 bit_{2-1}) 4x(12 bit_{2-1}) 6x(13 bit_{2-1}) 1x(15 bit_{2-1}) 2x(18 bit_{2-1})	1x(9 bit_{2-1}) 1x(10 bit_{2-1}) 3x(11 bit_{2-1}) 3x(12 bit_{3-1}) 2x(13 bit_{3-1}) 1x(14 bit_{2-1}) 1x(14 bit_{3-1}) 1x(17 bit_{2-1}) 1x(20 bit_{3-1})	1x(9 bit_{2-1}) 2x(10 bit_{2-1}) 3x(11 bit_{2-1}) 1x(12 bit_{2-1}) 1x(12 bit_{3-1}) 1x(13 bit_{2-1}) 1x(14 bit_{2-1}) 5x(15 bit_{2-1}) 1x(15 bit_{3-1}) 1x(17 bit_{3-1}) 1x(18 bit_{2-1}) 1x(19 bit_{3-1})	2x(9 bit_{2-1}) 1x(10 bit_{2-1}) 3x(11 bit_{2-1}) 3x(12 bit_{2-1}) 1x(12 bit_{3-1}) 1x(13 bit_{2-1}) 1x(13 bit_{3-1}) 2x(14 bit_{2-1}) 1x(14 bit_{3-1}) 1x(15 bit_{2-1}) 2x(15 bit_{3-1}) 1x(15 bit_{4-1}) 1x(16 bit_{2-1}) 1x(16 bit_{3-1}) 1x(17 bit_{4-1}) 1x(19 bit_{4-1})	2x(11 bit_{2-1}) 3x(11 bit_{3-1}) 2x(12 bit_{2-1}) 1x(12 bit_{4-1}) 1x(13 bit_{2-1}) 3x(13 bit_{3-1}) 1x(14 bit_{3-1}) 1x(14 bit_{4-1}) 1x(15 bit_{2-1}) 1x(15 bit_{3-1}) 1x(17 bit_{2-1}) 1x(17 bit_{3-1}) 1x(17 bit_{4-1}) 1x(18 bit_{4-1})
Symmetry Multiplexer	6x(17 bit_{2-1}) 2x(18 bit_{2-1}) 2x(19 bit_{2-1})	2x(16 bit_{2-1}) 4x(17 bit_{2-1}) 2x(18 bit_{2-1}) 2x(19 bit_{2-1})	2x(16 bit_{2-1}) 4x(17 bit_{2-1}) 2x(18 bit_{2-1}) 2x(19 bit_{2-1})	4x(16 bit_{2-1}) 4x(17 bit_{2-1}) 2x(19 bit_{2-1})	4x(16 bit_{2-1}) 4x(17 bit_{2-1}) 2x(19 bit_{2-1})

Multiple Output Reconfigurable Multiplication

As an example, a polyphase FIR decimation filter with decimation factor, M varies from 4 to 8 is chosen. The filter has a passband and stopband edges of $0.8 \cdot \pi/M$ rad and π/M rad, respectively, and passband and stopband ripples of 0.05. The filter length is chosen such that it meets the specification and $N = M \cdot n$. The filters of the respective length and equal weight of 1 for both passband and stopband are designed with `firpm`. The coefficients are

TABLE 4.9: Implementation cost for Polyphase FIR Filters with Different Decimation Factors

Decimation	**4**	**5**	**6**	**7**	**8**
Filter Length	**44**	**55**	**66**	**77**	**88**
Subfilter Length	**11**	**11**	**11**	**11**	**11**
MCM Block of Proposed Algorithm	21942	23197	26095	27814	27646
Distinct MCM Subfilters	12986 14095	12383 13822 9117	10785 11919 13223	11949 13218 11423 7926	11374 10402 12458 10348
Sum of Distinct MCM Subfilters	27081	35322	35927	44516	44582
Savings by Fusing	*5139*	*12125*	*9832*	*16702*	*16936*
Single MCM Block by MinLD [58]	23970	28098	29326	31250	33457
Reduction by Fusing over Single MCM Block	*2028*	*4901*	*3231*	*3436*	*5811*
Symmetry Multiplexers	4928	4872	4872	4760	4760
Structural Adder Reductions	*41001*	*57120*	*73373*	*89386*	*105399*

rounded to 14 bits, which results in a maximum of 13 active bits effectively. The filter input bit width is again set to 8 bits. The second and third rows of Table 4.8 and Table 4.9 show the filter lengths for the overall filter and the subfilters. The first row 'Arithmetic Operations' of Table 4.8 shows the required number of arithmetic operators for each decimation factor. In the subsequent rows, the breakdown of the arithmetic operators in terms of the number of adders, subtractors, adder/subtractors and multiplexers of each bit width and type are shown. In the rows 'multiplexers' and 'symmetry multiplexers', the multiplexers are specified in the notation $(p\,bit_{q-1})$, where p denotes the bit width of the multiplexer while q denotes the number of inputs of the q-to-1 multiplexer. The symmetry multiplexers are required to establish the symmetry as shown in Figure 4.5.

Based on the cost estimate from [99], the bit-level cost in square micrometer are shown in Table 4.9. The cost for the MCM block of the proposed ReMCM algorithm is shown in the first row below the table header. The next two rows 'Distinct MCM Subfilters' and 'Sum of Distinct MCM Subfilter' show the cost and sum of cost for implementing the filters from the proposed algorithm separately. The row 'Savings by Fusing' show the area savings by implementing the fused filters instead of implementing them separately. For comparison, the cost estimate of implementing the whole filter (non-reconfigurable) optimized by the algorithm presented in Section 4.2 ([58]) and the achieved reduction are presented in the rows 'Single MCM Block by MinLD' and 'Reduction by Fusing over Single MCM Block', respectively. The second last row shows the cost estimate for the 'Symmetry Multiplexers', while the last row 'Structural Adder Reduction' shows the area savings in the SA block between the polyphase implementation with distinct MCM blocks and the fused ReMCM MCM block.

From Table 4.9 it can be seen that if only the MCM block is of interest, the cost estimate of implementing the MCM blocks separately is higher than the implementation of a MCM block containing all coefficients, which again is higher than the cost estimate of the proposed algorithm. The overhead for implementing the subfilters independently ranges from 18% to

37% while the reduction of the MCM cost estimate between the fused and non-reconfigurable MCM block ranges from 8 and 17%. The latter is shallow comparison as it does not include the overhead of the symmetry multiplexer and the cost of the SA block for implementing a polyphase filter. If the cost of the symmetry multiplexers is included, the fused MCM is smaller only for $M = 8$. As the length increases from 44 with the decimation factor $M = 4$ to 88 with $M = 8$, the cost of the non-reconfigurable MCM increases faster than the cost of the fused decimation filter, which increases only 26% from $M = 4$ to $M = 8$. As seen in Figure 4.3, the results from the distinct MCM blocks have to be added together before they are fed to the adders in the SA block. In the reconfigurable MCM block operating at higher clock frequency, the addition is executed over time. The adders for summing the results are removed but a multiplexer is required for each SA. The estimated cost reduction increases linearly with the decimation factor to more than three times the size of the single non-configurable MCM block implementation for $M = 8$. Based on these figures, it is clearly advantageous to implement a polyphase FIR decimator at higher clock frequency to reduce the cost mainly due to the structural adder. The proposed algorithm further reduces the area cost by reusing the arithmetic elements in the MCM block by up to 17%.

4.4 Chapter Summary

This chapter presented a thorough analysis of the paper [23] about common subexpression elimination using binary representation to debunk the myths about the superiority of binary representation over canonical signed digits (CSD) for common subexpression elimination (CSE). The quantitative figures used to justify the merits of binary representation over CSD were shown to be inaccurate. From mere LO optimization perspective, slight statistical advantage of binary over CSD was demonstrated in [88], [151] and [20] but this minor advantage had been unduly amplified in [23] by incomplete analysis and partial comparisons. It was shown that the LD penalty of binary CSE had been deemphasized by the errors in the reported LD. Its contradictions with reference to the latest experimentation of binary, CSD and MSD number representations for CSE were pointed out. Upon correcting the error in the reported filter lengths for different stopband attenuations of D-AMPS specification, the LO and LD data of the CSE algorithms were recalculated using the corrected filter coefficient sets.

A detailed analysis of the CSD space was followed up to find out possible ways to add two numbers with minimal LD. The study concluded that CSE algorithm cannot find all beneficial shared patterns from any SD representation of the coefficient set as the nonzero digits in the addends may not be visible as they can be hidden, converted to different SD and switched position from the sum.

A minimal logic depth MCM optimization technique was proposed based on this detailed analysis of many possible ways to build an adder tree for a constant multiplication from the CSD space. The proposed new method requires less adders than CSE methods with the same minimal logic depth. Comparing to the AG algorithms, its number of adders increases only marginally but it has the minimum logic depth for all the coefficient multipliers. The small LO coupled with the minimal LD reduces the dynamic power of the proposed method by a factor of up to 3.4 times estimated by the Glitch Path Count and Glitch Path Score.

To optimize polyphase FIR filters and other multiple output MCM problems, an algorithm which implements optimized reconfigurable multiple constant multiplication was also presented. It is capable of generating a reconfigurable block that produces the product of an input multiplied by several fundamentals at any time. The algorithm is based on the minimal logic depth MCM algorithm and the evaluation results show some relation between filter length, decimation factor and MUX cost in polyphase filters, which hold promises for further optimization.

Chapter 5

Bit-Level MCM Optimization

One characteristic of the current approach to the optimization of MCM is that it is input independent and hence, the optimized shift-and-add network can be used for any input bit width. This also implies that some input variable correlated redundancy that is only visible at the bit-level will not be disclosed and cannot be eliminated. In Section 2.1.4, it was shown that the addition of two correlated signals contains some redundancy that does not exist in a generic addition. To exploit this redundancy is hard if the shift-and-add structure is to be preserved. If the MCM problem is solved differently, some additional optimization is possible. Two such approaches are presented in this chapter.

The advancement of FPGA technology enables the implementation of complex MCM instances on FPGA, but the shift-and-add network implementation does not make full use of the fundamental resources of FPGA, like the LUTs. Since bit-serial implementation optimized for FPGA is slow, an attempt for the bit-parallel LUT-based implementation of single constant multiplication has been made [152]. The first section of this chapter presents an extension to the LUT-based multiplication method for FPGA to multiple constant multiplications. Our study reveals an interesting and unexpected outcome that the maximum number of LUTs required can be limited far below the theoretical maximum by mere enumeration of all possible output combinations. If the multiplicands have a large bit width, this method has another advantage that the delay is maintained at the delay of a LUT and a CPA of length equal to the length of the product.

The second approach starts with a general purpose multiplier that contains a partial product matrix. Due to the constant, certain partial products can be eliminated. This was proposed for single constant multiplication by Pai *et al.* [153], [154] and is extended for MCM in this chapter. Two upper bounds for the number of partial product rows were derived. The general upper bound depends only on the bit width of the variable and the specific upper bound depends on both the bit width of the variable and the number of nonzero digits of the CSD encoded constant. Their implications on the logic depths of the carry propagate adder and the carry save adder tree implementation are analyzed. It is shown that if the constant requires more nonzero digits to be represented than the bit width of the input, the number of partial product rows can be reduced. The fact that the resulting implementation is faster and simpler than a direct CSD implementation is interesting. The improvements in terms of the maximal possible number and average number of partial product rows are also shown for some constant coefficient FIR filters examples.

5.1 Bit-Parallel Multiple Constant Multiplication using Look-Up Tables on FPGA

With the advancement of technology for FPGA, it is now possible to implement FIR filter on FPGAs at high speed and using bit-parallel arithmetic. For high-speed applications, bit-serial and distributed arithmetic implementations are not fast enough and the present shift-and-add optimization does not make full use of the FPGA fabrics. High-speed shift-and-add implementations based on CSA provide very high speed addition in ASIC but the mapping of CSA adders to FPGA is suboptimal, as one slice containing two LUTs is required per CSA. A CPA based on FAs allows fast carry logic, and two FAs can be fit into one logic slice. The reduction of slice utilization in FPGAs for the same filter requires the use of carry chain, which can be slower. Sometimes, the LUTs are not fully utilized.

Different approaches to single constant multiplication for many DSP algorithms have been studied [152], [155], [156]. Wirthlin [152] introduces a method to execute a constant multiplication on FPGA using LUTs for bit-parallel realization. The main idea is to split the input signal into several segments and then use 4-bit LUTs to generate the coefficient multiplier output bits. The most significant segment is viewed as a signed variable while the remaining parts are viewed as unsigned, as shown in Eq. (5) of [152]. To obtain the final result, the intermediate results from the split input signal are added by a CPA. Wirthlin noted the existence of some redundancies in a single constant multiplication [152] and categorized them into three different types. First, an output bit of the LUT can be always zero, which does not require any LUT. Second, two or more LUTs produce the same output, which can be replaced by a single LUT and appropriate routing. Third, the output of a LUT is identical to one of its inputs, which can be wired through directly from input to output without any LUT. In [155] a memory based approach for the implementation FIR filters is introduced. It uses a different approach to reduce the size of the LUT, which involves barrel shifters and adder-subtractors as well as control signals.

The main building blocks of FPGA are small LUTs, usually with four inputs. Newer generations of FPGA can have five or six input LUTs. CPA are mapped to slices using the high-speed carry propagation circuits within the FPGA, but this does not always lead to better utilization rate of the FPGA resources. It was shown in [152] that for single constant multiplication, the logic block resources in an FPGA are better utilized if a combination of LUTs and a few smaller input CPAs is used rather than a CPA-only implementation. Using the work of Wirthlin [152] as starting point, the reduction possibilities of sharing bit-slice LUTs for MCM are analyzed. Our analysis on the required LUTs revealed that when this concept was applied to MCM, many LUT outputs are identical and the number of distinct LUTs is limited and grows far slower with increasing LUT's bit width than expected, which contradicts the LUT estimates given in [155]. Based on this finding, an algorithm for the MCM optimization is proposed.

TABLE 5.1: LUTs Required under Different Conditions for Constants from 1 to 2^{12}

	Input	**1-bit**	**2-bit**	**3-bit**	**4-bit**	**5-bit**	**6-bit**	**7-bit**	**8-bit**
Output Bits	Unsigned	24575	26622	28669	30716	32763	34810	36857	38904
	Signed	24575	26622	28669	30716	32763	34810	36857	38904
[152] Single Constant	Unsigned	2047	9559	18527	24074	27572	30212	32531	34706
	Signed	2047	5909	14641	21543	25552	28300	30583	32720
Multiple Constants	Unsigned	2	8	36	144	616	2456	9598	15788
	Signed	2	6	16	52	176	680	2584	8295
Required LUTs	Unsigned	0	5	32	139	610	2359	5926	9887
	Signed	0	3	12	47	170	673	2470	6028

5.1.1 LUT based Multiplication of Multiple Constants

The number of n-bit LUTs required to obtain all products from the multiplications of an n-bit variable and all the odd numbers in the range of 1 to 2^{12} are shown in Table 5.1. For each category listed in the first column of Table 5.1, the analysis is performed for both signed and unsigned inputs. The first category refers to the total number of LUTs required without any optimization, which is equal to the number of output bits required to represent all possible products. For every input bit increment, there will be 2047 products with an output bit width incremented by one, which explains why there is a linear increment of the number of LUTs. The category 'Single Constant' refers to the number of LUTs required by applying the method of [152] to eliminate the redundant LUTs for each constant multiplication independently. The increment in the number of LUTs with its input bits is big for small n but tapers as n increases. This is because of the likelihood of finding identical output bits of the same weight among the products of an n-bit variable and a single constant decreases with increasing n. The category 'Mulitple Constants' indicates the number of LUTs required by considering the redundancies among all the constant multiplications as a whole. The category 'Required LUTs' is obtained by further removing a LUT with all zero entries and n LUTs that have outputs identical to those of the address bits. Table 5.2 lists the content of the LUTs in the Category 'Multiple Constants' for 3-bit signed and unsigned inputs. Each row represents the product bits obtained from the multiplication of one input combination of the variable and all odd integers from 1 to 4095, after removing duplicated columns. Each column corresponds to one 3-bit LUT. The first column of both signed and unsigned input cases corresponds to the trivial all zeros LUT. The columns highlighted in bold print are LUTs whose output bits can be obtained by directly hardwiring them to the input bits. Thus, only 12 LUTs for the signed case and 32 LUTs for the unsigned case are required. From Table 5.1, as n increases, the number of LUTs actually needed to implement all the constant multiplications increases almost linearly with a gradient of approximately four. This is rather unexpected as intuitively, the number of possible LUTs grows exponentially with n. The number of input combinations (rows) r of an integer variable is $r = 2^n$. Each of these r possible outputs of a LUT, can be either 0 or 1. Excluding the exceptional case of the zero input, where the output is always zero for any constant, an

exhaustive enumeration for the number of possible LUT outputs s gives

$$s = 2^{r-1} = 2^{2^n} \cdot 2^{-1} = 2^{2^n-1} \tag{5.1}$$

For $n = 8$, $s \approx 5.79 \cdot 10^{76}$. This number is much higher than the LUTs that are actually required. From Table 5.1, the number of LUTs required is only slightly more than $u = 4^n \cdot 2^{-1}$ for the unsigned case but slightly less for the signed case. The reason behind this is that many of these LUT output patterns are illegitimate under the multiplication of an n-bit variable with all odd integer constants. Generally, most of these legitimate patterns exhibit certain regularity. For example it is not possible to have an LUT with output $[01010100]^T$ for $n = 3$. If the last row is a 1, i.e., $[01010101]^T$, the output pattern becomes legitimate, which corresponds to the 11-th column for the signed input in Table 5.2. Patterns with only a single '1' also rarely occur. Only one case is found in Table 5.2, which appears in the second column for the unsigned input.

TABLE 5.2: Outputs for the Required 3-bit Input LUTs for the Signed and Unsigned Cases

Input	Signed	Unsigned
000	**0**0**0**000**0**000**0**00000	**0**0000**0**0000000000**0**000000**0**000000000000
001	**0**0**0**000**0**011**1**11111	**0**0000**0**0000000000**0**011111**1**111111111111
010	**0**0**0**011**1**100**0**01111	**0**0000**0**0001111111**1**000000**0**000111111111
011	**0**0**0**100**1**101**1**10011	**0**0000**0**1110000001**1**100011**1**1110001111111
100	**0**0**1**001**0**100**0**10101	**0**0001**1**1110011110**0**0111100**0**011000001111
101	**0**1**1**011**0**010**1**01100	**0**0011**1**0011100010**0**1011011**1**000110000111
110	**0**1**1**100**1**011**0**11000	**0**0111**1**0000001101**1**100010**0**111111000011
111	**0**1**1**111**1**100**1**00000	**0**1110**1**0100110110**1**0101001**1**001101010001

The difference in the number of required LUTs between the signed and unsigned input cases is due to the symmetry inherent in the range of signed integer number, which causes the range of magnitudes of the products to shrink. The fact is manifested in Table 5.1 where the number of required LUTs for the signed case of $n + 1$ approximates that of the unsigned case of n for $n > 3$.

The number of required LUTs shown in Table 5.1 are accurate for $n \leq 5$ but is likely to be underestimated for $n > 6$. This is because if the bit width m used to represent the constants is smaller than $2n + 1$, some overlapping bit patterns in the partial products will not be captured. Therefore, the figures presented for the 7- and 8-bit LUTs are likely to be too small. By increasing the range of odd integer constants to 2^{14}, the numbers of LUTs required for $n =$ 6, 7 and 8 are 2449, 9590 and 23770, respectively for the unsigned input and 673, 2576 and 9813, respectively for the signed input. When $m = 14$, the number of required LUTs reported for $n = 6$ will be accurate but it is still underestimated for $n = 7$ and 8.

To show the actual number of LUTs required on practical MCM problem, the filters from [21] and the 695 tap filter from [82] using four different coefficient bit widths were analyzed. All

filter coefficients are available in [122]. The results are shown in Table 5.3, where the columns '[152]' and 'Prop.' show respectively the numbers of 4-bit LUTs required when each constant multiplication is considered separately and when all constant multiplications are considered collectively. The percentage reduction is given in the column '%diff.'. Comparing with the required LUTs for $n = 4$ of Table 5.1, almost all possible LUTs are required for the case of signed input and between 48% and 82% of all possible LUTs are required for the case of unsigned input for the filters from [21]. On average, the number of LUTs required is reduced by 86.5% and 72.9% over [152] for the signed and the unsigned cases, respectively. The 695 tap filter example shows that increasing the constant bit width beyond 16-bit and the number of constants beyond 100 increases the number of LUTs required for [152], but the number of LUTs required in 'Prop.' only increases marginally from 16- to 20-bit and remains constant for coefficient bit width of 20- and 24-bit. For an input signal of 8-bit, an adder for each constant coefficient is needed to generate the final output from the LUTs. Adding an additional coefficient only requires one more adder, which is optimal in view of the lower bound for the MCM implementation given in [18]

TABLE 5.3: Comparison of LUT counts of [152] and proposed methods for 11 filters from [21] and a 695 tap filter from [82] with different coefficient bit widths

Filter	**unique coeff.**	**Signed**			**Unsigned**		
		[152]	**Prop.**	**%diff.**	**[152]**	**Prop.**	**%diff.**
1	19	263	43	83.7%	263	87	66.9%
2	39	578	47	91.9%	578	114	80.3%
3	14	174	42	75.9%	174	73	58.0%
4	33	427	46	89.2%	427	108	74.7%
5	18	232	46	80.2%	232	88	62.1%
6	36	474	47	90.1%	474	102	78.5%
7	19	242	42	82.6%	242	73	69.8%
8	29	407	46	88.7%	407	100	75.4%
9	14	252	45	82.1%	252	75	70.2%
10	16	398	46	88.4%	398	93	76.6%
11	12	173	39	77.5%	173	67	61.3%
Avg.	*22.6*	*329.1*	*44.5*	*86.5%*	*329.1*	*89.1*	*72.9%*
12-bit	32	320	44	86.3%	320	86	73.1%
16-bit	109	1270	46	96.4%	1270	131	89.7%
20-bit	271	3729	47	98.7%	3729	139	96.3%
24-bit	345	5937	47	99.2%	5937	139	97.7%

One important fact that was not mentioned in [152] is that using LUT-based multiplication shortens the delay compared with a CPA-based implementation when the CSD encoded constant contains many non zero digits. The delay of the latter is determined by the depth of the adder tree, which is given by $\lceil \log_2 (S(C)) \rceil$, where $S(C)$ is the number of nonzero digits of the

CSD constant C. If the input bit width is 8-bit and the signal is split into two 4-bit signals, two LUT blocks and one adder per constant are required. This delay is approximately equal to the delay of two cascaded adders. This means that for all cases where $S(C) > 4$, the delay will be reduced by using the proposed LUT technique.

5.1.2 Input Signal Splitting Considerations

If the bit width of the input signal is longer than that of the LUT, the input signal needs to be divided into two or more parts. For mapping onto FPGA, it makes sense to fit the bit width of the partial input signal to the size of the LUT or other size that will save on the number of CPAs required for LUT output merging. 4-bit LUT is typical but newer FPGAs could have five or six input LUTs. Beyond six inputs the number of possible LUTs is too big due to the lower number of redundant columns in the product bit matrix.

If the input has an odd bit width, it is beneficial to allocate the extra bit to the higher order part where the signed bit is located so that the maximum number of required LUTs is smaller. For a 7-bit input, the limit of LUTs for a $4 + 3$ splitting is $47 + 32 = 79$ while a splitting of $3 + 4$ requires $12 + 139 = 151$ LUTs maximally. The downside of allocating more bits to the higher order side is that a longer CPA is required for merging. No addition is required for the 3 and 4 LSBs for the $4 + 3$ and $3 + 4$ splitting respectively. The number of constants c can be used to decide which splitting is more beneficial. For the example, it is lucrative to allocate one more bit to the upper part as long as $c < 151 - 79 = 72$. For other bit widths, the same calculation can be applied. In general it is not beneficial to have more than 1 bit difference between the higher and lower order parts.

5.1.3 Simulation Results

The proposed LUT optimization for MCM has been coded in MatLab and integrated into our sysFIR automatic VHDL generator for FIR filter implementation (Chapter 3). Generic signed and unsigned adders were used and the design was functionally verified by ModelSim. An 8-bit input was assumed and it was split into two 4-bit signals. The 64-bit version of Xilinx ISE 11.2 (L.46) was used to compile and map the design onto the Xilinx Virtex-4 FPGA, xc4vlx200-10ff1513. The designs were optimized for 'speed' with effort level '2'.

Typically, one 4-bit LUT block will be consumed for each column of a 4-input product bit matrix for the MCM. As RCA adders are used, they can be implemented by using the fast carry propagation path. A reasonable estimate of approximately one LUT block per output bit is made. Using these estimates and the 'Macro Statistics' from the synthesis report of Xilinx ISE, the results in terms of LUT equivalent counts were generated for our proposed method, 'Prop.', the baseline implementation of CSD coefficient multipliers, 'CSD', and the minimal logic depth graph based MCM algorithm from [58], 'MinLD' and are given in Table 5.5. The MinLD algorithm was chosen because it outperforms all MCM CSE algorithms with minimal logic depth at a slightly higher adder cost than some graph based algorithms. For the filters from [21], the proposed LUT-based method for MCM has a clear advantage over CSD and on

TABLE 5.4: Simulation results for Xilinx Virtex-4 for 11 filters from [21] and a 695 tap filter from [82] of different coefficient bit widths

Filter	**unique coeff.**	**CSD**		**MinLD**		**Proposed**		**Red. over CSD**		**Red. over MinLD**	
		Slices	**Delay**	**Slices**	**Delay**	**Slices**	**Delay**	**Slices**	**Delay**	**Slices**	**Delay**
1	19	223	13.8	217	15.5	207	9.4	7.2%	31.8%	4.6%	39.3%
2	39	482	15.0	405	14.9	392	10.3	18.7%	31.8%	3.2%	30.9%
3	14	163	13.1	133	13.6	158	9.2	3.1%	29.8%	-18.8%	32.6%
4	33	333	14.3	308	15.2	316	10.1	5.1%	29.7%	-2.6%	33.5%
5	18	216	13.3	196	14.6	193	9.4	10.6%	29.3%	1.5%	35.3%
6	36	385	14.8	317	15.5	328	10.0	14.8%	32.3%	-3.5%	35.6%
7	19	200	13.6	185	15.4	187	9.4	6.5%	30.9%	-1.1%	39.0%
8	29	341	15.1	301	15.1	290	9.8	15.0%	35.4%	3.7%	35.3%
9	14	214	15.1	187	13.9	199	9.4	7.0%	37.7%	-6.4%	32.4%
10	16	334	14.5	293	15.4	284	9.9	15.0%	31.9%	3.1%	35.7%
11	12	174	13.5	150	12.7	148	9.2	14.9%	32.0%	1.3%	27.8%
Average	*22.6*	*278.6*	*14.2*	*244.7*	*14.7*	*245.6*	*9.6*	*11.8%*	*32.2%*	*-0.4%*	*34.5%*
12-bit	32	225	17.0	226	18.0	245	13.5	-8.9%	20.8%	-8.4%	24.9%
16-bit	109	783	19.5	699	21.3	763	14.2	2.6%	27.4%	-9.2%	33.4%
20-bit	271	2312	21.2	1867	22.5	2042	14.4	11.7%	32.2%	-9.4%	35.9%
24-bit	345	4309	23.5	3388	25.0	3146	15.2	27.0%	35.4%	7.1%	39.2%

average, it consumes 15.6% less LUT equivalents than the designs synthesized by [58]. The results of the DAmp filter from [82] show that the savings increase with increasing number of filter coefficients and precision of coefficients.

TABLE 5.5: Comparison of LUT equivalent counts for 11 filters from [21] and a 695 tap filter from [82] of different coefficient bit widths

Filter	**unique coeff.**	**LUT eqiv. count**			**% Reduction by Prop.**	
		CSD	**MinLD**	**Prop.**	**over CSD**	**over MinLD**
1	19	874	483	393	55.0%	18.6%
2	39	2120	926	739	65.1%	20.2%
3	14	535	300	289	46.0%	3.7%
4	33	1401	690	581	58.5%	15.8%
5	18	758	389	366	51.7%	5.9%
6	36	1612	731	623	61.4%	14.8%
7	19	757	408	357	52.8%	12.5%
8	29	1390	671	553	60.2%	17.6%
9	14	803	428	372	53.7%	13.1%
10	16	1366	696	537	60.7%	22.8%
11	12	590	309	279	52.7%	9.7%
Average	*22.6*	*1109.6*	*548.3*	*462.6*	*58.3%*	*15.6%*
12-bit	32	848	459	450	46.9%	2.0%
16-bit	109	3769	1768	1447	61.6%	18.2%
20-bit	271	12 267	5028	3915	68.1%	22.1%
24-bit	345	22 375	8567	6123	72.6%	28.5%

The MCM block of a FIR filter has a large number of outputs, which exceeds the number of I/O pins of the Virtex-4 FPGA for most of the example filters used in Table 5.4. Therefore, the synthesis results of Xilinx XST before mapping were shown. In terms of logic slices used, the simulation results show a marginal reduction of the proposed method compared with CSD, but for the 695 tap DAmps filter with 12-bit constants the proposed method requires slightly more slices. The slice count of the proposed method is somewhat inferior when it is compared with that of MinLD. This is due to the fact that the advanced low-level optimization performed by Xilinx XST can reduce the LUT equivalents for adders, while the LUT equivalents for the proposed LUTs cannot be further optimized. However, the delay of the proposed LUT-based method is reduced by approximately 1/3 comparing with those of CSD or MinLD. Although the reported delays may not be accurate before routing (which cannot be performed due to the IO constraint), they provide a relatively good indication of the trend. The shortened critical path is likely to reduce the power consumption.

5.2 Reduction of Partial Product Matrix for High-Speed Single or Multiple Constant Multiplication

The algorithms for optimization of MCM problems generally use CPA in the shift-and-add network. For high-speed implementation, the use of CSA is preferred [53]. The use of CSA within the MCM block was first considered by Bartlett *et al.* [157] and Gustafsson *et al.* [76]. In [157] it is found that the use of CSA reduces the power consumption for direct form and transposed direct form FIR filters. Adapting the solutions of MCM algorithms to CSA implementation is not optimal as the optimization originally targets two input adders which cannot be directly mapped to the three input CSA adders. In [76], a graph based minimum-adder CSA based constant multiplier is investigated and the possible adder graphs of up to five adders are presented. The approach is further refined in [158]. Using these graph reductions, it is shown that the number of adders can be reduced for constants with a bit width of more than nine compared to direct CSA implementation. The ideas from [76] are used in [77] to establish a MCM optimization algorithm which directly targets for CSA implementation. The results presented in [77] show that a direct CSA-based optimization is superior to a CPA based optimization that is remapped to CSA, but the CSA implementation requires more adders for coefficients with more than nine bits. The main difference between CPA and CSA based optimizations noted in [77] is that CPA usually consumes two intermediate values in one addition while CSA tends to add one intermediate value with a single input.

In [159], the problem of finding the minimum number of CSAs to implement a MCM is converted to a 0-1 ILP problem and then solved either exactly or heuristically. The conversion to a 0-1 ILP problem gives the algorithm the ability to solve the problem in Binary, CSD, MSD or any other number representation space. The algorithms of [159] outperform that of [77]. A common observation of the algorithms [76], [77], [157]–[159] is that they solve the problem at the arithmetic operator level, which implies that there is room for bit-level optimization.

CSAs have been widely employed in the design of fast two-variable multiplier, $X \times Y$, where $W(X)$, $W(Y)$ are the bit width of the two variable inputs, X and Y in two's complement representation. The multiplier can be implemented using $\min(W(X), W(Y))$ partial products rows, which are then reduced by a CSA tree and finally summed by a CPA. Using Booth recoding, the number of partial product rows can be reduced at the cost of additional Booth encoder and partial product generator circuits. Hashemian [160] introduced yet another possibility. Based on a ternary number system called Bit-Signed (BS), the CSD number representation is used to derive a CSD based multiplier. BS is a subset of Signed Digit (SD) [45]. In SD, a number a is represented by $a_{n-1}, \ldots, a_2, a_1, a_0$ with $a_i \in \{0, \pm 1, \pm 2, \ldots, \pm(r-1)\}$. In BS, the set of possible digits is restricted to only $\{\bar{1}, 0, 1\}$, where $\bar{1}$ represents -1, by setting $r = 2$. The CSD based multiplier includes a direct two's complement to CSD encoding and uses the minimal possible number of additions or subtractions, which is less than $\min(W(N), W(M))$. The multiplier consumes the bits from one multiplicand sequentially and is therefore more suitable for software or bit-serial implementation. This is because unfolding for parallel addition is inefficient due to the inevitable carry propagation in the CSD conversion. A similar approach

is presented in [161], where the algorithm is implemented in distributed arithmetic. A fully parallel high speed implementation is likewise not feasible. In [162], the authors compare the implementation of constant coefficient multipliers using CSD and Booth Recoding. Both constant codings reduce the number of partial products and hold a chance to find reusable three digit patterns in the recoded constant. They conclude that the use of CSD is advantageous in almost all cases, as CSD coding of the constant ensures that the number of partial products is minimal.

Pai *et al.* [153], [154] introduce a low power single constant coefficient multiplier generator, which uses CSD to represent the constant, and use bit-level equations to reduce the sign extension length. The reduced partial product matrix generated by the multiplier generator in [153], [154] is summed by a CSA tree and finally added by a CPA. This scheme allows the power consumption of a constant multiplier to be reduced without increasing the critical path delay. Based on the methods introduced in [153], [154], the following sections present a high-speed single constant multiplier and multiple constant multiplier. Also the upper and lower bounds for the partial product matrix height are proposed and analyzed. The implications of these bounds are discussed for constant multiplication and the required LD based on the statistics obtained for numbers below 2^{18}.

5.2.1 Height of Partial Product Matrix

For an unencoded digital multiplier, the achievable minimum number of partial product rows is given by $\text{rows}(X, Y) = \min(W(X), W(Y)) + 1$, where $+1$ is required only if the correction vector is used to avoid sign extension. Booth recoding or CSD recoding can be used to reduce this minimum number of rows. For a single or multiple constant multiplication, the focus of the optimization lies primarily on the constant C, while the input signal X is treated as a variable with its bit width $W(X)$ defined at design time. The recoding of the constant with CSD is straightforward and the number of partial product rows is estimated to be $\text{rows}(C, X) \approx S(C)/2 + 1$, which is simple for an unsigned input X. However, for a signed input X, the number of partial product rows can be reduced by transforming correlated bits (x_i and $\overline{x}_i$) and binary constants 0 and 1 in the same column into equivalent sum bits in the same column and carry bits in the next significant bit column.

The reduction rules can be defined by applying the HA addition on correlated input bits and binary constants. A Half Adder (HA) adds two input bits x and y and produces a sum bit s and a carry bit c, i.e.,

$$s = x \oplus y, \quad c = x \cdot y \tag{5.2}$$

where s has the same weight as the inputs and c is weighted one bit more significant. As noted in [153], [154] the following HA rules can be used to reduce the column height of the partial

product array under certain input conditions:

$$s = x \oplus x = 0,\ c = x \cdot x = x \tag{5.3}$$

$$s = x \oplus \overline{x} = 1,\ c = x \cdot \overline{x} = 0 \tag{5.4}$$

$$s = x \oplus 1 = \overline{x},\ c = x \cdot 1 = x \tag{5.5}$$

$$s = \overline{x} \oplus 1 = x,\ c = \overline{x} \cdot 1 = \overline{x} \tag{5.6}$$

$$s = 1 \oplus 1 = 0,\ c = 1 \cdot 1 = 1 \tag{5.7}$$

where $\overline{x}$ denotes the bit negation of x. If more than two correlated or constant input bits exist in the same column, Rules (5.3)-(5.7) can be applied repeatedly.

Based on the above transformation, the following theorem is formulated for the bounds of the number of partial products in the partial product array:

Theorem 1.
Trivial Exact Cases:

$$\text{rows}(C, X) = \begin{cases} 0 & \text{if } S(C) = 0 \\ 1 & \text{if } S(C) = 1,\ C > 0 \\ 2 & \text{if } S(C) = 1,\ C < 0 \end{cases} \tag{5.8}$$

Lower Bound ($S(C) > 1$):

$$\text{rows}(C, X) \geq 2 \tag{5.9}$$

Upper Bound ($S(C) > 1$):

- *Without exploring the property of CSD encoding,*

$$\text{rows}(C, X) \leq W(X) + 1 \tag{5.10}$$

- *By exploiting the property of CSD encoding,*

$$\text{rows}(C, X) \leq \max\left(\left\lceil \frac{W(X) - 1}{2} \right\rceil + 1, 2\right) \tag{5.11}$$

- *With the knowledge of the constant, C*

$$\text{rows}(C, X) \leq \max_n \left(\hat{C} \star d\right)[n] + 1 \tag{5.12}$$

where $\left(\hat{C} \star d\right)[n]$ is the cross-correlation of $\hat{C}$ with a constant string d defined by $\left(\hat{C} \star d\right)[n] = \left(\sum_{m=-\infty}^{\infty} |c_m| \times d_{m+n}\right)$, with $d_i = \begin{cases} 1 & \text{if } 1 \leq i \leq W(X) - 1 \\ 0 & \text{otherwise} \end{cases}$ and c_i is the zero padded CSD encoded constant C

Proof of (5.9). A signed input requires at least two bits, one for the sign, one for the magni-

tude, and sign extension is required. If $S(C) > 1$, i.e., the constant has more than one nonzero digit, there will be at least one partial product column with two distinct inputs, x_i and x_j. This means that Rules (5.3)-(5.7) cannot be applied and no column height reduction is possible. If the constant is negative, a constant 1 is added at the LSB position of the partial product due to the negation (two's complement) of the partial product generated by the sign digit of the constant. Hence, $\text{rows}(C, X) \geq 3$ is required. This can be avoided by inverting the constant and changing the adder in the next stage to a subtractor, which is hereafter considered to be the case. □

Proof of (5.10). Applying Rules (5.3)-(5.7) from the LSB to the MSB of the sign extended partial product matrix eliminates all input bits that exist more than once in a column. As only $W(X)$ distinct inputs are present and the constant 1s due to the negation of partial products can be summed to no more than one 1 per column, the maximum number of partial product rows is less than or equal to $W(X) + 1$. □

Proof of (5.11). As the constant C is encoded in CSD, two consecutive digits cannot be both nonzero, $c_i \times c_{i-1} = 0$. This implies that x_i and x_{i-1} ($x_i, x_{i-1} \neq x_{MSB}$) can never be in the same column. The shifts of the multiplicand X to form the partial products according to the weight of the CSD-encoded bits guarantee that $x_i \neq x_{MSB}$ cannot exist twice in the same row. The sign, x_{MSB}, has to be extended up to the output bit width and therefore can exist in any column to the left of its first occurrence. Based on the fact that x_i and x_{i-1} can never exist in the same column, and that x_{MSB} can coexist with any x_i in the same column, no two consecutive binary variables from $\{x_0, \ldots, x_{MSB-1}\}$ can appear in any column. This maximal number of multiplicand bits plus x_{MSB} plus a constant 1 can exist in one column. If both x_{MSB} and a constant 1 exist in the same column, they can be merged by using (5.5) or (5.6) such that only one row is required for x_{MSB} or 1. Merging a constant 1 with $x_i \in \{x_0, \ldots, x_{MSB-1}\}$ by the application of Rule (5.5) or (5.6) is only to be done when the next column can accommodate the carry bit generated by the application of Rule (5.5) or (5.6) without increasing the maximum height or when the rules can be applied again to further absorb the carry bit. Therefore at most half of the binary variables in $\{x_0, \ldots, x_{MSB-1}\}$ plus either x_{MSB} or a constant 1 can exist simultaneously in any one column of the partial product matrix. □

The upper bound (5.11) is independent of the constant C which means that even if only the input bit width is known but not the constant C it is possible to calculate an upper bound for the maximal LD for the multiplication by the constant C. This can easily be extendend to of a MCM block with multiple constants. The upper bound (5.11) can be further lowered when the constant C and its CSD representation are known.

Proof of (5.12). The upper bound in (5.11) can be further reduced when the constant C and its CSD representation are known. A CSD encoded number has the highest possible number of nonzero digits when it is composed of alternating zero and nonzero digits. This case is relatively rare as the average number of nonzero digits in CSD is 1/3. Knowledge of the

number of nonzero digits in m consecutive digits of a CSD number can be used to reduce the upper bound of the partial products. One way to find the maximal number of nonzero digit count in a CSD substring is to find the maximum cross correlation of the absolute digit value of the CSD encoded constant and a string of 1 digits of length n, both zero padded. The same arguments for the proof of (5.11) can be applied, except that the maximum number of nonzero digits in a given span further restricts the maximum possible number of occurrences of x_i in the same column. This is reflected in (5.12). □

An example for the partial product matrix of a constant multiplier with $X = x_3x_2x_1x_0$ and $C = 1593$ is shown in Figure 5.1. The CSD representation of 1593 is $10\bar{1}001001\bar{1}001$. Using (5.11) or (5.12), a maximum height of three is obtained. However, by inspecting the sections of C where the maximum cross correlation occurs, if none of the leading digits is negative, the additional row for the negation constant is not necessary and the reduced partial product matrix only requires two rows. The original partial product matrix is shown in Figure 5.1 while Figure 5.2 shows the partial product matrix reduced by Rules (5.3)-(5.7).

x_3	x_3	x_3	x_3	x_3	x_3	x_3	x_3	x_3	x_3	x_3	x_3	x_2	x_1	x_0
$\overline{x_3}$	$\overline{x_3}$	$\overline{x_3}$	$\overline{x_3}$	$\overline{x_3}$	$\overline{x_3}$	$\overline{x_3}$	$\overline{x_3}$	$\overline{x_3}$	$\overline{x_2}$	$\overline{x_1}$	$\overline{x_0}$			
x_3	x_3	x_3	x_3	x_3	x_3	x_2	x_1	x_0			1			
$\overline{x_3}$	$\overline{x_3}$	$\overline{x_3}$	$\overline{x_2}$	$\overline{x_1}$	$\overline{x_0}$									
x_3	x_2	x_1	x_0		1									

Figure 5.1: Partial Product Matrix for 4-bit Input Multiplied by 1593

1	$\overline{x_3}$	$\overline{x_3}$	$\overline{x_2}$	$\overline{x_1}$	$\overline{x_0}$	$\overline{x_3}$	$\overline{x_3}$	$\overline{x_3}$	$\overline{x_2}$	$\overline{x_1}$	$\overline{x_3}$	x_2	x_1	x_0
	x_2	x_1	x_0			x_2	x_1	x_0			$\overline{x_0}$			

Figure 5.2: Reduced Partial Product Matrix for 4-bit Input Multiplied by 1593

5.2.2 Implications from Upper Bound for CPA and CSA Additions

Using the aforementioned theorem, Equation (2.25) can be improved to

$$LD_{min}\left(C_i, X\right) = \min(\lceil \log_2\left(S(C_i)\right)\rceil, \lceil \log_2\left(\text{rows}(C_i, X)\right)\rceil) \tag{5.13}$$

which depends on the bit width of the input signal X. It is likely to improve the LD if the bit width of X is smaller than the nonzero count of the constants C_i. For example, a MCM block with an input bit width of seven can always be implemented with a logic depth of two, no matter how many bits are required to represent the constants.

Table 5.7 shows the maximum LD required for CPA based on Equation (5.11) with different input bit widths listed on the left column of the table and the nonzero digit count of the CSD coded constant listed on the top row of the table. The maximum LDs required are highlighted in italic for those combinations of $W(X)$ and $S(C)$ that are lower than the LD calculated based

TABLE 5.6: Number of Constants with a Given Partial Product Row Count for all odd Constants from 3 to $2^{18} - 1$

$W(X)$	Partial Product Row Count / Nonzero count for CSD									Reduced Rows		Reduced LD	
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	%	#	%	#
2-3	131 071	0	0	0	0	0	0	0	0	131 038	99.9%	131 038	99.9%
4	16 049	115 022	0	0	0	0	0	0	0	130 980	99.9%	129 872	99.0%
5	4216	126 855	0	0	0	0	0	0	0	130 926	99.8%	129 011	98.4%
6	1664	60 742	68 665	0	0	0	0	0	0	130 794	99.7%	128 405	97.9%
7	818	28 747	101 506	0	0	0	0	0	0	130 590	99.6%	128 004	97.6%
8	484	14 887	82 836	32 864	0	0	0	0	0	130 190	99.3%	101 517	77.4%
9	338	8653	57 520	64 560	0	0	0	0	0	129 480	98.7%	70 225	53.5%
10	252	5617	39 009	71 823	14 370	0	0	0	0	128 138	97.7%	48 506	37.0%
11	186	3846	26 975	64 824	35 240	0	0	0	0	125 696	95.8%	34 569	26.3%
12	135	2639	18 917	53 900	49 472	6008	0	0	0	121 342	92.5%	25 202	19.2%
13	97	1814	13 386	43 198	54 808	17 768	0	0	0	113 838	86.8%	18 770	14.3%
14	70	1251	9502	33 952	53 992	29 904	2400	0	0	101 374	77.3%	14 269	10.8%
15	52	875	6768	26 256	49 616	39 184	8320	0	0	81 922	62.5%	11 123	8.4%
16	41	638	4928	20 280	43 680	44 352	16 256	896	0	54 612	41.6%	8 128	6.2%
CSD	33	450	3276	13 728	33 264	44 352	28 800	6 912	256				

solely on CSD coding. For the commonly used bit widths of $\{8, 10, 12\}$, a LD reduction over CSD is guaranteed when $S(C) \geq 9$, which requires the constant to be at least 43691.

TABLE 5.7: Maximum CPA-LD Needed with the Application of Equation (5.13)

$W(X) \diagdown S(C)$	**1**	**2**	**3-4**	**5-8**	**9-16**	**17-32**
only Eq. (2.25)	0	1	2	3	4	5
1	0	1	***1***	***1***	***1***	***1***
2-3	0	1	***1***	***1***	***1***	***1***
4-7	0	1	2	***2***	***2***	***2***
8-15	0	1	2	3	***3***	***3***
16-31	0	1	2	3	4	***4***
32-63	0	1	2	3	4	5

The maximum heights of the CSA trees for different combinations of input bit widths and nonzero digit count of the constant are shown in Table 5.8. Again, the maximum heights of the CSA trees which are lower than the maximum heights of the CSA trees determined solely by CSD coding are highlighted. For the commonly used bit widths of 8 and 10, a reduction in CSA tree height is guaranteed when $S(C) \geq 10$ while for bit width of 12, reduction is guaranteed when $S(C) \geq 14$.

TABLE 5.8: Maximal CSA Tree Heights Needed Based on Equation (5.11)

$W(X) \diagdown S(C)$	**1-2**	**3**	**4**	**5-6**	**7-9**	**10-13**	**14-28**
based on CSD only	0	1	2	3	4	5	6
1-2	0	1	***0***	***0***	***0***	***0***	***0***
3	0	1	***1***	***1***	***1***	***1***	***1***
4-5	0	1	2	***2***	***2***	***2***	***2***
6-7	0	1	2	3	***3***	***3***	***3***
8-11	0	1	2	3	4	***4***	***4***
12-17	0	1	2	3	4	5	***5***
18-25	0	1	2	3	4	5	6

Tables 5.7 and 5.8 show the maximal possible number of adder levels in CPA and CSA additions, respectively. It should be noted that each level of CSA has the delay of only a FA but each level of CPA requires a full length carry propagation. The effective number of partial product rows can actually fall below the bound as demonstrated previously with the example of a 4-bit variable multiplied by a constant, $C = 1593$. For odd values of $n \in \{3, 5, \ldots, 2^{18} - 1\}$, the partial product matrices were generated and reduced by Rules (5.3)-(5.7) for input bit widths range from 2 to 16. The results are summarized in Table 5.6. For comparison, the last row shows the number of CSD coded constants that has the number of nonzero digits equal to the partial product row count. For all tested constant/input bit width combinations, a solution with equal or less rows than the upper bound predicted by Equation (5.12) were found. The

closer the input bit width to the constant bit width, the smaller the chance for optimization. For the commonly used bit widths of $8, 10$ and 12, there is a good chance to reduce the height of the partial product matrix for constants from 3 to $2^{18} - 1$. The larger the constant, the better the chance.

5.2.3 Experimental Results

Using the examples 'Filter 1' to 'Filter 5' from [21], the reductions in terms of the number of partial product rows with the application of Rules (5.3)-(5.7) were calculated for bit widths of 2 to 14. The results are shown in Table 5.9. The average and maximum numbers of partial product rows are provided for up to 9-bit inputs. When the reduced partial product matrices are added using either CPA or CSA adders, the power consumption will be reduced as there will be less glitches and shorter paths for the propagation of glitches, but the extent of reduction needs further investigation.

The sharing of HAs and FAs in the CSA implementation of 'Filter 1' from [21] with an 8-bit input was analyzed. The filter contains the following partial product columns: $3\times$ zero element, $79\times$ one element, $113\times$ two elements, $107\times$ three elements, $32\times$ four elements and $5\times$ five elements. A direct implementation of the first stage of the CSA adder tree by greedily reducing the bits in a column with either FAs or HAs wherever possible, requires 144 FAs and 113 HAs. Several identical HA or FA inputs can be found among the columns such that the number of columns is reduced to: 14 single element columns, 49 two element columns, 80 three element columns, 5 four element columns and no five element columns. This means that the number of FAs required are reduced by 41% and the number of HAs by 56%. A CPA based implementation reduces the adder count from 65 to 52, which is equivalent to a reduction of 20%, but the size of the CPA will deviate from a CPA/CSD based implementation as no sign extension is required here and the partial product matrix can be very sparse in the lower part. Further investigation into the CSD sharing, comparison with [76], [77], [157]–[159] and the potential sharing of intermediate CPA results is ongoing.

TABLE 5.9: Partial Product Row Count and Reduction for Example Filters from [21]

	Filter 1		**Filter 2**		**Filter 3**		**Filter 4**		**Filter 5**	
PP rows	**max**	**mean**	**max**	**mean**	**max**	**mean**	**max**	**mean**	**max**	**mean**
CSD	6	4.42	7	4.87	6	3.79	7	4.15	6	4.11
2-3	2 (67%)	2.00 (55%)	2 (71%)	2.00 (59%)	2 (67%)	2.00 (47%)	2 (71%)	2.00 (52%)	2 (67%)	2.00 (51%)
4	3 (50%)	2.68 (39%)	3 (57%)	2.72 (44%)	3 (50%)	2.43 (36%)	3 (57%)	2.64 (37%)	3 (50%)	2.72 (34%)
5	3 (50%)	2.95 (33%)	3 (57%)	2.87 (41%)	3 (50%)	2.57 (32%)	3 (57%)	2.76 (34%)	3 (50%)	2.83 (31%)
6	4 (33%)	3.26 (26%)	4 (43%)	3.28 (33%)	4 (33%)	2.71 (28%)	4 (43%)	3.09 (26%)	4 (33%)	3.22 (22%)
7	4 (33%)	3.32 (25%)	4 (43%)	3.59 (26%)	4 (33%)	3.07 (19%)	4 (43%)	3.27 (21%)	4 (33%)	3.44 (16%)
8	5 (17%)	3.74 (15%)	5 (29%)	3.90 (20%)	5 (17%)	3.21 (15%)	5 (29%)	3.46 (17%)	5 (17%)	3.67 (11%)
9	5 (17%)	3.95 (11%)	5 (29%)	4.18 (14%)	5 (17%)	3.43 (9%)	5 (29%)	3.73 (10%)	5 (17%)	3.89 (5%)
10	6 (0%)	4.26 (4%)	6 (14%)	4.36 (11%)	5 (17%)	3.57 (6%)	6 (14%)	3.88 (7%)	6 (0%)	4.06 (1%)
11	6 (0%)	4.37 (1%)	6 (14%)	4.56 (6%)	6 (0%)	3.71 (2%)	6 (14%)	4.03 (3%)	6 (0%)	4.06 (1%)
12	6 (0%)	4.42 (0%)	6 (14%)	4.74 (3%)	6 (0%)	3.79 (0%)	6 (14%)	4.09 (1%)	6 (0%)	4.06 (1%)
13	6 (0%)	4.42 (0%)	7 (0%)	4.85 (1%)	6 (0%)	3.79 (0%)	6 (14%)	4.09 (1%)	6 (0%)	4.11 (0%)
14	6 (0%)	4.42 (0%)	7 (0%)	4.85 (1%)	6 (0%)	3.79 (0%)	7 (0%)	4.15 (0%)	6 (0%)	4.11 (0%)

5.3 Chapter Summary

In this chapter two methods for bit-parallel MCM which is optimized at the bit-level were presented.

First, a LUT-based method for designing fully parallel multiple constant multiplication amenable to FPGA mapping was presented. It was shown that there exists an upper limit for the number of LUTs required for each input bit width. By splitting an 8-bit input signal into two 4-bit signals, simulation results for several FIR filters showed that the proposed method achieves the predicted hardware savings. The proposed LUT-based method is comparable with the current-art adder-based MCM optimization techniques in terms of the usage of logic slices and outperforms them by about 33% in terms of delay. The results also indicated that the proposed method is advantageous for MCM problems when the number of constants and their precision increase.

Second, an upper bound for constant multiplication was derived which reduces the maximum height of the partial product matrix to at most one more than half the bit width of the input variable irrespective of the value of the constant. A lower upper bound which relies on the input bit width and the number of nonzero digits of the constant is also derived if the constant is CSD encoded. Their implications on the logic depth for CPA and CSA adder implementation are examined. Based on the upper bound and a rule set that can be used to achieve the upper bound, the partial product height of several FIR filter examples were examined. Besides the reduction in the partial product height, which can reduce the logic depth, the number of adders/logic elements is also reduced by sharing. Using a FIR filter example with an 8-bit input, it is shown that the number of FAs and HAs required in the first stage of the CSA tree can be reduced by 41% and 56%, respectively.

Both approaches are promising for the problems where the input bit width is only single digit while the constant multipliers have larger bit width. One possible application can be found in digital filters that follow directly after a high speed ADC, which normally do not have high bit width, but are producing samples at a high bit rate. The most extreme case is where a sigma-delta ADC only produces a single digit output but does not require multiplication.

Chapter 6

Structural Adder Optimization

The redundancies in MCM block are mainly due to the multiplication of a single input signal with several constants. Such multiplication leads to additions of fully correlated signals. In the SA block, the signals to be added stem from different clock cycles and are considered to be independent, making the optimization of adders at word level impossible. At bit-level, the overall structure of the adder-delay line within the SA block reveals a number of properties that can be exploited for optimization. As the impulse response of FIR filter is a sinc-like function, the inputs from the MCM block has a $1/x$ envelope. The exact bit width required to represent the output of a structural adder grows towards the output of the filter till the middle tap but remains relatively constant from the middle tap to the output. For long filters, this phenomenon can be used for the area reduction of SA block directly and critical path reduction indirectly by splitting the data path and recombining it later. If offset representation is used, similar area and delay reduction can be achieved by forwarding the carry from one adder to the next instead of letting it propagate to the MSB at each adder. As this optimization is more localized, it may also benefit shorter filters. Pipelining to achieve higher throughput has been considered in MCM optimization but never for the SA block. In this chapter, a cost effective method to pipeline the SA block by adding only few registers to the existing registers in the adder-delay path is also proposed. Last but not least, based on the fact that the output bit width of a FIR filter is much larger than the input bit width, further area reduction of the SA block is possible at the cost of a small reduction in the output precision. The additionally introduced error can be contained to one LSB of the reduced bit width output, the saving in circuit area is still quite substantial.

The redundancies in the SA block may not be large relative to the area of SA, but the overall savings can be very significant. This is because the adders in the SA block are much larger than other adders in a FIR filter and consume the most area. With increasing filter length, the fraction of area [1] required for the SA block increases and the area saved by optimizing the structural adders alone could even exceed the area of the entire MCM block.

To the best of my knowledge, direct optimization of SA block has not been tackled other than my work [142], [145] or my co-authored work [44]. Indirect optimization via avoidance of sign extension [42]–[44] has been considered, but except for [44] quantitative area reduction in the SA block is not reported. In [43], different options for avoiding sign extension in the MCM block have been discussed, of which sign extension in the SA block is also eliminated by the offset

[1] area relative to the overall FIR filter

representation. For the structural adders after the middle tap to the output, this elimination of sign extension bits is quite significant as the bit widths of the partial sums are far bigger than those from the MCM block. In an earlier publication [42], a FIR filter in an all-digital modem system is analyzed as a whole with a sign extension avoidance technique called MSB-Fix. This technique is similar to the correction vector used in the partial product addition of two's complement multiplication. It inverts the MSB and sums up the corresponding correction values. This aggregated constant is added just before the last filter tap. As the partial sums are delayed in the transposed direct form FIR filters, it is highlighted in [42] that the first $N-1$ outputs are incorrect as the cumulative error has to propagate to the last tap. This is not a problem for the FIR filter in the all-digital modem of [42] with only $N = 20$ taps. As this system needs more than 20 clock cycles to start up, the first 19 incorrect outputs are not critical. In [142], it is argued that it can be a problem for generic FIR filter to flush the first $N-1$ incorrect outputs. This problem can be circumvented in [44] by initializing all delay elements to some specific values. The initial value of each register can be determined from the error introduced by the sign avoidance technique until the tap before the register. This error is equal to the value of the register if the filter is flushed with an all-zero input of at least $N-1$ bits long.

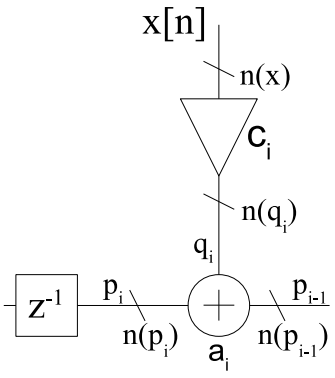


Figure 6.1: Structural Adder with Annotated Signal Paths

Figure 6.1 shows a schematic of a Structural Adder (SA) zoomed in from Figure 2.2. In Figure 6.1, a_i $(i = N-2, N-3, \ldots, 0)$ is the structural adder that adds the output of the register p_i to the output of the coefficient multiplier q_i in the tap delay line, and n(z) denotes the bit width of an arbitrary signal z.

The operands of a SA are binary numbers in two's complement or offset representation. Details for these representations were given in Section 2.1.6. After the middle coefficient of a filter, the bit width of the coefficient multiplier q_i is always smaller than that of the previous partial sum p_i, which results in long sign extension of the SA in two's complement representation. The sign bits of all positive constants in the MCM block are equal to the input sign bit. Therefore the long sign extensions of the SAs result in a high sign bit loading. This high sign bit loading can be overcome by using offset representation, but the carry will still be rippled to the MSB of the partial sum.

6.1 Probabilities in Structural Adders

By nature of the filter coefficients, the signals from the MCM block, which are inputs to the SA block, have special properties. While the range of the signal after constant multiplication is larger than the filter input signal range, the number of distinct values does not increase but are evenly spaced into a larger range. For example, if the input values $0, 1, 2, 3, 4, 5$ are multiplied by a coefficient 5, the inputs to the SA block are $0, 5, 10, 15, 20, 25$, other values like 1 or 6 are not possible.

The expanded input space with only sparse values poses some questions. Will this sparseness carry forward through the SA block or will the space of numbers become continuous at the output? What will be the probability distributions of the bits and values of the inputs to the SA block, the outputs of SAs in the SA block and the final output of the filter? Does the Central Limit Theorem (CLT) apply if the filter input is an independent random variable or if it is white noise? These questions will be discussed in this section.

6.1.1 Bit Probabilities Using Full Adders

The analysis of the bit probabilities is inspired by the FA based probabilistic model used by [163], [164] for the MCM block. For simplicity, all adders within the MCM and SA block are assumed to be implemented by RCA using FA as basic elements. Let $\alpha_0(X)$ and $\alpha_1(X)$ denote the probabilities that the signal X are 0 and 1, respectively. The XOR function within the FA has the tendency to equalize the probabilities of 0 and 1 outputs because

$$\alpha_0(A \oplus B) = \alpha_0(A) \cdot \alpha_0(B) + \alpha_1(A) \cdot \alpha_1(B) \tag{6.1}$$

$$\alpha_1(A \oplus B) = \alpha_0(A) \cdot \alpha_1(B) + \alpha_1(A) \cdot \alpha_0(B) \tag{6.2}$$

For example, if $\alpha_0(A) = 1/3$ and $\alpha_0(B) = 1/5$, then $\alpha_0(A \oplus B) = 3/5$ and $\alpha_1(A \oplus B) = 2/5$ which shows that the 0 and 1 output probabilities are closer to equality than the 0 and 1 probabilities of any of its inputs. If one input, for example A, has $\alpha_0(A) = 1/2$, the output will also have $\alpha_0 = 1/2$. These rules are always true, except when one of the inputs is a constant, then the probability of the variable input will be copied or inverted depending on whether the constant value is 0 or 1. On the other hand, the AND operation requires both inputs to be 1 for the output to be 1. Assuming both inputs A and B have equal probability of 0 and 1, then $\alpha_0(A \cdot B) = 3/4$ and $\alpha_1(A \cdot B) = 1/4$. A FA with $\alpha_0(c_{in}) = 1$ has $\alpha_0(s) = \alpha_1(s) = 1/2$ and $\alpha_0(c_{out}) = 3/4$ while a generic FA has $\alpha_0(s) = \alpha_1(s) = 1/2$ and $\alpha_0(c_{out}) = 1/4 + \alpha_0(c_{in}) \cdot 1/2$.

The following probabilities can be deduced from the analysis of a RCA that calculates $y = C \cdot x$, where y is the output with bit width v, x is the input signal with bit width u and $\alpha_0(x_n) = \alpha_1(x_n) = 1/2,\ n \in \{0, 1, \ldots, u-1\}$, and C is a positive, odd constant bigger than one.

- The probabilities for the internal carry signal can be calculated as $\alpha_0(c_m) = \frac{2^p+1}{2^{(p+1)}}$ and $\alpha_1(c_m) = \frac{2^p-1}{2^{(p+1)}}$, where m is the position of carry signal and $p = \min(m, v-1)$. Position $m = 0$ refers to the carry input to the LSB.

- The sign bit has $\alpha_0(y_{v-1}) = \alpha_1(y_{v-1}) = 1/2$ as it can be wired directly from the input x_{u-1}.
- All output bits from the LSB up to the MSB position of the input will have $\alpha_0(y_n) = \alpha_1(y_n) = 1/2,\ n \in \{0, 1, \ldots, u-1\}$
- For the bits between the MSB position of the input and the sign bit of the output $n \in \{u, u+1, \ldots, v-2\}$, the values depend on the binary representation of the multiplier constant C. Excluding the LSB of the binary represented multiplier, $\alpha_0(y_n) = \alpha_0(c_m)$ for all positions where the binary representation has a values of 1 and $\alpha_0(y_n) = \alpha_1(y_n) = 1/2$ for values of 0 in the binary representation. For example if $C = 11 = [1011]_2$ the probability α_0 for the five MSBs are $[\frac{1}{2}\ \frac{2^p+1}{2^{(p+1)}}\ \frac{1}{2}\ \frac{2^p+1}{2^{(p+1)}}\ \frac{1}{2}]$.

If the constant C is negative and odd (including -1), the logic for the last bullet point is reversed. Therefore, all inputs to the SA block have equal or near equal probability of being 0 or 1. If the filter input samples are independent, the same result is applicable to the FAs in the SA. A large deviation from the balanced probability of $\alpha_0 = \alpha_1 = 1/2$ of any input bit will not cause a large perturbation of the probability of $\alpha_0 = \alpha_1 = 1/2$ for all output bits. This was verified by a Monte Carlo simulation with white noise as input and 2^{20} input samples on different filters.

6.1.2 Value Distribution within the Structural Adder

Two questions of interest arise from an observation of the distribution of input values to the SAs within the SA block. First, the inputs to the SA block are sparse[1]. The question is whether the sparseness will be preserved after these signals have been summed. Second, how likely will these values fall near the borders of their range calculated by the minimum bit widths required to represent the signals (See Section 2.1.7). If it is very unlikely, then the corresponding ranges can be reduced with a small risk or low probability of obtaining an inaccurate results.

The question regarding sparseness of the signal at the output of a SA can be partially answered by observing the different combinations of legitimate values taken within the ranges of the two addends. If each addend is expressed as the product of a discrete constant integer (the constant multiplier) and a set of continuous integers (the input), then the addition of one element from one addend set to every element of the other addend set will produce a set of constant integers. This is repeated until all elements of the first addend set have been added to the other addend set. These resulting sets may have some common entries and their combination may be a set of continuous integers. If one constant coefficient multiplier is 1, the corresponding addend will possess a continuous set and the resulting set for the addition is continuous unless the other integer is larger than the input range. Due to the sinc function-like impulse response of a FIR filter, there is a high chance to have small values in the tail of the

[1] unless the magnitude of the constant coefficient multiplier is equal to 1, which can be directly fed from the input or its negated version

impulse response including ± 1. Therefore, within the SA block, most outputs of SAs will have a continuous range around the central value of the possible range of values.

It is too complex to mathematically derive the probabilistic distribution. A technique similar to the one used later in Section 6.4 can be applied to simplify the analysis. Assume that the input values of the filter are independently and uniformly distributed over all the integer values within its dynamic range. The possible output values of the first SA can then be calculated by adding the set of equally probably integer outputs from the first constant multiplier to every integer value from the set of possible integer outputs of the second constant multiplier. The possible values for all subsequent SAs can be calculated in the same way by adding the set of equally probable integer outputs from the current constant multiplier to the set of possible integer outputs from the previous SA. Using the example of $x \in \{0, 1, \ldots, 5\}$ and the constant coefficients of 3 and 5, the two sets of integers for the constant multipliers are $S_3 = \{0, 3, 6, 9, 12, 15\}$ and $S_5 = \{0, 5, 10, 15, 20, 25\}$, respectively. Each value has a probability of occurrence of $1/6$. If the values in S_5 are added to the set S_3, it will result in the following sets of integers:

$$\begin{aligned} S_3 + 0 &= \{0, 3, 6, 9, 12, 15\} \\ S_3 + 5 &= \{5, 8, 11, 14, 17, 20\} \\ S_3 + 10 &= \{10, 13, 16, 19, 22, 25\} \\ S_3 + 15 &= \{15, 18, 21, 24, 27, 30\} \\ S_3 + 20 &= \{20, 23, 26, 29, 32, 35\} \\ S_3 + 25 &= \{25, 28, 31, 34, 37, 40\} \end{aligned} \tag{6.3}$$

where all entries have a probability of occurrence of $1/6 \cdot 1/6 = 1/36$. The distribution is shown in Figure 6.2.

The union of all sets in Equation (6.3) is the output set $S_{3\times5} = \{0, 3, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 35, 37, 40\}$, where the values $\{15, 20, 25\}$ have the probability of $2/36$ as they exist twice in different sets of Equation (6.3). Furthermore, there is no void between integers 8 to 32 and the number of voids at the two tails are symmetrical. Even for this simple two-constant multiplier example, it is easy to see that the number of unique integer values grows very fast, but it does not grow as fast as would be expected. An input x with a bit width of 8 can have $2^8 = 256$ distinct integer values. A 121-tap FIR filter like the one from [28] has in principle $256^{121} \approx 2.5 \cdot 10^{291}$ possible distinct paths to generate the output values, but the actual possible number of different output values is smaller than $2^{25} \approx 3.3 \cdot 10^7$. This is because after some number of accumulations, the growth in the number of different integers in the sum has slowed down due to the increasing number of recurrent integer values. With the available computing power today, it is possible to calculate the probabilities of each possible integer output value of each SA for short filters[1] with a small

[1] e.g. less than 40 taps

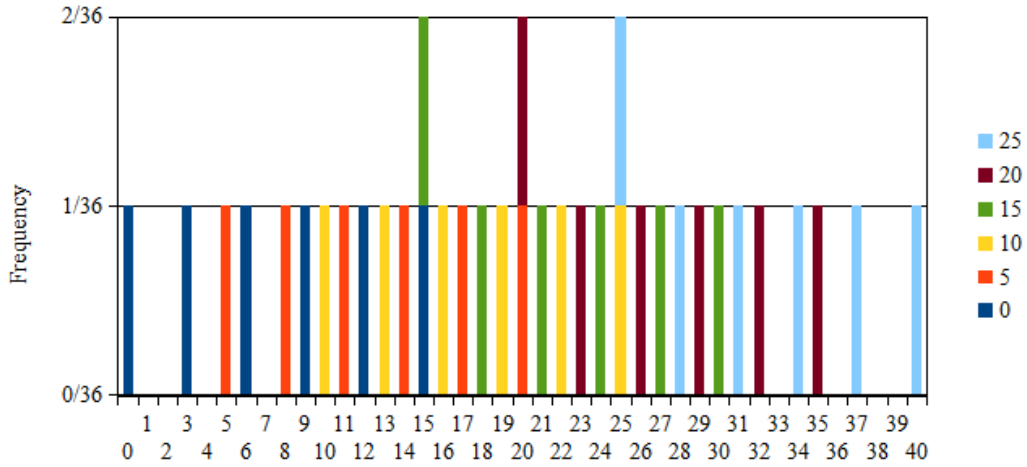


Figure 6.2: Histogramm of possible additions of $3 \cdot x$ and $5 \cdot x$ where $x \in \{0, 1, \ldots, 5\}$, different colors indicate the sums of S_3 and different values of S_5

input bit width[1] in reasonable time.

As an example, consider the FIR filter 'G1' from [34] with coefficients, $3, 6, 0, -16, -19, 12, 76, 128, 128, 76, 12, -19, -16, 0, 6$ and 3. The probabilities of occurrences of all possible output values of the structural adder at each tap is shown in Figure 6.3 for an input bit width of 2 and Figure 6.4 for an input bit width of 6. The distributions from the two different input bit widths and other bit widths differ mainly in the taps further from the output and the differences are prominent for smaller bit width. The differences in early taps are masked by the high frequency of recurrences of integers towards the output tap. The difference between the last distributions of Figures 6.3 and 6.4 is not significant. In general, a small input bit width can still be used as a good estimate for the distribution of the filter output value.

Comparing the resulting distributions to a normal distribution shows the CLT holds and the distribution of the SA output can be approximated by a normal distribution from the sixth tap onwards. Figure 6.5 shows that the output distributions of filter 'G1' from [34] can be matched to a normal distribution for bit widths 2 and 6, respectively. By plotting these distributions on a log scale in Figure 6.6 for comparison, it can be seen that the values in the tails of the actual distribution are even less likely to occur than those at the tails of normal distribution. Therefore it is safe to consider normal distribution for the estimation of the error probability in the bit width reduction of structural adder. Unfortunately, due to the accumulation of errors along the tapped delay line, the error increases fast and the bit width can hardly be reduced by more than one bit as noted in [44]. The mean and variance of the normal distribution can

[1] e.g. 4, 5 or 6 bits

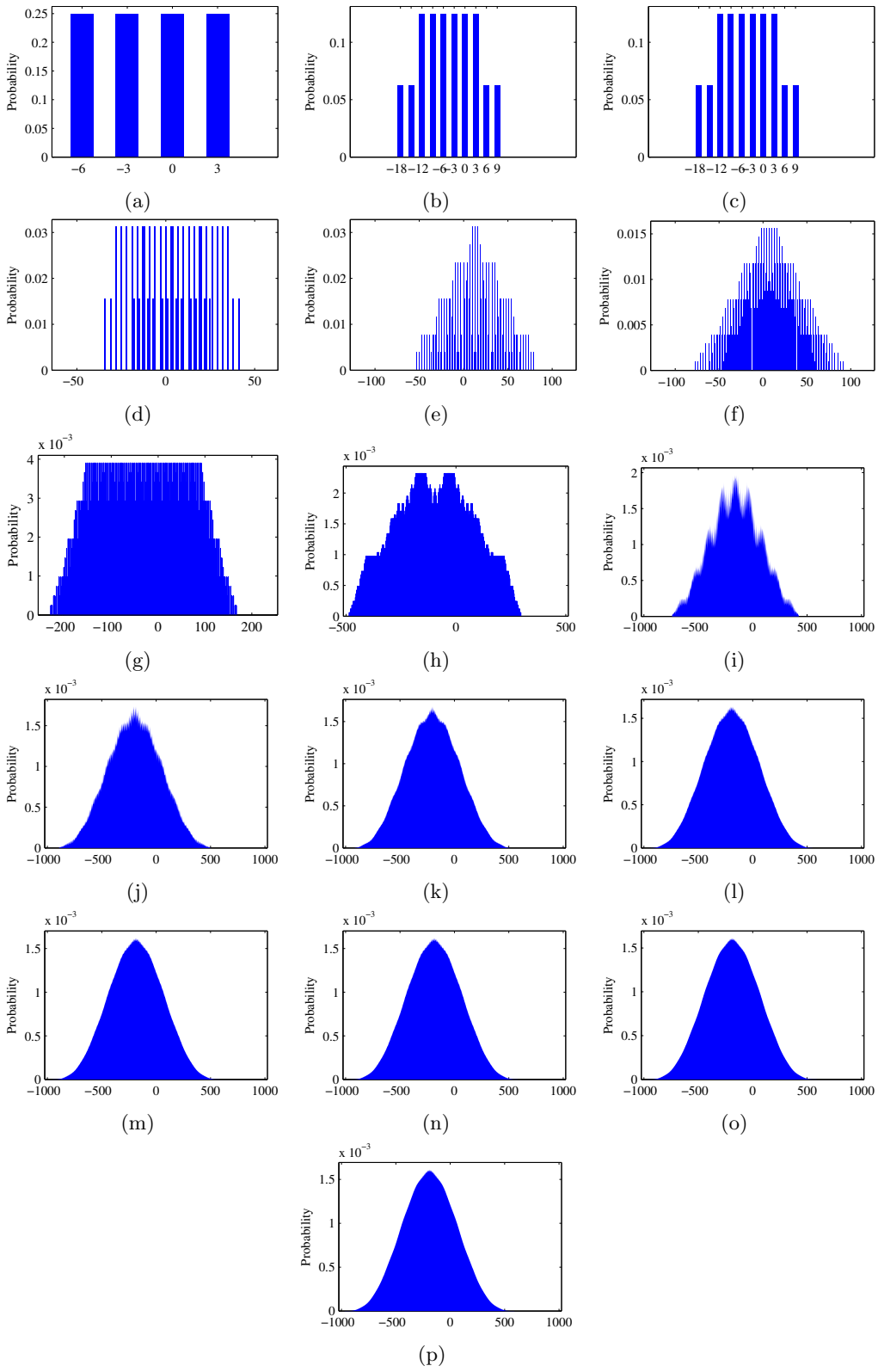


Figure 6.3: Probability distribution of structural adder output values for the filter 'G1' from [34] with input bit width of 2 from (a) tap 15 (input tap) to (p) tap 0 (output tap)

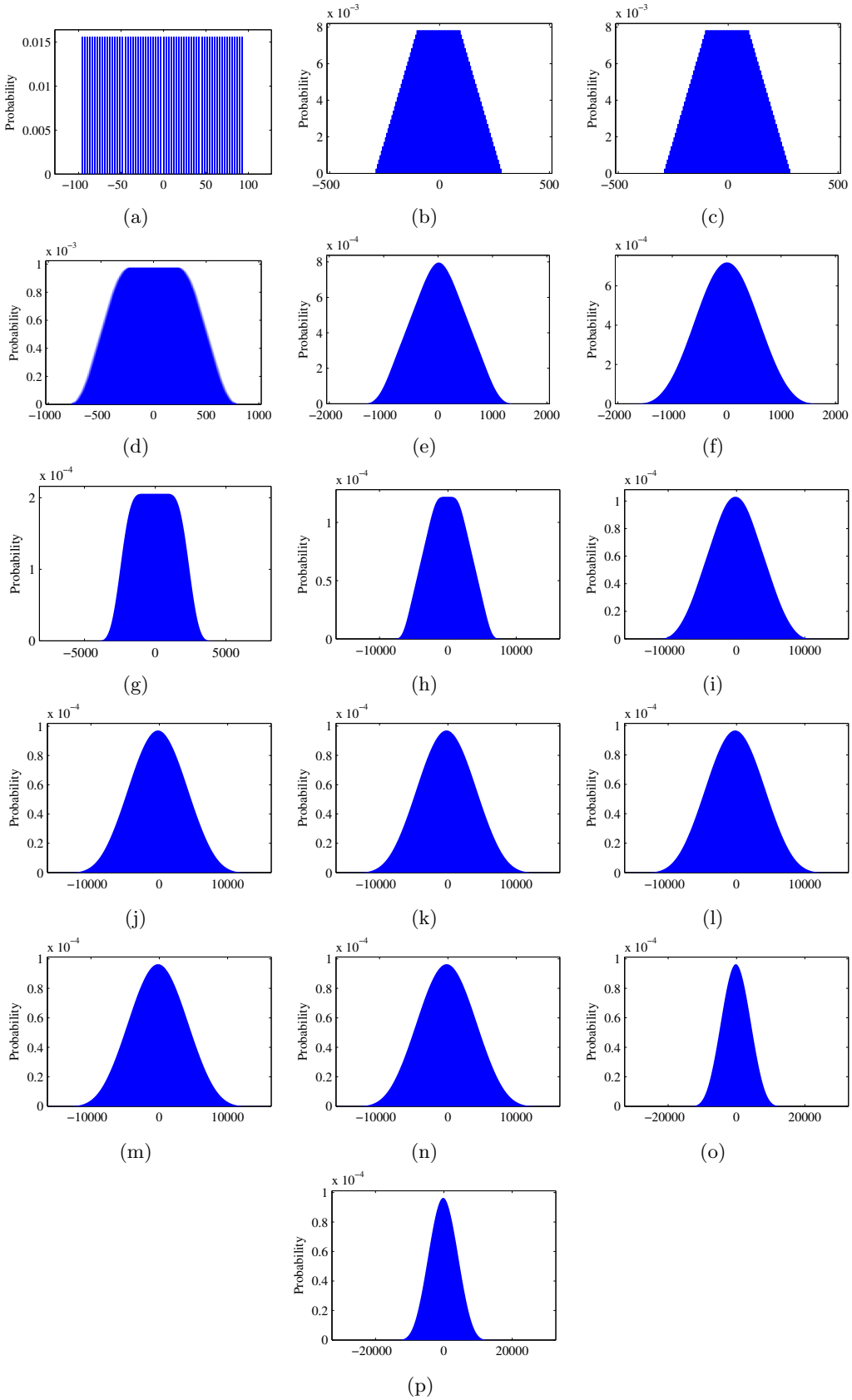


Figure 6.4: Probability distribution of structural adder output values for the filter 'G1' from [34] with input bit width of 6 from (a) tap 15 (input tap) to (p) tap 0 (output tap)

be estimated by:

$$\mu = \sum_{j=-2^{n-1}}^{2^{n-1}-1} \cdot \sum_{k=0}^{N-1} c(k) = -0.5 \cdot \sum_{k=0}^{N-1} c(k) \tag{6.4}$$

$$\sigma^2 = \sum_{k=0}^{N-1} c(k)^2 \cdot \sigma_{inp}^2 \tag{6.5}$$

where n is the bit width, N is the number of coefficients and

$$\sigma_{inp}^2 = \frac{1}{n} \sum_{i=-2^{n-1}}^{2^{n-1}-1} (i-\mu)^2 = \left(\frac{1}{n} \sum_{i=-2^{n-1}}^{2^{n-1}-1} i^2 \right) - \mu^2 = \left(\sum_{i=0}^{n-2} 4^i \right) + 0.25 \tag{6.6}$$

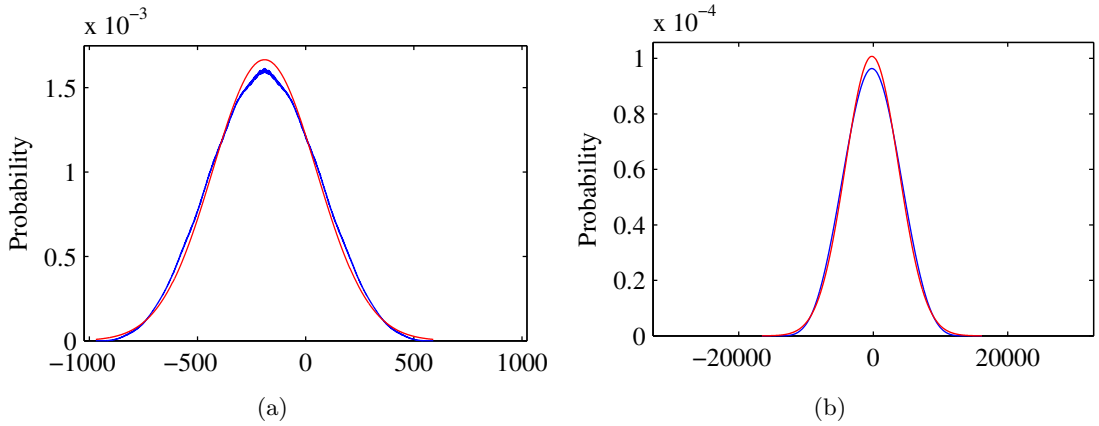


Figure 6.5: Gaussian curve fitting of output value distribution probability for filter 'G1' from [34] with input bit widths of (a) 2 and (b) 6

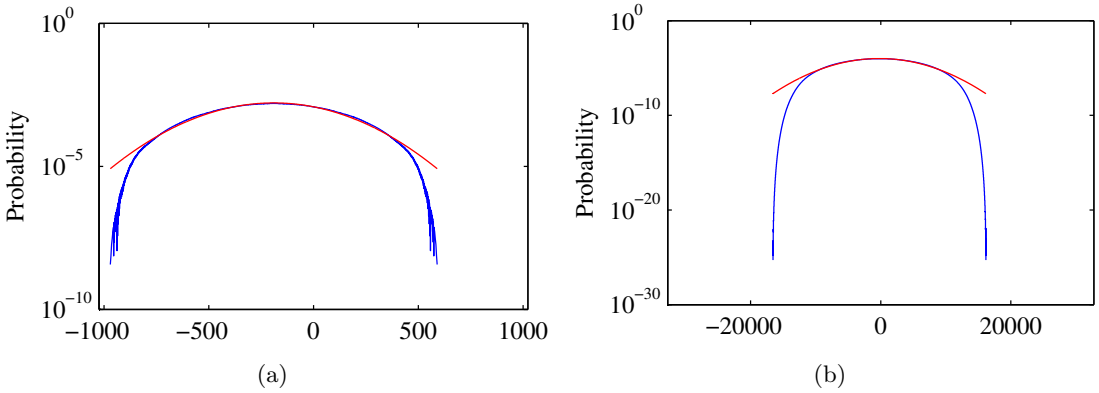


Figure 6.6: Gaussian curve fitting of output value distribution probability on log scale for filter 'G1' from [34] with input bit widths (a) 2 and (b) 6

The example filter used does not contain a coefficient with absolute value 1, which results in sparsity at the tails of the value range. Table 6.1 shows the range and its utilization for each structural adder output for an input bit width of 6. In Table 6.1, '$c(n)$' denotes the i-th filter tap coefficient, 'Range' is described by the range ('Size') of integer values, and the

minimum ('Min.') and maximum ('Max.') values of the product of $c(n)$ and input variable, 'Binary Range' refers to the 'Size', 'Min.' and 'Max.' of the binary word required to represent the value in the range of 'Size', 'Numbers Used' denotes the number of unique integers used in 'Range' and 'Numbers %' is the percentage ratio of 'Numbers Used' to 'Range Size', and 'Possible Comb.' indicates the total number of possible output values at each structural adder output. 'Possible Comb.' is estimated by $p_i = (2^w)^i$, where w is the input bit and i is the tap number. If the filter contains a coefficient with an absolute value of 1, the value range is continuous and therefore the usage % reaches 100%. This range utilization is calculated based on the smallest and biggest possible output values, without taking into account of the excess unused integer values if these boundaries do not coincide with the maximum and minimum values of a given integer bit width. The calculation time for the probabilities of occurrence for up to 32731 unique integers for all the 16 taps was about 1300 seconds by regular MatLab code without much optimization on a standard computer. The time taken to calculate the probabilities of occurrences of unique integers for each of the last few coefficients was less than 220 seconds. Unfortunately, it is not possible to calculate the result for a single SA without calculating those for all its preceding SAs. The normal distribution can be used as an estimate based on the simple value range boundary calculation.

TABLE 6.1: Ranges and utilization rate of output values of structural adders for filter 'G1' from [34] with input bit width of 6

c(n)	**Range**			**Binary Range**			**Numbers**		**Possible**
	Size	**Min.**	**Max.**	**Size**	**Min.**	**Max.**	**Used**	**%**	**Comb.**
3	190	-96	93	256	-128	127	64	33.68	64
6	568	-288	279	1024	-512	511	190	33.45	4096
0	568	-288	279	1024	-512	511	190	33.45	$2.62{\cdot}10^{5}$
-16	1576	-784	791	2048	-1024	1023	1546	98.10	$1.67{\cdot}10^{7}$
-19	2773	-1373	1399	4096	-2048	2047	2743	98.92	$1.07{\cdot}10^{9}$
12	3529	-1757	1771	4096	-2048	2047	3499	99.15	$6.87{\cdot}10^{10}$
76	8317	-4189	4127	16384	-8192	8191	8287	99.64	$4.40{\cdot}10^{12}$
128	16381	-8285	8095	32768	-16384	16383	16351	99.82	$2.81{\cdot}10^{14}$
128	24445	-12381	12063	32768	-16384	16383	24415	99.88	$1.80{\cdot}10^{16}$
76	29233	-14813	14419	32768	-16384	16383	29203	99.90	$1.15{\cdot}10^{18}$
12	29989	-15197	14791	32768	-16384	16383	29959	99.90	$7.38{\cdot}10^{19}$
-19	31186	-15786	15399	32768	-16384	16383	31156	99.90	$4.72{\cdot}10^{21}$
-16	32194	-16282	15911	32768	-16384	16383	32164	99.91	$3.02{\cdot}10^{23}$
0	32194	-16282	15911	32768	-16384	16383	32164	99.91	$1.93{\cdot}10^{25}$
6	32572	-16474	16097	65536	-32768	32767	32542	99.91	$1.24{\cdot}10^{27}$
3	32761	-16570	16190	65536	-32768	32767	32731	99.91	$7.92{\cdot}10^{28}$

6.1.3 Findings from the Distribution of SA Output Values

Based on the findings presented so far, it can be concluded that the probability of the output values of a filter can be approximated by a normal distribution. The calculation of its means and variance were presented. The same estimates are also applicable to the output values of a structural adder after the middle coefficient. Using the presented estimate, it is possible to estimate the error introduced by pruning the output ranges of the structural adders. If rarely occurred large errors are acceptable or small overflow or wrap around avoidance circuits are available, the approximation can be used to locate the bits for reduction to achieve the largest savings with the smallest error. Based on these insights, further research was directed towards the reduction of the bit width of SA without violating the required range.

6.2 Structural Adder Area Reduction

For fixed coefficient FIR filters, the bit widths of the input and all coefficients are known. This enables the bit width of the coefficient multiplier to be determined from its dynamic range (See Section 2.1.7). As the partial sums are delayed before they are added with the coefficient multiplier outputs in the SAs, the bit widths of the SAs increase monotonically from the first SA towards the output. Careful analysis revealed that for most filters, the bit width of the adder increases only from coefficient $N-2$ to about $N/2$, after which the bit width stays relatively constant and increases by no more than two bits. As the bit width of the coefficient multiplier output reduces towards the last tap, longer sign extension and carry chains are required for these SAs.

First, the dominance of the SA block in the overall cost for the implementation of a FIR filter with N taps is shown. The lower bound for single constant multiplier from [18] shown in Equation (2.28) is extended in [18] for the MCM as follows:

$$LO_{min,MCM} = \min_i\{\lceil \log_2 (S(C_i))\rceil\} + M - 1 \tag{6.7}$$

where M is the number of unique, positive and odd constants derived from the coefficients of the filter. Therefore the minimal bound to implement an MCM block is equal to the implementation cost of the simplest[1] constant C_i plus one additional adder per positive and odd constants. Because the magnitude of the impulse response of FIR filters is limited by a $1/|x|$ function due to its similarity to a sinc function, there are always small and simple coefficients. It can be concluded that the $LO_{min,MCM} \approx M$. Many MCM algorithms produce solutions that are very close to the lower bound. For linear phase filters, $M \leq \lceil {}^{N-Z}/_{2} \rceil$, where N is the number of taps and Z is the number of zero coefficients. Therefore, the approximate number of LOs of an optimized MCM block of a linear phase FIR filter can be written as:

$$LO_{MCM} \approx \left\lceil \frac{N-Z}{2} \right\rceil \tag{6.8}$$

Comparing the expected cost from Equation (6.8) with the word level cost of $N-Z-1$ adders and N delay elements in the SA block in terms of LO, it is obvious that the size of the SA block has to be bigger than the MCM block. The discrepancy is greater for longer filters which are more likely to have repeated constant multipliers. In addition, the bit widths of the adders and delay elements in the SA block have to be bigger than the bit widths of the respective adders in the MCM block:

$$\mathrm{w}(a_{MCM,i}) \leq \mathrm{w}(a_{SA,i}) \tag{6.9}$$

where $\mathrm{w}(x)$ is the bit width of the signal x, and $a_{MCM,i}$ and $a_{SA,i}$ denote the adders of the i-th tap for the MCM and SA blocks, respectively. The proof for Equation (6.9) is straightforward. $\mathrm{w}(a_{MCM,i})$ can be derived deterministically as it only depends on the input bit width and

[1] In the sense of the least cost for implementation

the constant multiplier generated by the addition $a_{MCM,i}$ as introduced in Section 2.1.7. For example if $a_{MCM,i}$ adds x and $5 \cdot x << 3$, the output is $41 \cdot x$, and the required output bit width is $\mathrm{w}(x) + \lceil \log_2(41) \rceil = w(x) + 6$. When the output of the mentioned MCM adder is fed to the structural block, it will be added by a structural adder to the partial sum received from the previous structural adder. As these signals are associated with different input samples, they have to be regarded as independent and uncorrelated. Therefore the ranges of both inputs add up to at least the same bit width as the MCM adder output, but the bit width is more likely to be larger. As the output of previous structural adder and the output from the MCM block are independent and uncorrelated, it is straightforward to show that $\mathrm{w}(a_{SA,i}) \leq \mathrm{w}(a_{SA,i+1})$[1], which means that the bit width of the structural adders grows monotonically from the first structural adder towards the output. Based on the calculations presented in Section 2.1.7 and Section 2.1.8, a careful analysis of the bit widths of the structural adders of real FIR filters from the benchmark filters from [122] showed that the bit width only increases from the first tap numbered $N-1$ to about the middle tap numbered $\lfloor N/2 \rfloor$, where it has reached the same or at most two bits less than the final output bit width. Therefore at least half of the structural adders and their corresponding delay elements have the same bit width of the output or at most two bits less. Based on this analysis, a first order estimate of the total number of one-bit adders (assume that the CPA is implemented by RCA) and one-bit delay elements required for the SA block is given by:

$$\mathrm{Bitadders}_{SA} \approx N \cdot \mathrm{w}(Y) - \frac{\mathrm{w}(Y) - \mathrm{w}(X)}{2} \cdot \frac{N}{2} \tag{6.10}$$

where X is the input and Y the output of a FIR filter.

Using Equation (6.9) and assuming an optimized MCM block of the lower bound in Equation (6.8), it can be concluded that the cost of the MCM block has to be smaller than the cost of all structural adders for any real FIR filter. Experimental results confirmed this and in some cases, the MCM block requires less than 10% of the total FIR filter area. With the optimization of the MCM block, the size of this block reduced significantly. The argument that MCM block dominating the cost of a FIR filter is no longer valid. Therefore optimization of the SA block by even a small percentage will save more area than a substantial LO reduction in the MCM block.

6.2.1 Structural Adder Area Reduction

A basis for the optimization of the SA block is the knowledge of the exact bit width needed to represent the signal. The bit width can be calculated exactly or approximated as in Section 2.1.7 if all coefficients and the bit width of the input are known. Special handling is required for the offset representation. If the SA block is implemented using CSA, the MSB requires special circuitry to avoid carry overflows as introduced in Section 2.1.8.

Because word level optimization is not possible, a bit-level approach was sought. Generally the adders cannot be optimized because the inputs to them are uncorrelated and independent.

[1] the structural adder closest to the output is $a_{SA,0}$, see Figure 2.2

Straightforward implementation of bypassing the adder where the LSBs of the signal from the MCM is left shifted is not considered an optimization here. The only way to optimize the SA block is to achieve a net gain from the trade-off between logical operations and registers. To establish a beneficial tradeoff, the ratio of these two component areas is required. The structural adders add two two's complement or two unsigned binary numbers. The simplest CPA to implement the addition is a RCA, which consists of one FA for each pair of addend bits and HA to propagate the carry to the next higher order bit when there is only one operand bit and the carry at the high order bit positions. Other faster adders like CLA require more area than a RCA. Therefore, the complexity of a SA is assessed based on the number of FAs and HAs required to implement the respective RCA. FFs are used for registers. The authors of [165] suggest a ratio of $1 : 0.6 \approx 0.8$ for FA : FF. TSMC $0.18\mu m$ standard cell library [166] is used for simulation throughout this thesis. It contains a variety of FFs with areas measured from $53.2\mu m^2$ to $86.5\mu m^2$. Its standard FA cell has an area of $69.9\mu m^2$. Based on these documented numbers, and the average area of $69.2\mu m^2$ for the FF as well as the FA in typical FIR filters synthesized with this standard cell library, a FA:FF ratio of $1 : 1(\rho = 1)$ is adopted.

As the structural adders are different for the cases of sign extended two's complement representation and MSB-Fix offset representation, it is not possible to find an optimization method that works for both number representations. In both representations, several successive structural adders have to be taken into consideration to achieve area reduction. The key factor to be considered in both cases is the aggregate difference in the bit widths between the delayed, partial sum and the outputs of the MCM block. In the following subsections, two optimization techniques are developed for the two's complement and offset representations.

6.2.2 Structural Adder Area Reduction for Two's Complement Number Representation

Figure 6.7 shows the analysis of Filter Example 1 of [28] using an 8-bit input signal. The bit width of the partial sum is the minimal bit width based on the range calculation as it was presented in the background Section. It remains constant after c_{60} even as the coefficient multiplier output's bit width reduces towards both ends. As the difference between the bit widths of the coefficient multiplier output and the partial sum increases towards the output tap, longer sign extension is required for these structural adders. Six or more bits are required for the sign extension for the structural adders from a_{52} to a_0.

To illustrate the fundamental concept of the optimization method for the sign extended case, an example in decimal representation is presented first. The set of outputs $\{610, -274, 2, 258\}$ from the MCM block is to be accumulated to a large partial sum 1234567 by the structural adders in a tapped delay line. Note that each value in this set of outputs need at most three digits to represent while the partial sum requires up to seven digits to represent. Three possible ways of accumulation are shown in Figure 6.8. The downright approach shown in Figure 6.8(a) adds one number at a time from the set to the large integer and stores the new large integer. When done in binary, it requires all numbers from the set to be sign extended to

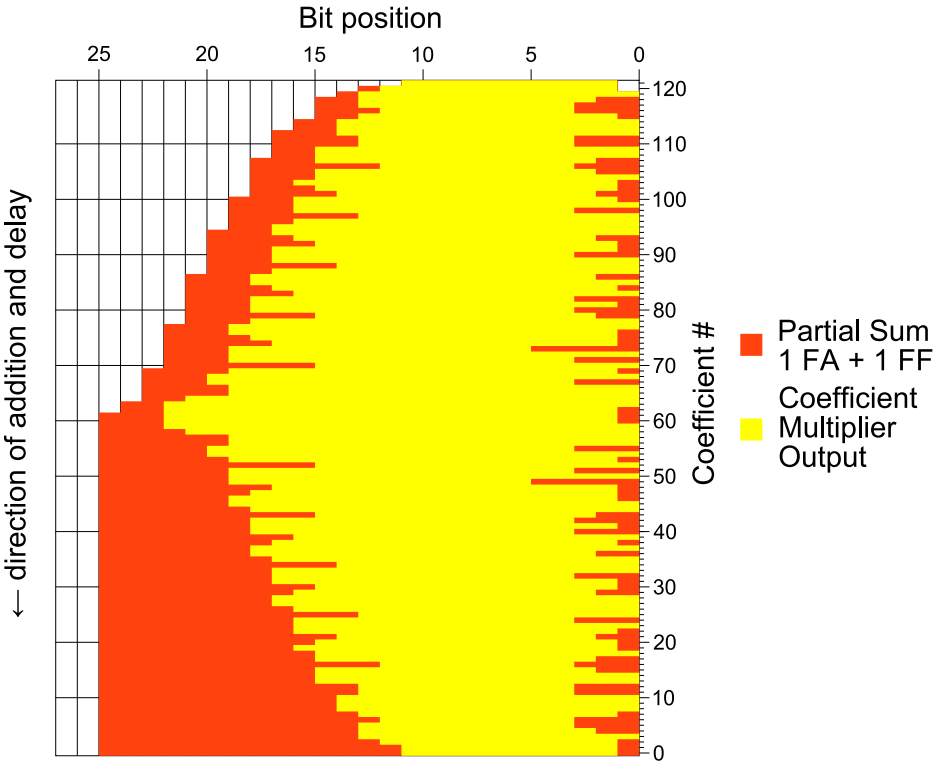


Figure 6.7: Bit width analysis for the structural adder for Filter 1 of [28] using an 8-bit input signal.

the length of the large number although the high order bits remain unchanged. Figure 6.8(b) shows a simple alternative where the integers in the set are summed without altering the large integer till the last step. This requires the same number of adders but might increase the delay as two numbers have to be added in the last tap. Furthermore, this method requires the storage of the large integer as well as the intermediate partial sums. Figure 6.8(c) shows the concept of the proposed method. Only the last three digits of the large integer are used for the summation while its higher order digits are directly forwarded to the next step. At the last tap, an additional adder is required, but it only needs to add four digits since the three least significant digits are all zeros in one of its inputs. (c) has the advantage that the adders are almost as short as those of (b) and it uses only slightly more delay elements than the case of (a). The register overhead is only incurred at the fourth digit position which overlaps between the higher digits and the small partial sum. The reduction of the dynamic ranges of the operands also simplifies the structural adder implementation and reduces the length of sign extension. The regularity of adders and registers involved in the structural adder block is kept. Thus the layout complexity will not be increased. In fact, it has been simplified because the large fan out due to sign extension has been relaxed.

Figure 6.9 shows the binary implementation of the proposed scheme on the last three coefficients of the filter example from Figure 6.7. In Figure 6.9, D denotes an one-bit FF. The black and gray dots represent a binary variable bit and the sign-extension bit of a variable, respectively. The dashed box represents a CPA. The box filled with upper left to lower right

1234567	1234567	1234000 567
610	610	610
1235177	1234567 610	1234___ 1177
-274	-274	-274
1234903	1234567 336	1234___ 903
2	2	2
1234905	1234567 338	1234___ 905
258	258	258
1235163	1234567 596	1234___ 1163
	1234567	1234000
	1235163	1235163
(a) downright approach	(b) sum small numbers first	(c) split large number

Figure 6.8: Three Different Possibilities to Accumulate a Set of Decimal Numbers

hatched lines indicates the reduced length of a CPA. The box filled with lower left to upper right hatched lines represents the register[1] and CPA[2] overhead. Orange, yellow, blue and green colors are used to identify the split and merged partial sum bits, coefficient multiplier output bits, reduced partial sum bits and the stored partial sum bits, respectively. A simplified diagram for the whole optimized filter by abstracting away the dots and 'D's is shown in Figure 6.10. The reduction of adder lengths is observed to be several times more than the register and adder overheads it incurres. Furthermore, the delays through the SAs, a_2 and a_1, have been reduced, while the delay through a_0 is increased by one FA delay. The slight increase in the delay through a_0 is not an issue as in most cases, there exists at least one tap $(i > 0)$ for which $\text{delay}(x \cdot c_0) < \text{delay}(x \cdot c_i)$. The reduction of FAs for the SAs can be offset by the increase in FF overhead, depending on the difference between the addends of the SAs.

With reference to Figure 6.1, p_i and q_i are the inputs to the i-th SA, a_i, where q_i is the i-th coefficient multiplier's output and a_0 is the last adder that produces the output y. Let $\text{n}(z)$ be the bit width of an arbitrary signal z and $\left[v_i^-, v_i^+\right]$ be the dynamic range of q_i. The selected partial sum p_i is bisected into two halves, i.e., $p_i = p_i^u \left| p_i^l \right.$ at bit position h_i such that $\text{n}\left(p_i^l\right) = h_i$ and $\text{n}\left(p_i^u\right) = \text{n}\left(p_i\right) - h_i$. If p_i^u is added, after m delays, to the sign extended p_{i-m+1}^l where $\text{n}(p_{i-m+1}^l) > h_i$, then the original SAs, a_{i-j+1} will be replaced by the reduced SAs $a_{i,j}$ for $j = 1, 2, \ldots, m$. The minimal bit width $\text{n}(a_{i,j})$ of each reduced adder can be obtained by the following range estimation

$$\text{n}\left(a_{i,j}\right) = \left\lceil \log_2 \left(\max \left(2^{h_i} - 1 + \sum_{k=0}^{j-1} \left(v_{i-k}^+\right), -\sum_{k=0}^{j-1} \left(v_{i-k}^-\right) \right) \right) \right\rceil + 1 \tag{6.11}$$

In Equation (6.11), the value $2^{h_i} - 1$ is the maximum possible value of the lower bisected

[1] Rightmost two 'D' in the light green area.
[2] Overlap blocks of dark green and blue colors and rightmost three bits of the 13-Bit CPA.

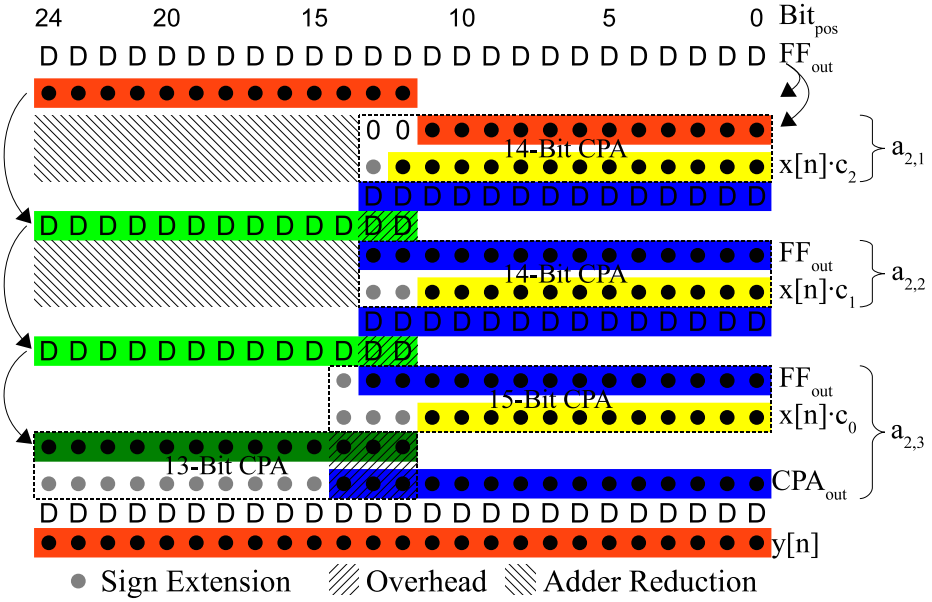


Figure 6.9: Binary example on the optimization of the last three adders of Figure 6.7 based on two's complement number representation

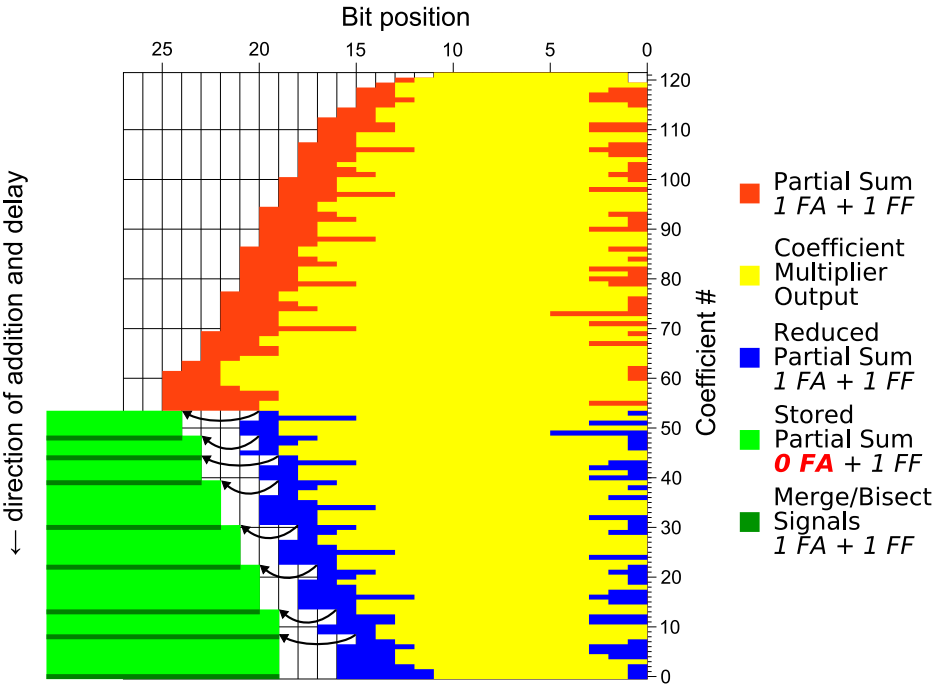


Figure 6.10: Optimized bit width for tap delay line signals of the example filter

partial sum p_i^l. Since the first term provides the bit width of the magnitude from the dynamic range, the constant '1' is added to account for the sign bit. As j increases, the value of $\mathrm{n}(a_{i,j})$ grows monotonically as $v_i^- \leq 0$ and $v_i^+ \geq 0$. The total number of FAs saved by the m successive

reduced SAs starting from the i-th tap is given as follows:

$$\Delta_{i,m} = \sum_{k=1}^{m-1} \left[\left(\mathrm{n}\left(p_i\right) - \mathrm{n}\left(a_{i,k}\right) \right) - \rho \left(\mathrm{n}\left(a_{i,k}\right) - h_i \right) \right] - \left(\mathrm{n}\left(a_{i,m}\right) - h_i \right) \tag{6.12}$$

where ρ is the FA:FF ratio. In (6.12), the term $\mathrm{n}\left(p_i\right) - \mathrm{n}\left(a_{i,k}\right)$ denotes the adder cost reduction and the term $\mathrm{n}\left(a_{i,k}\right) - h_i$ represents the register overhead. The last term $\mathrm{n}\left(a_{i,m}\right) - h_i$ refers to the overlapping digits for the addition of p_i^u. As $\mathrm{n}\left(a_{i,m}\right) > h_i,\ \forall c_i \neq 0$, it is apparent from (6.12) that there is no saving if $m = 1$.

Hypothetically, h_i can be any arbitrary value smaller than $\mathrm{n}(p_i)$. However, as $\mathrm{n}(a_{i,j})$ is dependent on v_i^+ and v_i^-, it can be optimally deduced. If $h_i > \max\left(\left\lceil \log_2\left(v_i^+\right) \right\rceil, \left\lceil \log_2\left(-v_i^-\right) \right\rceil \right)$, less reduction in terms of FA count is possible in the SAs block. If it is smaller, the register overhead and the FA cost of the final addition are higher. As $\mathrm{n}(q_i)$ is not monotonically decreasing with i, h_i needs to be selected by looking ahead into several coefficients. It was empirically determined that a look ahead of 5 coefficients is sufficient. This leads to

$$h_i = \max\left(\left\lceil \log_2\left(v_k^+\right) \right\rceil, \left\lceil \log_2\left(-v_k^-\right) \right\rceil \right),\ \forall\, k = i, i-1, \ldots, i-5 \tag{6.13}$$

Combining Equation (6.11) and Equation (6.13) it can be derived that $\mathrm{n}\left(a_{i,j}\right) \geq h_i + 2,\ \forall j > 0$. Defining $\delta_i = \mathrm{n}\left(p_i\right) - h_i$, from Equation (6.12), we have

$$\Delta_{i,j} \leq (m-1) \cdot (\delta_i - 2 - 2\rho) - 2 \tag{6.14}$$

To avoid increasing the area, the following lower bound can be derived from Equation (6.14).

$$\delta_i \geq 2\frac{m\left(1+\rho\right) - \rho}{m-1} \tag{6.15}$$

If $m \to \infty$, $\delta_i \geq 2\left(1+\rho\right)$. Usually, m is small and for TSMC $0.18\mu m$ standard cell library [166], $\rho \approx 1$ and $\delta_i \geq \frac{2(2m-1)}{m-1}$. For $m = 2$, $\delta_i \geq 6$ and for $m = 3$, $\delta_i \geq 5$.

The above formulae hold only if no zero valued coefficient is involved. Since no SA is required to add a zero coefficient, no saving can be gained from the SA but the overhead due to the overlapping bits remains. Experiment results showed that in many cases it is not beneficial to span the bisected partial sum addition process over zero coefficients. It was also observed that $\mathrm{n}\left(a_{i,j}\right) = h_i + 2$ is valid only for $j \leq 3$. This observation suggests the use of $\mathrm{n}(p_i) - \mathrm{n}(q_i) \geq 6$ to select the partial sum for bisection. In Figure 6.10, it can be seen that the first bisection of partial sum is made at the tap where the partial sums of the tap and all successive taps have $\mathrm{n}(p_i) - \mathrm{n}(q_i) \geq 6$. h_i is always one bit below $\mathrm{n}\left(q_i\right)$, as the sign bit is not included in the range $\left[v_i^-, v_i^+\right]$ calculated in Equation (6.13).

From Equation (6.11), $\mathrm{n}\left(a_{i,j}\right)$ grows monotonically with increasing j. This implies that the FF overhead is increasing as the FA reduction in the SA is declining. The gain diminishes with increasing m and can become negative. Furthermore, as $\mathrm{n}(q_i)$ decreases towards c_0, a smaller h_i can be chosen for better gain. This suggests that it is beneficial to make repeated

bisections. The bisection can be repeated in two ways. If only area reduction is of interest, it is more optimal to merge p_i^u at a tap and then bisect the partial sum again on the next tap. This will increase the path delay at each merging SA. To avoid the delay penalty, the merging and bisecting can be performed simultaneously at the same tap. For this concurrent merge/bisect SA a_i, the operand, p_i^l from the previous bisection is bisected again at h_i such that $p_i^l = p_i^{lu} \left| p_i^{ll} \right.$. p_i^{lu} is sign extended and added to p_i^u while p_i^{ll} is added to q_i. The register overhead will be increased by two FFs. Since the additions are independent, the operands are shortened by the simultaneous merging and bisection, thus the delay through the SA is also reduced. Experiment results showed that the delay reduction is more prominent than the cost of two FFs. Therefore, simultaneous merging and bisection is adopted. For the last tap, there is an increment of one FA delay due to the inevitable final merging as shown in Figure 6.9. In most cases, this is not a problem as c_0 does not fall in the critical path. In case timing closure is affected, the final merging could be moved to an earlier tap.

Algorithm for Finding the Best Trade-off

Using the formulae above, it is straightforward to calculate the savings for different starting points. The problem is relatively simple and there is no need for a heuristic as calculating all possibilities can be done in very short time. The calculation of Equation (6.12) is repeated for every $i > 2$ for which $\mathrm{n}(p_i) - \mathrm{n}(q_i) \geq 6$ is valid from $i > 2$ until the middle coefficient. Every $\{i, j\}$ pair is to be checked to determine if a repeated bisection will lead to a better result than the best result obtained from previous runs. The pseudocode for the algorithm is shown in Algorithm 3. The complexity of this search is $\mathrm{O}(n^2)$. On an Intel P4, 3.0 GHz PC with 2 GB RAM, the runtime is below 10 seconds even for filters with $N > 1000$.

Algorithm 3 Pseudocode for Finding the Maximum Saving in the SA Block

Inputs: Filter Length N, bit width information $\mathrm{n}(p_i), \mathrm{n}(q_i)\, \forall i$

OptimizeStructuralAdder

```
1: i = 1
2: while i ≤ N
3:    if n(q_i) < n(p_i) − 3 then
4:       calculate potential reduction for every bisected bit position n(q_i) ≤ j ≤ n(p_i) − 2 from
         i to 0
5:       use previous results and all possible merged positions i' < i
6:       save reduction and split/merge information of the best result for i
7:    else
8:       save zero as reduction information
9:    i = i + 1
10: select the best result ∀i
11: return split/merge information for the best result
```

Implementation Results

The method of optimization for two's complement representation was evaluated for many filters from the filter benchmark suite (see Section 3.2). The details of 44 longer filters that could be optimized are given in Table 3.1. The absolute reduction depends mainly on the difference between the bit widths of the delayed signal and the signal from the MCM block, as discussed in Section 2.1.7. The savings relative to the overall area of the filter depend on the input bit width. This is illustrated by an evaluation on 4- and 8-bit inputs. The number of FAs saved ('#FA') and the theoretically calculated ('Theo') percentage reduction of SA block in FA equivalent of each of the 44 filters are shown in the first two columns for each bit width in Table 6.2. As opposed to the results presented in [142] (which will not be repeated here), the results in Table 6.2 are obtained solely for the SA block by Design Compiler (DC). As MCM block is needed for the synthesis of the filter, direct CSD implementation was adopted. For the FPGA results, a separation of the MCM and SA block areas is not possible and the reduction reported is relative to the overall filter size. As mentioned before and in [142], it is possible to optimize the MCM block in addition to the SA optimization. This was not considered here as it has been shown that the savings due to the MCM block optimization is small.

The SA optimization for two's complement scheme was integrated into sysFIR (see Chapter 3). sysFIR was then used to generate the VHDL codes automatically for all filters. The VHDL descriptions was synthesized by Synopsys Design Compiler (Version 2007.12) using TSMC 0.18μm standard cell library [166] and Xilinx ISE targeting Virtex 5 FPGA, xc5vlx30. Unconstrained (no delay constraint - 'DC A') and fixed timing constraint (3ns delay constraint - 'DC D') were used for the technology mapping and logic optimization by Synopsys Design Compiler. In Xilinx ISE, no delay constraint was used and the results reported under 'FPGA' are the percentage savings in the number of equivalent two input gates used. From the FPGA utilization reports, the tradeoff between registers and LUTs can be seen easily. The register count was smaller than the LUT count for all filters in the original implementation before SA optimization. As the SA optimization trades logic for registers, the counts were more balanced for the optimized code and this reduced the number of slices used in most cases. Although such optimization was not always performed consistently, overall the slice utilization was reduced by 4.5% on average.

As the SA optimization techniques reduce the complexities of the SA and the carry adder chain, the logic optimization effort and hence the compilation time required by Design Compiler was substantially reduced. With the delay constraint, the step up of tool efficiency can be as high as 45.2%. This is significant in view of the several hours of runtime required originally to achieve timing closure for long filters.

From Table 6.2, the effective reduction after synthesis is bigger than the theoretical reduction. This discrepancy can be explained as follows. The optimization trades combinational area for registers and reduces the path delays in several taps. The reduced combinational logic requires less area and the area reduction is more prominent under tighter delay constraints. Moreover, the reduced path delays create more slacks, which enable slower but smaller cells to be used.

TABLE 6.2: Experimental Results for Two's Complement Structural Adder Optimization

Filter Name	**% red. for 4-bit input**					**% red. for 8-bit input**					**%**
	#FA	**Theo**	**DC A**	**DC D**	**FPGA**	**#FA**	**Theo**	**DC A**	**DC D**	**FPGA**	**MCM**
JAIN91_11	4	1.88	4.07	5.14	2.90	4	1.33	2.18	3.57	3.26	6.44
JANG02_11	2	0.91	2.34	1.81	1.84	2	0.64	1.62	2.52	1.20	6.61
DEMPSTER02_25	3	0.35	1.06	-0.19	0.28	3	0.28	0.70	1.48	0.68	24.99
ZAHOSAM89_25	7	1.27	1.07	3.10	0.69	7	0.93	0.29	2.51	1.27	7.66
VINOD03_26A	2	0.35	0.46	1.84	0.58	2	0.26	0.04	1.62	0.76	9.42
CHEN99_28A	5	0.64	1.44	1.11	1.11	5	0.50	1.08	2.70	1.66	10.27
CHEN99_28B	9	1.18	2.09	2.45	2.27	9	0.91	1.59	3.69	1.22	10.90
CHENYAO01_28A	9	1.19	2.90	2.36	2.00	9	0.92	1.80	2.44	1.85	9.70
CHENYAO01_28B	8	1.04	1.32	1.19	1.61	8	0.81	0.93	1.48	1.78	10.10
XU07_28	6	0.79	1.64	1.83	1.54	6	0.61	1.32	1.94	1.24	14.42
JHENG04_29A	11	1.50	2.61	3.63	2.41	11	1.14	1.73	2.38	1.76	10.46
JHENG04_29B	13	1.74	3.42	3.28	3.30	12	1.22	1.66	3.16	2.00	10.11
JHENG04_30	7	0.99	2.32	1.21	2.38	7	0.74	1.12	2.87	1.66	9.98
YLI01_30	10	1.34	3.73	4.01	3.88	10	1.01	2.19	2.66	2.92	9.75
BULL91_32	17	2.17	3.65	4.38	3.78	17	1.63	2.62	5.17	3.10	9.92
LIM83_36	10	1.16	2.93	3.24	5.85	9	0.78	1.88	3.42	4.71	4.25
LIM83A_36	15	1.82	3.94	4.67	4.42	17	1.52	3.13	5.01	3.72	3.90
LIM83_37	12	1.37	3.23	4.06	2.51	12	1.02	2.22	4.45	2.31	3.85
YEUNG04_40	38	2.31	3.64	7.18	5.17	38	1.93	2.81	5.51	3.45	21.13
KWENTUS97_47	5	0.30	0.87	2.12	2.34	5	0.25	0.70	2.02	1.74	13.80
ROSA04_49	28	2.39	2.94	4.74	3.96	31	1.96	3.49	4.55	4.04	8.30
SAMUELI89_60	53	2.61	3.68	7.69	5.51	51	2.03	2.56	6.94	3.91	10.63
LIM83_63	42	2.35	4.46	7.66	2.13	41	1.78	2.72	5.85	2.35	9.48
YOSHINO90_64	68	3.04	4.42	8.36	5.93	67	2.42	3.32	8.85	4.23	16.52
NIELSEN89_67A	54	2.65	3.92	8.05	4.35	52	2.04	2.84	6.87	4.45	10.73
NIELSEN89_67B	75	3.34	5.46	10.30	6.41	74	2.66	3.50	8.46	5.00	15.31
MASKELL07_A108	106	3.51	5.45	7.42	5.33	102	2.62	4.06	6.87	4.88	6.99
MASKELL07_B108	94	3.12	5.06	7.47	4.86	93	2.39	3.90	7.71	4.30	6.92
LIM83_121	140	3.27	4.75	10.28	3.84	138	2.63	3.91	9.06	3.87	13.47
LIMAKT08_121	144	3.47	4.39	10.25	5.92	142	2.77	3.26	9.21	5.35	10.38
LIMPASKO99_121	140	3.29	4.82	10.14	6.81	138	2.64	4.20	8.64	4.76	13.02
LIMYU07_121	146	3.47	4.57	9.80	5.52	143	2.76	4.34	8.94	4.99	11.65
AKSOY07_C180	276	4.64	7.21	13.54	8.97	273	3.78	5.68	10.92	7.16	12.72
AKSOY07_G200	377	5.50	7.13	12.84	10.26	371	4.45	6.52	11.61	7.81	11.55
AKSOY07_E240	411	5.20	7.68	11.95	10.07	408	4.28	6.16	11.79	7.62	11.73
AKSOY07_H240	430	5.34	7.69	13.44	n/a	422	4.32	6.25	11.92	7.59	10.95
AKSOY07_A200	362	4.71	5.91	10.97	8.76	349	3.75	4.40	10.23	6.31	13.50
AKSOY07_D200	282	3.54	5.25	9.29	7.07	272	2.84	3.99	8.59	5.36	13.94
AKSOY07_F300	468	4.75	7.19	11.97	9.92	467	3.91	5.91	10.64	7.13	10.74
MASKELL07_222	360	5.00	5.90	11.23	9.99	360	3.99	4.63	11.04	7.61	6.85
AKSOY07_I240	549	6.22	8.46	13.68	11.51	542	5.03	6.74	12.44	8.16	9.83
AKSOY07_B240	517	5.74	7.72	11.77	10.51	505	4.62	5.30	11.02	7.32	12.19
AKSOY07_J300	642	6.01	8.42	13.31	11.55	634	4.87	6.75	12.45	8.25	9.06
MASKELL07_441	912	5.67	6.59	12.58	10.73	900	4.58	5.07	11.83	8.22	6.18
AVERAGE	*156*	*2.80*	*4.27*	*6.98*	*5.13*	*154*	*2.22*	*3.21*	*6.52*	*4.16*	*10.69*

One effect that has not been considered is the influence of the additional clock buffers and the marginally more complex clock network. It is not expected to have a large impact, but might reduce the savings slightly.

To show the dominance of SA block, the percentage area occupied by the MCM block ('%MCM') is also evaluated. The area for the MCM block is higher when the delay constraint is imposed as the data paths of the MCM block are more closely coupled and area is traded for the critical path delay by the tools. Furthermore, the area for 8-bit input is always bigger than that for the 4-bit input as the latter has less overlapping digits between the operands to be added after the shifts. The area saved by the optimized SA block and the area of the MCM block are compared. It is remarkable to note that the area reduction of the optimized SA block is larger than the area of the MCM block in 9 cases and larger than half the MCM area in 29 cases for 8-bit input with delay optimization. For 8-bit input with area optimization, the area saved by SA optimization is larger than the area of the MCM block in 5 cases and larger than half the MCM area in 20 cases. For 4-bit input, the area saved by the optimized SA block is almost always larger than the area of the MCM block with only a few exceptions.

As expected, the absolute savings are very close between the 4-bit input and the 8-bit input results. This means that if the input bit width is increased further, the absolute savings will be very similar, but due to the larger adders the relative savings will be smaller. This was verified by running the optimization for bit widths of 12 and 16 and looking at the area results for the SA block alone. The results for the example filter with 4-bit input show an additional area cost of $\approx 71800\mu m^2$ with a standard deviation of $478\mu m^2$ for non optimized SA block and $\approx 71500\mu m^2$ with a standard deviation of $248\mu m^2$ for the optimized case. For all tested filters, the additional area cost increases linearly with the input bit width. The absolute savings are on average constant for input bit widths of $4, 8, 12$ and 16, though some filters may exhibit a slight increment in absolute area savings from smaller to bigger bit width while some others may exhibit similar but opposite trend.

Figure 6.11 shows a plot of the percentage reduction in equivalent FA by the proposed technique for different filter sizes. It is observed that the area savings increase with increasing filter length. The increment per additional filter tap declines with growing filter length. The reason of this trend is unclear but the curve $y = -1.4684 + 0.5092 \cdot \sqrt{x} - 0.0091x$ provides a good fit for the data points plotted in Figure 6.11.

Summary

A method to reduce the total area of a fixed coefficient transposed direct form FIR filter by minimizing the bit widths of the SAs in two's complement representation was presented. For two's complement representation, the method reduces the required sign extensions by bisecting the partial sums and shortens the delay path in some sections at the expense of additional registers and a few adders to merge the bisected partial sums. It is possible to obtain a trade-off that leads to area reduction. The optimal bisection was determined analytically. Theoretical calculation shows an area reduction of up to 6.2% (4-bit input) and 5.0% (8-bit input) for the

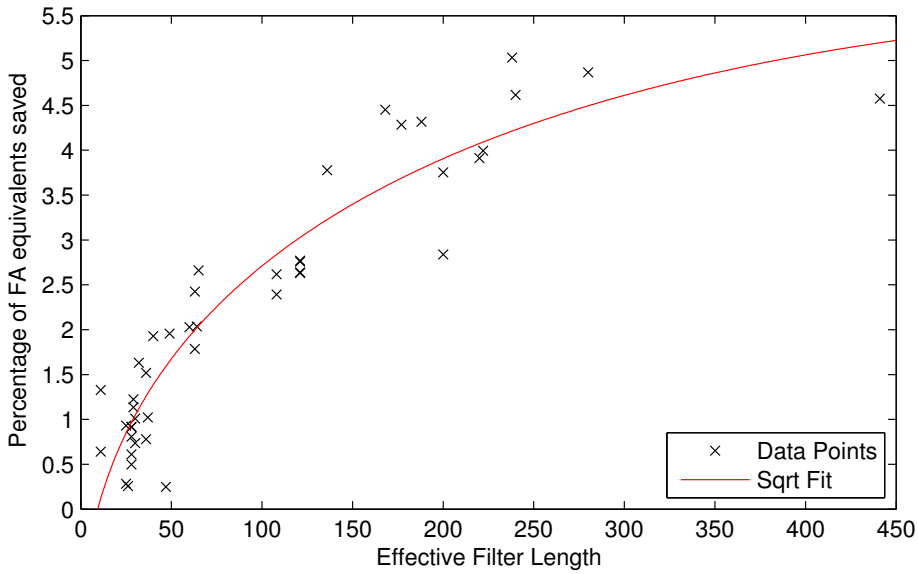


Figure 6.11: Relation Between Filter Length and Percentage of Saved FA Equivalents for 4-bit Input

SA block for a set of benchmark filters. The proposed methods can be combined with an MCM optimization algorithm to further reduce the required chip area. Synthesis results showed that with merely SA optimization, an area reduction of up to 13.68% for timing constrained ASIC implementation, and a reduction of 11.55% of equivalent gates for FPGA implementation were achieved.

6.2.3 Structural Adder Area Reduction for Offset Representation

The offset representation, also called MSB-Fix [42] or Correction Vector (CV) method, shifts the signed value range to an all positive value range by a constant value by inverting the MSB. In offset representation, all values are positive, so the adders can be simplified as the need for sign extension has been eliminated. This will shorten the critical path of the SA block. For many adders of the SA block, the carry needs to be rippled through only the difference in length between $n(p_i)$ and $n(q_i)$ due to the leading zeros of the shorter operand. This means that only HAs instead of FAs are required for these bits of the SA. This simplification of SA comes with some prices. A subtractor is needed to subtract the sum of all constant offsets after the last addition. The required bit width of the structural adder may also be increased by one or two bits (See Section 2.1.6).

In offset representation, the reduction in the higher order bits can only eliminate HAs as opposed to the elimination of FAs in two's complement when the same technique is applied. A FA consists of two HAs. The area ratio of FA:HA in the TSMC 0.18μm standard cell library [166] is about 2:1. Using the area estimate of 2 HAs $\approx$ 1 FA $\approx$ 1 FF, at least two HAs have to be saved to compensate for an additional FF. The bisection technique presented earlier for two's complement representation could only save half of the area for the offset representation

than what it could save with the two's complement. The HA reduction could only outweigh the FF increment when the difference δ_i between $\mathrm{n}(p_i)$ and $\mathrm{n}(q_i)$ is very large. Hence a different approach is needed. The HAs to the left of the MSB of the shorter operand for the SA could be replaced by a FA to add one more bit. This can be used to stop the carry from rippling through the HA chain in one of the previous SA. There the carry is saved and forwarded, resulting in a partial carry-save structure.

If δ_i and δ_{i-1} are both greater than 3, the carry output of the FA at the MSB of the shorter operand of SA a_i is not ripple-added till the MSB of the longer operand but saved for the next SA a_{i-1} by a FF. This reduces the HA count of the current SA by at least 4 by adding one FF and replacing the HA by a FA in the next SA. This is repeated until the difference δ_i is smaller than 3. In the last SA for which this condition is met, the HA carry chain is retained except with a FA to add the carry forwarded from the previous SA. Due to the small minimal difference required, it is possible to save HAs when the bit width for the SAs increases. Careful analysis of a number of example filters revealed that if the carry is forwarded to the next SA instead of allowing to ripple in the current SA, the increment from $\mathrm{n}(p_i)$ to $\mathrm{n}(p_{i+1})$ will also be postponed to the next SA. This gives a second optimization opportunity. The condition for this companion optimization is that if the bit width of the SA a_i is increased from SA a_{i+1} and $\delta_i > 2$, then the carry is forwarded to the next SA. This saves at least one HA, as the bit width is not increased. Consequently, one FF is freed, which can be used to save the carry. For cases with $\delta_i = 2$, no saving is possible, but postponing the bit width increment of a tap reduces the delay of the SA and leaves more room for area optimization under delay constraint.

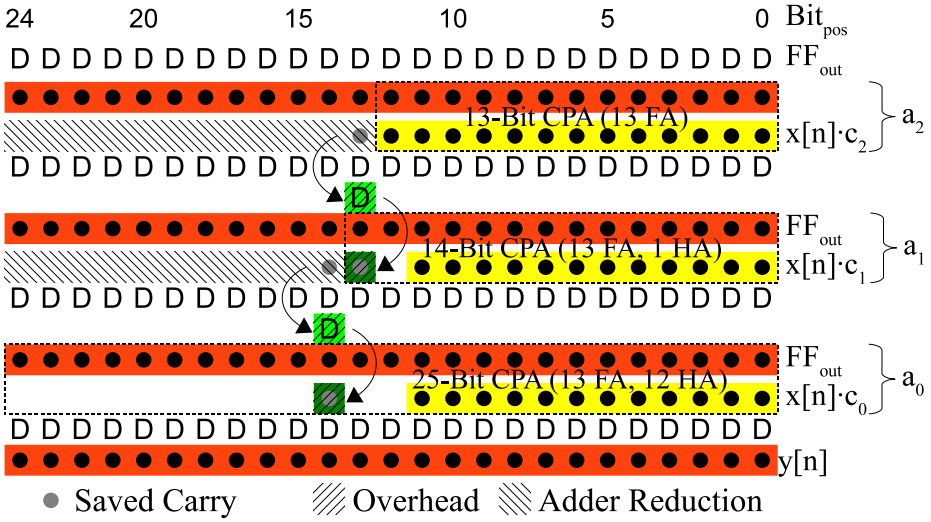


Figure 6.12: Binary example on the optimization of the last three adders of Figure 6.7 for offset representation

Figure 6.12 shows the optimization on the last three taps of the example filter. The CPA for the SA a_2 is shortened to the length of the multiplier output, as the carry is saved at the cost of one additional FF. For the SA a_1, the CPA is also shortened up to the position where

the carry forwarded from a_2 is encountered. At that position, the HA is replaced by a FA for which the saved carry is used as second input and the carry output is again saved. The last SA a_0 adds the saved carry with the carry propagating from the MSB of its shorter operand by a FA and then propagate the carry till the MSB of the longer operand with HAs. The delay of this SA is only increased slightly by the additional FA. This is insignificant as the constant multiplier for the last tap is generally small and therefore the total amount of FAs is smaller than other SAs.

With carry saved, the position of the carry output from the current SA h_i is increased by one. Hence the difference δ_i between the preceding SA output $\mathrm{n}(p_i)$ and h_i is reduced by one. To maximize the savings, it is important to determine when to stop saving the carry continuously and let it ripples to the MSB to allow the saving of carry at a lower position to be reinitiated. Under the assumption that the difference δ_i is constant for several SAs before applying the proposed method, the ideal number of steps g between merges is calculated by the following surprisingly simple equation.

$$g = \mathrm{round}\left(\sqrt{2 \cdot \delta_i}\right) \tag{6.16}$$

Ideally, g is a square root function of δ_i, which means that it will increase slowly with δ_i. A closed form expression for the case of not constant δ_i has not been established. However, if δ_i increases, the motivation to merge the saved carry and start anew is higher. If the merge is postponed past the ideal number of SA g, the possible saving declines slowly. To reduce the runtime, the search for possible merge points should be stopped some time after g, in the proposed method $2 \cdot g$, which is its next integer multiple, was used.

Due to the large δ_i closer to the output, it can be advantageous to skip a merge of the saved carry and delay the merge till the last tap by saving the carry without addition. If a saved carry is not merged, the HAs for ripple carry addition of the SA at the current tap can be eliminated. However, this saving is offset by the FF required to store the saved carry. To limit the number of additional FFs, the merging of saved carry can only be skipped at the last few taps.

Based on the above discussions, Algorithm 3 can be modified to Algorithm 4 for finding the most optimal partial sum splitting (for carry saved and forwarding in this case) and merging points. Similar to Algorithm 3, the search for the optimal solution can also be performed exhaustively in a very short time. The worst case complexity of the search is $\mathrm{O}(n^2)$ but it can be reduced to linear complexity by limiting the repetition of the saving of the carry by $a \cdot g$ with $a > 1$.

This algorithm was implemented and realized as a module for sysFIR. A graphical chart was generated for the visualization of the optimized filter. Figure 6.13 shows the characteristics of the example filter. The direction of signal flow in the SA block is from top to bottom, where tap 0 is the output tap. Color lines are used to demarcate the bit width boundaries of various signals or binary strings. Bits on the right of the red lines are the trailing zeros. The blue and the green lines mark the MSB positions of the coefficient multiplier outputs and the

Algorithm 4 Pseudocode for Finding the Carry-Saved and Carry-Merged Points for the Optimization of SA Block with Offset Representation

Inputs: Filter length N, bit width information $\mathrm{n}(p_i), \mathrm{n}(q_i)\ \forall i$

OptimizeStructuralAdderCV

```
1: i = 1
2: while i ≤ N
3:    if n(q_i) + 2 ≤ n(p_i) or n(p_i) + 2 ≤ n(p_{i+1}) then
4:       calculate reduction and cost to save the carry at bit j for n(q_i) ≤ j ≤ n(p_i) − 2
5:       thereby use previous results to evaluate merge position from 0 ≤ k < i
6:       save best reduction and carry save/merge points for i
7:       if n(p_i) + 1 ≤ n(p_{i+1}) then
8:          calculate reduction and cost to save the carry at bit n(q_i)
9:          save reduction, carry save/merge points and modified bit width information
10:      if i < 2 · (n(p_i) − n(q_i)) then
11:         evaluate reduction and cost by forwarding carry all the way to last tap
12:         save reduction and carry save/merge points if it is better to forward the carry to
            the last tap.
13:   else
14:      save zero for no reduction
15:   i = i + 1
16: select best result ∀i
17: return carry save/merge information, reduction and cost
```

outputs of SAs. The light blue rectangles show the saved (i.e., stored and forwarded) carries and 'X' represents an eliminated HA. Two leftmost eliminated HAs of the reduced CPA are not marked as 'X' as they are used to offset the FF added to store the carry. On the upper half of the figure, the growth of SA bit width is delayed[1] and the bottom half shows the normal savings with frequently repeated merging of forward carry. In the last tap, two light blue boxes are present. The right light blue box represents the saved carry from the former tap and the left light blue box shows the saved carry that was forwarded from two taps before it.

The offset representation method is able to save 246 HAs compared to the saving of 140 FAs for the two's complement representation method for the example filter. The effective saving (246 HAs $\approx$ 123 FAs) is smaller. The overall cost of the SA is decreased by the reduction from FA to HA due to the avoidance of sign extension. Meantime, the constant adder for the correction vector and the higher bit width of the partial sums incurs additional cost (see Section 2.1.6). All these factors make a fair comparison of the cost savings of SA block by these two methods difficult as the strength of these counteracting effects varies with the filters.

Implementation Results

The SA optimization for offset representation was integrated into sysFIR (see Chapter 3) to generate the VHDL codes automatically. The VHDL descriptions were synthesized by Synopsys Design Compiler (Version 2007.12) using TSMC 0.18μm standard cell library [166] and Xilinx

[1]The figure shows the non-optimized bit width of the SA. At where the carry is saved and the bit width is increased, the increment would have happened at the next SA.

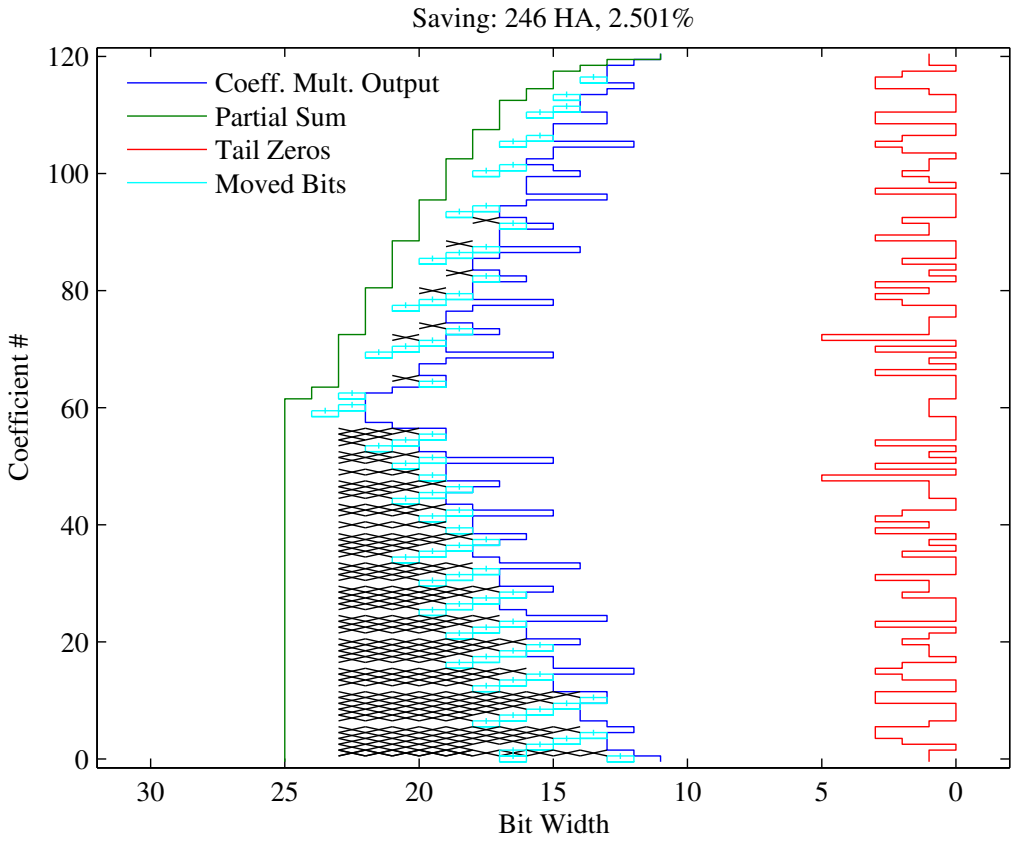


Figure 6.13: Visualization of Structural Adder Optimization for Offset Representation

ISE targeting Virtex 5 FPGA, xc5vlx30. Unconstrained (no delay constraint - 'DC A') and fixed timing constraint (3ns delay constraint - 'DC D') were used for the technology mapping and logic optimization by Synopsys Design Compiler. In Xilinx ISE, no delay constraint was used and the results reported under 'FPGA' are the savings in terms of the number of equivalent two input gates used. As MCM block is needed for the synthesis of the filter, direct CSD implementation was adopted. For the FPGA results, a separation of the MCM and SA block areas was not possible, and the reported percentage area reduction is relative to the overall filter size. The theoretical savings and percentage area reduction from the synthesis results for 4- and 8-bit input signals were listed in Table 6.3 for the same set of filters as Table 6.2. The number of HAs saved '#HA' was used for the theoretically estimated percentage reduction 'Theo'. Comparing it with the results in Table 6.2 for the two's complement optimization method, the estimated savings are larger for filters with fewer taps and smaller for filters with over hundred taps taking into account that 2 HAs *approx* 1 FA. The synthesis results in Table 6.3 were obtained by Design Compiler for only the SA block. These results do not match the theoretical estimates. One reason is the carry only ripples through the HAs in the original SA block with offset representation, which has less impact on delay constraint. This gives Design Compiler more room for logic optimization to reduce both the area and delay. So

the elimination of the HAs from the SA will not further reduce the area and delay as much as would be expected. As the offset representation reduces the combinational logic but not the FF count, the imbalance between logic and registers is larger than for the two's complement SA. The net gain is lowered by the optimization method, which trades registers for logic. This is also one of the reasons why the percentage area reduction results for FPGA exceed those of ASIC implementation and the theoretically estimated savings.

Summary

A method to reduce the total area of a fixed coefficient transposed direct form FIR filter by minimizing the bit widths of the SAs in offset representation was presented. For offset representation, the method reduces the carry propagation by saving and forwarding the carry to the next tap at the expense of additional registers and replacement of a few HAs to FAs in order to add/merge the saved carries. To have an area reduction, the condition that the bit width difference between the two inputs of the SA has to be at least 3 has been established. Average area reductions of the SA block of up to 4.95% (for 4-bit input) and 3.96% (for 8-bit input) for a set of benchmark filters have been achieved based on the theoretical estimation. Synthesis results showed that with mere SA optimization method for the offset representation, an area reduction of up to 6.23% for timing constrained ASIC implementation, and a reduction of 8.52% of equivalent gates for FPGA implementation were achieved. Further chip area can be reduced by combining it with an MCM optimization algorithm.

6.2.4 Structural Adder Area Reduction - A Holistic Comparison

This section makes further comparison of the two different number representations and SA optimization techniques for these representations. Table 6.4 shows the comparison of the percentage area reduction of the offset representation scheme over two's complement scheme for 4 and 8 input bits filters. The percentage area reductions were obtained without applying specific delay constraints and the only design goal for the synthesis by Design Compiler was area minimization. The theoretical estimation of the difference in area of offset representation over two complement's representation are presented in the column 'Equiv.'. The calculation is based on the total number of HAs, FAs and FFs within the SA block. As these elements have different costs, they are converted to HAs by by 1 FF = 1 FA = 2 HAs. On average, the theoretical savings of offset representations over two's complement representation are nearly 9% for 4-bit input and 7.5% for 8-bit input. This is because no sign extension is required for offset representation and only HAs are present for the difference between the bit widths of the two inputs for each SA. The columns 'SA A' and 'Filt. A' show the percentage area savings of offset representations over two's complement representation obtained by Design Compiler for the SA block only and for the whole filter, respectively. The column 'FPGA' shows the percentage area reduction of offset representation over two's complement representation for FPGA implementation. It should be noted that for the percentage reduction results reported in 'Filt. A' and 'FPGA', the offset representation is applied to only the SA block. Two's

TABLE 6.3: Experimental Results for Offset Representation Structural Adder Optimization

Filter Name	% red. for 4-bit input					% red. for 8-bit input					%
	#HA	Theo	DC A	DC D	FPGA	#HA	Theo	DC A	DC D	FPGA	MCM
JAIN91_11	7	1.76	0.75	0.30	2.79	7	1.22	0.47	0.11	2.36	6.81
JANG02_11	7	1.70	0.37	0.05	2.04	5	0.85	0.33	1.07	1.37	6.39
DEMPSTER02_25	22	1.34	0.04	1.71	2.01	22	1.08	0.04	0.30	1.26	23.34
ZAHOSAM89_25	19	1.86	0.42	-0.29	2.84	19	1.34	0.36	0.43	2.27	7.10
VINOD03_26A	19	1.81	0.23	-0.25	2.90	19	1.30	0.18	0.79	1.85	8.99
CHEN99_28A	14	1.01	-0.02	-0.28	2.00	21	1.13	0.35	0.99	1.69	9.61
CHEN99_28B	25	1.79	0.54	0.19	3.02	25	1.36	0.43	-0.03	2.13	9.59
CHENYAO01_28A	19	1.39	0.29	0.86	2.66	19	1.05	0.21	0.09	1.71	9.00
CHENYAO01_28B	21	1.51	0.50	0.35	2.79	21	1.15	0.36	0.29	1.92	8.99
XU07_28	16	1.17	0.34	0.90	2.23	16	0.89	0.27	1.83	1.48	12.68
JHENG04_29A	20	1.50	0.27	0.03	2.86	20	1.12	0.07	1.25	2.00	10.08
JHENG04_29B	19	1.40	0.48	0.15	2.60	19	1.05	0.31	-0.90	1.61	8.74
JHENG04_30	18	1.42	0.28	0.29	2.11	18	1.04	0.19	0.45	1.81	8.87
YLI01_30	22	1.63	0.73	1.42	2.93	22	1.22	0.58	1.14	1.90	8.83
BULL91_32	38	2.61	0.75	0.81	3.38	38	1.93	0.53	0.58	2.54	9.88
LIM83_36	38	2.39	0.24	-0.69	5.18	38	1.75	0.13	0.96	3.71	3.30
LIM83A_36	47	3.08	0.94	0.12	4.83	47	2.23	0.65	1.47	3.37	3.25
LIM83_37	40	2.47	0.37	0.60	2.82	40	1.81	0.23	0.38	2.64	3.11
YEUNG04_40	68	2.20	1.09	1.18	3.74	68	1.82	0.89	3.08	2.68	21.42
KWENTUS97_47	15	0.53	0.22	-0.21	1.38	15	0.43	0.55	1.23	1.05	12.94
ROSA04_49	60	2.82	1.22	0.72	3.94	77	2.59	1.14	1.10	3.92	7.67
SAMUELI89_60	112	2.95	1.29	1.90	4.57	113	2.37	2.00	3.26	3.31	11.81
LIM83_63	97	2.97	0.89	0.95	2.84	97	2.27	0.68	-0.58	2.75	9.95
YOSHINO90_64	116	2.82	1.45	2.20	5.16	118	2.29	2.24	4.05	3.56	17.06
NIELSEN89_67A	111	2.95	1.10	1.12	4.82	111	2.32	0.82	1.73	3.27	10.88
NIELSEN89_67B	123	2.98	1.59	1.61	5.31	125	2.41	2.23	3.36	3.68	15.92
MASKELL07_A108	215	3.95	1.79	1.38	6.71	215	3.00	1.28	2.77	4.70	7.36
MASKELL07_B108	201	3.71	1.69	0.69	6.41	207	2.90	1.31	2.30	4.63	6.77
LIM83_121	246	3.12	1.34	2.27	3.35	246	2.50	2.16	3.93	3.13	14.29
LIMAKT08_121	249	3.25	1.42	1.43	5.52	249	2.59	2.52	4.26	3.95	11.26
LIMPASKO99_121	246	3.13	1.32	1.76	5.28	246	2.51	2.22	3.83	3.77	13.56
LIMYU07_121	276	3.57	1.57	2.16	5.96	278	2.88	2.50	3.56	4.29	12.70
AKSOY07_G200	548	4.50	2.75	4.12	7.98	550	3.64	2.15	3.64	5.96	11.35
AKSOY07_H240	623	4.39	4.79	5.29	7.99	625	3.57	3.85	5.62	5.95	11.84
AKSOY07_A200	582	4.12	2.21	2.00	6.98	582	3.36	3.15	4.46	5.15	14.43
AKSOY07_D200	493	3.36	1.51	2.22	5.86	494	2.76	2.63	4.65	4.39	14.50
AKSOY07_F300	661	3.85	2.38	2.55	7.24	663	3.14	3.33	5.85	5.43	11.68
MASKELL07_222	600	4.63	2.51	0.92	8.18	599	3.63	3.42	5.22	6.26	7.20
AKSOY07_I240	787	4.95	4.99	5.18	8.52	781	3.96	3.94	6.17	6.35	10.36
AKSOY07_B240	756	4.63	2.61	1.83	8.20	769	3.81	3.36	5.57	6.07	13.34
AKSOY07_J300	909	4.80	2.57	3.14	8.47	906	3.85	3.84	5.13	6.40	9.47
AVERAGE	*249*	*2.73*	*1.26*	*1.29*	*4.55*	*250*	*2.18*	*1.46*	*2.40*	*3.43*	*10.57*

TABLE 6.4: Area Reduction by Offset Representation Implementation Results over Two's Complement Implementation with SA Optimization

Filter Name	**% red. for 4-bit input**				**% red. for 8-bit input**			
	Equiv.	**SA A**	**Filt. A**	**FPGA**	**Equiv.**	**SA A**	**Filt. A**	**FPGA**
JAIN91_11	6.81	12.18	11.59	-12.25	4.82	10.65	9.81	-8.17
JANG02_11	6.59	10.82	10.33	-11.96	5.45	10.40	9.61	-8.00
DEMPSTER02_25	4.65	11.00	10.49	-4.10	4.14	10.63	8.93	-2.95
ZAHOSAM89_25	7.53	12.95	12.62	-4.27	5.53	10.95	10.27	-2.30
VINOD03_26A	8.39	12.72	12.33	-5.66	6.67	10.77	9.93	-4.05
CHEN99_28A	10.96	12.95	12.55	-0.83	7.26	9.67	8.93	-3.72
CHEN99_28B	8.53	13.13	12.76	-1.83	6.85	11.70	10.79	-1.93
CHENYAO01_28A	9.22	11.18	10.89	-2.75	8.11	10.18	9.44	-2.76
CHENYAO01_28B	9.40	11.10	10.81	-3.94	8.11	10.49	9.70	-1.96
XU07_28	9.83	9.66	9.32	-3.05	8.99	9.33	8.38	-2.11
JHENG04_29A	9.32	11.04	10.74	-3.85	8.10	10.09	9.25	-3.08
JHENG04_29B	8.85	10.38	10.08	-3.53	7.76	9.58	8.84	-2.80
JHENG04_30	10.20	10.62	10.33	-3.92	9.11	9.71	9.06	-2.87
YLI01_30	9.58	10.45	10.15	-4.10	8.84	9.89	9.17	-2.49
BULL91_32	7.21	13.65	13.29	-1.75	5.47	11.80	10.95	-1.63
LIM83_36	7.66	13.20	13.03	1.78	6.07	11.66	11.30	1.27
LIM83A_36	7.51	12.91	12.73	0.84	5.94	11.52	11.17	0.72
LIM83_37	7.79	13.40	13.24	-1.84	5.83	11.30	10.97	-1.30
YEUNG04_40	6.11	13.35	12.86	0.78	5.41	12.62	11.01	0.67
KWENTUS97_47	14.58	8.00	7.78	-1.51	14.21	1.79	1.61	-1.08
ROSA04_49	9.31	14.75	14.44	-0.32	6.25	11.77	11.12	-0.93
SAMUELI89_60	6.59	12.87	12.63	0.40	5.29	3.66	3.36	-0.47
LIM83_63	8.56	15.24	14.98	-0.29	6.85	13.48	12.54	-0.53
YOSHINO90_64	8.24	14.38	13.98	0.88	6.98	4.94	4.46	0.56
NIELSEN89_67A	7.80	14.39	14.12	0.10	6.31	13.15	12.21	0.11
NIELSEN89_67B	8.14	14.85	14.49	1.32	6.81	5.30	4.82	1.08
MASKELL07_A108	9.94	16.52	16.29	1.61	8.07	14.75	14.03	1.84
MASKELL07_B108	10.15	16.05	15.88	1.34	8.26	14.19	13.54	1.56
LIM83_121	7.67	14.72	14.43	-2.89	6.27	5.15	4.75	-1.10
LIMAKT08_121	7.59	14.54	14.38	1.39	6.34	4.95	4.65	1.48
LIMPASKO99_121	7.71	14.79	14.52	1.70	6.29	5.30	4.90	1.21
LIMYU07_121	8.19	15.22	14.96	1.83	6.81	5.72	5.32	1.74
AKSOY07_G200	11.02	15.93	15.69	3.46	9.46	14.45	13.54	2.69
AKSOY07_H240	11.77	7.65	7.56	3.43	10.38	6.18	5.80	2.70
AKSOY07_A200	8.23	15.89	15.66	2.91	6.88	5.93	5.51	2.07
AKSOY07_D200	7.91	15.45	15.23	2.87	6.68	5.46	5.04	2.27
AKSOY07_F300	12.89	15.02	14.86	3.82	11.52	6.15	5.80	2.85
MASKELL07_222	10.03	16.81	16.69	3.53	8.48	7.19	6.95	3.12
AKSOY07_I240	9.96	8.78	8.69	4.42	8.40	7.06	6.69	3.33
AKSOY07_B240	9.26	16.73	16.49	4.13	7.70	6.73	6.29	2.87
AKSOY07_J300	11.25	17.07	16.92	4.83	9.69	7.52	7.16	3.62
MASKELL07_441	10.81	17.60	17.52	4.81	9.16	7.53	7.32	4.12
AVERAGE	*8.90*	*13.33*	*13.06*	*-0.54*	*7.42*	*9.08*	*8.45*	*-0.34*
OVERALL[1]	*9.91*	*14.63*	*14.43*	*2.62*	*9.91*	*7.71*	*7.24*	*2.05*

complement arithmetic operations are used in the MCM block of both methods consistently for an unbiased comparison. The area reduction by offset representation is significant for standard cell implementation but the absolute savings between 4-bit and 8-bit inputs are very similar. Due to the increase of the operator bit widths with input bit width, the saving becomes smaller for higher input bit width (8.45% for 8-bit input compared with 13.06% for 4-bit input). FPGA implementation do not benefit from the implementation of the SA block in offset representation at all. By considering the average savings over all filters tested, the reduction is only about 2%. The reason for this is that the reduction from FAs to HAs does not translate into area savings in FPGA as only the complexity of the LUT is reduced, but no logic slice can be eliminated.

Table 6.5 shows a comparison of the complete filter implementation against different baseline implementations. The first row of the header indicates the optimizations applied to the implementations, which can be a combination of either two's complement or offset representation with the corresponding proposed SA optimization method. 'TC Opt.' denotes that the SA block of the filter is implemented with the proposed optimization technique in two's complement representation; 'OR' denotes a plain offset representation implementation. The arithmetic operations in the SA block of the filter are implemented by offset representation without applying the proposed SA optimization technique and without register initialization, i.e.,there will be $N-1$ incorrect output samples after startup; 'OR Opt.' denotes that the SA block of the filter is implemented with the proposed optimization technique in offset representation; 'OR Opt w Init.' denotes that the SA block of the filter is implemented with the proposed optimization technique in offset representation and in addition, all registers are initialized with the error vectors upon reset so that the outputs of the FIR filter are correct from the first sample. The second row in the table header denotes the corresponding baseline method that is being compared with the method indicated in the first row. All MCM blocks except the 'CSD' are optimized by the 'MinLD' algorithm (See Section 4.2). 'CSD' denotes the implementation of a straightforward CSD coefficient in the MCM block. The bit width of the input signal is 8 bits for all filters. All data presented refer to the total area of the complete filter consisting of both MCM and SA blocks synthesized by Synopsys Design Compiler with area optimization. The first column 'MinLD/CSD' shows the percentage area reduction achieved by the MinLD (See Section 4.2) MCM algorithm over the CSD baseline. On average, the optimization of MCM block by MinLD algorithm results in a reduction of less than 1% of the total filter area. This is mainly due to the fact that logic optimization performed by Design Compiler can easily remove most redundant logics of the baseline CSD MCM block. The next two columns ('TC Opt / MinLD' and 'TC Opt / CSD') show the percentage area reductions of the SA optimization in two's complement representation with MinLD optimized MCM against the CSD baseline and the MinLD optimized MCM, both without SA optimization, respectively. The average area reduction of 'TC Opt' in excess of the MinLD MCM optimization for all filters is 2.89%, which is more than three times the area reduction of MCM optimization over CSD. A close examination of the percentage area reductions of all filters shows that not all filters gained as much from the SA optimization. For example, filter XU07_28 gains more

from MCM optimization than from SA optimization. For filters with more than hundred taps, SA optimization clearly reduces more areas. The columns 'OR/MinLD' and 'OR/CSD' refer to the percentage area reduction of 'OR' over 'MinLD' and CSD baseline, respectively. The average area reduction over 'MinLD' implementation reaches 8.32%, which is comparable to the average area reduction of 8.45% obtained from Table 6.4. The difference is most likely due to the heuristic of Design Compiler. The next two columns 'OR Opt / OR' and 'OR Opt / CSD' show the percentage area reductions for offset representation with SA optimization over the plain offset representation and CSD baseline, respectively. Optimizing the SA block in offset representation gives a further area reduction of 1.44% on average and 2.64% on weighted average[1]. The last two columns 'OR Opt w Init / OR Opt' and 'OR Opt w Init / CSD' show the percentage reduction of 'OR Opt w Init' over the optimized offset representation and CSD baseline, respectively. Interestingly, with initialization, the area increases by up to 2.44% over the 'OR Opt' for 27 filters and reduces by up to 1.45% for the other 17 filters, resulting in an average area increment of 0.53%. Surprisingly, the areas of large filters with many taps were reduced though many registers have to be preset. As a result, the weighted average area reduction of 0.51% is obtained over 'OR Opt'.

Overall, the presented methods for optimization of the structural adder block can achieve an average area reduction of nearly 10% for the tested benchmark filters for 8-bit input signal.

[1] weighted average is the average over all filters weighted by their respective areas.

TABLE 6.5: Percentage Area Reduction of Filters with SA Block Optimization in Two Different Number Representations and MCM Block Optimization

Filter Name	**MinLD**	**TC Opt**		**OR**		**OR Opt**		**OR Opt w Init**	
	CSD	**MinLD**	**CSD**	**MinLD**	**CSD**	**OR**	**CSD**	**OR Opt**	**CSD**
AKSOY07_A200	1.44	3.81	5.19	4.97	6.33	2.88	9.03	1.17	10.10
AKSOY07_B240	1.40	4.58	5.92	5.79	7.11	3.45	10.32	1.45	11.62
AKSOY07_C180	1.16	5.41	6.51	5.03	6.13	3.06	9.01	1.21	10.11
AKSOY07_D200	0.99	3.86	4.81	4.51	5.46	2.46	7.79	1.35	9.03
AKSOY07_E240	0.53	6.08	6.58	5.70	6.21	3.25	9.26	1.19	10.34
AKSOY07_F300	1.27	5.34	6.54	5.32	6.52	3.27	9.57	1.04	10.51
AKSOY07_G200	1.30	5.95	7.17	5.33	6.56	3.46	9.79	1.26	10.93
AKSOY07_H240	0.75	5.98	6.69	5.29	6.00	3.51	9.30	1.04	10.25
AKSOY07_I240	0.41	6.43	6.82	6.34	6.72	3.79	10.26	1.23	11.36
AKSOY07_J300	0.74	6.58	7.27	6.68	7.37	3.60	10.70	1.13	11.71
BULL91_32	1.00	2.49	3.47	10.74	11.63	0.49	12.06	-1.90	10.40
CHEN99_28A	1.45	0.45	1.89	8.46	9.79	0.32	10.08	-1.45	8.78
CHEN99_28B	0.83	1.29	2.11	10.19	10.94	0.40	11.29	-1.57	9.90
CHENYAO01_28A	1.20	1.50	2.68	8.83	9.93	0.19	10.10	-1.24	8.99
CHENYAO01_28B	0.90	0.79	1.68	8.98	9.80	0.33	10.09	-1.07	9.13
DEMPSTER02_25	1.98	0.69	2.66	8.64	10.45	0.01	10.46	-2.44	8.28
JAIN91_11	0.00	2.05	2.05	9.81	9.81	0.43	10.19	-1.75	8.62
JANG02_11	0.46	1.68	2.13	9.17	9.59	0.30	9.86	-1.22	8.76
JHENG04_29A	0.84	1.40	2.23	8.98	9.75	0.06	9.81	-1.62	8.35
JHENG04_29B	1.05	1.32	2.35	8.71	9.66	0.28	9.92	-2.25	7.89
JHENG04_30	0.27	0.92	1.19	8.99	9.24	0.17	9.39	-1.61	7.93
KWENTUS97_47	0.76	0.31	1.06	1.10	1.85	0.43	2.28	1.07	3.32
LIM83_121	1.13	3.77	4.86	12.11	13.10	0.90	13.88	-1.67	12.44
LIM83_36	0.30	1.82	2.11	11.12	11.39	0.13	11.50	-1.64	10.05
LIM83_37	0.00	2.14	2.14	10.97	10.97	0.23	11.18	-1.59	9.78
LIM83_63	0.61	2.30	2.90	12.27	12.81	0.63	13.36	-1.41	12.14
LIM83A_36	0.12	2.99	3.10	11.13	11.23	0.63	11.79	-1.49	10.47
LIMAKT08_121	0.97	2.66	3.60	4.37	5.30	2.19	7.38	1.03	8.33
LIMPASKO99_121	1.56	3.73	5.24	12.05	13.42	0.98	14.27	-1.62	12.89
LIMYU07_121	1.03	4.15	5.14	12.43	13.34	1.20	14.37	-1.63	12.98
MASKELL07_222	0.25	4.39	4.62	6.80	7.03	3.27	10.07	1.37	11.30
MASKELL07_441	0.23	4.82	5.04	7.19	7.40	4.08	11.18	1.25	12.29
MASKELL07_A108	0.76	3.92	4.65	13.79	14.45	1.34	15.60	-1.32	14.48
MASKELL07_B108	0.51	3.82	4.31	13.54	13.98	1.29	15.09	-1.46	13.85
NIELSEN89_67A	1.16	2.52	3.65	3.80	4.91	2.03	6.84	1.17	7.94
NIELSEN89_67B	1.21	2.76	3.94	4.06	5.22	2.16	7.27	1.29	8.47
ROSA04_49	0.69	3.12	3.79	11.12	11.73	1.08	12.68	-1.76	11.15
SAMUELLI89_60	-0.05	2.35	2.30	10.76	10.71	0.92	11.54	-1.46	10.24
VINOD03_26A	1.55	-0.15	1.40	10.08	11.47	0.00	11.47	-1.96	9.74
XU07_28	3.12	0.62	3.71	7.59	10.47	0.24	10.68	-2.00	8.90
YEUNG04_40	-0.13	2.52	2.39	2.64	2.52	1.47	3.95	1.28	5.17
YLI01_30	1.16	1.94	3.08	8.93	9.99	0.54	10.47	-1.51	9.11
YOSHINO90_64	0.93	2.01	2.92	11.50	12.32	0.75	12.98	-1.56	11.62
ZAHOSAM89_25	0.62	0.24	0.86	10.25	10.81	0.34	11.11	-1.83	9.48
AVERAGE	*0.87*	*2.89*	*3.74*	*8.32*	*9.12*	*1.24*	*10.44*	*-0.53*	*9.98*
OVERALL	*0.85*	*4.39*	*5.21*	*7.14*	*7.93*	*2.64*	*10.36*	*0.51*	*10.82*

6.3 Structural Adder Pipelining

Pipelining of MCM blocks is a recent research topic [123], [167], [168]. The target is to increase the speed of the filter when reducing the adder depth is not sufficient to achieve the throughput requirement. In FPGA, the word-level pipelining of adders can be done at no additional cost as the registers present in the committed logic slice can be used. During the research for the subject [168], it was found that the critical path always lies within the SA block with word-level pipelining on MCM block as the longest adders are in the SA block. If the use of high speed adders likes CLA still cannot meet the throughput target, the RCAs will need to be pipelined. This pipelining is rather expensive, as both input signals to the RCA have to be latched as shown in Figure 6.14. For this reason, pipelining on the SA block has not been considered in the literature to date.

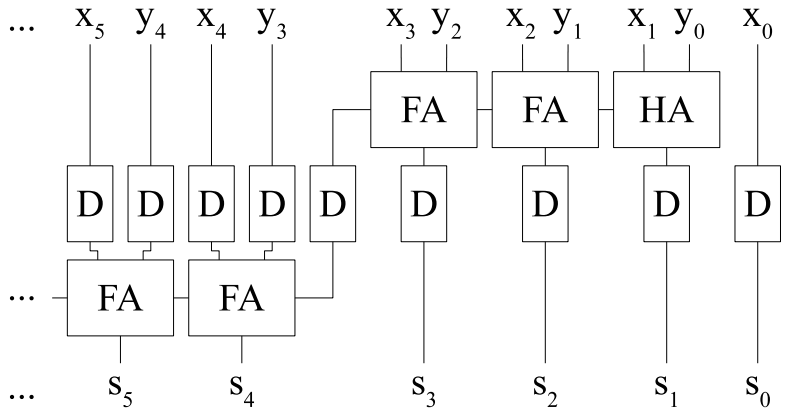


Figure 6.14: Pipelined RCA After Every Three Full Adders

The shortest delay option is to implement the filter in CSA structure. For filter implemented in the transposed direct from, the register cost for the SA block doubles, as the data path is split into the sum and carry signals. The MSB also needs to be handled with care (see Section 2.1.8). An ILP formulation and bit-level optimization for a complete decimation filter was presented in [123], but this method is designed for dedicated circuit structure likes decimation filer and is not applicable to large filters and filters with input bit width larger than three or four.

With the insight gained from the development of the structural adder optimization algorithms, a low overhead pipelining scheme for the SA block is proposed. For simplicity two's complement representation is considered.

6.3.1 Method for Pipelining of Structural Adders

The idea behind the proposed pipelining method is very similar to the optimization of structural adders in offset representation (see Section 6.2.3) presented earlier, where the carry is saved in one structural adder and then reinserted into the subsequent structural adder. The original registers between two structural adders of the filter will be reused as much as possible. The pipelining is carried out across SAs instead of within a single SA. The SA block is split

into pipelined blocks where each pipelined block contains a maximum number of RCAs (or faster implementation of CPAs). The carry after a smaller number of consecutive FAs is saved and forwarded to the next SA instead of continuing to propagate to the next FA of the current RCA. No bit width extension is required in general for the structural adder involved except for potential overflow cases where the value range is very close to the original bit width. At the last tap where the split signals are merged, extra number of stages are required. One disadvantage of this method is the challenge of verifying a more complicated signal flow as the otherwise easy to determine signals in the structural adder have to be derived with a lot more calculations.

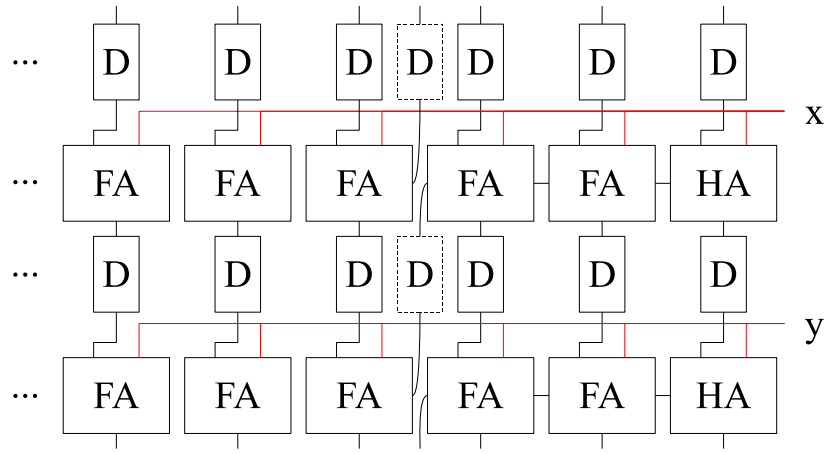


Figure 6.15: Proposed Pipelined RCA in SA Block with a Throughput of Three 1-bit Adder Delay

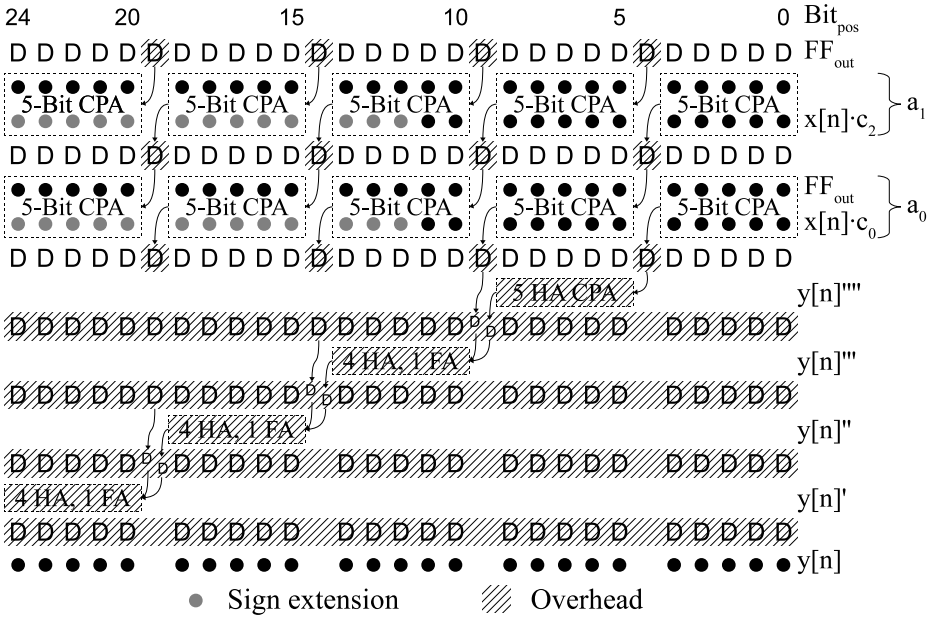


Figure 6.16: Last two taps of the pipelined structural adder block for Filter 1 of [28] with an 8-bit input signal and a 5-bit pipelining

Figure 6.15 shows two consecutive SAs of the proposed 3-bit RCA pipelining scheme and Figure 6.16 shows the last two taps of an example filter with a 5-bit RCA pipelining. From Figure 6.15, the only overhead (dotted register 'D') is the extra registers and no additional HA or FA is needed throughout the SA block until the last tap. The required overhead for the last two taps to establish the final result is shown in Figure 6.16. For this example of a 5-bit pipelining, four additional pipelined stages are required. One 5-bit segment is added and its output carry and the carry saved in the previous pipelined stage are saved in each additional stage. The additional saved carry can be added in the 5-bit RCA of the next stage by replacing the HA in its LSB position by a FA.

The pipelining changes the ranges of the RCA outputs for the two's complement representation due to the sign extension. The bit width calculation of the signals within the SA block has to account for the range expansion contributed by the weight of the saved carry. If it exceed the minimum bit width requirement (See Section 2.1.7), sign extension error will occur. For example, if a saved carry (or a borrow from a subtraction) is 1, the range will have to be extended by 2^m, where m is the weight of the saved carry. If the regular range expansion reaches the final output bit width, there is no need to extend the bit width of the SAs within the pipelined block. As no further sign extension of the data path can occur, the potentially incorrect sign bits can be corrected by subsequent calculation. This leads to a small variable overhead that is hard to be expressed in closed form. Therefore, only the deterministic overheads introduced by pipelining will be estimated.

Given the bit width $\mathrm{n}(a_i)$ of the structural adder a_i and the maximum allowable bit width of each CPA s_{CPA} after pipelining, the number of CPA blocks is given by:

$$n_{b,i} = \left\lceil \frac{\mathrm{n}(a_i)}{s_{CPA}} \right\rceil \tag{6.17}$$

The overhead incurred for pipeling a single SA is given by:

$$overhead_{FF,a_i} = \left\lceil \frac{\mathrm{n}(a_i)}{s_{CPA}} \right\rceil - 1 = n_{b,i} - 1 \tag{6.18}$$

Additional pipelined stages are required for the final calculation of $y[n]$. The overheads due to the additional FFs, HAs and FAs are given by:

$$overhead_{FF,y} = (n_{b,0} - 1) \cdot \mathrm{n}(a_0) + \frac{(n_{st,0} - 1)^2 + n_{st,0} - 3}{2} \tag{6.19}$$

$$overhead_{HA,y} = (n_{b,0} - 2) \cdot (s_{CPA} - 1) + (\mathrm{n}(a_0) \bmod s_{CPA}) \tag{6.20}$$

$$overhead_{FA,y} = n_{b,0} - 2 \tag{6.21}$$

Thus, the total overhead can be calculated as:

$$overhead_{total} = \sum_{i=0}^{N-1} overhead_{FF,a_i} + overhead_{FF,y} + overhead_{HA,y} + overhead_{FA,y} \tag{6.22}$$

Based on the area cost estimate of one FA is equivalent to one FF and two HAs, Equation (6.22) can be simplified to:

$$\begin{aligned} overhead_{total} &= \sum_{i=0}^{N-1} \left\lceil \frac{\mathrm{n}(a_i)}{s_{CPA}} \right\rceil - N + overhead_{FF,y} + overhead_{HA,y} + overhead_{FA,y} \\ &= \sum_{i=0}^{N-1} \left\lceil \frac{\mathrm{n}(a_i)}{s_{CPA}} \right\rceil + n_{b,0} \left(\mathrm{n}(a_0) + \frac{n_{b,0}}{2} + \frac{s_{CPA}}{2} \right) + \mathrm{n}(a_0) - s_{CPA} \\ &\quad + \frac{\mathrm{n}(a_0) \bmod s_{CPA}}{2} - N - 2 \end{aligned} \tag{6.23}$$

Depending on the length of the allowable bit width of each pipelined CPA block and the filter length, the overhead is mainly contributed by either extra registers between the structural adders (for small s_{CPA} and long filters) or the registers and adders in the additional pipelined stages required for the calculation of the final output (for larger s_{CPA} or short filters). Still this is comparatively smaller than the CSA implementation or straightforward pipelining of each structural adder.

This method can also be applied to the previously presented SA optimization in offset representation by combining the carry saving for pipelining with the carry saving for SA optimization in offset representation. However, the complexity of this method is high and will not be considered.

6.3.2 Implementation Results

To analyze the area overhead of different granularity of pipelining, Filter 1 of [28] was analyzed with an 8-bit input, the corresponding 25 output bits and different maximum allowable bit width of pipelinedCPA from 2 to 24. In Table 6.6, 'Blocks' refer to the total number of pipelined CPAs; 'FA equiv.' refers to the total area of a non-pipelined SA block implementation in terms of number of FAs; 'PL FF', 'PL FA' and 'PL HA' refer to the extra number of 1-bit registers, full adders and half adders, respectively, required for pipelining. 'PL FA equiv.' refers to the total area overhead incurred by pipelining the SA block in terms of the number of FAs. 'Inc. %' refers to the percentage area increment of pipelined structural adder block and 'WL Incr.' refers to the word length increment by summing up the bit width increment of every SA. The results show that for $s_{CPA} = 2$, thirteen blocks are required for pipelining. With increasing s_{CPA}, the number of blocks reduces rapidly to two eventually when $s_{CPA} \geq 13$. For $s_{CPA} > 13$, the implementation are not very efficient as the two adder blocks have different sizes and the reduction in area overhead is insignificant. In the case of this filter, its area overhead actually increases initially with larger s_{CPA} due to the overhead required for the word length extension. If s_{CPA} is increased further, the area overhead reduces because the taps further from the output are smaller than s_{CPA}, and need not be pipelined.

Ideally, the adders are split into equally sized blocks. For the example filer and input bit width of eight, $s_{CPA} \in \{2, 3, 4, 5, 7, 9, 13\}$ are the most ideal.

The additional area required for pipelining the example filter is less than 38% if the maximum

TABLE 6.6: Theoretical results for pipelining Filter 1 of [28] with different maximum allowable bit widths of pipelined CPAs

s_{CPA}	Blocks	FA equiv.	PL FF	PL FA	PL HA	PL FA equiv.	Inc. %	WL Incr.
2	13	5247	1815	131	12	1952.0	37.20	120
3	9	5247	1151	75	15	1233.5	23.51	68
4	7	5247	839	51	16	898.0	17.11	46
5	5	5247	587	38	12	631.0	12.03	35
6	5	5247	538	30	16	576.0	10.98	27
7	4	5247	421	27	16	456.0	8.69	25
8	4	5247	390	18	15	415.5	7.92	16
9	3	5247	295	23	15	325.5	6.20	22
10	3	5247	280	22	14	309.0	5.89	21
11	3	5247	251	11	13	268.5	5.12	10
12	3	5247	239	7	12	252.0	4.80	6
13	2	5247	152	8	12	166.0	3.16	8
14	2	5247	153	10	11	168.5	3.21	10
15	2	5247	151	12	10	168.0	3.20	12
16	2	5247	153	16	9	173.5	3.31	16
17	2	5247	155	23	8	182.0	3.47	23
18	2	5247	146	21	7	170.5	3.25	21
19	2	5247	141	22	6	166.0	3.16	22
20	2	5247	130	19	5	151.5	2.89	19
21	2	5247	117	15	4	134.0	2.55	15
22	2	5247	102	8	3	111.5	2.13	8
23	2	5247	90	2	2	93.0	1.77	2
24	2	5247	86	0	1	86.5	1.65	0

bit width of the pipelined CPA is limited to only two. For a full CSA implementation, two CSAs are required for each SA which lead to an area overhead of 100%. With an area overhead of less than less than 4%, the throughput of the SA block can be doubled and with less than 9% area overhead, the throughput can be quadrupled. Comparing with the possible savings from using offset representation and optimization of the SA block, the cost for pipelining is small and could be further reduced by combining the pipelining and optimization methods.

In Table 6.7 shows the synthesis results of the SA block of the same Filter 1 of [28] obtained by Design Compiler. 's_{CPA}' refers to the maximum allowable bit width of pipelined CPA. The last row 'TC' show the results of a non-pipelined two's complement implementation. Three sets of synthesis results were generated. The first set 'Min. Area' was obtained by synthesizing the SA block with pure area optimization. The columns 'Area' and 'Delay' show the area in μm^2 and critical path delay in ns, respectively of the SA block, whereas the column 'Inc.' shows the percentage area increment of the SA block due to pipelining. The synthesis results matches well with the area overhead predicted by the theoretical results. Due to the logic optimization performed by Design Compiler, the area overhead does not increase monotonically with shorter

TABLE 6.7: Synthesis results of area and delay for pipelining Filter 1 of [28] with different number of maximum allowable bit widths of pipelined CPAs

s_{CPA}	Min. Area			Min. Delay		Delay goal 1ns	
	Area $[\mu m^2]$	**Inc. [%]**	**Delay** $[ns]$	**Area** $[\mu m^2]$	**Delay** $[ns]$	**Area** $[\mu m^2]$	**Delay** $[ns]$
2	491056	33.66	1.22	706697	0.84	507343	1.00
3	448645	22.12	1.43	648691	0.86	465506	1.00
4	425586	15.84	1.78	640342	0.84	492716	1.00
5	406812	10.73	1.91	626507	0.88	485262	1.00
6	405422	10.35	2.33	640392	0.91	509322	1.00
7	395605	7.68	2.47	622097	0.96	513486	1.00
8	394770	7.46	2.88	601832	0.96	531505	1.00
9	388114	5.64	3.05	628031	0.97	553144	1.00
10	387210	5.40	3.35	597678	1.02	557621	1.01
11	383793	4.47	3.59	622303	1.02	580999	1.03
12	387532	5.49	3.99	622010	1.03	583953	1.06
13	380876	3.67	4.20	625786	1.04	577197	1.10
14	377756	2.82	4.17	642181	1.04	599354	1.06
15	378381	2.99	4.65	647610	1.05	582792	1.10
16	378415	3.00	4.80	620234	1.08	593862	1.10
17	379389	3.27	5.16	627419	1.11	602847	1.11
18	378561	3.04	5.39	612194	1.12	594315	1.12
19	378288	2.97	5.67	621727	1.14	594694	1.14
20	377270	2.69	5.93	633673	1.13	604567	1.14
21	376073	2.37	6.19	613671	1.17	616372	1.15
22	374609	1.97	6.47	601875	1.18	597877	1.18
23	372653	1.44	6.70	637528	1.18	619662	1.16
24	374213	1.86	7.04	642730	1.18	625629	1.17
TC	367381	-	6.94	620314	1.20	606629	1.20

pipelined CPA. The same is true for the delay. The second set of results are obtained by synthesizing the SA block under the goal of delay minimization. Due to the aggressive logic optimization of Design Compiler, the same trend is observed for the 'Delay' but it is gentle than that obtained by the area-optimized synthesis results. The 'Area' results are inconsistent due to the optimization heuristic of the Design Compiler. The last set of 'Area' and 'Delay' results under 'Delay goal 1 ns' refer to the area and delay obtained by synthesizing the SA block with a target delay of 1 ns. The first eight pipelined filters with the shorter pipelining bit width meet the timing closure and also have smaller area than the rest. For the other pipelining bit widths, their areas and delays are comparable with those obtained with the minimum delay goal synthesis. Interestingly, the 3-bit pipelining has the smallest area in this set of synthesis. The reason for this is that the overhead due to pipelining increases when the bit width of the pipelined CPA reduces from three to two bits and the increase is more than the amount of logic that can be simplified by Design Compiler with the increase in timing slack. In conclusion,

the optimal size of pipelined CPA depends on the timing constraint. With the short VHDL code generation and verification time of less than three minutes, and the short synthesis time of less than five minutes per filter, it is possible to synthesize of several different bit widths of pipelining for a target timing constraint.

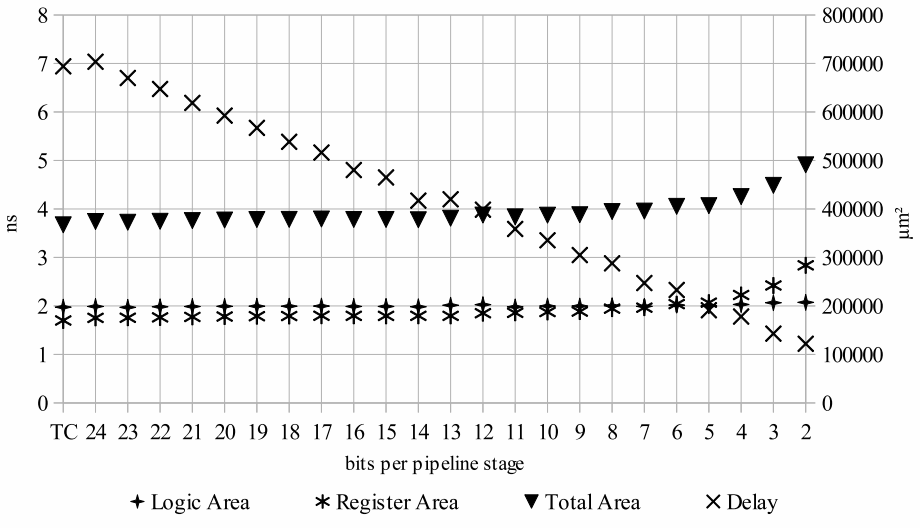


Figure 6.17: Plot of the Area-Minimization Synthesis Results for Different Pipelining Bit Width

Figures 6.17, 6.18 and 6.19 show the plot of the components' and total areas. 'Logic Area' refers to the area of the adders, 'Register Area' refers to the area of the flip-flops and 'Total Area' refers to the total area. The scale on the right vertical axis is the area in μm^2. The scale on the left vertical axis is the delay in ns. The scale on the horizontal axis is the column 's_{CPA}' of Table 6.7. Comparing the 'Register Area' among the three figures for the 1 ns delay constrained synthesis, the total area overhead of shorter pipelining is reduced by the possibility of using slower and smaller flip-flops.

The proposed method for pipelining the SA block shows that with small area increments, the delay can be reduced easily. For area minimization synthesis option, the delay is reduced by more than 80% with about 1/3 area increment. For delay minimization synthesis option, pipelining can reduce the delay by 30% while the area is increased by less than 4% with ideal pipelining bit width.

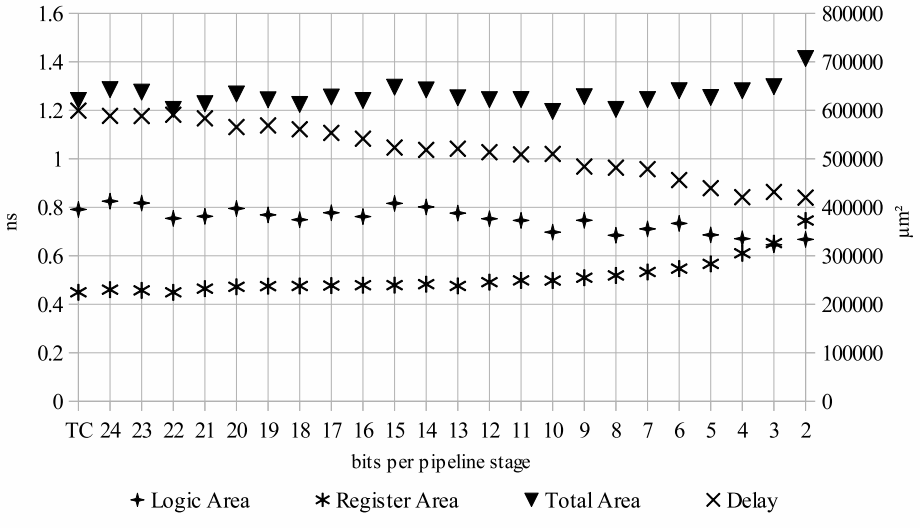


Figure 6.18: Plot of the Delay-Minimization Synthesis Results for Different Pipelining Bit Width

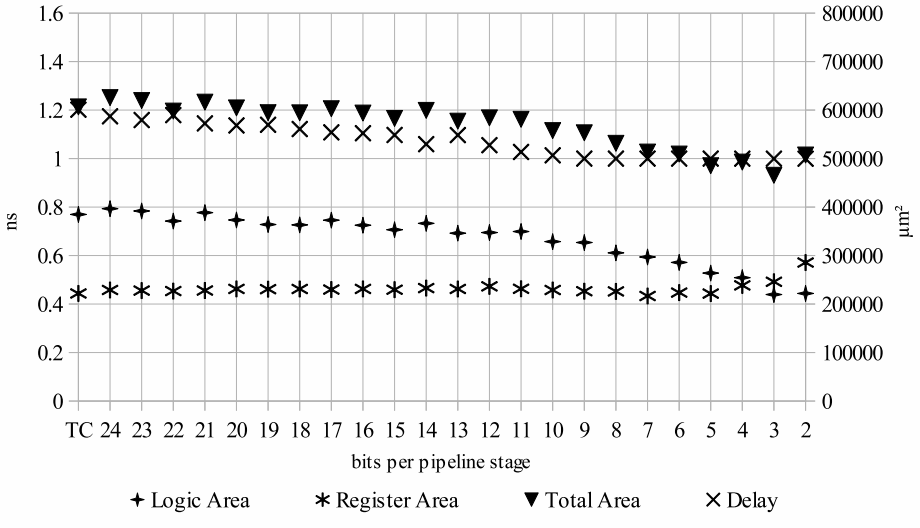


Figure 6.19: Plot of the Synthesis Results for Different Pipelining Bit Width with a Delay Constraint of Synthesis Set to 1 ns

6.4 Output Bit Width Truncation in Structural Adder

In constant coefficient FIR filters, the less expensive fixed point representation is used to avoid the larger area, delay and power consumption of floating point arithmetic. The numerical range of the fixed point signal can be chosen to meet the required accuracy by quantization of real coefficients or filter design algorithms like [28]. The constants are, for simplicity, treated as integers in MCM optimization as the exact position of the decimal point has no impact on the logic cost as it only causes all coefficients to be shifted consistently by a fixed amount from its least significant position. Most MCM optimization are carried out at a high level of abstraction without involving the bit width of the input signal (e.g., Chapter 4). Some MCM optimization algorithms (e.g., Chapter 5) that operate at the bit-level may require the bit width of the input signal but the algorithm do not depend on the position of the decimal point.

By defining the range of input x as $[-1, 1)$, the decimal point is positioned directly after the sign bit or the MSB of its two's complement representation. In general, a passband gain of approximately 0 dB is desired for a passive FIR filter. For such filters, the magnitudes (or absolute values) of the constant coefficients are less than or equal to 1. Therefore the decimal point for the coefficients is usually fixed at the position just before the MSB of the two's complement representation of the largest coefficient or immediately after the MSB if the largest coefficient is one. As the multiplier and multiplicand are both smaller than or equal to one, the constant multiplications of the MCM may produce results with additional bits appear after the LSB of the input. Due to the addition of many statistical independent signals in the SA block, a few additional higher order bits than the MSB of the input can also be generated. These excess bits above the MSB and below the LSB of the input may have no significance to the final output precision but only temporary extend the precision of some intermediate computations within the filter. The simplest way to remove the excess bits of higher weight than the input MSB is through an overflow limiter. This limiting circuit introduces errors. The error magnitude could be significant but they lie in the tails of the bell shaped probability distribution curve of the output values with very low probability of occurrence (See Section 6.1). To reduce such error, the limiter circuit should only be placed at the last structural adder. The potential gain of a limiter circuit is very limited. First it can only save one or two bits at the MSB side. Second, if it is placed at a SA before the final adder, the gain is marginal as the additional bit will be reintroduced at the next SA and the error accumulates. It is more advantageous to limit the bits of weight lower than LSB of the input. These bits can be removed by either rounding or truncation. By truncation, the bits below the desired precision are simply ignored, which leads to a biased or asymmetric error. If rounding towards zero or rounding to the nearest even or odd integer is used, the error introduced is symmetrical but these types of rounding require additional circuits. The aforementioned rounding methods introduce the smallest possible error. If the rounding takes place at the output of the SA block, the error is at most half a LSB of the output of the rounding circuit. This is true under the assumption that the truncated bits have equal probability to be 0 or

1 and are from independent samples. The former is strongly suggested by Section 6.1.1 while the latter is generally assumed for digital filters.

In [84], Hartley proposes a truncation scheme within the MCM block, which is very similar in concept to the design of truncated or fixed width multipliers. In [169], [170], a pattern modification technique is used to analyze the input pairs of the adders and subtractors in the MCM adder graph so as to reduce the signal bit width appropriately. However, as the input is generally a series of independent samples, the errors introduced before the structural adders will be accumulated by the structural adders in the tap-delay chain. The resulting error magnitude at the output will have a bell shaped probability distribution based on the central limit theorem, and the error will exceed half LSB.

In what follows, a method to gradually reduce the fractional bits of the output back to the number of fractional bits of the input is presented. By allowing the error to exceed half a LSB but not more than a full LSB, the implementation cost can be reduced for the SA block and some adders within the MCM block can even be saved.

6.4.1 Low Error Bit Width Tapering

In the proposed method, the constant multiplications within the MCM block are executed at full precision and the filter input is assumed to have uniformly distributed values of a series of independent samples, which leads to equal probability of 0 or 1 for the LSBs according to Section 6.1.1. Truncation is only introduced in the structural adder block. The idea is to gradually step up the reduction of bit width towards the last structural adder, such that the number of fractional bits of the output of the last structural adder is exactly the same as the number of fractional bits of the input. By applying truncation T_i to the output r of the i-th structural adder a_i to obtain $r_T = \lfloor r \rfloor$, and ceiling functions C_i to obtain $r_C = \lceil r \rceil$ at the appropriate positions, the mean error is close to 0. The hardware overhead of this bit width reduction is only an AND gate. The idea behind applying truncation and ceiling functions is that these functions exhibit opposite mean errors. Therefore, a solution where a ceiling function of the same bit width exists for each truncation is sought. To achieve this, the bit width of the structural adder is reduced at each tap where the output from the MCM block is reduced. Otherwise the mean error would increase. Under this constraint, it is not optimal to reduce the bit width by more than one bit at one time or the reduction of bit width at earlier taps will become impossible.

Figure 6.20 shows the last five structural adders where the LSBs of the adder outputs are to be removed progressively. For simplicity, the LSB of the full precision output is considered to have the weight of 1 in the following discussion. The ceiling function C_4 removes the LSB of the output of the structural adder a_4 and introduces an error of 0 or 1 with an equal probability. One FF is saved in the delay element that registers the truncated output. The input to a_3 from the MCM block is truncated by T_3 such that the weight of its LSB is the same as the other input of a_3 after C_4. This truncation by T_3 introduces an error of 0 or -1 with an equal probability. As both signals are obtained from the input to the filter sampled at different time, all possible

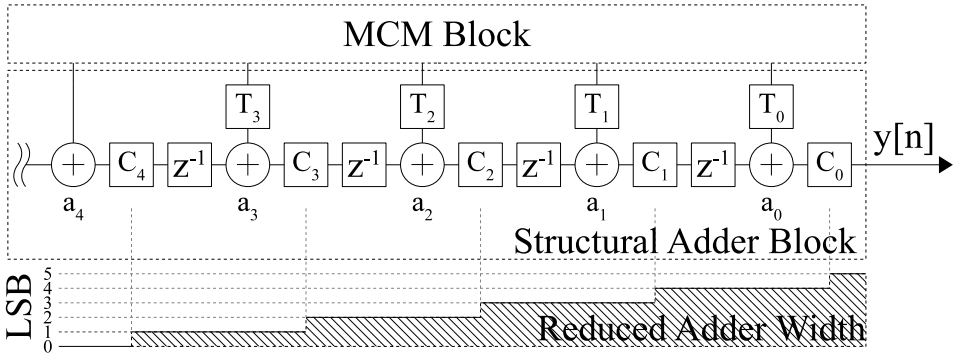


Figure 6.20: Example Showing the Lower Bits of a Reduced Bit Width Structural Adder

combinations of errors have to be considered. This results in an error vector of $[-1\ 0\ 1]$ with their corresponding probabilities given by the vector $[1/4\ 1/2\ 1/4]$. C_3 removes another bit from the LSB of the output of a_3, which introduces an equally probable error of 0 or 2. Accounting for all possible combinations of the addition of two error results in an error vector of $[-1\ 0\ 1\ 2\ 3]$ with its associated probability vector of $[1/8\ 1/4\ 1/4\ 1/4\ 1/8]$. Similarly, T_2 truncates the output from the MCM block before it is fed to a_2. The LSB of the truncated signal will have the same weight as the other input of a_2 from C_3, which results in possible error values of $[-3\ -2\ -1\ 0]$ each with a probability of 25%. This process is continued until after the last structural adder a_0, where C_0 reduces its output to the same number of fractional bits as that of the input. The resulting error vector for this example is $[-26\ -25\ \ldots\ 30\ 31]$. The weight of the LSB of the truncated output is 32 relative to the weight of the LSB of the full precision output and equal to the weight of the LSB of the input. Therefore the output can deviate from the symmetric arithmetic rounded[1] output by no more than ± 1 LSB. Unfortunately the range is not fully symmetric around 0 which introduces a small error bias, but this bias is less than $1/100$ of LSB for 9 or more steps and less than $1/1000$ of the LSB for 13 or more steps. Figure 6.21 shows the error probability if 15 reduction steps are applied. The curve resembles a normal distribution except that the center is flattened. This is because the error introduced by the ceiling function on the LSB has only two distinct error values $[0\ 2^m]$, where m is the number of of SA output truncation steps before the ceiling function. The dotted line in Figure 6.21 shows the error probability obtained by mathematical rounding to the same precision.

No logic is required to truncate the signals from the MCM block. Otherwise, there will be a cost penalty for symmetric FIR filters as the full precision MCM output cannot be reused for the symmetric coefficient multipliers. The partial-sum signal between the structural adders is reduced by one bit at each structural adder. Because of this, the ceiling function is like a round-half-up-asymmetric function, where the bit to be truncated (or the MSB among the bits to be truncated) is added to the next higher order bit. The asymmetric rounding is due to the different rounding of 0.5 and -0.5. The former is rounded to 1 while the latter is rounded to 0 instead of -1. For our method, this is actually an advantage as there is only one error value to be considered.

[1] also known as Round-Half-Up-Symmetric or Conventional Rounding

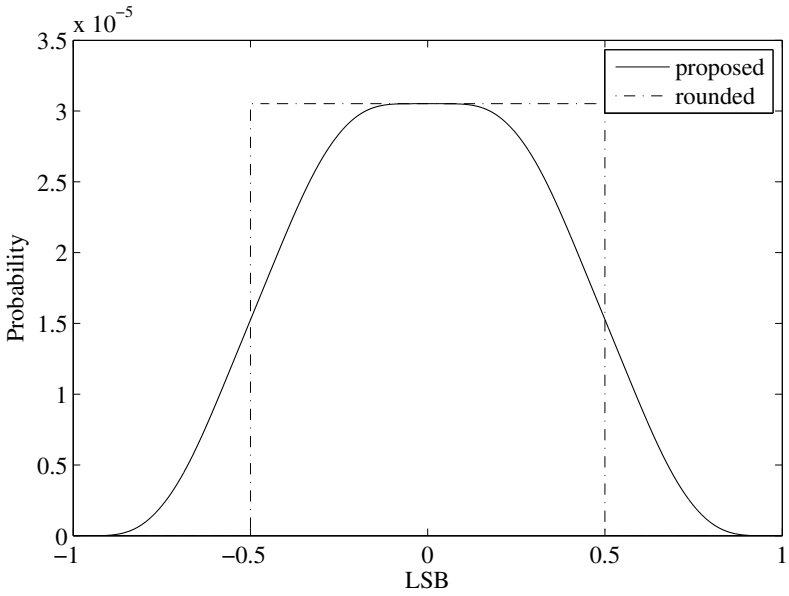


Figure 6.21: Error Probability Predicted for the Proposed Method and for Rounding After 15 SA Truncation Steps in Terms of the Weight of LSB of Input Signal

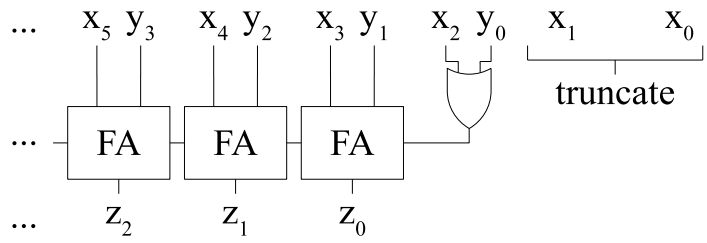


Figure 6.22: Example of the Bit Width Reduction of Structural Adder

In term of implementation, the T_i function is simply a matter of signal rewiring while the C_i function is obtained by simplifying the structural adder a_i to produce a shorter rounded sum. To realize a round-half-up function, the sum output of the last FA of a_i has to be added to the carry input of the FA to its left. The rightmost FA does not have any carry input, i.e., $c_0 = 0$. If the output x from the MCM block is to be added to the delayed output y from the previous SA, then either the sum or the carry signal but not both are high at the same time for the LSB bit adder. This can be used to simplify the Boolean equation for c_1 to $x_p \vee y_0$, where $\vee$ is a logical OR function, x and y are the input operands to the adder, and the index p of x is equal to the number of truncated bits of x. If the signal x from the MCM block is to be subtracted from instead of added to y by the structural adder, there is a probability that both sum and carry bits are 1 at the same time. As more bits are truncated, this probability is lowered. Therefore, no special handling is needed. The reduced bit width structural adder is shown in Figure 6.22 where two LSBs of the signal x from the MCM block are truncated, $p = 2$, and y is the delayed output from the previous SA. The output z of the adder is a

shorter rounded sum of x and y.

6.4.2 Implementation Results

The effect of the proposed method is demonstrated on two example filters, A and B, taken from [21] (Filter 1 and 10). Three different implementations of these two filters are compared: (1) 'CSD': the baseline CSD implementation without optimization (2) [58]: optimization of the MCM block by the MinLD algorithm [58] and (3) 'Prop.': application of proposed bit width reduction on the SA block after the MCM block has been optimized by the MinLD algorithm. The specifications of the two example filters are given in Table 6.8. The passband ripple and the stopband attenuation are obtained from the transfer function. The filters were originally designed with the *remez* algorithm with equal weights for both bands. As the Signal-to-Quantization-Noise Ratio (SQNR) is increased by ≈ 6 dB for each additional bit, an input bit width 'In' of 8 bits (1 sign bit, 7 data bits) provides a possible SQNR of ≈ 42 dB, which is sufficient to ensure that the SQNR does not interfere with the attenuation specification of the transfer function. Table 6.8 also shows the maximum bit width 'c_n' of the filter coefficients and the output bit width 'Out' for the full precision output as well as the reduced output with the same number of fractional bits as the input.

Table 6.9 shows the estimated hardware reduction and the synthesis results obtained by Synopsys Design Compiler (Ver. 2007.12) using TSMC $0.18\mu m$ standard cell library. The column 'Delay' specifies the three timing constraints used for the optimization, which are with no timing constraint (∞) and with fixed delay constraints of 3 ns and 2.8 ns, respectively.

The estimated reduction in terms of 1-bit adders, 'FA' and flip-flop 'FF' for each filter is shown in the second column of Table 6.9. The reduction in FF can be calculated by the sum of an arithmetic series

$$\sum_{k=1}^{M} k = \frac{1}{2}M(M+1) \tag{6.24}$$

where M is the bit width reduction of the output. For Filter A, the output is reduced by 16 bits and

$$\sum_{k=1}^{16} k = \frac{1}{2}16(16+1) = 136 \tag{6.25}$$

As for the number of 1-bit adders (FA) saved, the same equation can be applied with the following two considerations:

A) If the output from the MCM block is shifted to the left (e.g., the coefficient is even), there are trailing zeros. Those 1-bit adders have already been reduced in the baseline implementation.

B) If the coefficient is positive, the last FA can be reduced to a HA. For negative coefficient, this last FA can be a simplified FA with one constant input '1'.

For simplicity, the second factor is neglected and all 1-bit adders are counted as FAs. Based on data provided in [166], each FA or FF occupies approximately $70\mu m^2$. This is used to estimate

TABLE 6.8: Specifications of Test Filters 1 (A) and 10 (B) from [21]

Filter	ω_P [π]	ω_S [π]	**Tap** #	**Word Length** c_n	**In**	**Out**	A_p [dB]	A_s [dB]
A	0.10	0.15	40	14	8	25 / 9	≈ 1.1	> 23
B	0.20	0.25	60	16	8	27 / 10	≈ 0.4	> 32

TABLE 6.9: Experimental Results for Output Bit Width Truncation in Structural Adder

Filter	**Estimated Reduction**	**Delay** [ns]	**Total Filter Area [μm^2]**			**Area Reduction**			**Compilation Time [s]**		
			CSD	**[58]**	**Prop.**	**[58]/CSD**	**Prop./CSD**	**Prop./[58]**	**CSD**	**[10]**	**Prop.**
A	126 FAs 136 FFs 18340μm^2	∞	144 961	143 887	126 011	0.74%	13.07%	12.42%	38	36	33
		3	191 750	188 920	166 523	1.48%	13.16%	11.86%	161	143	125
		2.8	204 820	198 406	174 929	3.13%	14.59%	11.83%	235	185	159
B	133 FAs 153 FFs 20020μm^2	∞	224 745	218 957	202 375	2.58%	9.95%	7.57%	61	55	52
		3	302 596	293 222	272 169	3.10%	10.06%	7.18%	334	289	275
		2.8	325 445	311 494	286 712	4.29%	11.90%	7.96%	471	375	367

the area reduction in the second column of Table 6.9.

From '[58]/CSD' of Table 6.9, the benefit of algorithmic optimization of the MCM block is not as conspicuous as expected after the gate-level optimization and technology mapping made by Synopsys Design Compiler. The reduction achieved by the proposed bit width reduction of structural adders after synthesis is close to the estimate. For Filter A, the reduction is below the estimate if the synthesis is performed with unconstrained delay ($17876\mu m^2$) and above it with delay constraints ($22397\mu m^2$ and $23478\mu m^2$). The same is true for Filter B with area reductions of $16582\mu m^2$, $21053\mu m^2$ and $24782\mu m^2$ for 'Delay' $= \infty$, 3ns and 2.8 ns, respectively. It should be noted that the area of the MCM block with no delay constraint is $18335\mu m^2$ and $23471\mu m^2$ for Filter A and Filter B, respectively. The areas reduced by the proposed bit width reduction method are 97.4% and 70.6% of the respective MCM blocks of these filters. As the critical path passes through the MCM block, the area of the MCM block grows tremendously when delay constraint is applied, causing these area savings to become approximately 60% and 40% of the MCM blocks of Filter A and B, respectively.

Besides the area reduction, the time required by Synopsys Design Compiler to synthesize the filters is also reduced partially due to the optimization by MinLD and the proposed method. The percentage reduction in compilation time of the proposed method ranges from 8% to 52%. The compilation time was reported by Design Compiler (Ver. 2007.12) on Red Hat Enterprise Linux Server (Release 5.4) running on two Xeon 5460 @ 3.16GHz with 40GB RAM.

The area reduction and the error (relative to the weight of the LSB of the input) obtained by the proposed method are independent of the length of the filter and the input bit width. Therefore the absolute area savings will not change if the input bit width or the filter length changes. The only factor that has a direct impact on these results is the number of fractional bits in the coefficients. The more the number of fractional bits, the bigger the savings. Therefore, the savings are more significant for short filters with high coefficient bit width.

TABLE 6.10: Output Accuracy Normalized to the Weight of the LSB of Input

Filter	**Mean**		**Variance**		**#errors**		
	Opt.-FP	**Opt.-Rnd**	**Opt.-FP**	**Opt.-Rnd**	**−1**	**0**	**1**
A	0.00404	0.00179	0.11	0.14	710	8602	728
B	0.00042	0.00219	0.11	0.14	689	8660	711

Table 6.10 shows the statistics obtained by comparing the output of the optimized filter with (a) the full precision (FP) output and (b) the output rounded mathematically (Rnd) to the same precision. Both filters were supplied with 10000 uniformly distributed input samples followed by N zeros, where N is the filter order. The maximum output error introduced by the proposed method is ± 1 and the frequencies of occurrences of zero and ± 1 errors are shown in the column '#errors'. About 14% of the outputs exhibit errors. The bias introduced by the proposed rounding method is, as predicted, very small and the variance is within reasonable bounds.

Figure 6.23(a) and (b) show the histograms of the rounding errors obtained by symmetric

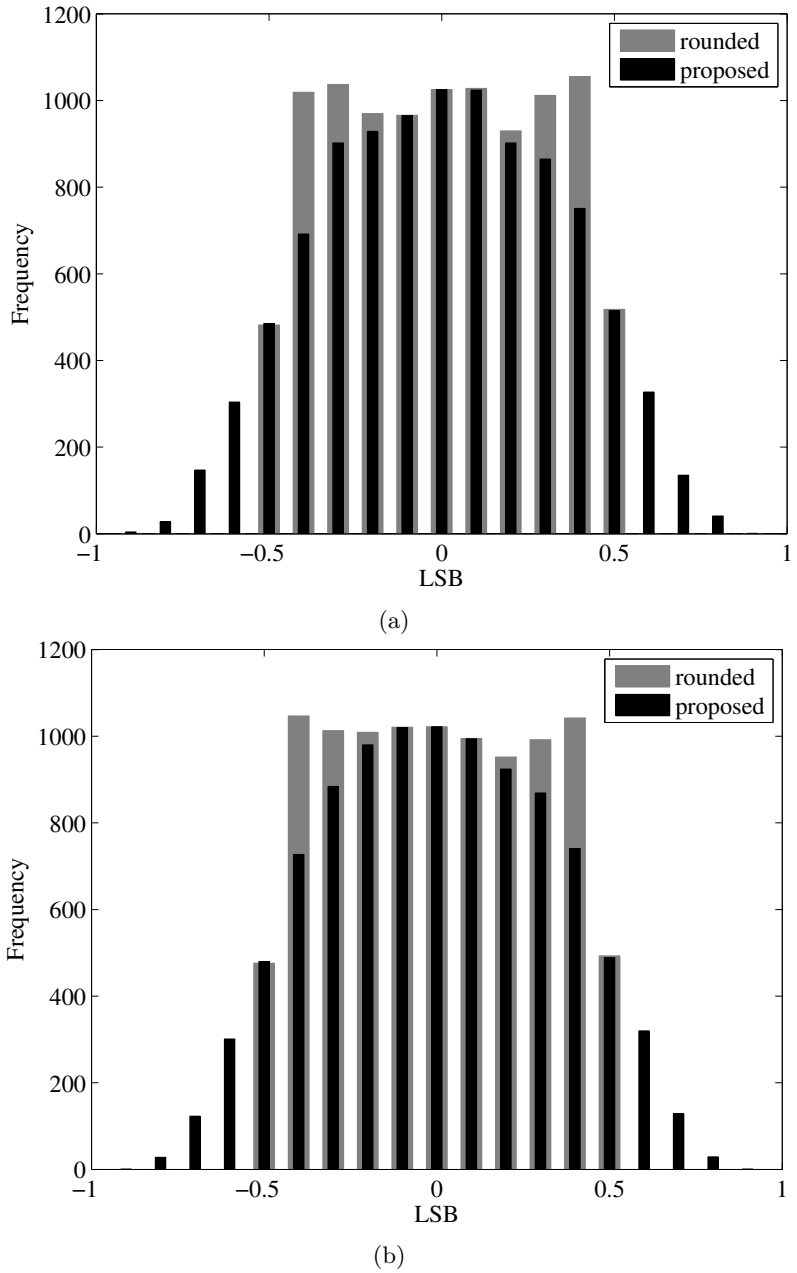


Figure 6.23: Histograms of the errors in terms of LSB introduced by the proposed methods and rounding for $10000 + N$ random input samples: (a) Filter A and (b) Filter B

arithmetic rounding and the proposed method measured in terms of the LSBs of input signal for Filter A and Filter B, respectively. The tails of the histogram of the proposed method beyond ± 0.5 LSB result in output errors of ± 1. The errors from the simulation results agree with the prediction in Figure 6.21.

The difference between the outputs of the proposed optimization method and the rounded results is limited to one LSB. Therefore, the difference in SQNR_{dB} is expected to be below 6 dB as a precision of one digit contributes 6 dB of SQNR. The impact of the quantization on

the SQNR is calculated as follows:

$$\mathrm{SQNR}_{\mathrm{dB_{Diff}}} = 10\log_{10}\left(\frac{E[x^2]}{E[(x-x_r)^2]}\right) - 10\log_{10}\left(\frac{E[x^2]}{E[(x-x_q)^2]}\right) = 10\log 10\left(\frac{E[(x-x_q)^2]}{E[(x-x_r)^2]}\right) \tag{6.26}$$

where x is the full precision signal, x_r is the rounded output and x_q is the output of the proposed method.

Using the filter results in Table 6.10 the SQNRs for the rounded and optimized outputs and their difference were calculated. The latter is calculated based on Equation (6.26), and the results are shown in Table 6.11. The SQNR reduction of less than 1.3 dB is considered acceptable for the achieved area reduction.

TABLE 6.11: SQNR in dB for rounded and optimized results

Filter	**SQNR [dB]**		
	Rounded	**Optimized**	**Difference**
A	38.86	37.58	1.28
B	42.77	41.56	1.20

6.5 Chapter Summary

The properties of the structural adder block were analyzed by its adder bit and value distributions. Based on the analysis, it was concluded that the central limit theorem is valid and the value distribution of the structure adder can be estimated by normal distribution with the presented properties. Unfortunately there is limited potential to optimize the structural adders by cutting the tails of the normal distribution.

The possibilities for area reduction of the SA block were discussed. Based on the tradeoff between combinational logic and registers, a holistic comparison is made between the proposed SA optimization methods for two's complement representation and offset representation. For two's complement representation, the proposed method reduces the sign extension by bisecting the saved sum at the cost of a few additional registers. The tradeoff reduces the area by up to 13.68% as observed from the synthesis results of a set of benchmark filters. The area reduction is nearly constant for different input bit widths and increases with the length of the filters. For offset representation, the method reduces the carry propagation in the higher order bit position of the adder by saving and forwarding the carry to the next adder. The trading of HA cost for registers is beneficial even if the difference between the bit widths of the two adder inputs is small. Synthesis results of a set of benchmark filters showed an area reduction of up to 6.23% for ASIC implementation and up to 8.52% for FPGA implementation. When the optimization of the SA block is combined with the MCM optimization, up to 14.48% of area reduction can be obtained without changing the output value of the filter. MCM optimization contributes only a small fraction of this area reduction, up to 3.12% of the overall filter area. This is due to two factors. First, the adders in the SA block are longer than those in the MCM block which leads a maximum area ratio of SA block to MCM block of 9:1. Secondly, the synthesis tools like Design Compiler and Xilinx ISE are better at optimizing the fully combinational logic circuit of the MCM block but are less efficient in optimizing the adder logic between registers in the SA block than the proposed SA optimization algorithms.

Pipelining of SA block is also proposed to increase the throughput of very high-speed filter. By saving the intermediate carry signal, the proposed pipelining technique for SA block can be achieved with a much smaller area overhead than expected. For an example filter with 121 taps, it is shown that the most expensive pipelining obtained by dividing the SAs into blocks of only two RCAs results in an area overhead of less than 40%. Delay minimization synthesis showed that a delay reduction of 30% with an area increment of less than 4% can be achieved.

Finally, a new method using truncation and round-half-up was proposed to gradually reduce the fractional part of the signal in the structural adder block to the same number of factional bits as the input signal. The error introduced by the proposed method was analyzed and confirmed by simulation results. The results showed that the proposed method introduced a small mean error of less than 0.25% of the LSB and the error was confined to only ± 1 LSB. Meantime the area required for circuit implementation was reduced by up to 12.42%. The absolute area reduction and the error are both independent of the filter length and the input bit width, but dependent only on the precision of the filter coefficients.

Chapter 7

Binary-to-CSD and CSD-to-Binary Converter

In many arithmetic problems, the properties of Canonical Signed Digit (CSD) can be exploited to provide advantages over the use of binary representations as highlighted in Section 2.1.6. In some applications, the CSD properties are not only useful as constant representation during the design phase, but also make physical implementation of actual arithmetic operation efficient. This makes it necessary to either store a CSD number, which costs two bits per CSD digit for direct calculation or only a small overhead over the binary bit count plus a converter. To overcome the storage overhead, the CSD representation can be generated on the fly, but this requires a high speed CSD encoder. Unfortunately, the properties of CSD do not allow for a fully parallel encoding due to the carry propagation chain. For calculations that are executed fully in CSD representation, there might be a need to convert the output from CSD back to the binary representation.

This chapter consists of two main parts. The first part introduces a bypass technique for binary to CSD conversion based on the property that no two consecutive digits in CSD can be nonzero. The conversion algorithm can be realized with simple combinational circuit components and is therefore amenable to standard cell based implementation. The proposed converter uses less logic elements than the existing carry propagating converters in ASIC or FPGA. Instead of a carry signal, a bypass signal is output to the next digit slice to bypass the evaluation of the inputs in the next slice if a nonzero digit is output in the current slice. This bypass signal can be generated with minimal delay by only a single NOR gate in each digit slice. The bypass chain can be further sped up by a look-ahead circuit. The design is simple and scalable as n digit-slices can be cascaded to generate an n-digit CSD. The second part proposes a CSD to binary converter. Unfortunately, this conversion is unable to take full advantage of the CSD properties when the output representation contains long series of ones. Nevertheless, a solution with only two NAND gates in the propagation path was proposed, and a simple look ahead recursion method was suggested to further simplify the converter design.

7.1 Binary-to-CSD Converter

7.1.1 CSD Review

CSD representation [41] has been used for digital arithmetic operations to reduce the average and maximum numbers of nonzero digits to $n/3 + 1/9$ and $\lceil \frac{n+1}{2} \rceil$, respectively for an n-bit binary encoded number. In CSD, two adjacent digits cannot be both nonzero, i. e., $x_i \times x_{i-1} = 0$, where x_i is the i-th digit of a CSD number. This helps to reduce the number of additions and subtractions required in constant and variable multiplications. Exponentiation in cryptography can also benefit from the use of CSD representation [171].

The requirement that the nonzero digits must not be adjacent makes CSD representation unique and differentiates it from the non unique MSD representation [172], which also has minimal Hamming weight. A major drawback for the representation of a variable in CSD is that the conversion from binary to CSD can only be generated recursively, whether it is determined from the LSB to the MSB, i.e., right-to-left conversion or from the MSB to the LSB, i.e., left-to-right conversion. Most left-to-right conversion [49], [171] and bidirectional conversion [173] methods generate a MSD encoded output. This is due to the fact that the least significant digit can influence all digits to its left during the conversion to CSD. The most extreme example for this problem is shown in Figure 7.1.

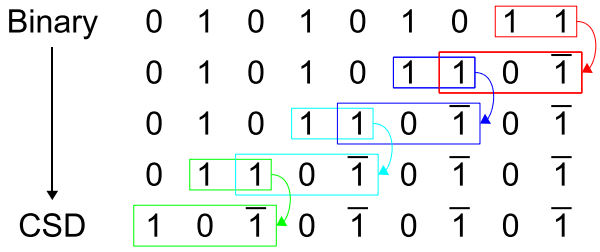


Figure 7.1: Example of the Rippling of Adjacent Nonzero Bits in Binary-to-CSD Conversion

In this example, the two LSB bits are both 1. Replacing them by $[10\bar{1}]$ results in a 1 digit at the third LSB. This newly generated 1 forms a [11] pair with the next higher digit, which has to be replaced again. This continues until the entire bit width has been evaluated and consecutive nonzero digits have been sequentially replaced.

Some conversions to other SD representations can be executed fully in parallel, for example, Booth encoding, but these SD representations do not have the minimum number of nonzero digits and could even have more nonzero digits than binary representation for some numbers.

Conventional approaches [41], [174], [175] to the binary to CSD transformation use an intermediate binary carry signal. These approaches do not take advantage of the non consecutive nonzero digit properties of CSD numbers. In [171], a left-to-right sliding window method is proposed to convert a binary number into MSD. The sliding window scans three consecutive bits at a time and produces one or two output digits. It produces two output digits whenever one of the outputs is 1 or -1. The simplicity of the sliding window algorithm in [171] triggered

a search for a similar hardware implementation and led to the proposed method. The only similarity in both methods is they both use the fact that $x_i \cdot x_{i-1} = 0$.

7.1.2 Processing Element of Binary-to-CSD Converter

In the proposed method, the two's complement number to be converted is partitioned into overlapping three-bit groups with two overlapping bits between two adjacent groups. One bit $b_{-1} = 0$ is appended at the LSB and 0s are prepended to the MSB to complete its three-bit group. Each digit slice of the converter recodes three binary bits, b_{i+1}, b_i and b_{i-1} into a signed digit, x_i which is binary encoded into a sign bit, $x_{i,s}$, and a magnitude bit, $x_{i,m}$. Under the sign-magnitude encoding (equivalent to two's complement) $\{0 \equiv 00, 1 \equiv 01, -1 \equiv 11\}$. The truth table for the Processing Element (PE) of the i-th digit slice is shown in Table 7.1, where p_i and p_{i+1} are the bypass input into and output from the PE, respectively. Without the bypass signals, redundant nonzero digits will be generated when the PEs of all digit-slices execute concurrently. For example, the binary number $[0101011]_2$ will be recoded into $[11\bar{1}1\bar{1}0\bar{1}]$ when the bypass signals $p_i = 0$, while the correct CSD output based on a recursive right-to-left application of the CSD identities to make $x_i \times x_{i-1} = 0$ yields $[10\bar{1}0\bar{1}0\bar{1}]$. In this example, two extraneously generated nonzero digits exist at bit positions $i = \{3, 5\}$ (where $i = 0$ for the LSB). This breach of the non-adjacency criterion of CSD is detected and corrected after one gate delay per digit by cascading the PEs using the bypass signals.

TABLE 7.1: Truth Table for CSD Encoding with Bypass Logic

p_i	b_{i+1}	b_i	b_{i-1}	x_i	$x_{i,s}$	$x_{i,m}$	p_{i+1}
0	0	0	0	0	0	0	0
	0	0	1	1	0	1	1
	0	1	0	1	0	1	1
	0	1	1	0	0	0	0
	1	0	0	0	0	0	0
	1	0	1	-1	1	1	1
	1	1	0	-1	1	1	1
	1	1	1	0	0	0	0
1	d	d	d	0	0	0	0

When $p_i = 0$, the recoded digit x_i is generated from the input bits b_{i+1}, b_i and b_{i-1} according to the first eight rows of Table 7.1, and a bypass signal is generated for the next PE, $p_{i+1} = |x_i|$, which can be directly hardwired from $x_{i,m}$. The magnitude bit is dependent only on b_i and b_{i-1}, and $x_{i,m} = 1 \iff b_i = 1$ or $b_{i-1} = 1$. The sign bit is dependent on b_{i+1} and $|x_i|$, and $x_{i,s} = 1 \iff b_{i+1} = |x_i| = 1$. When $p_i = 1$, the outputs x_i and p_{i+1} are forced to zero irrespective of the other inputs as depicted by the don't care inputs, d, in the last row of Table 7.1. The inputs to the PE are said to be bypassed or ignored by the PE, while setting p_{i+1} to zero allows a new bypass signal to be generated at the next PE. The PE circuit that realizes Table 7.1 is very simple, as shown in Figure 7.2. Comparing the proposed PE with the

bit-parallel implementation in Fig. 5 of [49], the simplicity of the circuit is obvious. The bypass signal can be generated or propagated with only a NOR gate delay as opposed to the carry propagation delay of at least two cascaded two-input gates. The same bypass technique is also applicable to the CSD digit encoded in negative-positive binary code $\{0 \equiv 00, 1 \equiv 01, -1 \equiv 10\}$ but additional logic gates will be required. Specifically, the output $x_{i,m}$ is not equal to p_{i+1} but calculated as $x_{i,m} = p_{i+1} \cdot (\overline{b_{i+1}})$ instead.

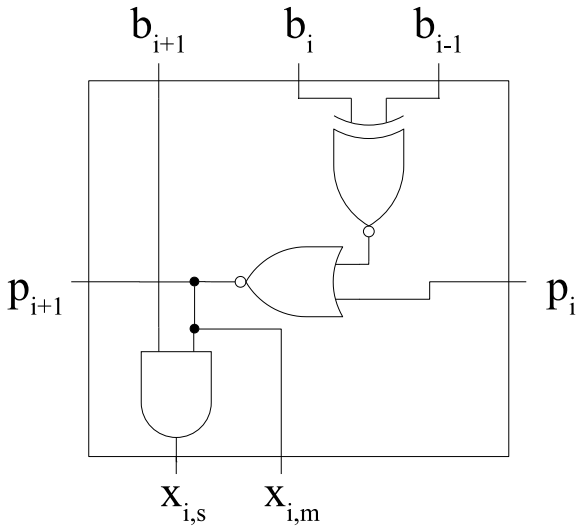


Figure 7.2: Processing Element for one Digit of Binary-to-CSD Conversion

Input paddings are required for the two end PEs at the least and most significant digit positions, i.e., $i = 0$ and $i = n - 1$. The inputs b_{-1} and p_0 to the rightmost PE are set to zero. For the leftmost PE, if the input is signed and represented in two's complement form, the MSB b_{n-1} is sign extended to b_n. If the input is unsigned, then an additional PE will be needed to generate x_n to cover the largest positive number and the inputs b_n and b_{n+1} are set to 0. In both cases, p_{i+1} of the leftmost PE is discarded.

7.1.3 Lookahead Circuit for Bypass Generation

The bypass signal p_{i+1} depends on the output of the XOR, $v_i = b_i \oplus b_{i-1}$, and the bypass signal p_i by the relation $p_{i+1} = v_i \overline{p_i}$. This can be unfolded and rearranged as it is done for the carry in [175] or carry look-ahead adder circuits. The unfolded bypass signal is given by

$$p_i = \sum_{j=0}^{i/2} \overline{v_{2j}} \prod_{k=j}^{i/2} v_{2k+1} \tag{7.1}$$

$$p_i = \prod_{m=0}^{i/2} v_{2m} + \sum_{j=1}^{i/2} \overline{v_{2j-1}} \prod_{k=j}^{i/2} v_{2k} \tag{7.2}$$

where (7.1) and (7.2) are used for the even and odd i, respectively. The Boolean sum of products can be implemented as a balanced tree. Therefore, the bypass signal p_i can be

calculated with the delay of O(log(n)). As b_i is a common input in v_i and v_{i+1}, the following simplification is possible $\overline{v_i}v_{i+1} = \overline{b_{i-1}}\,\overline{b_i}\,b_{i+1} + b_{i-1}\,b_i\,\overline{b_{i+1}}$, but as v_i is required anyway, the simplification requires more area and the speedup is marginal for shorter bit widths.

7.1.4 Results and Comparison

Comparison of the proposed method and a general carry propagation method for different input bit widths is shown in Tables 7.2, 7.3 and 7.4, where the first table shows the results for a Virtex 4 FPGA implementation while the latter two show results for ASIC implementations optimized for area and delay, respectively. The results were generated from VHDL code compiled by Xilinx ISE 10.1 for Virtex 4 (xc4vlx15-12sf363) and Synopsys Design Compiler v2010.03-SP3 with the TSMC 0.18μm standard cell library.

TABLE 7.2: Results of Binary-to-CSD Converters on Xilinx Virtex 4 FPGA, Optimized for Speed

	[41]		Proposed		Reduction	
Bits	**Slices**	**LUT**	**Slices**	**LUT**	**Slices**	**LUT**
PE	2	3	1	2	50.0%	33.3%
8	12	21	12	21	0.0%	0.0%
16	28	49	18	31	35.7%	36.7%
32	58	102	36	63	37.9%	38.2%
64	119	210	73	127	38.7%	39.5%
128	242	427	147	255	39.3%	40.3%

For FPGA implementation, the PE of the proposed method fits into one single slice using only two LUTs, while the general implementation requires three LUTs. For $n = 8$, the results are identical, due to the logic flattening of the Xilinx ISE compiler. For bit widths above 16, logic flattening is not executed and the proposed method outperforms the conventional method by 35% or more.

TABLE 7.3: Area Optimized Synthesis Results Based on CMOS 0.18μm Standard Cell Library

	[41]		Proposed		Reduction	
Bits	[μm^2]	[ns]	[μm^2]	[ns]	**Area**	**Delay**
PE	66	0.25	39	0.25	40.0%	−1.7%
8	449	1.58	279	0.84	37.8%	46.5%
16	981	3.38	598	1.69	39.0%	50.0%
32	2045	6.97	1237	3.38	39.5%	51.5%
64	4174	14.16	2514	6.76	39.8%	52.3%
128	8432	28.54	5069	13.52	39.9%	52.6%
256	16948	57.31	10178	27.04	39.9%	52.8%

For area optimized ASIC implementation, the PE is 40% smaller while being only 1.7%

slower. For different word lengths of the conversion, the area savings reach almost 40% while the delay is reduced by half. The increment in area and delay is, as expected, almost linearly related to word length. The small nonlinearity is due to the possible simplifications at both ends of the conversions because of the constant or identical inputs to the PEs.

TABLE 7.4: Delay Optimized Synthesis Results Based on CMOS 0.18μm Standard Cell Library

Bits	**[41]**		**Proposed**		**Reduction**	
	$[\mu m^2]$	**[ns]**	$[\mu m^2]$	**[ns]**	**Area**	**Delay**
PE	236	0.11	136	0.15	42.3%	−40%
8	945	0.74	609	0.39	35.6%	47.3%
16	2235	1.46	1054	0.84	52.8%	42.9%
32	5212	2.83	1943	1.71	62.7%	39.7%
64	9790	5.64	4201	3.39	57.1%	39.9%
128	18714	11.32	8206	6.78	56.2%	40.1%
256	37725	22.55	16353	13.59	56.7%	39.7%

When optimizing for minimum delay, the carry based PE is faster than the bypass based PE due to the longer delay from the binary inputs to the digit output, but consumes about twice the area. However, for $n = 16$ to 256, the proposed converter has approximately 40% shorter delay than the conventional method due to the shorter delay of the bypass signal and uses less than 50% of its area. For $n = 8$, the savings are higher for delay and lower for area. With an average delay of approximately 0.053 ns per digit, the conversion is very fast. This opens up new opportunities to use CSD instead of Booth encoding to reduce the switching activity due to fewer nonzero digits in multiplication.

7.1.5 Credit to Reitweisner

Due to the difficulty to get hold of the old works from Reitweisner, the work in [39] was only reviewed after the publication of [50]. A careful review of [39] actually revealed that in [39, Section 8.8 - Recapitulation, Page 255] a group of equations are given that describe the equations for Figure 7.2. Following the equations 8.8.3β - 8.8.3ξ from [39] transferred to the notation used here, where y_i is an intermediate signal, that is not visible in the proposed method.

$$p_{-1} = b_{-1} = 0 \tag{7.3}$$

$$y_i = b_i \oplus b_{i-1} \tag{7.4}$$

$$p_{i+1} = \overline{p_i} y_i \tag{7.5}$$

$$x_i = (1 - 2 \cdot b_{i+1}) p_{i+1} \tag{7.6}$$

From these equations, it can be seen that the recursion proposed by Reitweisner in 1960 also contains only an inversion and an AND gate. On hindsight I would like to acknowledge the

pioneering contribution of Reitweisner [39] for the technique with which we optimize for the VLSI implementation.

7.1.6 Post Publication Discussion

Most of the above presentation had been published in [50]. In [176], our proposed method was cited but unfortunately there is no comparison against it. The idea in [176] is similar as the carry is replaced by two carries, which can be processed with only a NAND or a NOR gate in the ripple carry path. Implementation wise, the method presented above contains only a single type of PE while [176] requires two types, one PE for even digits and one PE for odd digits. Furthermore, the two methods use different encodings for the CSD digits. As described above, the presented method uses two's complement representation while [176] uses negative-positive binary code. To produce two's complement representation from the latter, an OR gate has to be inserted into the magnitude output to OR with the sign output. Therefore, irrespective of the number representation used, additional overhead is needed in one method to produce the same output format as the other, which makes the comparison based on any single encoding format biased. Taking each method by itself and using the standard cell measurement from the TSMC $0.18\mu m$ library, the PE of the presented method needs $49.9\mu m$ and the one from [176] needs on average $46.6\mu m$ or 6.66% less area. To match the output format of one method to the other, the overhead is approx. $13.3\mu m$ for either method. In other words, either method requires more area than the other when the output is not expressed in its native format. Therefore it can be said that for ripple carry implementation, each method has been designed to best suit its own signed digit encoding format.

7.2 CSD-to-Binary Converter

7.2.1 Origin and Application

Originally the intention was to build a fast adder to add two CSD numbers and produce an output that is CSD encoded. To add two CSD numbers and have a SD output is rather straightforward, but to have the SD output comply to the CSD properties is complicated. This is because the carries can propagate from the LSB to the MSB while the borrows can propagate from the MSB to the LSB. One approach was to convert the two numbers back to binary, add them and convert them back to CSD. Unfortunately, there is no strong theoretical underpinning and good techniques that could simplify this process to an extent that is acceptably efficient.

In [177], the authors propose to use CSD encoding for data communications on the buses to reduce crosstalk and other effects between adjacent wires. If a signal on a bus is encoded in CSD, it needs to be converted back for binary processing, for which a CSD-to-Binary converter is required.

7.2.2 Processing Element of Binary-to-CSD Converter

When developing design algorithms for MCM problems, one often needs to verify the correctness of the CSD encoded numbers. On the high level program platform such as MatLab, this can be done rather easily as the decimal value can be generated by summing every digits of the CSD vector multiplied by their respective positional weights. In binary arithmetic, this is more complicated. The CSD number is split into two binary vectors. One vector contains all positive nonzero digits while the other vector contains a one for each negative nonzero digits. Then the vector containing the ones in the negative nonzero digits vector is inverted and a one is added at the position of the nearest negative nonzero digit to the LSB. These two vectors are then added with a regular RCA to produce the number in two's complement binary representation. This conversion requires a circuit to detect the negative nonzero digit closest to the LSB, an inverter and a full adder for each digit.

Since not all input combinations are legitimate for the two inputs to the RCA based on the properties and encoding of the CSD number in the abovemented conversion, there is possibility to optimize the converter design. Again using the sign-magnitude encoding for CSD, a simple PE is proposed. It is not possible to use a bypass signal, as every digits of the output will be affected, but there is a direct relation between the CSD digits and the output when a carry signal is used. The truth table for the PE is shown in Table 7.5, where x_i consists of $x_{i,s}$ and $x_{i,m}$ for the representation of the sign and magnitude, respectively of the i-th digits of CSD encoded number x. For consistency p_i is used as the i-th digit of the borrow signal p.

The truth table in Table 7.5 does not show all possible input combinations, as it is not allowed to have $x_{i,s} = 1$ when $x_{i,m} = 0$. These two rows can be used to simplify the equations to form the two outputs. From the truth table, it can be seen that the output b_i is equal to the borrow input p_i if $x_{i,m} = 0$ and the inverse of the borrow input if $x_{i,m} = 1$. This leads to

TABLE 7.5: Truth Table for Binary Encoding a CSD Number

x_i	$x_{i,m}$	$x_{i,s}$	p_i	b_i	p_{i+1}
0	0	0	0	0	0
0	0	0	1	1	1
1	0	1	0	1	0
1	0	1	1	0	0
−1	1	1	0	1	1
−1	1	1	1	0	1

the following equation.

$$b_i = x_{i,m} \oplus p_i \tag{7.7}$$

The borrow signal can be easily generated if $x_i = -1$ and propagated if $x_i = 0$. This can be expressed as:

$$p_{i+1} = x_{i,s} + (\overline{x_{i,m}} p_i) \tag{7.8}$$

From Equation (7.8), it can be seen that at least two logic gates are in the propagation path, which means the conversion from CSD back to binary is at best half as fast as the forward conversion. Still the proposed PE is much faster than the generic method described earlier. Figure 7.3 shows two possible implementations for the processing element. Figure 7.3(a) is optimized for area and Figure 7.3(b) is optimized for propagation delay. In the minimization for area, the signal $\overline{x_{i,m}} p_i$ is part of the XOR function of Equation (7.7) and can be reused in Equation (7.8) to further reduce the area.

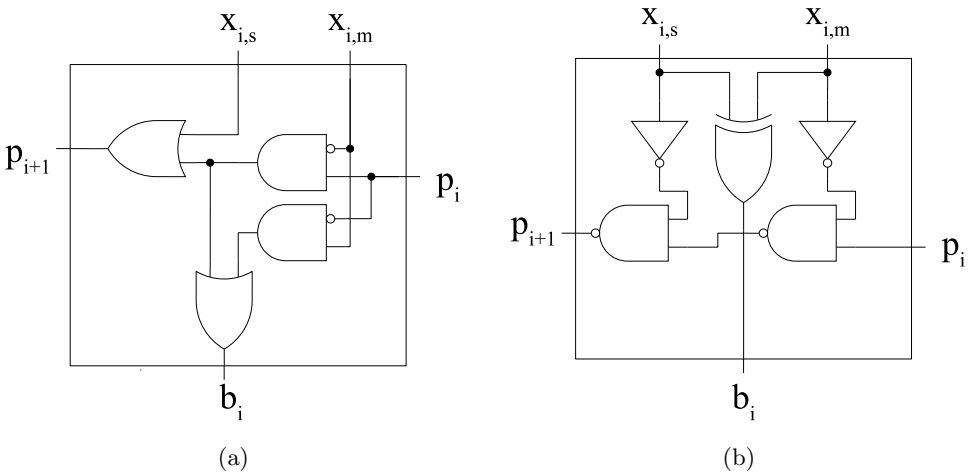


Figure 7.3: Processing elements for one digit of CSD-to-binary conversion, (a) optimized for minimal area, (b) optimized for minimal delay

7.2.3 Lookahead Circuit for Bypass Generation

As mentioned in the previous subsection, the borrow signal is generated when $x_i = -1$ and propagated if $x_i = 0$. Unfortunately, the definition of CSD only restricts the maximal density of nonzero digits, but has no restrictions on consecutive zero digits, therefore no further optimization can be done. By unfolding Equation (7.8), each borrow input can be generated individually with a delay of $n \log_2(n)$ when using balanced trees, but this comes at a high cost for larger bit width. Partial speedup methods like carry look-ahead [25] or parallel prefix [178] computations as in carry-propagation adder can be used with $G(x) = x_s$ and $P(x) = \overline{x_m}$.

7.2.4 Results and Comparison

The proposed CSD to binary conversion method was implemented in VHDL and compiled using Synopsys Design Compiler v2007.12 with the TSMC 0.18μm standard cell library. The results for only area minimization as well as combined area and delay optimization are shown in Table 7.6.

TABLE 7.6: Results for CSD to Binary Conversion for Area Optimization only and Area-Delay Optimization Based on CMOS 0.18μm Standard Cell Library

Bits	**Area Opt.**		**Area & Delay Opt.**	
	[μm^2]	**[ns]**	**[μm^2]**	**[ns]**
PE	53	0.18	96	0.11
8	346	1.05	672	0.65
16	772	2.24	1660	1.28
32	1623	4.63	3476	2.66
64	3326	9.41	7458	5.40
128	6733	18.97	14516	10.89
256	13545	38.09	28244	21.88

As expected, the area and delay grow linearly with the bit width of the converter. The difference in area between the area and delay optimized versions is smaller than for the binary to CSD converter, which is particularly obvious from the sizes of the PE elements for the forward and reverse converters. The smallest forward converter PE element is smaller than the area optimized reverse converter PE while the fastest forward converter PE is bigger than the delay optimized reverse converter PE. In terms of delay, the reverse converter is slower than the forward converter as expected due to the two logic gates in the critical path. Design compiler uses a faster special complex gateto replace the two individual logic gates, which explains why the delay for the reverse converter is not double that of the forward converter. In a system where both forward and reverse conversions are used, the reverse converter is a more critical overhead. With only less than 0.1 ns / bit, this delay overhead is not a severe performance bottleneck.

7.3 Chapter Summary

In this chapter, a method for binary to CSD conversion using a bypass signal was proposed. With the use of a bypass signal instead of a carry signal, the proposed method outperforms other carry propagation methods. It is also shown that the generation of the bypass signal to each PE can be further sped up by using relatively simple look-ahead logic. Its implementations on FPGA and ASIC both showed significant area and delay savings.

A method for CSD to binary conversion using a borrow signal and a low number of logic elements was also proposed. With the reduced number of logic elements and only two logic gates in the critical path, similar results to the binary to CSD converter are obtained. The hardware efficiency makes the use of CSD representation for data communication on system bus feasible. To further speed up the converter, techniques used to accelerate two operand addition can be used to improve its speed to match with that of the forward converter. This argument was supported by the results of ASIC implementation.

Chapter 8

Conclusion and Future Works

8.1 Conclusion

In this thesis, different design methodologies to reduce the complexity of digital FIR filters have been developed.

The establishment of 'sysFIR' as a platform for design automation of FIR filters has helped tremendously in the process of developing, testing, verifying and the result generation and comparison of the new design methodologies proposed and in this thesis. Based on the extensive literature review and a careful appraisal of the number representations used by existing MCM optimization algorithms, the myth of Binary Common Subexpression Elimination was debunk and new methods of MCM optimization have been proposed with the aid of sysFIR. The proposed MCM optimization method leverages the strength of graph based algorithms but limits the maximal delay of each coefficient multiplier to its minima such that the resulting filter achieve smaller area without compromising the delay. Due to its flexibility and efficiency in searching for the intermediate values for adder minimization, the new algorithm is extended to deal with the reconfigurable MCM problem, where one operator[1] can be reused or time-multiplexed to produce different outputs. Two bit-level algorithms were proposed. One algorithm targets FPGA implementation, where the LUT are used for the implementation of multiplications and an upper bound on the LUTs required was established. The other algorithm targets high-speed ASIC implementation by removing the redundancies. An upper bound for the maximum height of the partial product matrix of reconfigurable multiplier is established.

The analysis of and optimization methods for the SA block contributed significantly to the reduction of area and delay of FIR filters. Based on probabilistic analysis, it has been shown that central limit theorem is applicable to the word level signals of the SA block. The mean and variance of the approximated normal distribution has been derived. It is shown that the area ofSA block can be optimized by shortening the sign extension and carry ripple chain. These are accomplished by appropriately bisecting the operands or saving the carry for the next addition. The cost of pipelining the RCA adders in the SA has been known to be considerably high. A new technique has been developed in this thesis with an astonishingly low cost for pipelining the SA block.

CSD has been used widely in the design algorithms, but not directly in the hardware im-

[1] Adder, Subtractor or fused Adder/Subtractor

plementation of the arithmetic operations. To use CSD numbers as operands in an arithmetic operation, conversion between CSD and Binary number representation must be performed on the fly. These conversions, unfortunately cannot evade the carry propagation problem. High-speed converters for the conversions to and fro CSD representation have been presented. The critical path consists of only one NOR gate for the conversion to CSD and two NAND gates for the conversion from CSD.

The salient results obtained from the main contributing chapters of this thesis is summarized as follows.

sysFIR - A Platform for Design Automation of FIR Filters

The implementation and extension of sysFIR provided many valuable insights that lead to findings presented in this thesis. It has been the basis for the development of new algorithms. One of the most beneficial aspect of sysFIR is it's ability to not only produce syntactically correct VHDL code along with the scripts for synthesis by Design Compiler and Xilinx ISE, but also its ability to verify that the VHDL code is functionally correct. This leads to the convenience of the bit width analysis and the calculation of minimum word lengths for the signals within the SA block and ensured that the proposed algorithms produce correct results. Due to the complexity of VHDL implementation of FIR filter, manual coding and functional verification is very cumbersome and time consuming. As the published results were not always clearly and unambiguously presented, sysFIR helps to functionally verify the design with common set of inputs to obtain a better pictures of presented results.

Word Level MCM Optimization

The optimization of the Multiple Constant Multiplication (MCM) problem have received a lot of attention in the past two decades and the associated LO reduction problem can be considered well solved as guaranteed optimal algorithms were proposed in recent publications, although their runtime is prohibitively long for most real word problem size. The heuristic CSE or AG algorithms have also been developed to a very advanced level and it was shown that they produce solutions with quality very close, if not equal, to the optimal results. By allowing the use of non unique number representations like MSD, the CSE algorithms can expand their search space to improve the quality of their results but the computational complexity will also grow towards that of AG algorithms. An in depth analysis of a recent binary based CSE [23] reveals the errors in its presented results. Several inconsistencies were detected and summarized in Section 4.1.

The application of additional constraints, such as limiting the maximal LD for all coefficients opens up a new avenue for algorithmic development. The proposed 'MinLD' algorithm requires less LO than CSE methods with the same minimal logic depth. When comparing it against AG algorithms, the number of adders increases only marginally. The small number of LO combined with minimal LD results in a dynamic power reduction of up to 3.4 times as estimated by the Glitch Path Count and Glitch Path Score. The same optimization approach was extended to

polyphase FIR filters and other multiple output MCM problems. The presented reconfigurable MCM algorithm is capable of generating a reconfigurable block that produces the product of an input multiplied by several fundamentals at any time with promising results comparing with the existing reconfigurable MCM algorithms.

Bit-Level MCM Optimization

LUT-based distributive arithmetic is amenable for FPGA. A method for designing fully parallel MCM based on LUT has been presented. It has been shown that the number of LUTs required for each input bit width can be limited to a number far below the general theoretical expectation. The proposed method is experimented by splitting the 8-bit input signal of a filter into two 4-bit signals for the addresses to the LUTs. The results showed that the LUT-based method is on par with state of the art MCM optimization techniques in terms of the number of logic slices used, but outperforms them by 33% in terms of delay.

An upper bound of the maximum height of the partial product matrix for constant multiplication has been established and was found to be at most one more than half the bit width of the input variable irrespective of the value of the constant. An even more tight upper bound which relies on the input bit width and the number of nonzero digits of CSD encoded constant multiplier has also been derived. Further reduction has been made possible by sharing adders and logic elements from the partial product matrix. A FIR filter example showed that the proposed method based on the above study reduces 41% of FAs and 56% of HAs in the first stage of the CSA tree of the shift-and-add network.

Both methods are more advantageous if the input bit width is small and the precision of the constant multipliers is much larger. Such condition is found in the digital filters that follow directly after a high speed ADC.

Structural Adder Analysis/Optimization/Pipelining

It has been shown by analysis of the signal properties within the SA block that the central limit theorem is valid and the value distribution can be approximated by normal distribution. Formulae are provided for the estimation of its mean and variance.

Although the inputs to the SA in the SA block are not correlated, optimization of the adders in the SA block is possible. Depending on the number representation, the sign extension (for two's complement) or the carry ripple (for offset representation) can be reduced by bisecting the saved sum or by saving and forwarding the carry to the next SA. The SA block has been shown to be area dominant and even a small percentage of its area reduction can be more significant than a larger percentage of MCM area reduction in the optimization of the whole FIR filter. A holistic comparison of the SA optimizations for the two number representations combined with a MCM optimization showed an area reduction of up to 14.48%, where the optimization of the MCM block only contributed up to 3.12% of the overall filter. Both SA optimizations strives for a positive tradeoff between the use of logic gates and registers, which is less likely to be the strategy used by the synthesis tools as the synthesis tool tends to eliminate

logic redundancies.

Pipelining for the SA block has not been reported in the literature. In this regard, an area efficient method has been proposed to pipeline the adders in the SA block by saving and forwarding the carry into the next tap. As the proposed method uses mainly the existing registers, the overhead is small and for not aggressive pipelining it is dominated by the registers required for the additional number of pipelined stages introduced by the final merging before the output. Synthesis results for a 121 tap filter have shown that the delay can be reduced by 30% while the area has only increased by 4%.

A bit width truncation method has also been proposed by introducing truncation and round-half-up operations into the inputs of SA to reduce the fractional bits of the signals in the SA block progressively to the input bit width. This method reduces the area required for the filter and introduces a small error of only ± 1 LSB comparing with performing a full precision calculation in the SA and then rounding the final result at the output.

The optimization and truncation methods have one thing in common, which is the absolute savings are almost independent of the input bit width of the filter, but dependent heavily on the number of taps and the envelope of the impulse response of the FIR filter. FIR filters with short transition band and high stop band attenuation can gain the most from the SA optimization, while short filters gain the most from the SA output truncation.

Binary-to-CSD and CSD-to-Binary Converter

By replacing the carry signal by a bypass signal, a new circuit architecture has been proposed for the conversion from Binary to CSD. The proposed circuit requires only a single logic gate in the critical path and outperforms existing conversion circuits. A new circuit architecture for the reverse conversion from CSD to binary has also been proposed. It uses a borrow signal and requires only two logic gates in the critical path. Both methods are well suited for ASIC implementation. The CSD signals can be generated on-the-fly to reduce the storage requirement if CSD numbers are required. Storing a CSD number requires two registers per digit, whereas the cost of the proposed high-speed Binary to CSD converter is approximately equivalent to one register per bit based on Complementary Metal-Oxide-Semiconductor (CMOS) $0.18\mu m$ standard library and has a delay of only 0.05 ns per digit.

Summary

The objectives set forth in this thesis in Section 1.2 for the design methodologies for complexity reduction of digital FIR filters have been fulfilled with the five major contributions highlighted in Section 1.3 and summarized in this Chapter. In conclusion, the outputs of this thesis have achieved the milestones of advancing the design methodologies for smaller and faster digital FIR filters for ASIC and FPGA based implementations. In fact, the optimization of structural adder block of transpose direct form digital FIR filters is an uncharted new research area with no available literature.

8.2 Future Works

Multiple Constant Multiplication

The research in MCM optimization has reached a mature state and smaller problems can now be solved optimally, and excellent heuristic algorithms are available for bigger problems. By minimizing the LD, the power consumption based on glitch generation and propagation estimates has been reduced, but more accurate power estimates are rarely used in the cost function of filter design and optimization algorithms. Bit-level optimization guided by more accurate gate-level power estimates may produce better results than the existing solutions, although the improvement may not be significantly great.

The influence of the compilers used for the mapping to ASIC or FPGA targets could be analyzed. With the computing power available today, the MCM problems can be optimized down to the logic gate level efficiently by the tool. The effect of the high-level MCM optimization algorithm is diminishing or has been overwhelmed by the logic optimization of the tools in some cases except only the visible reduction in the compilation time. This is true for as long as there is no delay element present in the MCM block. If pipelining is used, the effect of MCM optimization is still significant. Such pipelining optimization algorithms could lead to more optimized circuits both in terms of throughput and area than the current capability of logic synthesis tool. Besides the two methods presented in this thesis for bit-level optimization methods, further research and enhancement could derive more merits for specific application conditions, like small input bit width or high coefficient precision.

The multi-input MCM problems like matrix multiplication or complex signal processing in FFT can be solved more efficiently and effectively by CSE algorithms, as the search space for AG algorithms is too big. Possibilities to reduce the search space for better solutions using AG methodologies could be explored for these applications. A relevant topic is the graph inversion problem. In MCM problems, one input variable is manipulated and produce several outputs. The inverse problem where several input variables contributed to a single output is harder to solve, although its shift-and-add network may be viewed as an inverted graph of an MCM problem. One possible research direction is to convert the original problem into an MCM problem with a different set of constraints for the recast MCM problem formulation so that result has a multiple-input single-output architecture or can be easily mapped to such an architecture.

Correlated Addition/Subtraction

The addition of two correlated signals leads to simplifications in the adders as not all output combinations are possible. The same is true for cascaded additions of correlated signals, which can be found in the shift-and-add networks for constant multiplications. With high probability, current synthesis tools will eliminate the bit-level redundancies in each adder independently, but are less likely to eliminate correlated redundancies effectively for cascaded adders. An analysis of the relationship between the correlated inputs and the possible simpli-

fication omitted by the tools would bolster the understanding of the lower-level properties of constant multiplications. The understanding will help to derive new methods that are capable of producing solutions with higher parallelism and lower power consumption.

Reconfigurable Multiple Constant Multiplication

Reconfigurable MCM optimization is a relatively new research and the method presented in Section 4.3 considers reconfigurability exists among several MCM blocks. One possible improvement is to relax the minimal LD constraint by allowing some adder paths to have longer LD as long as it is not larger than the maximal LD among all adder paths. Another improvement is the different approach to the merging of several outputs. It was noted during the development of the algorithm that the merging of outputs requires a significant number of MUX if an incorrect position is selected. It could happen that implementing an addition twice at different locations is advantageous compared with the use of MUX.

Structural Adder Optimization

Many real world signals do not exhibit white noise properties as in the analysis of the value distribution of SA block. An analysis based on sinusoid signals could lead to further insights. The possibilities to optimize the structural adder block of a FIR filter has been demonstrated in the methods presented in Chapter 6. The presented optimizations mainly focus on the area reduction of area. Other cost functions may be more useful if the presented techniques are to be applied to other add-and-delay subsystems to improve their performance. It is a promising research direction to integrate the presented method for pipelining the structural adders block with other optimization techniques and pipelined MCM blocks. Further investigation can be carried out on the benchmark filter set with comparison made between the use of pipelining and high-speed CPA like CLA and for different types of FPGA implementation.

Signal Bit Width Reduction in FIR Filters

There exists some disagreement in the FIR filter community with regards to the bit width reduction of arithmetic operations within the FIR filter. There is no dispute on the influences of truncations within the MCM block over several paths as the internal signals of MCM block are reused. The controversy is on whether the errors introduced by these truncations are random, or correlated due to the reuse of the signals within the MCM block. This can be further investigated in order to better utilize the additional dimension of output precision as a trade-off for area, delay and power minimization.

Another idea is to investigate the relation between dynamic range of the filter input/output and the tolerance of FIR filter transfer function specification. A simple example is that it does not make sense to implement a filter with 80 dB attenuation in the stop band when only an 8-bit signal is used for input and output. The dynamic range of an 8-bit signal is only about 48 dB but 14 bits or more are required. Even if this design criterion has been resolved, the internal bit widths of a FIR filter are still calculated to the full precision. The noise of

the input LSB is attenuated by 80 dB even though the output does not require or cannot be afford to have this level of precision. This fact should be exploited to reduce the internal bit width of the FIR or modify the constant multipliers such that signals with larger magnitude are attenuated more than signals with low magnitude by the amount just sufficient to push them into the ineluctable quantization noise.

CSD Arithmetics

CSD representation is not only good for constant representation in MCM problem, but also useful for cost reduction in general multiplications. It could be interesting to apply the very fast binary to CSD forward and reverse converters developed in this thesis to develop an adaptive FIR filter that uses CSD representation for the variable coefficients. Two arithmetic elements in addition to the converters have to be developed. First, efficient operators have to be developed for the addition/subtraction of two CSD numbers with the sum expressed in CSD or the addition/subtraction of one CSD number and one binary (two's complement) number with a CSD output. Second, the behavior of CSD signals upon scaling and rounding need to be studied and efficient CSD scalers need to be developed.

Author's Publications

This chapter lists all publications where the author of this thesis was either the first author or contributing author. First, three journal papers are listed, of which [82] is an extended comment on [23]. Due to the extent of the discussions on the problems found in [23], it was necessary to submit a short paper instead of a comment. One journal papers [179], based on the content Chapter 6, is currently in preparation. One book chapter [180], based on the content of Chapter 2 is scheduled for publication.

Six of the seven conference paper listed were contributed by the author of this thesis as the principal author. Three of the papers were presented at the ISCAS with an average acceptance rate of about 40% over the years.

Last but not least, the website FIRsuite with benchmark filters for MCM problems is listed. It was established at the latest stage of the author's PhD candidature. FIRsuite found a small group of contributors and the repercussion from the research community has been very positive.

Journal Papers

[82]: C. H. Chang and M. Faust, "On "A new common subexpression elimination algorithm for realizing low-complexity higher order digital filters"," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 5, pp. 844–848, May 2010.

[50]: M. Faust, O. Gustafsson, and C. H. Chang, "Fast and VLSI efficient binary-to-CSD encoder using bypass signal," *Electronics Letters*, vol. 47, no. 1, pp. 18–20, Jan. 2011.

[44]: R. Huang, C. H. Chang, M. Faust, N. Lotze, and Y. Manoli, "Sign-extension avoidance and word-length optimization by positive-offset representation for FIR filter design," *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 58, no. 12, pp. 916–920, Dec. 2011.

Book Chapter

[180]: P. K. Meher, C. H. Chang, O. Gustafsson, A. P. Vinod, and M. Faust, "Shift-add circuits for constant multiplications," in *Arithmetic Circuits for DSP Applications*. Wiley-IEEE Press, 2014, in press.

In Preparation

[179]: M. Faust and C. H. Chang, "Analysis and optimization of structural adders in fixed coefficient transposed direct form FIR filters," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2014, in preparation.

Conference Papers

[142]: M. Faust and C. H. Chang, "Optimization of structural adders in fixed coefficient transposed direct form FIR filters," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Taipei, Taiwan, 24-27 May 2009, pp. 2185–2188.

[58]: M. Faust and C. H. Chang, "Minimal logic depth adder tree optimization for multiple constant multiplication," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Paris, France, 30 May - 2 Jun. 2010, pp. 457–460.

[143]: M. Faust and C. H. Chang, "Reduction of partial product matrix for high-speed single or multiple constant multiplication," in *Proc. Asia Pacific Conf. on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, Shanghai, China, 22-24 Sep. 2010, pp. 416–420.

[105]: M. Faust, O. Gustafsson, and C. H. Chang, "Reconfigurable multiple constant multiplication using minimum adder depth," in *Proc. 44th Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA, 7-10 Nov. 2010, pp. 1297–1301.

[144]: M. Faust and C. H. Chang, "Bit-parallel multiple constant multiplication using look-up tables on FPGA," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Rio de Janeiro, Brasil, 15-18 May 2011, pp. 657–660.

[145]: M. Faust and C. H. Chang, "Low error bit width reduction for structural adders of FIR filters," in *Proc. European Conf. on Circuit Theory and Design (ECCTD)*, Linköping, Sweden, 29-31 Aug. 2011, pp. 713–716.

[168]: M. Kumm, P. Zipf, M. Faust, and C. H. Chang, "Pipelined adder graph optimization for high speed multiple constant multiplication," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Seoul, Korea, 20-23 May. 2012, pp. 49–52.

Website

[122]: FIRsuite, "Suite of constant coefficient FIR filters," 2014. [Online]. Available: `http://www.firsuite.net`

References

[1] 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. Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 2, pp. 151–165, Feb. 1996.

[2] D. R. Bull and D. H. Horrocks, "Primitive operator digital filter synthesis using a shift biased algorithm," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 2, Espoo, Finland, 7-9 Jun. 1988, pp. 1529–1532.

[3] ——, "Primitive operator digital filters," *IEE Proc. G*, vol. 138, no. 3, pp. 401–412, Jun. 1991.

[4] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 42, no. 9, pp. 569–577, Sep. 1995.

[5] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, no. 10, pp. 677–688, Oct. 1996.

[6] N. Sankarayya, K. Roy, and D. Bhattacharya, "Algorithms for low power and high speed FIR filter realization using differential coefficients," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 44, no. 6, pp. 488–497, Jun. 1997.

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

[8] H.-J. Kang and I.-C. Park, "FIR filter synthesis algorithms for minimizing the delay and the number of adders," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, no. 8, pp. 770–777, Aug. 2001.

[9] Y. Jang and S. Yang, "Low-power CSD linear phase FIR filter structure using vertical common sub-expression," *Electronics Letters*, vol. 38, no. 15, pp. 777–779, Jul. 2002.

[10] M. Martínez-Peiró, E. I. Boemo, and L. Wanhammar, "Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 49, no. 3, pp. 196–203, Mar. 2002.

[11] A. P. Vinod, E. M. K. Lai, A. B. Premkuntar, and C. T. Lau, "FIR filter implementation by efficient sharing of horizontal and vertical common subexpressions," *Electronics Letters*, vol. 39, no. 2, pp. 251–253, 23 Jan. 2003.

[12] C.-Y. Yao, H.-H. Chen, T.-F. Lin, C.-J. Chien, and C.-T. Hsu, "A novel common-subexpression-elimination method for synthesizing fixed-point FIR filters," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 51, no. 11, pp. 2215–2221, Nov. 2004.

[13] N. Boullis and A. Tisserand, "Some optimizations of hardware multiplication by constant matrices," *IEEE Trans. Comput.*, vol. 54, no. 10, pp. 1271–1282, Oct. 2005.

[14] M. D. Macleod and A. G. Dempster, "Multiplierless FIR filter design algorithms," *IEEE Signal Processing Letters*, vol. 12, no. 3, pp. 186–189, Mar. 2005.

[15] A. P. Vinod and E. M. K. Lai, "On the implementation of efficient channel filters for wideband receivers by optimizing common subexpression elimination methods," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 2, pp. 295–304, Feb. 2005.

[16] Y. Wang and K. Roy, "CSDC: a new complexity reduction technique for multiplierless implementation of digital FIR filters," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 52, no. 9, pp. 1845–1853, Sep. 2005.

[17] F. Xu, C. H. Chang, and C. C. Jong, "Contention resolution algorithm for common subexpression elimination in digital filter design," *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 52, no. 10, pp. 695–700, Oct. 2005.

[18] O. Gustafsson, "Lower bounds for constant multiplication problems," *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 54, no. 11, pp. 974–978, Nov. 2007.

[19] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. on Algorithms*, vol. 3, no. 2, p. 11, May 2007.

[20] L. Aksoy, E. da Costa, P. Flores, and J. Monteiro, "Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 6, pp. 1013–1026, Jun. 2008a.

[21] L. Aksoy, E. O. Gunes, and P. Flores, "An exact breadth-first search algorithm for the multiple constant multiplications problem," in *Proc. IEEE NORCHIP Conf.*, Tallinn, Estonia, Nov. 2008b, pp. 41–46.

[22] C. H. Chang, J. Chen, and A. P. Vinod, "Information theoretic approach to complexity reduction of FIR filter design," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 55, no. 8, pp. 2310–2321, Sep. 2008.

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

[24] 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 and Systems I: Regular Papers*, vol. 55, no. 2, pp. 559–571, Mar. 2008.

[25] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms and Applications*, 4th ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2007.

[26] G. De Micheli, *Synthesis and optimization of digital circuits*, ser. McGraw-Hill series in electrical and computer engineering. Electronics and VLSI circuits. New York: McGraw-Hill, 1994.

[27] L. Wanhammar, *DSP integrated circuits*, ser. Academic Press series in engineering. San Diego, Calif.: Academic Press, 1999.

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

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

[30] Y. C. 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, Jun. 1983.

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

[32] 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, Reg. Papers*, vol. 54, no. 10, pp. 2330–2338, Oct. 2007.

[33] F. Xu, C. H. Chang, and C.-C. Jong, "Hamming weight pyramid - A new insight into canonical signed digit representation and its applications," *Computers & Electrical Engineering*, vol. 33, no. 3, pp. 195–207, May 2007.

[34] D. Shi and Y. J. Yu, "Design of linear phase FIR filters with high probability of achieving minimum number of adders," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 58, no. 1, pp. 126–136, Jan. 2011.

[35] Y. J. Yu, D. Shi, and Y. C. Lim, "Design of extrapolated impulse response FIR filters with residual compensation in subexpression space," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 56, no. 12, pp. 2621–2633, Dec. 2009.

[36] M. Aktan, A. Yurdakul, and G. Dundar, "An algorithm for the design of low-power hardware-efficient FIR filters," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 6, pp. 1536–1545, Jul. 2008.

[37] D. Shi and Y. J. Yu, "Design of discrete-valued linear phase FIR filters in cascade form," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 58, no. 7, pp. 1627–1636, jul. 2011.

[38] P. R. Cappello and K. Steiglitz, "Some complexity issues in digital signal processing," *IEEE Trans. Acoustics, Speech and Signal Processing*, vol. 32, no. 5, pp. 1037–1041, Oct. 1984.

[39] G. W. Reitweisner, "Binary arithmetic," *Adv. Comput.*, vol. 1, pp. 232–308, 1960.

[40] H. L. Garner, L. A. Franz, and R. Morris, "Number systems and arithmetic," in *Advances in Computers*. New York: Academic Press, 1965, vol. 6, pp. 131–194.

[41] K. Hwang, *Computer Arithmetic: Principles, Architecture and Design*. John Wiley & Sons, Inc., 1979.

[42] B. C. Wong and H. Samueli, "A 200-MHz all-digital QAM modulator and demodulator in 1.2-μm CMOS for digital radio applications," *IEEE J. of Solid-State Circuits*, vol. 26, no. 12, pp. 1970–1980, Dec. 1991.

[43] O. Gustafsson, K. Johansson, and L. S. DeBrunner, "Techniques for avoiding sign-extension in multiple constant multiplication," in *Proc. 43rd Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA, 1-4 Nov. 2009.

[44] R. Huang, C. H. Chang, M. Faust, N. Lotze, and Y. Manoli, "Sign-extension avoidance and word-length optimization by positive-offset representation for FIR filter design," *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 58, no. 12, pp. 916–920, Dec. 2011.

[45] A. Avizienis, "Signed-digit number representation for fast parallel arithmetic," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 389–400, Sep. 1961.

[46] S. Arno and F. S. Wheeler, "Signed digit representations of minimal Hamming weight," *IEEE Trans. Comput.*, vol. 42, no. 8, pp. 1007–1010, Aug. 1993.

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

[48] A. G. Dempster and M. D. Macleod, "Generation of signed-digit representations for integer multiplication," *IEEE Signal Processing Letters*, vol. 11, no. 8, pp. 663–665, Aug. 2004.

[49] Y. C. Lim, J. B. Evans, and B. Liu, "Decomposition of binary integers into signed power-of-two terms," *IEEE Trans. Circuits and Systems*, vol. 38, no. 6, pp. 667–672, Jun. 1991.

[50] M. Faust, O. Gustafsson, and C. H. Chang, "Fast and VLSI efficient binary-to-CSD encoder using bypass signal," *Electronics Letters*, vol. 47, no. 1, pp. 18–20, Jan. 2011.

[51] J. Thong and N. Nicolici, "Time-efficient single constant multiplication based on overlapping digit patterns," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 9, pp. 1353 – 1357, Sep. 2009.

[52] M. Faust and M. Rähle, "An integrated environment for automatic VHDL generation for FIR filters," Diploma Thesis (DA-ES-142, requestable from erwin.braendle@hsr.ch), University of Applied Sciences Rapperswil (HSR), Nanyang Technological University Singapore (NTU), Dez. 2006.

[53] T. G. Noll, "Carry-save architectures for high-speed digital signal processing," *The Journal of VLSI Signal Processing*, vol. 3, no. 1-2, pp. 121–140, Jun. 1991.

[54] K. Nakayama, "Permuted difference coefficient realization of FIR digital filters," *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. 30, no. 2, pp. 269–278, Apr. 1982.

[55] A. Matsuura and A. Nagoya, "Formulation of the addition-shift-sequence problem and its complexity," in *Algorithms and Computation*, ser. Lecture Notes in Computer Science, H. Leong, H. Imai, and S. Jain, Eds. Springer Berlin / Heidelberg, 1997, vol. 1350, pp. 42–51.

[56] L. Guerra, M. Potkonjak, and J. Rabaey, "Concurrency characteristics in DSP programs," in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP-94)*, vol. II, Adelaide, SA, 19-22 Apr. 1994, pp. 433–436.

[57] K. D. Cooper, L. T. Simpson, and C. A. Vick, "Operator strength reduction," *ACM Trans. Program. Lang. Syst.*, vol. 23, pp. 603–625, Sep. 2001.

[58] M. Faust and C. H. Chang, "Minimal logic depth adder tree optimization for multiple constant multiplication," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Paris, France, 30 May - 2 Jun. 2010, pp. 457–460.

[59] R. Bernstein, "Multiplication by integer constants," *Softw.-Pract. Exper.*, vol. 16, no. 7, pp. 641–652, Jul. 1986.

[60] V. Lefèvre, "Multiplication by an integer constant," Institut national de recherche en informatique et automatique (INRIA), Rapport de recherche RR-4192, May 2001. [Online]. Available: http://www.vinc17.org/research/papers/rr_mulcst2.pdf

[61] A. G. Dempster and M. D. Macleod, "Using all signed-digit representations to design single integer multipliers using subexpression elimination," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 3, Vancouver, Canada, 23-26 May 2004, pp. 165–168.

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

[63] O. Gustafsson, A. G. Dempster, and L. Wanhammar, "Extended results for minimum-adder constant integer multipliers," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 1, Scottsdale, AZ, 26-29 May 2002, pp. 73–76.

[64] A. G. Dempster and M. D. Macleod, "General algorithms for reduced-adder integer multiplier design," *Electronics Letters*, vol. 31, no. 21, pp. 1800–1802, 12 Oct. 1995.

[65] D. Li, "Minimum number of adders for implementing a multiplier and its application to the design of multiplierless digital filters," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 42, no. 7, pp. 453–460, Jul. 1995.

[66] R. Pinch, "Asymptotic upper bound for multiplier design," *Electronics Letters*, vol. 32, no. 5, pp. 420–421, Feb. 1996.

[67] J. Thong and N. Nicolici, "A novel optimal single constant multiplication algorithm," in *Proc. 47th ACM/IEEE Design Automation Conf. (DAC)*, Anaheim, CA, 13-18 Jun. 2010, pp. 613–616.

[68] A. Chatterjee, R. K. Roy, and M. A. d'Abreu, "Greedy hardware optimization for linear digital circuits using number splitting and refactorization," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 4, pp. 423–431, Dec. 1993.

[69] M. D. Macleod and A. G. Dempster, "Common subexpression elimination algorithm for low-cost multiplierless implementation of matrix multipliers," *Electronics Letters*, vol. 40, no. 11, pp. 651–652, May 2004.

[70] H. Safiri, M. Ahmadi, G. A. Jullien, and W. C. Miller, "A new algorithm for the elimination of common subexpressions in hardware implementation of digital filters by using genetic programming," *The J. of VLSI Signal Processing*, vol. 31, no. 2, pp. 91–100, Jun. 2002.

[71] O. Gustafsson and A. G. Dempster, "On the use of multiple constant multiplication in polyphase FIR filters and filter banks," in *Proc. 6th Nordic Signal Processing Symposium (NORSIG)*, Espoo, Finland, 9-11 Jun. 2004, pp. 53–56.

[72] F. Xu, C. H. Chang, and C. C. Jong, "Modified reduced adder graph algorithm for multiplierless FIR filters," *Electronics Letters*, vol. 41, no. 6, pp. 302–303, Mar. 2005.

[73] J.-H. Han and I.-C. Park, "FIR filter synthesis considering multiple adder graphs for a coefficient," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 5, pp. 958–962, May 2008.

[74] Spiral, "Software/hardware generation for DSP algorithms - multiplierless constant multiplication," 2014. [Online]. Available: http://www.spiral.net/hardware/multless.html

[75] A. G. Dempster, S. S. Dimirsoy, and I. Kale, "Designing multiplier blocks with low logic depth," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 5, Scottsdale, Arizona, 26-29 May 2002, pp. 773–776.

[76] O. Gustafsson, H. Ohlsson, and L. Wanhammar, "Minimum-adder integer multipliers using carry-save adders," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 2, Sydney, NSW, 6-9 May 2001, pp. 709–712.

[77] O. Gustafsson, A. G. Dempster, and L. Wanhammar, "Multiplier blocks using carry-save adders," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 2, Vancouver, Canada, 23-26 May 2004, pp. 473–476.

[78] O. Gustafsson, "Comments on 'A 70 MHz multiplierless FIR Hilbert transformer in 0.35 μm standard CMOS library'," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E91-A, no. 3, pp. 899–900, Mar. 2008.

[79] K. Johansson, "Low power and low complexity shift-and-add based computations," Ph.D. dissertation, Linköping University, 2008, Linköping Studies in Science and Technology. Dissertations.

[80] K. Johansson, O. Gustafsson, L. S. DeBrunner, and L. Wanhammar, "Minimum adder depth multiple constant multiplication algorithm for low power FIR filters," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Rio de Janeiro, Brazil, 15-18 May 2011.

[81] O. Gustafsson, K. Khursheed, M. Imran, and L. Wanhammar, "Generalized overlapping digit patterns for multi-dimensional sub-expression sharing," in *Proc. Int. Conf. on Green Circuits and Systems (ICGCS)*, Shanghai, China, 21-23 Jun. 2010, pp. 65–68.

[82] C. H. Chang and M. Faust, "On "A new common subexpression elimination algorithm for realizing low-complexity higher order digital filters"," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 5, pp. 844–848, May 2010.

[83] A. V. Aho, S. C. Johnson, and J. D. Ullman, "Code generation for expressions with common subexpressions," *J. of the ACM*, vol. 24, no. 1, pp. 146–160, Jan. 1977.

[84] R. Hartley, "Optimization of canonic signed digit multipliers for filter design," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 4, Singapore, 11-14 Jun. 1991, pp. 1992–1995.

[85] M. Mehendale, S. D. Sherlekar, and G. Venkatesh, "Synthesis of multiplier-less FIR filters with minimum number of additions," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, San Jose, CA, 5-9 Nov. 1995, pp. 668–671.

[86] R. Pasko, P. Schaumont, V. Derudder, and D. Durackova, "Optimization method for broadband modem FIR filter design using common subexpression elimination," in *Proc. 10th Int. Symposium on System Synthesis*, Antwerp, Belgium, 17-19 Sep. 1997, pp. 100–106.

[87] A. P. Vinod and E. M.-K. Lai, "Comparison of the horizontal and the vertical common subexpression elimination methods for realizing digital filters," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Kobe, Japan, May 2005, pp. 496–499.

[88] P. Flores, J. Monteiro, and E. Costa, "An exact algorithm for the maximal sharing of partial terms in multiple constant multiplications," in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, San Jose, CA, 6-10 Nov. 2005, pp. 13–16.

[89] K. G. Smitha and A. P. Vinod, "Low power realization and synthesis of higher-order FIR filters using an improved common subexpression elimination method," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E91-A, no. 11, pp. 3282–3292, Nov. 2008.

[90] A. G. Dempster, M. D. Macleod, and O. Gustafsson, "Comparison of graphical and subexpression methods for design of efficient multipliers," in *Proc. 38th Asilomar Conf. on Signals, Systems and Computers*, vol. 1, Pacific Grove, CA, 7-10 Nov. 2004, pp. 72–76.

[91] H. Choo, K. Muhammad, and K. Roy, "Complexity reduction of digital filters using shift inclusive differential coefficients," *IEEE Trans. Signal Processing*, vol. 52, no. 6, pp. 1760–1772, Jun. 2004.

[92] O. Gustafsson, H. Ohlsson, and L. Wanhammar, "Improved multiple constant multiplication using a minimum spanning tree," in *Proc. 38th Asilomar Conf. on Signals, Systems and Computers*, vol. 1, Pacific Grove, CA, 7-10 Nov 2004, pp. 63–66.

[93] O. Gustafsson, "A difference based adder graph heuristic for multiple constant multiplication problems," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, New Orleans, LA, 27-30 May 2007, pp. 1097–1100.

[94] K. Muhammad and K. Roy, "A graph theoretic approach for synthesizing very low-complexity high-speed digital filters," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 2, pp. 204–216, Feb. 2002.

[95] A. P. Vinod, A. Singla, and C. H. Chang, "Low-power differential coefficients-based FIR filters using hardware-optimised multipliers," *Circuits, Devices & Systems, IET*, vol. 1, no. 1, pp. 13–20, Feb. 2007.

[96] S. S. Demirsoy, A. G. Dempster, and I. Kale, "Design guidelines for reconfigurable multiplier blocks," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 4, Bangkok, Thailand, 25-28 May 2003, pp. 293–296.

[97] S. S. Demirsoy, I. Kale, and A. G. Dempster, "Efficient implementation of digital filters using novel reconfigurable multiplier blocks," in *Proc. 38th Asilomar Conf. on Signals, Systems and Computers*, vol. 1, Pacific Grove, CA, 7-10 Nov. 2004, pp. 461–464.

[98] S. Demirsoy, I. Kale, and A. G. Dempster, "Reconfigurable multiplier blocks: Structures, algorithm and applications," *Circuits, Systems, and Signal Processing*, vol. 26, no. 6, pp. 793–827, Dec. 2007.

[99] P. Tummeltshammer, J. C. Hoe, and M. Puschel, "Time-multiplexed multiple-constant multiplication," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 9, pp. 1551–1563, Sep. 2007.

[100] J. Chen and C. H. Chang, "High-level synthesis algorithm for the design of reconfigurable constant multiplier," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 12, pp. 1844–1856, Dec. 2009.

[101] R. Mahesh and A. P. Vinod, "New reconfigurable architectures for implementing FIR filters with low complexity," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 2, pp. 275–288, Feb. 2010.

[102] R. Gutierrez, J. Valls, and A. Perez-Pascual, "FPGA-implementation of time-multiplexed multiple constant multiplication based on carry-save arithmetic," in *Proc. Int. Conf. on Field Programmable Logic and Applications (FPL 2009)*, Prague, Czech Republic, 31 Aug.-2 Sep. 2009, pp. 609–612.

[103] N. Sidahao, G. Constantinides, and P. Cheung, "Power and area optimization for multiple restricted multiplication," in *Proc. Int. Conf. on Field Programmable Logic and Applications*. Tampere, Finland: IEEE, 24-26 Aug. 2005, pp. 112–117.

[104] C. Howard and L. DeBrunner, "High speed DSP block for FPGA devices using a programmable adder graph," in *Proc. 13th IEEE Digital Signal Processing Workshop and 5th IEEE Signal Processing Education Workshop (DSP/SPE 2009)*. Marco Island, FL: IEEE, 4-7 Jan. 2009, pp. 490–494.

[105] M. Faust, O. Gustafsson, and C. H. Chang, "Reconfigurable multiple constant multiplication using minimum adder depth," in *Proc. 44th Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA, 7-10 Nov. 2010, pp. 1297–1301.

[106] A. Yurdakul and G. Dündar, "Multiplierless realization of linear DSP transforms by using common two-term expressions," *The J. of VLSI Signal Processing*, vol. 22, no. 3, pp. 163–172, Sep. 1999.

[107] O. Gustafsson and L. Wanhammar, "ILP modelling of the common subexpression sharing problem," in *Proc. 9th Int. Conf. on Electronics, Circuits and Systems*, vol. 3, Dubrovnik, Croatia, 10 Dec. 2002, pp. 1171–1174.

[108] O. Gustafsson, "Towards optimal multiple constant multiplication: A hypergraph approach," in *Proc. 42nd Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA, 26-29 Oct. 2008, pp. 1805–1809.

[109] L. Aksoy, E. O. Gunes, and P. Flores, "Search algorithms for the multiple constant multiplications problem: Exact and approximate," *Microprocessors and Microsystems*, vol. 34, no. 5, pp. 151–162, Aug. 2010.

[110] K. S. Yeung and S. C. Chan, "The design and multiplier-less realization of software radio receivers with reduced system delay," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 51, no. 12, pp. 2444–2459, Dec. 2004.

[111] A. P. Vinod and E. M. K. Lai, "Hardware efficient DCT implementation for portable multimedia terminals using subexpression sharing," in *Proc. IEEE Region 10 Conf. (TENCON)*, vol. 1, Chiang Mai, Thailand, 21-24 Nov. 2004, pp. 227–230.

[112] S. Vijay and A. P. Vinod, "A new algorithm to implement low complexity DCT for portable multimedia devices," in *Proc. Int. Conf. on Signal Processsing and Communication System*, Gold Coast, Australia, Dec. 2007, pp. 171–176.

[113] M. D. Macleod, "Multiplierless implementation of rotators and FFTs," *EURASIP J. on Applied Signal Processing*, vol. 2005, no. 17, pp. 2903–2910, 2005.

[114] W. Han, A. T. Erdogan, T. Arslan, and M. Hasan, "High-performance low-power FFT cores," *ETRI Journal*, vol. 30, no. 3, pp. 451–460, Jun. 2008.

[115] U. Meyer-Bäse, H. Natarajan, and A. G. Dempster, "Fast discrete Fourier transform computations using the reduced adder graph technique," *EURASIP J. on Advances in Signal Processing*, vol. 2007, no. 67360, p. 8, May 2007.

[116] P. P. Dang and P. M. Chau, "A high performance, low power VLSI design of discrete wavelet transform for lossless compression in JPEG 2000 standard," in *Proc. Int. Conf. on Consumer Electronics (ICCE)*, Los Angeles, CA, 19-21 Jun. 2001, pp. 126–127.

[117] H. Nguyen and A. Chattejee, "Number-splitting with shift-and-add decomposition for power and hardware optimization in linear DSP synthesis," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 4, pp. 419–424, Aug. 2000.

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

[119] S. A. White, "Applications of distributed arithmetic to digital signal processing: a tutorial review," *IEEE ASSP Magazine*, vol. 6, no. 3, pp. 4–19, Jul. 1989.

[120] U. Meyer-Bäse, *Digital signal processing with field programmable gate arrays.* Heidelberg: Springer, 2007.

[121] S. Udit, F. Xu, C. H. Chang, C. C. Jong, and K.-S. Yeo, "sys-FIR: A compiler for evaluating VLSI performance metrics of reduced adder cost FIR filters," in *Proc. IEEE Int. Symp. on Low-power and High-speed Chips (Cool Chips VIII)*, Yokohama Joho Bunka Centre, Yokohama, Japan, Apr. 20-22 2005, pp. 339–346.

[122] FIRsuite, "Suite of constant coefficient FIR filters," 2014. [Online]. Available: http://www.firsuite.net

[123] A. Blad and O. Gustafsson, "Integer linear programming-based bit-level optimization for high-speed FIR decimation filter architectures," *Circuits, Systems, and Signal Processing*, vol. 29, pp. 81–101, 2010.

[124] V. S. Dimitrov, J. Eskritt, L. Imbert, G. A. Jullien, and W. C. Miller, "The use of the multi-dimensional logarithmic number system in DSP applications," in *Proc. IEEE Symp. on Comput. Arithmetic*, Vail, CO, 11-13 Jun. 2001, pp. 247–254.

[125] C. H. Chang, J. Chen, and A. P. Vinod, "Maximum likelihood disjunctive decomposition to reduced multirooted DAG for FIR filter design," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Island of Kos, Greece, 21-24 May 2006, p. 4 pp.

[126] A. G. Dempster and M. D. Macleod, "Digital filter design using subexpression elimination and all signed-digit representations," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 3, Vancouver, Canada, 23-26 May 2004, pp. 169–172.

[127] D. Goodman and M. Carey, "Nine digital filters for decimation and interpolation," *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. 25, no. 2, pp. 121–126, Apr. 1977.

[128] R. Jain, P. T. Yang, and T. Yoshino, "FIRGEN: a computer-aided design system for high performance FIR filter integrated circuits," *IEEE Trans. Signal Process.*, vol. 39, no. 7, pp. 1655–1668, Jul. 1991.

[129] A. Y. Kwentus, Z. Jiang, and J. Willson, A. N., "Application of filter sharpening to cascaded integrator-comb decimation filters," *IEEE Trans. Signal Process.*, vol. 45, no. 2, pp. 457–467, Feb. 1997.

[130] C.-L. Chen and J. Willson, A. N., "A trellis search algorithm for the design of FIR filters with signed-powers-of-two coefficients," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 46, no. 1, pp. 29–39, Jan. 1999.

[131] C.-Y. Yao, "A study of SPT-term distribution of CSD numbers and its application for designing fixed-point linear phase FIR filters," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 2, Sydney, NSW, 6-9 May 2001, pp. 301–304 vol. 2.

[132] K.-Y. Jheng, S.-J. Jou, and A.-Y. Wu, "A design flow for multiplierless linear-phase FIR filters: from system specification to Verilog code," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 5, Vancouver, Canada, 23-26 May 2004, pp. V–293–V–296 Vol.5.

[133] J. Yli-Kaakinen and T. Saramaki, "A systematic algorithm for the design of multiplierless FIR filters," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 2, Sydney, NSW, 6-9 May 2001, pp. 185–188 vol. 2.

[134] S. Rosa, Vagner, E. Costa, J. C. Monteiro, and S. Bampi, "An improved synthesis method for low power hardwired FIR filters," in *Proc. 17th Symp. on Integrated Circuits and System Design.* Pernambuco, Brazil: ACM, 7-11 Sep. 2004, pp. 237 – 241.

[135] A. Shahein, Q. Zhang, N. Lotze, and Y. Manoli, "A novel hybrid monotonic local search algorithm for FIR filter coefficients optimization," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2011.

[136] T. Yoshino, R. Jain, P. T. Yang, H. Davis, W. Gass, and A. H. Shah, "A 100-MHz 64-tap FIR digital filter in 0.8-μm BiCMOS gate array," *IEEE J. Solid-State Circuits*, vol. 25, no. 6, pp. 1494–1501, Dec. 1990.

[137] J. J. Nielsen, "Design of linear-phase direct-form FIR digital filters with quantized coefficients using error spectrum shaping," *IEEE Trans. Circuits Syst.*, vol. 37, no. 7, pp. 1020–1026, Jul. 1989.

[138] F. Xu, C. H. Chang, and C. C. Jong, "I^{2}CRA: contention resolution algorithm for intra- and inter-coefficient common subexpression elimination," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Kobe, Japan, 23-26 May 2005, pp. 1823–1826 Vol. 2.

[139] J. Chen and C. H. Chang, "A tool for automated synthesis and optimization of integrated FIR filters," in *IET Workshop Microelectron. and Embedded Syst.*, Singapore, Jan. 2007.

[140] J. Chen, C. H. Chang, and H. Qian, "New power index model for switching power analysis from adder graph of FIR filter," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Taipei, Taiwan, 24-27 May 2009, pp. 2197–2200.

[141] N. Chen and Z. Yan, "Cyclotomic FFTs with reduced additive complexities based on a novel common subexpression elimination algorithm," *IEEE Trans. Signal Processing*, vol. 57, no. 3, pp. 1010–1020, Mar. 2009.

[142] M. Faust and C. H. Chang, "Optimization of structural adders in fixed coefficient transposed direct form FIR filters," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Taipei, Taiwan, 24-27 May 2009, pp. 2185–2188.

[143] ——, "Reduction of partial product matrix for high-speed single or multiple constant multiplication," in *Proc. Asia Pacific Conf. on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, Shanghai, China, 22-24 Sep. 2010, pp. 416–420.

[144] ——, "Bit-parallel multiple constant multiplication using look-up tables on FPGA," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Rio de Janeiro, Brasil, 15-18 May 2011, pp. 657–660.

[145] ——, "Low error bit width reduction for structural adders of FIR filters," in *Proc. European Conf. on Circuit Theory and Design (ECCTD)*, Linköping, Sweden, 29-31 Aug. 2011, pp. 713–716.

[146] L. Rabiner and O. Herrmann, "The predictability of certain optimum finite-impulse-response digital filters," *IEEE Trans. Circuit Theory*, vol. 20, no. 4, pp. 401–408, Jul. 1973.

[147] S. S. Demirsoy, A. Dempster, and I. Kale, "Transition analysis on FPGA for multiplier-block based FIR filter structures," in *Proc. 7th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS)*, vol. 2, Jounieh, Lebanon, 17-20 Dec. 2000, pp. 862–865.

[148] S. S. Demirsoy, A. G. Dempster, and I. Kale, "Power analysis of multiplier blocks," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 1, Scottsdale, Arizona, 26-29 May 2002, pp. 297–300.

[149] N. Sidahao, G. A. Constantinides, and P. Y. Cheung, "Multiple restricted multiplication," *Lecture Notes in Computer Science, Field Programmable Logic and Application*, vol. 3203/2004, pp. 374–383, 2004.

[150] K. Johansson, O. Gustafsson, and L. Wanhammar, "A detailed complexity model for multiple constant multiplication and an algorithm to minimize the complexity," in *Proc. European Conf. on Circuit Theory and Design*, vol. 3, Cork, Ireland, 28 Aug.-2 Sep. 2005, pp. 465–468.

[151] L. Aksoy, E. O. Gunes, E. Costa, P. Flores, and J. Monteiro, "Effect of number representation on the achievable minimum number of operations in multiple constant multiplications," in *Proc. 2007 IEEE Workshop Signal Process. Syst.*, Shanghai, China, 17-19 Oct. 2007, pp. 424–429.

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

[153] C.-Y. Pai, A. J. Al-Khalili, and W. E. Lynch, "Low-power constant-coefficient multiplier generator," in *Proc. IEEE Int. ASIC/SOC Conf.*, Washington, DC, 12-15 Sep. 2001, pp. 185–189.

[154] ——, "Low-power constant-coefficient multiplier generator," *The J. of VLSI Signal Processing*, vol. 35, no. 2, pp. 187–194, Sep. 2003.

[155] P. K. Meher, "New approach to look-up-table design and memory-based realization of FIR digital filter," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 3, pp. 592–603, Mar. 2010.

[156] ——, "LUT optimization for memory-based computation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 57, no. 4, pp. 285–289, Apr. 2010.

[157] V. A. Bartlett and A. G. Dempster, "Using carry-save adders in low-power multiplier blocks," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, vol. 4, Sydney, NSW, Australia, 6-9 May 2001, pp. 222–225.

[158] O. Gustafsson and L. Wanhammar, "Low-complexity constant multiplication using carry-save arithmetic for high-speed digital filters," in *Proc. 5th Int. Symp. on Image Signal Process. Analysis (ISPA)*, Istanbul, Turkey, 27-29 Sep. 2007, pp. 212–217.

[159] L. Aksoy and E. O. Gunes, "Area optimization algorithms in high-speed digital FIR filter synthesis," in *Proc. 21st Annu. Symp. on Integr. Circuits Syst. Des.*, Gramado, Brazil, 1-4 Sep. 2008, pp. 64–69.

[160] R. Hashemian, "A new number system for faster multiplication," in *Proc. IEEE Midwest Symp. on Circuits Syst. (MWCAS)*, vol. 2, Ames, IA, 18-21 Aug. 1996, pp. 681–684.

[161] Y. Wang, L. S. DeBrunner, D. Zhou, and V. E. DeBrunner, "A multiplier structure based on a novel real-time CSD recoding," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, New Orleans, LA, 27-30 May 2007, pp. 3195–3198.

[162] A. Hu and A. J. Al-Khalili, "Comparison of constant coefficient multipliers for CSD and booth recoding," in *Proc. 14th Int. Conf. on Microelectron., 2002. ICM 2002.*, Beirut, Lebanon, 11-13 Dec. 2002, pp. 66–69.

[163] K. Johansson, O. Gustafsson, and L. Wanhammar, "Switching activity estimation for shift-and-add based constant multipliers," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Seattle, WA, 18-21 May 2008, pp. 676–679.

[164] K. Johansson, O. Gustafsson, and L. S. DeBrunner, "Estimation of the switching activity in shift-and-add based computations," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Taipei, Taiwan, 24-27 May 2009, pp. 3054–3057.

[165] K. Suzuki, H. Ochi, and S. Kinjo, "A design of FIR filter using CSD with minimum number of registers," in *Proc. IEEE Asia Pacific Conf. on Circuits and Systems*, Seoul, South Korea, 18-21 Nov. 1996, pp. 227–230.

[166] *TSMC 0.18µm Process 1.8-Volt SAGE-XTM Standard Cell Library Databook*, 4th ed. Sunnyvale, CA: Artisan Components Inc., Sep. 2003.

[167] U. Meyer-Bäse, J. Chen, C. H. Chang, and A. G. Dempster, "A comparison of pipelined RAG-n and DA FPGA-based multiplierless filters," in *Proc. IEEE Asia Pacific Conf. on Circuits and Systems (APCCAS)*, Singapore, Dec. 2006, pp. 1555–1558.

[168] M. Kumm, P. Zipf, M. Faust, and C. H. Chang, "Pipelined adder graph optimization for high speed multiple constant multiplication," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Seoul, Korea, 20-23 May. 2012, pp. 49–52.

[169] R. Guo, L. Wang, and L. DeBrunner, "A novel FIR filter implementation using truncated MCM technique," in *Proc. 43rd Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA, 1-4 Nov. 2009, pp. 718–722.

[170] R. Guo, L. DeBrunner, and K. Johansson, "Truncated MCM using pattern modification for FIR filter implementation," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Paris, France, May 30 - Jun. 2 2010, pp. 3881–3884.

[171] K. Okeya, K. Schmidt-Samoa, C. Spahn, and T. Takagi, "Signed binary representations revisited," in *Advances in Cryptology*, ser. Lecture Notes in Computer Science, vol. 3152. Springer Berlin / Heidelberg, 2004, pp. 119–142.

[172] B. Phillips and N. Burgess, "Minimal weight digit set conversions," *IEEE Trans. Computers*, vol. 53, no. 6, pp. 666–677, Jun. 2004.

[173] E. Backenius, E. Säll, and O. Gustafsson, "Bidirectional conversion to minimum signed-digit representation," in *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Island of Kos, Greece, 21-24 May 2006, pp. 2413–2416.

[174] A. Herrfeld and S. Hentschke, "Look-ahead circuit for CSD-code carry determination," *Electronics Letters*, vol. 31, no. 6, pp. 434–435, 1995.

[175] S. Das and M. Pinotti, "Fast VLSI circuits for CSD coding and GNAF coding," *Electronics Letters*, vol. 32, no. 7, pp. 632–634, 1996.

[176] G. A. Ruiz and M. Granda, "Efficient canonic signed digit recoding," *Microelectronics Journal*, vol. 42, no. 9, pp. 1090 – 1097, 2011.

[177] S. Saini and S. Mandalika, "A new bus coding technique to minimize crosstalk in vlsi bus," in *Proc. 3rd Int. Conf. on Electronics Computer Technology (ICECT)*, vol. 1, 8-10 Apr. 2011, pp. 424–428.

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

[179] M. Faust and C. H. Chang, "Analysis and optimization of structural adders in fixed coefficient transposed direct form FIR filters," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2014, in preparation.

[180] P. K. Meher, C. H. Chang, O. Gustafsson, A. P. Vinod, and M. Faust, "Shift-add circuits for constant multiplications," in *Arithmetic Circuits for DSP Applications*. Wiley-IEEE Press, 2014, in press.