ASIC Implementation of a Delayless Acoustic Echo Canceller

ASIC Implementation of a Delayless Acoustic Echo Canceller

Architecture and Arithmetic

Anders Berkeman

Department of Electroscience, Lund University

Department of Electroscience
Lund University
Box 118, S-221 00 Lund
Sweden

This thesis was set in Computer Modern 10,
using the LaTeX Documentation System.

The image featured on the front cover is a micro photograph of the acoustic echo canceller chip presented in paper II.

Printed in Sweden by KFS AB, Lund
November 2002

No. 32
ISSN 1402-8662
ISBN 91-628-5463-1

Abstract

Application specific digital signal processors are superior compared to standard digital signal processors in a number of application fields, mainly due to high throughput and low power consumption traded for flexibility.

This thesis deals with two areas related to hardware implementation of custom digital signal processors: design methodology and efficient implementation of arithmetic circuits. A delayless acoustic echo canceller is chosen as an example algorithm for custom hardware implementation. The canceller algorithm with no signal path delay is suitable in telecommunication applications, and has a high implementation complexity both in number of operations per second and in the variety of signal processing elements it is composed of. The design methodology developed and applied during the echo canceller hardware implementation is presented together with a number of optimizations applicable to the algorithm, architecture, and arithmetic design levels.

The work on digital arithmetic circuits includes efficient implementation of dividers and complex multipliers. A configurable divider architecture for use in a wide range of applications is proposed. The divider is based on digit recurrence algorithms. A parameterized complex multiplier designed for low power consumption and high throughput applications is presented. The multiplier is based on distributed arithmetic, offset binary coding, and adder trees. Furthermore, an arithmetic co-optimization between two algorithms, the fast Fourier transform and the FIR filter, is proposed.

The acoustic echo canceller chip has been fabricated and verified for functionality, throughput, and power consumption.

Contents

Preface

This thesis summarizes most of my work in the Digital ASIC group at the Department of Electroscience, Lund University. Parts of the work has been published or is submitted as the following papers:

A. Berkeman, V. Öwall, and M. Torkelson, "A Low Logic Depth Complex Multiplier using Distributed Arithmetic," *IEEE Journal of Solid-State Circuits,* Vol 35, No. 4, pp. 656–659, Apr. 2000.

A. Berkeman, V. Öwall, and M. Torkelson, "Co-Optimization of FFT and FIR in a Delayless Acoustic Echo Canceller Implementation," in *Proceedings of IEEE International Symposium on Circuits and Systems,* Geneva, Switzerland, May 2000.

A. Berkeman, V. Öwall, and M. Torkelson, "A Prestudy of an Echo Canceller Implementation," in *Proceedings of the International Conference on Signal Processing Applications and Technology,* Orlando FL, USA, Nov. 1999.

A. Berkeman and V. Öwall, "A Configurable Divider using Digit Recurrence," Submitted to *IEEE International Symposium on Circuits and Systems*, 2003.

A. Berkeman and V. Öwall, "Efficient Implementation of an FFT-FIR Structure Using a Distributed Arithmetic Multiplier," Submitted to *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*

A. Berkeman and V. Öwall, "Custom Silicon Implementation of a Delayless Acoustic Echo Canceller Algorithm," in preparation.

During my time at the Department, the following papers have also been presented, but are not considered to be part of the thesis:

A. Berkeman and V. Öwall, "Architectural Tradeoffs for a Custom Implementation of an Acoustic Echo Canceller," in *Proceedings of the Nordic Signal Processing Symposium,* Hurtigruten from Tromsø to Trondheim, Norway, Oct. 2002.

A. Berkeman, V. Öwall, and M. Torkelson, "Implementation Issues for Acoustic Echo Cancellers," in *Proceedings of the 42th Midwest Symposium on Circuits and Systems,* Las Cruses NM, USA, Aug. 1999.

A. Berkeman, V. Öwall, and M. Torkelson, "Implementation of Delayless Echo Cancellers for acoustic echoes," in *Proceedings of RVK Radio Science Conference,* Karlskrona, Sweden, June 1999.

A. Berkeman, V. Öwall, and M. Torkelson, "An Adder Tree Based Complex Multiplier," in *Proceedings of RVK Radio Science Conference,* Karlskrona, Sweden, June 1999.

A. Berkeman, V. Öwall, and M. Torkelson, "A Low Logic Depth Complex Multiplier," in *Proceedings of the 24th IEEE European Solid-State Circuits Conference,* The Hague, The Netherlands, Sept. 1998.

A. Berkeman, V. Öwall, and M. Torkelson, "A Complex Multiplier with Low Logic Depth," in *Proceedings of the 5th IEEE International Conference on Electronics, Circuits, and systems,* Lisbon, Portugal, Sept. 1998.

A. Berkeman, V. Öwall, and M. Torkelson, "A Fast Complex Tree Multiplier using Distributed Arithmetic," in *Proceedings of the NORCHIP Conference,* Tallin, Estonia, Nov. 1997.

A. Berkeman, P. Nilsson, V. Öwall, P. Åström, and M. Torkelson, "A Bit-Serial Implementation of a Wavelet Filter-Bank," in *Proceedings of the Nordic Signal Processing Symposium,* Espoo, Finland, Sept. 1996.

Acknowledgments

Looking back at my time as a graduate student, there are so many people I have met and worked with that have altered the course of my work and life. Trying to mention all of them here is not possible.

Without doubt, I am most grateful to my advisor, Viktor Öwall. Although a busy man, he has always had time for me and my problems, and I have always felt confident with his knowledge and support. During my time at the Department, I have also had the opportunity to travel, and Viktor has shown me many interesting places — from Vesuvio in San Francisco to Clube de Fado in Lisbon — that I will never forget.

To work in the inspiring atmosphere of the Digital ASIC Group has been a pleasure. I am grateful to Peter Nilsson for enlightening discussions about everything from deep submicron silicon technology to carpentry; Mats Torkelson for always being around, giving advice and helping me see things in a different perspective; Shousheng He for giving me a head-first start into the topics of fast Fourier transforms and Distributed Arithmetic. I also want to thank my colleague graduate students, both past and present, and Martin Nilsson, for long nights of discussions about important matters such as the mathematics of juggling and early analog sound synthesizers.

I wish to thank Francky Catthoor for giving me the opportunity to visit the VSDM section of the IMEC Laboratory in Leuven during the fall of 1998.

I am grateful to the technical and administrative staff at the department. Especially Lars Hedenstjerna for always helping me with chip micro photographs and other technical matters; Erik Jonsson for a stable computer environment, at least until he gave up on me and handed me a `root` password of my own; and Britta, Pia, and Elsbieta for helping me out with the paperwork.

Finally, I would like to thank my parents and my sister, for always standing by my side, and Gisela for her constant support and endurance throughout these years.

Abbreviations

ASDSP	Application Specific Digital Signal Processor
ASIC	Application Specific Integrated Circuit
CMOS	Complementary Metal Oxide Semiconductor
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing *or* Digital Signal Processor
DTSE	Data Transfer and Storage Exploration
DVB-T	Terrestrial Digital Video Broadcast
ERLE	Echo Return Loss Enhancement
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HDL	Hardware Description Language
IC	Integrated Circuit
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
ITU	International Telecommunication Union
LMS	Least Mean Squares
LVDS	Low-Voltage Differential Signal

MAC	Multiply and Accumulate
MMSE	Minimum Mean-Squared Error
NLMS	Normalized Least Mean Squares
PCB	Printed Circuit Board
PSTN	Public Switched Telephone Network
RAM	Random Access Memory
RLS	Recursive Least Squares
ROM	Read Only Memory
SIMD	Single Instruction Multiple Data
VLIW	Very Long Instruction Word

Part I

General Introduction

Chapter 1

Motivation

The field of modern digital signal processing applications includes image processing, active suppression of noise or echoes, audio and video processing, telecommunications, medical implants, etcetera. The reason for the popularity is partly due to the advances in silicon device manufacturing technology. Device development makes low cost integrated circuits for digital signal processing algorithm implementations possible. As manufacturing technology evolves, more complex algorithms are realized on silicon, which, in turn, paves the way for new applications and development in already well-established areas. Only two decades ago, a majority of the signal processing applications was performed in the analog domain. Today, signal processing is more or less synonymous with signal processing in the digital domain. Reasons for the popularity of digital signal processing compared to the analog counterpart are, that a digital implementation is [53]:

Capable of being reprogrammed

Capable to handle complicated algorithms

Insensitive to environmental conditions

Insensitive to component tolerances

Predictive and has a repeatable behavior

To solve a digital signal processing problem using silicon technology, the task is expressed as an algorithm. The algorithm is mapped to a target silicon device where it is executed. Silicon devices are either standard off-the-shelf compo-

nents, or Application Specific Integrated Circuits (ASIC[1]). Combinations are also possible. Off-the-shelf components include microprocessors, Digital Signal Processors (DSPs), and reconfigurable logic circuits, for example Field Programmable Gate Arrays (FPGAs). The common denominator of these devices is that they are designed to handle a wide range of applications. The ASIC, on the other hand, is developed for a specific application, giving a less flexible but far more efficient solution.

The number of available components per silicon area unit appears to increase in an exponential fashion over time. Approximately every second year, the number of components on an Integrated Circuit (IC) is doubled. This observation was reported almost four decades ago [59] and is still valid. As integrated circuits are getting more complex, delivering more performance for each generation, a question is why not every digital signal processing application can be implemented on a standard programmable circuit. The answer is that also the development of Digital Signal Processing (DSP) applications and algorithms appear to grow in complexity in an exponential fashion, probably with an even higher rate than the development of circuit technology. For the most computationally demanding applications, an ASIC solution is the only choice. However, within a few circuit generations, today's state-of-the-art applications will be implemented using standard components. ASICs are also used for economic and low power reasons.

Today, the maximum number of components available on single piece of silicon (a "chip") is for most applications not a limiting factor. On one small chip, a huge amount of functionality can be incorporated. However, a number of factors limiting complexity of silicon designs can be identified. Three of the most important factors are:

Design Complexity. As design complexity increases, the problem of managing large projects becomes apparent. These problems have been a subject of research for decades in the software community [23]. However, hardware design has the additional issue of precise signal timing not present in software design. Also, once the hardware is fabricated, it is impossible to do any modifications to the design. For software development, changes can always be made.

Control and Dataflow. To keep the dataflow between memories and computational units constant at a sufficient pace by designing controller units that performs efficient memory addressing. As will be seen in chapter 3.3.2, memories are commonly a limiting factor of digital designs.

[1]The definition of ASIC is an integrated circuit designed for a specific application. However, the acronym is also used to denote the silicon process on which the circuit is fabricated.

Controller design is a tedious work, increasing rapidly with the complexity of the circuits to be controlled.

Verification and Testability. As part of the design process, functionality it added to make sure the circuit can be tested and verified once fabricated. Measuring signals on a fabricated chip is both difficult and expensive, even if the signals are connected to external pads. Adding a number of external connections just for the purpose of testing if a device contains fabrication faults is in most cases not possible, due to increased manufacture cost. With more external connections on the package, packaging cost and size of the target Printed Circuit Board (PCB) where the device will be mounted will be larger. Therefore, more advanced on-chip testing functionality has to be included, like Built-In Self-Test (BIST).

1.1 Implementation Technologies

There are a number of possible implementation target technologies to choose from for mapping a digital signal processing algorithm to a silicon device. Each technology has specific features and drawbacks, and to decide which to choose will include more parameters than covered here, for example time to market, and overall features of the target DSP system. Below are the main device types and technologies:

Microprocessors, ranging from small controllers for embedded systems to high performance processors for use in workstations. The microprocessor is in general optimized to *make the most common case fast* [42], making it flexible and suitable for general purpose computing. Thus, for a specific algorithm, the implementation is sub-optimal, resulting in a relatively expensive and power consuming solution.

Digital Signal Processors (DSPs), is a special set of microprocessors, dedicated to solve the most common DSP problems in an efficient manner. The datapath of a DSP is designed to fit digital signal processing applications. Common features of a DSP include bit-reversed and cyclic addressing, a Multiply and Accumulate (MAC) unit, and special buffer memories.

Reconfigurable logic circuits, are a fast growing set of components with generic logic elements that can implement any logic function. Field Programmable Gate Array (FPGA) technology families range from devices with low to high number of programmable elements, and can be used in a wide range of applications. Modern FPGAs contain microprocessor

cores together with memory instances, and are shipped with extensive input/output capabilities.

Application Specific Integrated Circuits (ASICs), that is, integrated circuits custom designed for specific tasks. The design methodology can be full-custom, giving the ability to control every wire and component on the chip; semi-custom, or standard-cell design, where a design description is synthesized to cells from a cell library; or a combination or standard-cells and full-custom, where the critical parts are designed on a low level.

In this thesis, the terminology Application Specific Digital Signal Processor (ASDSP) is used for an architecture tailored for solving a specific DSP problem. Target implementation technology can be either as ASIC or on a reconfigurable logic device. ASIC technology offers the most design flexibility and performance, and is therefore the target technology for this thesis work.

The main advantage with microprocessors and DSPs is flexibility to solve any computational problem. The flexibility is achieved by having a datapath that can be controlled by a program, consisting of a list of ordered instructions. Such a program can be modified with a short iteration cycle. The limitation of a microprocessor or DSP solution is performance, which is the trade-off for the high flexibility. A non-realtime solution will be the result when performance is insufficient.

Except for the different datapaths of the microprocessor and the DSP, they differ in memory accessing. Microprocessors commonly apply the von Neumann architecture, while DSPs are more or less Harvard architectures [53]. The von Neumann computer reads both data and instructions from one memory. Operations are performed on the data according to the instruction, and computed results are written back to the same memory. This approach is straightforward, and feasible for standard microprocessors designed with general computing in mind.

For digital signal processors, with a datapath tailored for the common multiply and accumulate operation, performance will be limited using the von Neumann memory structure. In fact, it can be shown that four clock cycles are necessary to perform a single MAC operation using the von Neumann architecture [53]. Thus, performance of a tailored datapath capable of executing a MAC in one clock cycle is negligible in comparison to memory accessing. A common solution to increase memory bandwidth is to use the Harvard architecture, which in its native form has separate memories for data and instructions. This device doubles the bandwidth of the processor, being able to read instructions and data in parallel. A common modification of the Harvard architecture is to allow data accesses not only from one, but from both memories. Using a repeat instruction and a singe-instruction buffer memory, two words of data can

be read on every clock cycle throughout the repetition. Memory bandwidth is further increased by additional buffers and special memories with dual accesses per clock cycle [53], designed to decrease the execution time of common DSP algorithms.

Devices based on reconfigurable logic have the capability to map any architecture, if the device is large enough. The device technology is based on small, versatile configurable logic blocks, connected to each other in what can be described as a programmable matrix. This design strategy results in a high flexibility, at the cost of hardware overhead. Every small logic block has a generic structure that will be suitable for a large number of logic or arithmetic operations. A mapping of a specific function to the block will imply that a large part of that block is disconnected, and thus is useless for that application. Furthermore, it is not possible to design an ideal connection matrix, and different connections will differ in latency and load. For performance reasons, the device has to be sufficiently larger than actually required for the target application. Therefore, targeting an ASDSP to a reconfigurable logic device is not optimal in device utilization, which for a specific application is apparent in device cost and power consumption.

Modern FPGA circuits with on-chip dedicated microprocessor kernels, memory cores, and dedicated general purpose arithmetic circuits are removing the borders between processors and reconfigurable logic. Furthermore, there is no strict distinction between general purpose microprocessors and DSPs. There are microprocessors with DSP functionality, and DSPs with a close to general purpose instruction set.

Development of general purpose microprocessors have evolved since the early 1970s, and DSPs and reconfigurable logic since the 1980s — it is impossible for most departments and companies to compete with the huge effort and number of man-years spent in developing these components. With such competitors, it is obvious that an ASIC solution has to have extraordinary features that make it stand out in performance. Comparing application specific hardware to a standard DSP is difficult in general — the result will depend on the algorithm, the manufacture process, and how much time that is spent on the implementation. In [64], typical telecommunications and signal processing algorithm implementations are investigated. It is shown that an ASIC implementation consumes almost three levels of magnitude less energy per operation compared to a high-performance or low-power DSP.

1.2 Application Specific DSPs

The nomenclature Application Specific Digital Signal Processor (ASDSP) is used in this thesis to denote the architecture designed to solve a DSP algorithm. If nothing else is stated, it is assumed that the architecture is implemented in ASIC technology.

The advantage of ASDSP design is the possibility to implement *only* the logic necessary to realize a specific functionality, and design flexibility permits thorough exploration of the design space. For example, the architecture can be a hardware-mapped realization of the algorithm. A Finite Impulse Response (FIR) filter is an example, giving maximum throughput when all taps are computed in parallel. A different approach is to reuse hardware in a time-multiplexed scheme, which results in lower throughput, and smaller area compared to the parallel hardware-mapped implementation.

For ASIC implementation, maximum performance in speed and throughput is achieved by taking special care when designing critical parts. It is clear that the philosophy of tailoring the implementation to the application using ASIC technology gives an advantage compared to implementation on competing target technologies, which are aiming at solving a computation task in general. However, a tailored architecture will lack the flexibility of a general purpose component, and can not be reprogrammed to perform other tasks than designed for.

The ASDSP can be more or less designed to work in conjunction with other processors and devices. It can be a stand alone kernel to solve one dedicated problem autonomously, or it can be a tailored datapath working intimately with a microprocessor. The latter is more flexible, in that it co-operates with a programmable processor, while the first is often more efficient since only hardware necessary to solve the problem has to be implemented. While a standard microprocessor might not have the necessary performance for a digital signal processing application, it is suitable for control and high level protocol handling. Examples of such protocols are Ethernet, JPEG/JFIF, MPEG, etcetera. Therefore, combining a microprocessor with a dedicated hardware accelerator tailored for the specific application is a favorable solution. By giving the custom datapath accelerator a basic configuration interface, it has sufficient performance while maintaining the flexibility of the processor architecture.

To conclude, an ASDSP implementation gives by far the highest performance and room for design creativity. It can be designed to work alone or in conjunction with for example a microprocessor. Although it is dedicated hardware, it can be configurable or programmable to a certain extent. This thesis covers implementation methodology for design of large and efficient ASDSP circuits.

1.3 Overview of the Thesis

The thesis is divided into two parts. Part I is the general introduction, covering motivation, background, related research, and basic theory. Part II is composed of included papers.

An acoustic echo canceller for handheld applications is chosen as a target example where an ASDSP solution is in favor of a solution based on a standard DSP. A modern DSP, as well as an ASDSP solution, will meet the realtime requirements of the application, but the ASDSP will have significantly lower power consumption. Furthermore, the echo canceller algorithm only will use all resources of the standard DSP.

Acoustic echo cancellers are necessary for communication systems where there exists a direct acoustic path from the loudspeaker to the microphone of the communication device. This includes applications such as telephone conferences, hands-free communications, and future applications such as communication using stationary or handheld computers, where earphones, cables, and external microphones will be considered circumstantial.

The chosen echo canceller is a computationally expensive algorithm with no delay in the signal path [60]. A delayless approach is important in telecommunication systems, where the signal is delayed not only from transmission in the air but from all parts of the digital transmission chain such as speech codecs, channel codecs, and so on. A reception of the user's own voice looped and delayed some tenths of milliseconds is considered to be extremely annoying. Therefore, delay of the communication system should be as short as possible, motivating the fact that the chosen algorithm has a higher complexity compared to cancellers adding delay to the signal path.

The introduction ranges over a number of topics. Chapter 2 gives a background to the basic concepts of signal estimation and echo cancellation. Theory regarding efficient realization of filterbanks is also included. In chapter 3, design methodologies for ASDSP design are presented. The focus is on design space exploration, hierarchy, controller design, design flow and standard-cell synthesis, and cache memory strategies. Chapter 4 gives a basic background to Complementary Metal Oxide Semiconductor (CMOS) technology, the concepts of power and energy dissipation, and memory technology. In chapter 5, digital arithmetic for ASDSP design is presented. In particular, arithmetic functions not available in standard processor implementations, distributed arithmetic and offset binary coding, are investigated. Finally, a conclusion and summary of papers is presented in chapter 6.

As part of the thesis work, the echo canceller algorithm is implemented as an Application Specific Digital Signal Processor (ASDSP). The chip has been fabricated and successfully verified. Paper I and II covers algorithm analysis

and implementation of the echo canceller chip. Paper I presents an initial study of what parts of the canceller algorithm that are critical for an implementation, and in paper II, the architecture and optimizations of the implemented design together with measured results of the manufactured chip are presented.

Paper III, IV and V describe efficient implementation of arithmetic components required for the hardware implementation. In paper III, and efficient implementation of a complex multiplier is presented. Paper IV, presents a co-optimization of two algorithms, the fast Fourier transform and the FIR filter, based on a special arithmetic component. In paper V, an efficient configurable divider implementation is proposed.

Chapter 2

Introduction to Echo Cancellation

The problem of acoustic echoes in telecommunication systems arises when a loudspeaker and microphone are positioned such that the microphone picks up the acoustic waves emitted from the loudspeaker, and there is an electrical path from the microphone back to the loudspeaker. Depending on the turnaround time of the system, this is perceived as reverberation, or for longer delays, as an annoying echo [79]. The situation appears in telecommunication systems and sound amplification applications if not proper steps have been taken to prevent it. A related problem is cancellation of line echoes in the Public Switched Telephone Network (PSTN). Line echoes appear in long telephone wires, due to long wave propagation time and impedance mismatch [78]. The line echo problem is easier to solve, since wires are stationary, and the echo path changes little over time, as opposed to the acoustic counterpart.

Typical applications for acoustic echo cancellers include hearing aids, hand-held telephones, hands-free systems, and audio/video teleconferencing systems. For the latter, stereophonic systems are desired to increase perception and sense of a talkers spatial location. A characteristic of telecommunication systems is the rather long system round-trip time; and for devices not depending on head-sets or telephone receivers, a long acoustic impulse response. Since users of such systems are annoyed by listening to their own speech delayed by the round-trip time of the system, the need for acoustic echo cancellers is apparent.

Acoustic echoes can be reduced by sufficient attenuation of the acoustic path between loudspeaker and microphone. Historically, this is achieved by acoustic means, for example by highly directionally sensitive microphones and loud-speakers, or by using headsets providing the required attenuation. For hands-free telecommunication systems, where no attenuation of the signal path is possible, the situation is more complicated. The acoustic path may constantly

change due to variation of the enclosure and location of objects interacting with the sound waves. Therefore, it is not possible to have sufficient attenuation from loudspeaker to microphone — the loudspeaker and microphone can not be arranged in a way that they do not "see" each other. In the past, the problem has been solved by the use of half-duplex channels, allowing the signal to go in only one direction at a time using voice controlled switches. However, controlling these switches is a difficult task and requires highly disciplined speakers [21]. More advanced methods have to be used that can cope with rapidly changing echo paths and the long impulse responses of for example conference rooms.

As indicated, delay is an important aspect in speech communication. The International Telecommunication Union (ITU) restricts the maximum delay of a stationary telephone to 2 ms, and for a mobile telephone to 39 ms [21]. Since a telecommunication system is a complex device consisting of channel coders, speech coders, error correcting devices, and more, the total delay of the communication chain is a sum of all sub-block delays, and the echo canceller is only one of the contributors. Therefore, it is desired to keep the canceller delay as short as possible.

2.1 Typical Echo Canceller Scenario

A typical echo canceller scenario is depicted in figure 2.1. The signal $x(n)$ is assumed to be generated by a system similar to the one in the figure. This other system is denoted "the far end", and accordingly, the system in the figure is "the near end". The signal $x(n)$ is therefore denoted the *far end signal*, feeding the near end loudspeaker. Acoustic waves emitted from the loudspeaker are reflected by the enclosure and objects present within, and noise plus a possible near end talker signal $s(n)$ is added. The signal received by the microphone is denoted $y(n)$, and the acoustic path from $x(n)$ to $y(n)$ can be modeled as a linear time-variant system with a kernel $h(n)$, representing the acoustic impulse response. The shape of $h(n)$ is determined by the enclosure and location of relevant objects for the sound waves to interact with. Using $h(n)$, the function from loudspeaker to microphone can be written as

$$y(n) = h(n) * x(n) + s(n), \tag{2.1}$$

where $*$ denotes a convolution. The idea of the canceller is to suppress the impact of $x(n)$ on $y(n)$, ideally returning only the signal $s(n)$ to the far end. This can be achieved by a filter in parallel to the acoustic path, holding an approximation $\hat{h}(n)$ of the acoustic impulse response $h(n)$. By subtracting the estimate $\hat{y}(n)$ from $y(n)$, ideally the part dependent on $x(n)$ is cancelled out.

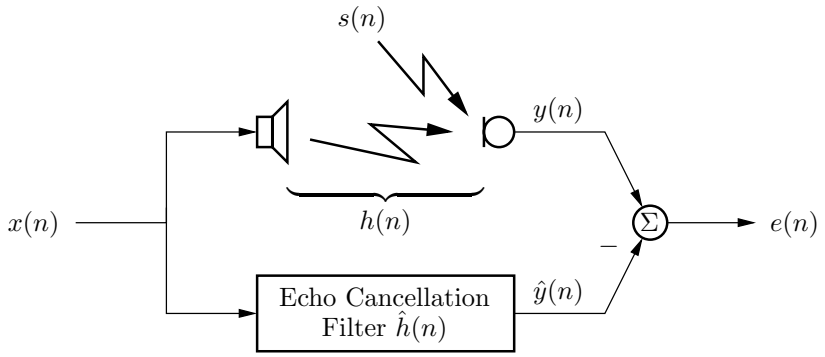


Figure 2.1: Echo canceller scenario. The far end signal $x(n)$ is fed to both the loudspeaker and a cancellation filter. As an acoustic wave, the signal propagates through the acoustic path, where noise and a near end talker signal is added. The acoustic path impulse response is denoted $h(n)$. If the cancellation filter holds an estimate $\hat{h}(n)$ of $h(n)$, a subtraction of the cancellation filter output $\hat{y}(n)$ from the microphone signal $y(n)$ ideally leaves only the near end talker signal $s(n)$ plus noise.

This leaves $s(n)$, which is fed back to the far end. The problem for the echo canceller is then to find the estimate $\hat{h}(n)$. Since sound travels at a speed of about $340\,\mathrm{m/s}$, a sample rate of $16\,\mathrm{kHz}$[1] corresponds to a distance of about $2\,\mathrm{cm}$. Thus, a small change in the length of the acoustic path or shape of the enclosure results in a large change of the impulse response $h(n)$. Therefore, the acoustic impulse response is considered to be time-variant, and an adaptive circuit keeping track of changes in the echo path is necessary.

For sufficient suppression, the length of the adaptive filter should be in the range of the reverberation time of the enclosure, depending on the size of the room, objects within, and what materials it is build of. An empty room with concrete walls, for example, have a rater long reverberation time compared to a furnished room with carpeted floor and wood-paneled walls. Typical values for the size of the adaptive filter, N, is in the range of 1024 to 4096 taps. This corresponds to an acoustic echo of 1/16 to 1/4 seconds at a sample rate of $16\,\mathrm{kHz}$ [79].

[1] The mentioned sample rate of $16\,\mathrm{kHz}$ is a typical value. It gives a relatively high quality, since it is twice the rate of the PSTN, and sufficient to represent the spectrum of human speech.

2.2 Echo cancellers

Echo canceller algorithms are based on estimation of the acoustic path impulse response, and cancellation (removal) of the loudspeaker signal from the microphone signal. There are two requirements for the estimation of the acoustic path to work. First, it is essential that there is a signal emitted from the loudspeaker. Second, estimation can not be performed while there is sound, such as a near end speaker signal, added in the acoustic path. When the near end talker is speaking, estimation must be turned off.

Assuming the far end talker is speaking, which can be easily detected by measuring the power of the loudspeaker signal, there are two cases. The first is that the near end talker is silent and there is no explicit noise within the enclosure. The second condition is the opposite, that is, there is sound that can not be derived from the loudspeaker generated within the enclosure. Under such a condition as the latter, popularly denoted "double-talk", it is not possible for the estimation algorithm to work properly. A double-talk detector has to be used to find these events [38]. The design of double-talk detectors is not within the scope of this thesis.

Echo canceller systems can be built based on estimators and adaptive filters. In section 2.3, theory of signal estimation and adaptive filters is presented. Several configurations are possible concerning both estimation and cancellation. The design space is vast and performance in terms of convergence rate and maximum echo suppression, as well as design cost in terms of silicon area or power consumption, depends strongly on how the algorithm is constructed.

A number of estimation algorithms are theoretically applicable for acoustic echo cancellation purposes [79, 80]. However, in most cases, a simple and robust algorithm outperforms more sophisticated solutions [21]. Although the more sophisticated solution might have better performance in theory, the simple and robust algorithm is easier to get to work in an implementation. Furthermore, the simpler algorithm is easier to analyze, and is associated with a lower implementation cost. Therefore, the Normalized Least Mean Squares (NLMS) algorithm has been selected as the estimation algorithm of choice throughout this work. Performance of the NLMS is sufficient for a reasonable implementation cost while stability and convergence can be guaranteed.

2.2.1 estimation domain

Acoustic impulse response estimation, can be performed either in the time domain, subband domain, or frequency domain. Frequency domain estimation is based on processing of blocks of signals, making estimation delay significant [55, 79], and for that reason not included in the further analysis. A

subband domain is defined by the filters of an analysis filterbank. The outputs of a filterbank are defined on such a domain.

A basic canceller using the (N)LMS estimation algorithm on speech signals in the time domain will result in poor performance. The reason is the strong correlation properties of the speech signal resulting in an ill-conditioned autocorrelation matrix [21]. A way to overcome the problem is to initially decorrelate $x(n)$ and $y(n)$ prior to adaption [21, 79]. This "pre-whitening" can be either adaptive or stationary. Stationary decorrelation filter coefficients have a low implementation cost, but they are constructed using a representative set of speech sequences and has limited performance. Adaptive coefficients are, on the other hand, renewed continuously to conform to the instantaneous properties of the speech signal. Performance is improved, but the adaption comes at a higher cost compared to stationary coefficients. Thus, the choice between adaptive or static decorrelation filters is a trade-off between performance and processing power [21].

Performing estimation in a subband domain is an alternative to time domain estimation. Subband estimation results from splitting the input signals $x(n)$ and $y(n)$ into subbands using analysis filterbanks, and performing estimation independently in each band. Since a subband signal contains only part of the total signal information, the filterbank is downsampled. With a lower sample rate, it is enough for each subband estimator to hold only a fraction of the total length of the corresponding time domain impulse response. In total, the subband solution reduces implementation cost significantly.

Furthermore, rate of convergence for the subband LMS is faster, because the spectral range is greatly reduced in each subband [47, 60]. In practice, this advantage is partially lost due to non-ideal filtering in the filterbanks, for two reasons: there is aliasing between the different subbands, making the subbands "leak" into each other, and the filterbank can not have infinite filter lengths. For an analysis of LMS convergence rate in the subband domain, see [79]. Another issue of the subband approach is that a causal fullband impulse response function transforms into non-causal subband impulse responses [47]. In order to achieve adequate echo suppression, this effect has to be considered. A practical solution is to compensate the non-causal taps by delaying the microphone signal [21]. Therefore, a careful design of the filterbanks is crucial for performance of the canceller algorithm [47]. For design of filterbanks, see for example [33].

Subband estimation is faster and comes at a lower cost, and there are more advantages. Since all subband estimators work independent of each other, they can be controlled independently. The possibility to control each subband is beneficial when for example a double-talk situation occurs, or if the input signal energy is too low for adaption to be reliable. Double-talk and energy

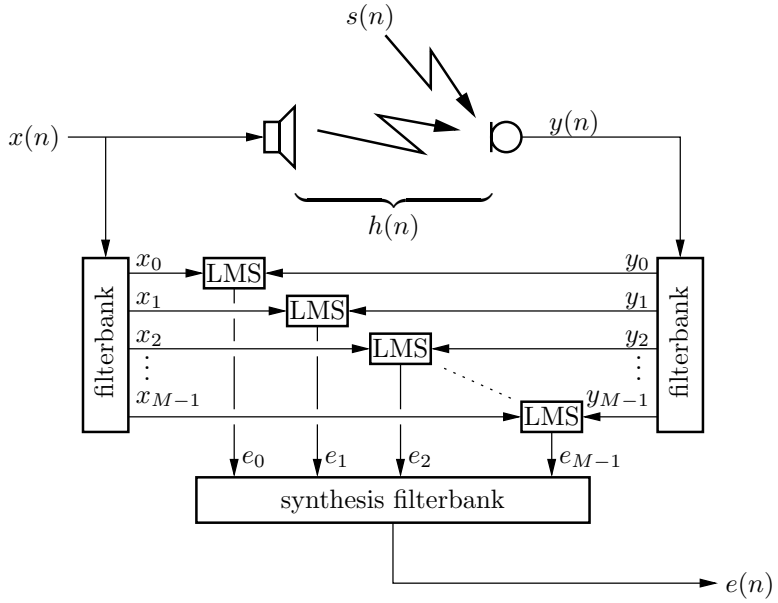


Figure 2.2: Open loop echo canceller using subband decomposition of the near and far end signals. The adaption is optimal in each subband, and not for the complete system, due to the open loop structure. Therefore, the output signal $e(n)$ is a suboptimal solution. In this algorithm, the signal is delayed by the analysis and synthesis filterbanks.

detectors can work independently in each subband, and convergence rate will increase while the total number of operations per sample is lowered.

2.2.2 SUBBAND DOMAIN CANCELLATION

In figure 2.2, a basic structure of an echo canceller with estimation and cancellation in the subband domain is shown [47]. The loudspeaker signal $x(n)$ and the microphone signal $y(n)$ are fed to two analysis subband filterbanks. The filterbanks can be uniform or have any frequency selection desired. For each subband, adaptive LMS filters perform estimation and cancellation independent of each other. Each adaptive LMS filter has a fast and reliable convergence since it is operating on subband filtered signals [79]. Because of the possibility to reduce the sampling rate of the subband signals the adaption within each subband can be performed with shorter filters at a lower rate. This reduces the number of operations per second considerably, compared to a

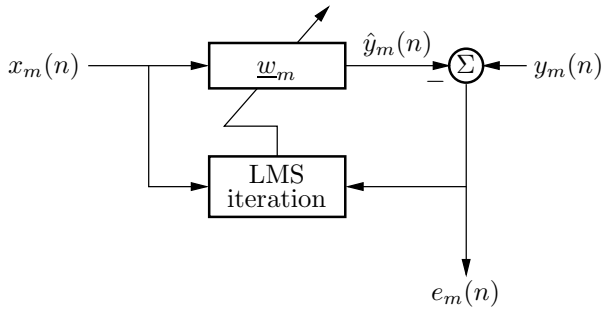


Figure 2.3: Subband LMS adaptive filter. The subscript m indicates the subband version of the loudspeaker, microphone and error signals $x(n)$, $y(n)$ and $e(n)$. The subband part of the fullband acoustic impulse response is here denoted $\underline{w}_n$.

fullband solution [15]. The error signals generated by each subband adaptive LMS filter are assembled into a time domain error signal using a synthesis filterbank. Characteristics of the synthesis filterbank is matched to the analysis filter banks for minimum aliasing and leakage effects [47].

A schematic illustration of a subband LMS adaptive filter is depicted in figure 2.3. Subband versions $x_m(n)$ of $x(n)$ and $y_m(n)$ of $y(n)$ are input to the left and right. Using $x_m(n)$ and the error signal $e_m(n)$, the LMS algorithm estimates the subband portion of the acoustic echo path impulse response. An adaptive filter using the impulse response estimate as filter kernel is fed by the subband loudspeaker signal $x_m(n)$. The output is a signal $\hat{y}_m(n)$, which is the estimate of the subband microphone signal $y_m(n)$. The subband error signal $e_m(n)$ is then calculated as the difference between the subband microphone signal and its estimate. Note that the time scale may be different in the subband domain due to downsampling.

There is no global feedback of the error signal $e(n)$ in the algorithm of figure 2.2. Each subband processor has its own loop. This device leads to a convergence in mean in each subband, and not necessarily to a optimal solution for the whole canceller. A problem with the subband canceller of figure 2.2 is the signal delay associated with the analysis and synthesis filterbanks connected in cascade. For high quality telecommunication applications, this delay might be too long to meet the target system constraints.

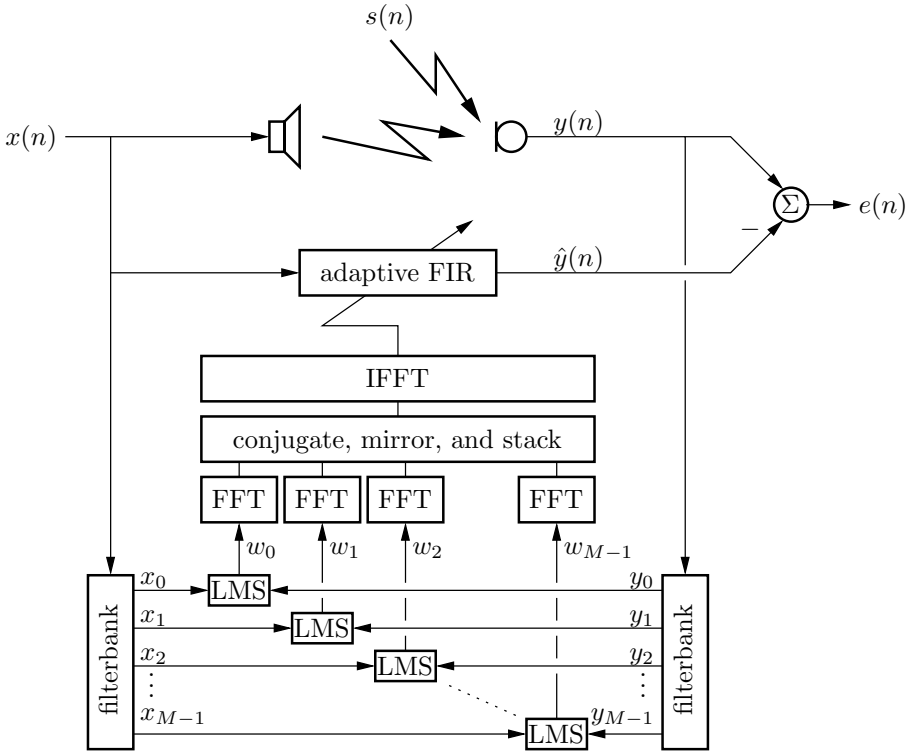


Figure 2.4: Open loop subband canceller with no signal path delay. Estimation in the subband-domain and cancellation in the time domain using a fullband FIR filter. See text for details.

2.2.3 TIME DOMAIN CANCELLATION

The algorithm by Morgan and Thi [60] solves the problem of signal path delay, at a higher implementation cost. The main features are depicted schematically in figure 2.4. An estimate of the impulse response is kept in the fullband adaptive filter. By time-domain filtering and subtraction, there is no delay in the signal path. The fullband filter has a high number of operations per sample compared to the LMS filters, since it holds a complete acoustic impulse response of considerable length, running at full sample rate [15].

The impulse response is estimated in the subband domain, but cancellation is performed in the time domain. An adaptive FIR filter holds the estimate of the time domain acoustic impulse response, and a subtraction removes the

impact of the far end signal to the return signal. This subtraction is the only processing element in the signal path, making the canceller a "zero-delay" echo canceller. As before, there is delay in the analysis filterbanks, and the estimate of the impulse response is delayed correspondingly.

The calculation of the fullband impulse response estimate is distributed to LMS estimators in all subbands, similar to the canceller in figure 2.2. A fullband impulse response estimate is assembled in the frequency domain by a mirror and stacking device [60]. This device operates on frequency transforms of each subband estimate. The output of the assembling box is transformed back to time domain and used as coefficients of the fullband adaptive filter.

The canceller presented in figure 2.4 is an open loop configuration, that is, there is no feedback except for within each subband. Thus, each subband LMS converges in the mean to an optimal result, but the fullband impulse response estimate might be suboptimal. As an alternative, it is also possible to have a closed loop configuration of the canceller by simply feeding back the error signal $e(n)$ instead of the microphone signal $y(n)$. In the long term, a closed loop canceller reaches a higher Echo Return Loss Enhancement (ERLE) and better echo suppression, but has an initial slower convergence rate. In a real application, the acoustic path fluctuates continuously, and it is unlikely to get to a state benefiting from the high suppression rate from the closed loop [60]. Instead, a faster convergence rate is preferred. However, it is possible to design a switch that changes from open to closed loop when appropriate, to get the best from both parts, but the problem of controlling such a switch remains.

2.2.4 HYBRID TIME AND SUBBAND DOMAIN CANCELLATION

The approach of the algorithm by Dörbecker and Vary [34] addresses the high complexity of the time domain FIR filter of the Morgan and Thi algorithm. The idea is to split each subband estimate of the acoustic impulse response into two parts, the early and the late part. The echo canceller algorithm consists of both a time domain and a subband domain canceller, where the early part of the impulse response estimate is used in the time domain canceller. The late part corresponds to the impulse response taps having indexes larger than the delay of the analysis and synthesis filterbanks, and are used to cancel echoes in the subband domain. In total, the canceller is without delay in the signal path. The transformation of the early taps from subband to time domain is implemented through a synthesis filterbank with a composite prototype filter. If the analysis and synthesis filterbanks are well designed to work together, they ideally calculate a lagged version of the early taps of the impulse response.

In reality, a well-designed analysis filterbank has a considerable number of taps, and a correspondingly large delay. A long filterbank delay implies that

a considerable part of the subband domain impulse response has to be transformed to the time domain cancellation filter. Taking into account the special upsampling synthesis filterbank required to compute the early filter taps, while still having a fair fullband FIR filter makes the solution less attractive. Also, for a hardware implementation of the algorithm, both time and subband canceller hardware need to be implemented. This extra amount of hardware is difficult to motivate without a significant gain in performance.

2.3 Theoretical Background

In this section, the LMS and Normalized Least Mean Squares (NLMS) estimation algorithms are derived and compared to another popular estimation algorithm, the Recursive Least Squares (RLS). Initially, properties of speech signals and a cancellation quality measurement are presented.

2.3.1 properties of speech signals

Performing signal processing on speech signals is considered to be difficult [21]. The reason is mainly that speech signals have a high correlation between neighboring samples. Estimation algorithms such as the Least Mean Squares (LMS) suffer from low convergence rate for correlated input [29]. Speech signals have a widely fluctuating envelope, consisting of segments of nearly periodic sequences, noise segments, and pauses. Sample rate of speech signals is typically in the range of 8 kHz in telephone systems to 40 kHz in high-fidelity applications, but even in the case of an 8 kHz sample rate, consecutive samples are highly correlated [21]. In general, algorithms for adaptive filter update suffer from low convergence rate and performance when operating on speech signals, due to the correlation.

2.3.2 quality measurement

A number of quality measurements have been derived for determining performance of an echo canceller algorithm. As such algorithms are designed to work with speech signals, it is important to perform quality tests based on human perception and sense for sound fidelity. For sound processing, as well as image and video processing, there are no good objective quality measurements. The human brain may be sensible to one kind of distortion, but forgiving to some other, and it is difficult to find a measurement capable of telling the difference.

A simple measurement producing exact numbers that can easily be used for comparison purposes is the Echo Return Loss Enhancement (ERLE). ERLE is

expressed in decibels (dB) and defined using the expectation operator $E\{\cdot\}$ [29] as

$$\mathrm{ERLE} = 10 \lg \frac{\mathrm{E}\left\{y^2(n)\right\}}{\mathrm{E}\left\{\left(y(n) - \hat{y}(n)\right)^2\right\}}, \tag{2.2}$$

where $y(n)$ is the microphone signal and $\hat{y}$ is the corresponding estimate generated by the echo canceller. This measure is calculated under the assumption that the signal $y(n)$ received by the microphone only contains signal contribution from the loudspeaker signal. In other words, the loudspeaker is the only signal source within the acoustic enclosure.

2.4 Signal Estimation

The general estimation problem can be stated as follows, given a signal $x(n)$, find an estimate of the desired, or ideal, signal $d(n)$ that is optimal in some sense. The estimate is expressed as a function of $x(n)$. There exists many estimators that solve the problem [29, 80], but the linear Minimum Mean-Squared Error (MMSE) estimator is the most practical one, since it is based on the assumption that the estimate $y(n)$ can be written as

$$y(n) = \sum_i w(i)x(n-i), \tag{2.3}$$

that is, as an Finite Impulse Response (FIR) filter. Such filters are thoroughly analyzed in the literature, and they are stable and easy to implement in software as well as hardware. The filter length and the coefficients $w(i)$ determine the quality of the estimate. In order to track changes in the desired signal, the filter coefficients are updated using an adaptive algorithm.

A basic constellation of an adaptive filtering system is presented in figure 2.5. The signal $y(n)$ is generated from the input signal $x(n)$ by the FIR filter as

$$y(n) = w(n) * x(n). \tag{2.4}$$

The signal $y(n)$ is compared to the input signal $d(n)$, and an error signal $e(n)$ is constructed as the difference

$$e(n) = d(n) - y(n). \tag{2.5}$$

The error signal is used to control the filter coefficients to move in a direction that minimizes the error signal itself. Thereby, matching $y(n)$ to the desired

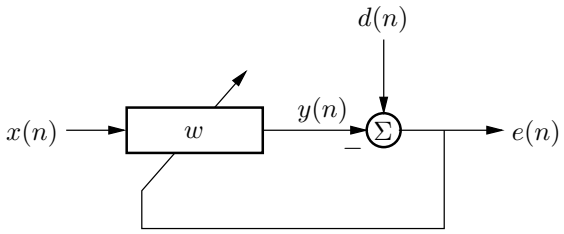


Figure 2.5: Basic adaptive filtering system. Input to the system is $x(n)$ and $d(n)$. The filter generates the output y from input x and filter coefficients w. The error signal $e(n)$ is used to control the filter coefficients.

signal $d(n)$. How the minimization is carried out is determined by an adaption algorithm. The most common adaption algorithms, the Recursive Least Squares (RLS) and Least Mean Squares (LMS), are presented next [29, 85]. Both algorithms are based on iteration formulas. The RLS has a theoretical fast and optimal convergence, but suffers from a high complexity and large requirements on signal dynamics. Therefore, the RLS is mostly used in theoretical simulations, as it is hard to apply in practical applications. The LMS on the other hand is less computationally expensive, but it has slower convergence for non-Gaussian input signals. In order to describe the algorithms, the subject of the *normal equations* is introduced first.

2.4.1 THE NORMAL EQUATIONS

An adaptive circuit optimizing in the least squares sense minimizes the mean energy of the error between desired signal and estimate. The mean energy can be written as

$$J = \mathrm{E}\left\{e^2(n)\right\}. \tag{2.6}$$

The use of the expectation operator is for generality and clarity. For deterministic sequences, J can be defined analogously as

$$J = \sum_n e^2(n), \tag{2.7}$$

and expectation as well as correlations can be expressed using summations. The error signal $e(n)$ is written using equations (2.4) and (2.5) and figure 2.5 as

$$e(n) = d(n) - w(n) * x(n). \tag{2.8}$$

Writing the filter operation explicit, the expression becomes

$$e(n) = d(n) - \sum_i w(i)x(n-i), \tag{2.9}$$

where the summation is over all of the coefficients of the filter w. The idea of adaptive filtering is to design a method to find the w that minimizes J. The optimal solution is found by differentiating J with respect to each filter coefficient $w(i)$ and equating the result to zero,

$$\frac{\partial J}{\partial w(j)} = 2\mathrm{E}\left\{ e(n)\frac{\partial e(n)}{\partial w(j)} \right\} = 0. \tag{2.10}$$

Differentiating equation (2.9) gives the interior derivative in (2.10) as

$$\frac{\partial e(n)}{\partial w(j)} = -x(n-j), \tag{2.11}$$

and thus

$$\frac{\partial J}{\partial w(j)} = -2\mathrm{E}\left\{ e(n)x(n-j) \right\}. \tag{2.12}$$

Taking $e(n)$ from equation (2.9) yields

$$\frac{\partial J}{\partial w(j)} = 2\sum_i \mathrm{E}\left\{ w(i)x(n-i)x(n-j) \right\} - 2\mathrm{E}\left\{ d(n)x(n-j) \right\}. \tag{2.13}$$

By identifying the correlation between d and x as $r(n-j,n)$ and the autocorrelation of x as $R(n-i,n-j)$, and equating to zero results in

$$\sum_i w(i)R(n-i,n-j) - r(n-j,n) = 0. \tag{2.14}$$

This is referred to as the *normal equations*. The solution to these equations results in an optimal filter in the least squares sense. It is convenient to express the normal equations in matrix form. Assuming the length of the causal FIR filter w to be N taps, the indices i and j from equation (2.14) are limited to the set

$$i, j \in \{0, 1, \ldots, N-1\}. \tag{2.15}$$

The filter taps $w(i)$ are then represented by a column vector $\underline{w}$ as

$$\underline{w} = \begin{bmatrix} w(0) & w(1) & \cdots & w(N-1) \end{bmatrix}^{\mathrm{T}}. \tag{2.16}$$

The set of equations in (2.14) is written in matrix notation as

$$\mathbf{R}\underline{w} - \underline{r} = \underline{0}, \tag{2.17}$$

using the following additional definitions

$$\begin{cases} \mathbf{R} &= \text{the } N \times N \text{ autocorrelation matrix with elements} \\ & \quad \mathbf{R}_{ij} = R(n-i, n-j) \\ \\ \underline{r} &= \begin{bmatrix} r(n,n) & r(n-1,n) & \cdots & r(n-N+1,n) \end{bmatrix}^{\mathrm{T}} \end{cases} \tag{2.18}$$

A straightforward solution to the normal equation (2.17) involve inversion of the autocorrelation matrix $\mathbf{R}$. This inversion is complicated by the fact that the matrix is positive semi-definite [29], and thus possibly singular and non-invertible. Furthermore, matrix inversion is a computationally expensive operation to realize in hardware. Partly due to the high number of computations involved, and partly due to the high dynamics required of the values of the matrix, especially if the matrix is close to singular. Since limited dynamics is desired in a custom hardware implementation to reduce power consumption, matrix inversion is in general avoided.

2.4.2 LEAST MEAN SQUARES

The normal equations can be solved by an iterative approach using the steepest descent formula [29]. Let $\underline{w}$ be a column vector of the filter taps $w(i)$ as before. The steepest descent formula is then written as

$$\underline{w}_{n+1} = \underline{w}_n - \frac{\alpha}{2}\nabla J_n, \tag{2.19}$$

where the subscript of w denotes iteration number, that is n and $n+1$, and α is the step-size parameter affecting convergence rate and stability. The steepest descent is a straightforward approach to the minimization problem, and it turns out to be a practical solution. The philosophy can be described as follows: The gradient ∇J_n points in the direction where J_n is increasing the fastest. Therefore, at each iteration, a move in the opposite direction to the gradient with a step size proportional to the magnitude of the gradient is made. The further away from the minimum, the larger the step size. For each iteration, the solution gets closer to the minimum of J.

The calculation of ∇J_n is expensive and can be approximated by a simpler expression for practical use. If ∇J_n is approximated by

$$\nabla \hat{J}_n = \frac{\partial e^2(n)}{\partial \underline{w}_n} \tag{2.20}$$

the result is the LMS filter algorithm. The mean value operation from equation (2.6) or (2.7) is approximated by the instantaneous value. The gradient vector is found from the derivatives of equation (2.12) by removing the expectation operator accordingly. By defining a column vector $\underline{x}_n$ as

$$\underline{x}_n = \left[\begin{array}{cccc} x(n) & x(n-1) & \cdots & x(n-N+1) \end{array} \right]^{\mathrm{T}}, \tag{2.21}$$

the iteration formula is conveniently written as

$$\underline{w}_{n+1} = \underline{w}_n + \alpha e(n) \underline{x}_n. \tag{2.22}$$

Approximation of mean values by instantaneous values results in a simple and practical formula, but by approximating ∇J with $\nabla \hat{J}$, equation 2.22 no longer solves the same problem. For each single iteration, it tries to find the instantaneous solution to the normal equations instead of a solution in the mean.

For non-Gaussian signals, such as speech signals, convergence rate of the LMS algorithm is non-uniform. This is referred to in the literature as the eigenvalue disparity problem, since the elements of the adaptive filter has a convergence rate that depends on the eigenvalue spread of the autocorrelation matrix [29]. Since the eigenvalues are proportional to the variance, or power, of the input signal, the spread can be reduced by normalizing the input signals prior to adaption [79]. The normalized LMS (NLMS) is derived by replacing the factor α by

$$\alpha' = \frac{\alpha}{P_x}. \tag{2.23}$$

In practice, the power is approximated by a time average. A practical approximation is to use

$$\hat{P}_x = \underline{x}_n^T \underline{x}_n, \tag{2.24}$$

since $\underline{x}_n$ is already available in the calculation of the iteration formula. The iteration step of the NLMS-algorithm then becomes

$$\underline{w}_{n+1} = \underline{w}_n + \frac{\alpha'}{\varepsilon + \underline{x}_n^T \underline{x}_n} e(n) \underline{x}_n. \tag{2.25}$$

A small positive number ε is added to the energy before division. This is to avoid the possibility of division by zero, as well as avoiding division by a far to small number. Such a division might lead to a too large quotient to be represented with a fixed point number, causing overflow and accuracy problems.

Furthermore, it is common to make the (N)LMS leaky by multiplying the old estimate by a leakage factor, γ, as

$$\underline{w}_{n+1} = \gamma \underline{w}_n + \frac{\alpha'}{\varepsilon + \underline{x}_n^T \underline{x}_n} e(n) \underline{x}_n. \tag{2.26}$$

The leakage factor γ should be in the range of $0 < \gamma < 1$, but in practice it is set close to one. Leakage forces the estimate to decrease to zero if for example the gradient estimate is constantly zero. It is also advantageous to cancel out impact of iterations in the wrong direction, caused by for example noise or signal errors, or as in the echo canceller application, a present near-end talk signal.

2.4.3 RECURSIVE LEAST SQUARES

The RLS algorithm is derived without the approximations used for the LMS. This makes the solution for the RLS optimal in each iteration. Furthermore, the RLS does not suffer from the eigenvalue disparity problem of the LMS. Instead, convergence is fast and uniform [29].

The basis of the algorithm is recursive calculation of the correlations, and inversion of the correlation matrix. These operations are computationally expensive compared to the LMS. Furthermore, the initial estimate of the autocorrelation is crucial, since an inaccurately initialized matrix adds a bias to all following iterations. Initial conditions have to be derived from the input data characteristics in order to guarantee stability and convergence. Finding adequate characteristics for speech signals is difficult, due to the varying envelope and the segments of periodic sequences, noise, and pauses. In theory though, the bias approaches zero for a converging solution when the number of iterations approach infinity.

As stated earlier, the autocorrelation matrix is ill-conditioned for speech signals. Inversion of such a matrix is difficult without a large precision unavailable for most implementations. Especially fast versions and fixed point implementations are in general difficult to realize in hardware. At least single precision floating point number representation is required for a reliable implementation of the RLS [76]. The RLS suffers from poor tracking ability of non-stationary data due to the equal weighting applied to all previous input data in the correlation calculations. This effect can be reduced by an exponential forgetting-factor similar to the leaky LMS described earlier. However, choosing an appropriate forgetting-factor is difficult, and it also degrades the quality of the cancellation.

To summarize, the RLS implementation suffers from stability problems, high requirements on dynamics, and a high complexity. The LMS is easier to

analyze and implement, and is known to be stable for certain easy-attainable conditions. Thus, for a hardware implementation, the LMS is preferred.

2.5 Polyphase Filterbanks

The number of arithmetic operations necessary to perform a straightforward M-band filterbank operation is limited by $\mathcal{O}(MK)$, if each filter has K taps. For a subband echo canceller with long filters and a large number of subbands, this is a substantial cost of the total complexity [15], especially since a subband canceller require two or three filterbanks.

The polyphase filterbank is an ingenious device to reduce the computational complexity of a filterbank [82]. It is based on a *prototype filter* $\{h(n)\}_{n=0}^{K-1}$. The prototype filter is shifted in the frequency domain and used to filter all subbands. The filter kernel used for subband m can be expressed using the prototype filter $h(n)$ as

$$h_m(n) = h(n)\,\mathrm{e}^{\mathrm{j}2\pi\frac{mn}{M}}. \tag{2.27}$$

This is validated by the time-frequency relation

$$\mathrm{e}^{\mathrm{j}\omega_0 n}x(n) \quad \stackrel{\mathcal{F}}{\longleftrightarrow} \quad X(\omega-\omega_0), \tag{2.28}$$

and setting the frequency translation $\omega_0 = 2\pi m/M$. The filter output at subband m can then be expressed as the convolution $y_m = h_m * x$, or by using summations as

$$y_m(n) = \sum_{k=0}^{K-1} h(k)\,\mathrm{e}^{\mathrm{j}2\pi\frac{mk}{M}}\,x(n-k). \tag{2.29}$$

Assume that M is a factor of the filter length K, that is, K can be written as $K = LM$. Writing equation (2.29) with sums over L and M yields

$$\begin{aligned} y_m(n) &= \sum_{i=0}^{M-1}\sum_{\ell=0}^{L-1} h(i+\ell M)\,\mathrm{e}^{\mathrm{j}2\pi\frac{m(i+\ell M)}{M}}\,x(n-i-\ell M) \\ &= \sum_{i=0}^{M-1} \mathrm{e}^{\mathrm{j}2\pi\frac{mi}{M}} \sum_{\ell=0}^{L-1} h(i+\ell M)\,x(n-i-\ell M). \end{aligned} \tag{2.30}$$

This can be identified as the Inverse Discrete Fourier Transform (IDFT) of the convolution for every Mth sample of x and h. It is convenient to write the

equation on matrix form, by introducing the following symbols

$$\begin{cases} \underline{x}_n &= \left[\begin{array}{cccc} x(n) & x(n-1) & \cdots & x(n-K+1) \end{array}\right]^{\mathrm{T}} \\ & \text{(the latest } K \text{ samples of } x(n), \text{ compare equation (2.21))} \\ \underline{y}(n) &= \left[\begin{array}{cccc} y_0(n) & y_1(n) & \cdots & y_{M-1}(n) \end{array}\right]^{\mathrm{T}} \\ & \text{(one value per subband)} \\ \mathbf{W} &= \text{the Fourier matrix with elements} \\ & \mathbf{W}_{ij} = \mathrm{e}^{-\mathrm{j}2\pi ij/M} \\ \mathbf{H} &= \left[\begin{array}{cccc} \mathbf{H}_0 & \mathbf{H}_1 & \cdots & \mathbf{H}_{L-1} \end{array}\right], \\ & \text{with } \mathbf{H}_i = \mathrm{diag}\left(h(Mi), h(Mi+1), \ldots, h(Mi+M-1)\right) \end{cases}$$

Now, the polyphase filtering operation is expressed using the introduced matrices as

$$\underline{y}(n) = \mathbf{W}^{-1}\mathbf{H}\,\underline{x}_n. \tag{2.31}$$

There are K non-zero position in the $\mathbf{H}$ matrix. Thus, the number of FIR multiplications is reduced from MK to K, multiplication of the Fourier matrix $\mathbf{W}$ not counted. If the Fourier transform is implemented using the radix–2 Fast Fourier Transform (FFT) algorithm, the additional number of multiplications per filter operation is $M/2\ \log_2 M$.

Running the filterbank on every input sample results in an M time increase in generated output sample. No new information is added in the filtering process, therefore, the extra amount of samples are redundant. Combining the filterbank with downsampling is a means to reduce the redundancy, while at the same time lowering the update rate of the signal processing following the filterbank. Critical downsampling is optimal, generating M output samples for every block of M input samples. However, since the prototype filter has finite length, and is non-ideal, there are aliasing effects and leakage between the subband outputs. The downsampling factor is a trade-off between operations per sample after the filterbank, and prototype filter length and dynamics in the filterbank.

An illustration of the downsampled polyphase filterbank is shown in figure 2.6. In 2.6(a), the straightforward approach is depicted. Only every Mth filter output have to be calculated, since the others are discarded by the downsampling. If the filter length is K, this yields MK multiplications every M samples, or K multiplications per sample. Figure 2.6(b) is the corresponding

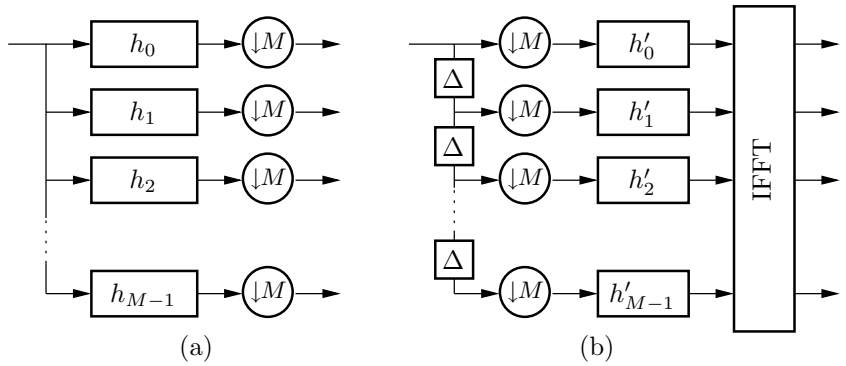


Figure 2.6: Critically downsampled M-band filterbanks. (a) straightforward approach, (b) polyphase implementation. The input-output relation is the same for both implementations, if the filter coefficients are chosen according to equation (2.27). The Δ boxes have an impulse response of $\delta(n-1)$, and delays the signal one sample.

polyphase implementation. Each filter $\{h'_i\}_i$ has only one Mth of the taps of the corresponding filter h. Therefore, complexity is reduced to $K+M/2\ \log_2 M$ multiplications per M samples. This corresponds to only $K/M+1/2\ \log_2 M$ multiplications per sample. The term containing the logarithm is due to the Inverse Fast Fourier Transform (IFFT), if a radix–2 implementation is chosen.

An examination of the subband impulse responses for the polyphase filterbank, equation (2.27), shows that impulse response for band zero and $M/2$ are real valued, and the remaining are complex valued. Therefore, most of the outputs of the filterbank are complex, and subband estimators and cancellers have to be designed for operating on complex numbers.

In the subband echo canceller application, input to the filterbanks are sampled sound signals. Sound signals are real-valued, and the corresponding complex spectrum has a symmetry property along the frequency $f = 1/2$. Therefore, for an M band analysis filterbank, output band $M/2+1$ to $M-1$ does not contain any information not present in band zero to $M/2$. These bands are thus not needed in the subband estimation and cancellation calculations. Furthermore, from an implementation point of view, band $M/2$ can be neglected since it represents the highest sampled frequencies of the input signals. Signals at these frequencies are attenuated in the analog domain prior to sampling in order to avoid aliasing effects.

2.6 Summary

In this chapter, the problem of acoustic echoes in telecommunication systems is presented together with some common solutions. The importance of a low delay in the signal path for telecommunication systems is pointed out. An acoustic echo canceller consists of an estimation part and a cancellation part.

In the estimation part, the acoustic channel is estimated. Estimation is carried out either in the time, subband, or frequency domain. Time domain estimation has a high complexity, and require decorrelation filters to precede the actual estimation. Due to the strong correlation of neighboring samples in speech signals and the lower implementation complexity, subband estimation is preferred.

The cancellation part can also be implemented in time, subband, or frequency domain. Subband cancellation implies that the output signal has to be transformed back to time domain before transmission, and this inevitably introduces delay in the signal path. Frequency domain cancellation is performed block-wise, and thus adds to the signal path delay. For these reasons, cancellation in the time domain is preferred for applications where signal path delay is a crucial design parameter, although the complexity is higher. Doing part cancellation in the time domain, and part in the subband domain is also achievable, but this requires a hardware implementation to have both subband and a time domain cancellers, and is considered to be unnecessarily complicated.

Chapter 3

ASDSP Design Methodology

Design of an Application Specific Digital Signal Processor (ASDSP) is a complex task, ranging from initial problem definition down to structures on a silicon chip. To end up with a working circuit, constraints and limitations on all levels of the design process have to be taken into consideration. Therefore, a thorough design methodology is required.

This chapter summarizes the design methodology applied and developed during development of the acoustic echo canceller chip presented in paper II.

3.1 Design Space Exploration

Given a Digital Signal Processing (DSP) design problem, the first task is to find a solving algorithm. Most likely, there are many candidate algorithms with different complexity and resource requirements, giving different performance of a hardware implementation. Unfortunately, the consequences of the choice of algorithm can not be fully explored without actually doing the implementation and examine the result. For ASDSP implementation on ASIC, iterating the implementation process from initial algorithm to layout is a time-consuming task. Therefore, it is important that as much information as possible is gathered before the implementation phase is initialized. At least quantitative comparisons can be done by analyzing properties having impact on the implementation.

If the application is to run on a generic DSP or microprocessor, the design process is almost complete when the algorithm is determined. In order to execute the algorithm on a processor, it has to be written in a low level language, such as the `C` programming language [50]. The description is then compiled and executed on the target processor, and performance is measured.

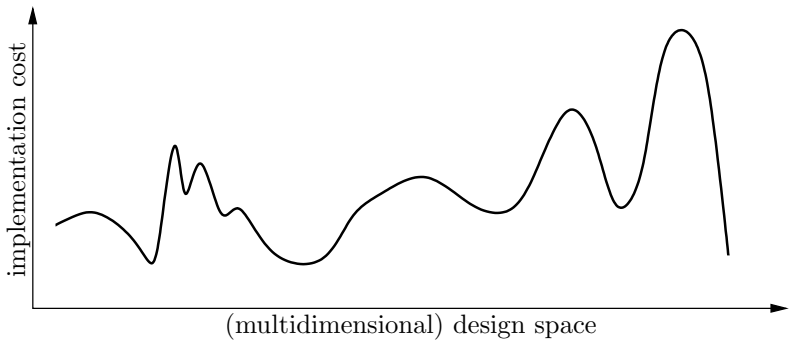


Figure 3.1: The idea of the design space. The graph illustrates implementation cost for a given DSP problem as a function of the (multidimensional) design space.

For implementation on ASIC or reconfigurable hardware, there is the additional step of finding a suitable hardware architecture. The best implementations are found by having the architecture in mind while choosing the algorithm, and optimize the algorithm and architecture together.

In both the DSP and the ASDSP algorithm implementation cases, the algorithm is mapped to a datapath. The complexity of the mapping problem is limited if the datapath is fixed, as for the DSP case. Mapping to the ASDSP, on the other hand, includes design of a tailored datapath, making it a different and more difficult problem. Therefore, it is not certain that the same algorithm is equally suitable for both implementations.

The concept of a design space is a useful analogy to describe the impact of design decisions to design cost. Figure 3.1 illustrates what a design space might look like for a given DSP problem. On the horizontal axis, which is multi-dimensional in reality, different algorithms and possible architectures are plotted. The vertical axis represents the design cost, in area, throughput, power consumption, money, etcetera. The graph illustrates that a small leap on the horizontal axis might imply a gigantic step in cost measure. Therefore, it is important to do a thorough search of the design space prior to making design decisions, if possible. It is of course not practicable to cover the complete space, but significant subsets of the space have to be identified and explored. It is probably so that a more experienced designer is capable of doing a better search than the novice. Design automation tools help calculating properties for a solution, but can not compete with the human brain when it comes to creative exploration of the design space.

3.2 Hierarchy

A powerful method to handle large and complex designs is to apply the divide and conquerer approach, expanding the design into a hierarchy of subdesigns. On the highest level of the hierarchy, the top level, interfaces to surrounding devices and instantiations of the main submodules appear. Each submodule, in turn, may contain instantiated submodules from a further lower level. Dividing the design into a hierarchy has several benefits:

> It is a comprehensible view of the design, where it is easy to focus on the relevant parts.
>
> The design process can evolve top-down, bottom-up, or as a mix of both. Designing top-down makes it possible to create an initial high level description of the complete circuit for system simulation. By designing submodules bottom-up, it is possible to verify and debug each submodule before instantiation in the real design.
>
> Typical synthesis tools work faster on a design split into submodules.
>
> If the interfaces between submodules are specified, several designers can contribute to the same design, since there are tight borders between the different modules, separating one sub-block from the other.

In the methodology used for this thesis work, the design is split into a hierarchy of autonomous submodules, where each submodule executes a part of the total algorithm, a subalgorithm. A submodule is then a hardware realization of a subalgorithm. Each submodule has its own controller — processing is carried out independent of other submodules. Thereby, communication between a module and its lower level submodules becomes limited. This reduces the complexity of the implementation task, since less time has to be spent on inter-module communication and design of large complicated controllers.

The definition of a subalgorithm has to be elaborated. A multiplication operation is in most cases implemented as a combinational hardware multiplier unit, not depending on any controller whatsoever. The multiplication is considered to be too simple to be a subalgorithm. An Finite Impulse Response (FIR) filter on the other hand, based on iterative use of a multiplier and an accumulator (Multiply and Accumulate, MAC) for several clock cycles, requires some control circuit, and is a typical example of a subalgorithm.

A more advanced algorithm, such as the Normalized Least Mean Squares (NLMS) filter, can be divided into several subalgorithms, or hardware submodules: an FIR filter, a coefficient update unit, an energy calculator, and a divider. The four hardware submodules have one controller each, and the

NLMS unit has a controller of its own, running the four submodules in turn. Furthermore, the NLMS is a subalgorithm of a complete echo canceller, for example. Exploiting this hierarchy, a tree of autonomous function blocks is derived, and the resulting hierarchy reveals a natural way for the controllers to communicate to each other.

In the case of the echo canceller implementation, submodules operate on chunks of data. Therefore, intermediate storage of data is required before and after most of the submodules. For these reasons, memory buses are a natural way for data transfer between submodules. An efficient implementation strategy for sharing memories have been developed, see paper II.

3.3 Datapath, Memory, and Controller

An ASDSP hardware implementation of a DSP algorithm is composed of a number of hardware units, each belonging to one of three[1] main disjoint sets, based on functionality:

Datapath

Memory

Controller

A datapath performs the actual arithmetic or logic operations of the algorithm implementation. The datapath can be streamlined for one particular arithmetic operation in the algorithm, or designed to compute a set of different operations, depending on how it is controlled. Datapath building blocks are arithmetic and logic computational units and pipeline registers for storage and performance purposes.

Most algorithms entail intermediate storage of data. Data storage is implemented using instances of Random Access Memory (RAM) or banks of flip-flops. There are limitations on access time, bandwidth, and latency for the RAM memories, that have to be taken into consideration when designing a datapath.

Since datapaths are data processors only, and memories are locations of storage, some kind of controlling hardware is required. A controller manages read and write data accesses to the memories, making sure the datapaths execute the correct operation on the accurate data at the right time. The controller might be either externally programmable or hardwired for a specific behavior.

[1] External interfaces is a fourth set, not covered by this thesis.

Depending on the complexity and required throughput of a hardware realization of an algorithm, it is said to be either control-dominated or data-dominated. Typical properties of a control-dominated implementation is datapath hardware reuse and storage of large data sets. The controller is responsible for transferring data between intermediate storage and the datapaths, and making the datapath execute the correct operations. A data-dominated hardware realization typically consists of a multistage pipeline datapath, operating on high speed datastreams.

3.3.1 DATAPATH

The computational operations of a DSP algorithm implementation is performed in datapaths. Here, arithmetic and logic operations are executed. An implementation may contain only one datapath, but it is common to have several, depending on the complexity of the implemented algorithm and the design constraints. Data is fed to a datapath from either another datapath, a memory, or from an off-chip device. The datapath itself operates independent of what kind of data it is or from where it originates. A controller is working in conjunction with memories and datapaths, and is responsible for data scheduling between the units. However, it is possible to design a datapath with memory addressing capabilities, if it brings special advantages.

For datapath design, the design space is vast. In one end of the space is the hardware-mapped datapath. If the algorithm is expressed as a signal-flow graph [31], composed of basic hardware realizable arithmetic functions and interconnections, the hardware-mapped datapath is the one-to-one mapping of the graph to hardware units and wires. In this scheme, every operation in the algorithm is implemented using a unique hardware unit. Control requirements for such a datapath are minimal.

In the other end of the design space is the generic datapath, as used in general purpose microprocessors and DSPs. This datapath is designed to perform all possible operations present in the algorithm. Such a design decision results in a programmable solution, with a wide application area. Data to and from the datapath does for most applications need intermediate storage. Most likely, there will be a memory bottleneck as there is in the Harvard or von Neumann schemes discussed in section 1.1. Therefore, the flexibility added results in a lower performance.

For a given DSP algorithm, such as an FIR filter, there is no point in implementing it using a general purpose datapath, unless the datapath is shared with some other algorithm in the design. For a tailored FIR filter datapath, there is the option of doing either a hardware-mapped or time-multiplexed implementation.

The N tap FIR will be implemented using N multipliers for a hardware-mapped realization. With proper pipelining [69], such a parallel solution results in high throughput. A time-multiplexed implementation, on the other hand, is composed of only one multiplier instance. This multiplier is used to execute all FIR multiplications of the filter in a sequential manner. Throughput will be lower, but the realization is more compact in size. In-between these two realizations, there is a number of hybrid parallel/time-multiplexed solutions, all with different performance and design cost. Which solution to choose depends on the surrounding circuits and the required performance of the implementation.

3.3.2 MEMORY

The use of on-chip intermediate data storage significantly increases the set of algorithms possible to implement on a single chip. However, it is important to understand the restrictions and limitations of on-chip memories to an AS-DSP design in order to achieve an efficient implementation. Typically, on-chip memories occupy 50% of the chip area and about half of the power consumption [25]. Thus, efficient handling of memory instances is a powerful means to achieve high performance.

Memory for intermediate storage can be either implemented as a RAM or, for smaller entities, using flip-flops. Memory for storage of constants can be implemented using random logic, Read Only Memory (ROM), or FLASH techniques. This section deals with problems associated with random access memory. CMOS memory technology is the subject of section 4.2.

On-chip RAM has either asynchronous or synchronous access operation. The synchronous memory behaves similarly to the common edge-trigged flip-flop, and is preferred in a synchronous design methodology. The asynchronous memory, on the other hand, can be converted to a synchronous access behavior by adding external flip-flops. Thus, if a design is based on synchronous memories, it can be adopted to work with asynchronous memories as well, whereas the other way around is not true. This is important when designing for portability, since different silicon device vendors use different memory generators.

An on-chip memory can have more than one interface port, allowing parallel access to different locations of the memory simultaneously. This is a convenient feature, since the data rate is multiplied while the data maintains local into one memory instance. Again, for a design to remain portable, it is safe to rely on single port memories only, since multiport devices are not available in all processes. Furthermore, the area of a memory with two interface ports is larger than the single port counterpart. For the memory generator used for the echo canceller chip, the dual port memory is significantly larger, as shown in [52].

In figure 3.2, a graph of the area versus storage capacity for three memory

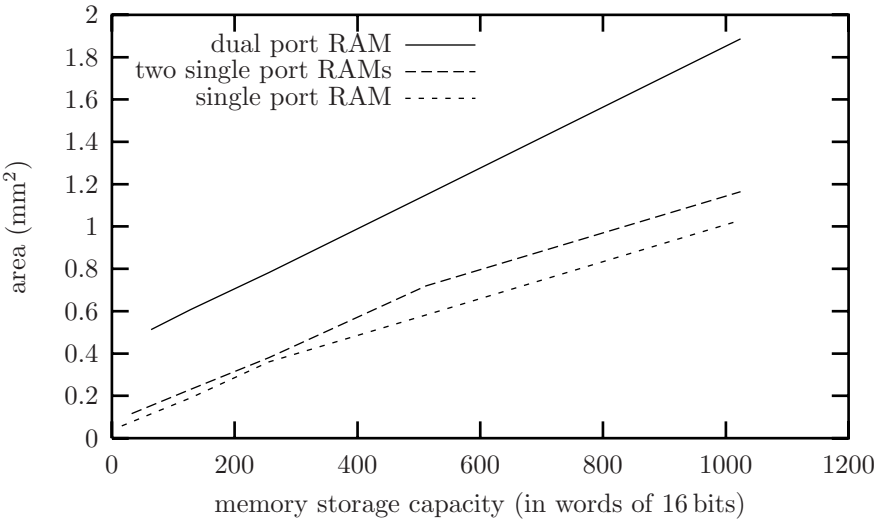


Figure 3.2: Memory area versus storage capacity for different implementation strategies.

configurations, generated using the same generator as in [5, 52], is shown [1]. The bottom dotted line is the reference static single port memory. The other two lines represent memory configurations with higher bandwidth. The top solid line is a dual port synchronous memory instantiated using the same memory generator family. As shown, the dual port memory has at least twice the area for the same storage capacity. The middle line is calculated by replacing the single port memory with two half-sized single port memories. This efficiently doubles the memory bandwidth, although the division of address space makes it impracticable for some applications. The double-memory approach is only slightly larger (approximately 20%) than using only one single port memory, and much smaller compared to the dual port alternative. Thus, the dual port memory should be avoided if possible.

Memory access diagrams for a single port synchronous RAM are shown in figure 3.3. It is assumed that all synchronous transactions appear on the rising edge of the clock. To perform a read operation, the synchronous operation of the memory requires the address to be latched on a rising edge before data lookup takes place. The next rising clock edge, output data is latched by the datapath to be used for calculations the next clock cycle. As the output data lags the input address by one clock cycle for the read access, a useless turnaround cycle has to be added when switching from write to read opera-

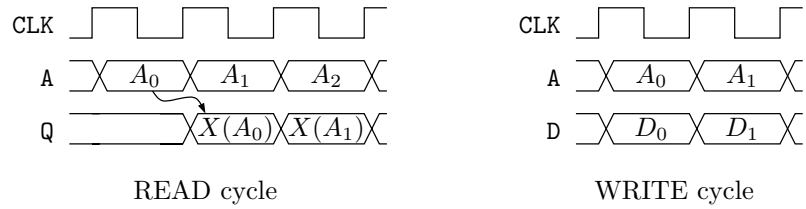


Figure 3.3: RAM access waveform for read and write operations.

tion. It is common that the data output at the turnaround cycle is the same as was written the previous clock cycle, but it depends on the implementation, and should not be counted on. As single-port RAM only has capacity to either perform a read *or* a write operation each clock cycle, the cost of this turnaround cycle is evident in algorithms with an extensive use of read-modify-write operations, such as the Least Mean Squares (LMS) or the Fast Fourier Transform (FFT). To maximize the utilization of the hardware, the implementation has to be design so that the amount of stall cycles are kept to a minimum.

3.3.3 CONTROLLER

The controller is responsible for data transfers between memories and datapaths, as well as making sure that datapaths with control input signals execute the correct operations on the data.

By making sure that only controllers can do memory accesses — and datapaths can not — simplifies the design process. The division into dedicated controllers, datapaths, and memories speeds up development and verification. Breaking the boundaries is allowed if it can be motivated for example by increased performance or simpler control.

Since digital signal processing is used in for a wide range of applications, there is a large variation in the complexity of a controller. Some applications are computationally extensive with high throughput datapaths and simple control requirements, while others have a time-shared datapath and rely on complicated control structures. The span ranges from autonomous massively pipelined computational units to the microprocessor architecture, with general datapath and versatile control capabilities.

A controller behaves similar to a Finite State Machine (FSM). For every time instance, it has a well-defined state, and state transitions are scheduled to the next time instance depending on certain input signal conditions. The outputs from the controller depend on the state and possibly on the input

signals. If the outputs depend on the current state only, it is a Moore machine, otherwise it is a Mealy machine.

It is common to model behavior of a controller as an FSM description. Controller descriptions based on FSMs grow rapidly in complexity for larger control tasks. For N states and M input signals, there are $(N^2 - N)/2$ possible transitions, each with 2^M possible input signal conditions. So, the number of possible transitions grow with $\mathcal{O}(N^2)$, each with exponential input signal condition complexity. Even if the practically used transitions only grow in, say, a linear fashion, a large FSM is difficult to design, verify, and maintain. A large design may have hundreds or thousands of control signals to be read and written at every clock cycle for sustained operation. Thus, using a single controller for a large design results in complicated problems.

As a means to reduce the number of control signals, an instruction decoder can be inserted between the controller and the datapath. The datapath is then programmed using Very Long Instruction Word (VLIW) instructions, similar to some standard processors. A more complex instruction decoder result in shorter instructions and less flexibility, while a wider instruction word allows more versatile operation at the cost of managing longer instruction words.

Another solution to the problem of large and complex controllers has already been mentioned — the divide and conquerer approach. By dividing the controller into a number of disjoint subcontrollers, each tailored to a specific subpart of the algorithm, control design becomes more manageable. The drawback with the divide and conquerer method is that inter-controller communications becomes increasingly difficult with more controller units. The inter-controller communication also grows as $\mathcal{O}(N^2)$, and fatal situations like deadlock [74] may occur. The phenomenon of deadlock appears for example when two communicating units both expect the other to respond to some message or situation, and execution stalls.

An easier way to describe controller behavior compared to the FSM description, is to use some kind of description language, with well-known sequential statements such as:

`if(` *expression* `)` *statement*`;`

`while(` *condition* `)` *statement*`;`

`for(` *initialization*`;`*condition*`;`*increment* `)` *statement*`;`

Such high level descriptions are easier to write, debug, and maintain. LAGER is a tool for automated control and datapath generation and has user interfaces on several design levels [77].

A tool focused on controller synthesis given knowledge about the datapath is an efficient means to design and maintain controllers [67]. The controller

synthesis tool parses the high level input control program and automatically generates a controller unit, given knowledge of datapath control signals. By this method, the design focus is on the behavior of the controller — and not the actual implementation, which is less relevant.

3.4 Synthesis and Standard-Cell Libraries

To have a hardware solution with the lowest cost in terms of clock speed, area, and power consumption, optimization on all design levels has to be considered [27]. In most cases, time-to-market is an important design parameter, that is, if the design process takes too long the solution might not be competitive. For that reason, it is common that design methodologies does not employ as thorough optimization on all design levels. High level design tools pay little attention to the lower levels. This is reasonable, however. For multi-million transistor designs, as are common today, it is not possible to do individual transistor sizing and layout.

Clearly, not all parts of a circuit are equally critical. Therefore, it is a waste of time and resources to do too much optimization on the layout level. To decrease design time, non critical parts, or even complete designs, are designed on a higher design level. This is commonly done by relying on a *cell library*. The cell library is a set of small circuits, to which a design description is mapped. Typical circuits include various logic gates and flip-flops, but it may also contain higher level components, such as simple arithmetic functions.

A design is mapped to a cell-library using a *synthesis tool*. Input to the tool is a description of the design in a Hardware Description Language (HDL), for example VHDL [18] and Verilog [19]. VHDL has the most functionality, making it more flexible. On the other hand, a more advanced language may be slower in simulation, and a bit detailed. Only a minority of the constructs of VHDL are actually valid for synthesis.

Which language to choose is a matter of taste — a successful design methodology does not depend on the chosen language, but on *how* it is used. Describing a large design in a HDL requires disciplined design entry. Extensive use of hierarchy simplifies both design description and code comprehensibility, while improving performance of logic optimization tools.

Hardware implementations will loose performance by the synthesis method compared to the full-custom technique. The designer has to rely on the efficiency of the components in the cell library, and that the synthesis tool is capable of handling low level design issues such as load matching and wire length calculations. For performance critical applications, synthesis to a standard cell-library may not fulfill the design constraints. In these cases, either

the parts of the design that limit performance have to be re-designed using full-custom techniques, or a different design decision at a higher design level has to be taken. Identification of the critical part is important. Time consuming full-custom techniques should only be performed where necessary, as design, simulation, and verification procedures are far more complicated and time-consuming for full-custom compared to a standard cell synthesis approach.

Today, a number of high-level design tools and libraries are available. With these tools, the design is described in terms of high-level objects, such as FIR filters. It is tempting to design on this level, not having to care about logic style or finite state machines. For a design with loose constraints, it may work satisfactory, and save design time. Unfortunately, for critical circuits, a more detailed control of the design is necessary.

The methodology used in this work is based on standard VHDL, making it independent of any specific synthesis or design automation tools. The code is written on a level where logic and arithmetic operations are separated from sequential functions. The use of a HDL allows the logic, arithmetic, and sequential units to be written on a behavioral level. A division into logic and sequential units makes the design easier to verify and maintain, since all time-depending functionality is located in the sequential part of the code.

3.5 Example: FFT Architecture Exploration

In this section, an example of an architecture level design space exploration is performed. The well-known radix–2 Fast Fourier Transform (FFT) algorithm [20] is chosen as subject for the exploration. As part of the echo canceller project, an 1024-point FFT testchip has been fabricated, see figure 3.4. Design of the testchip initiated investigation of efficient FFT architectures.

Maximum throughput implementations of the FFT are based on pipelined datapaths and/or systolic array architectures. A pipelined FFT processor has a datapath consisting of several butterfly processor instances, one for every butterfly stage, resulting in a high utilization at the cost of a large area. For the radix–2^2 algorithm [40], based on a "radix–4 architecture with radix–2 butterflies", multiplier utilization is 75%, and an N bin FFT is computed in only N clock cycles. Performance of this pipelined FFT is thus $\mathcal{O}(N)$. Though it is possible, for most applications there is little use in going faster. For small N, high throughput FFTs can be realized using signal-flow graph hardware mapping.

For implementations with lower requirement on throughput, such as the adaptive filter update unit in the echo canceller, the datapath can be re-used, or time-multiplexed, to save area. Six possible time-shared schemes based on

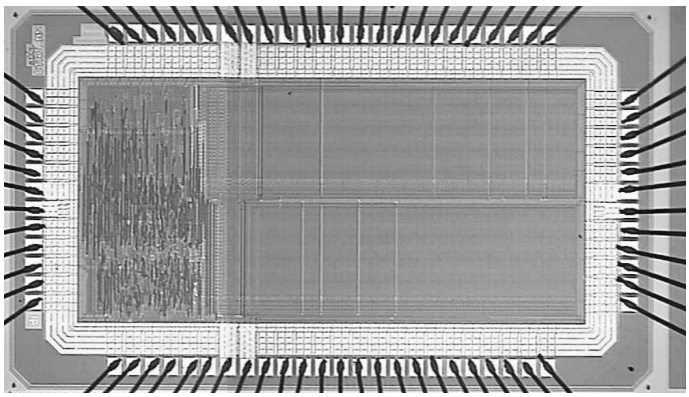


Figure 3.4: FFT testchip. The chip performs the fast Fourier transform on data sets of size 1024, with a wordlength of 12+12 real and imaginary bits.

the use of synchronous RAMs are presented and evaluated in this section. They all use the same datapath, a radix–2 butterfly, as depicted in figure 3.6. Inputs to the butterfly are $X(s)$, $X(t)$, and a twiddle factor W. The butterfly datapath computes one addition, one subtraction, and one multiplication per operation. In fact, the multiplication is a rotation, since the twiddle factor is a root of unity. The indices s and t refer to locations of data in a data set. Thus, data is read, altered by the butterfly operation, and written back to the original locations. To compute one FFT operation using a single butterfly instance, a number of butterfly operations have to be performed iteratively on the input data set. For a size-N FFT, $\log_2 N$ stages of $N/2$ butterflies each is required. In figure 3.5, an eight bin radix–2 Fast Fourier Transform (FFT) is depicted. It consists of twelve butterflies connected in three stages, that depend on each other.

A straightforward time-shared radix–2 FFT architecture is based on one memory instance and one butterfly datapath, as shown in figure 3.7. This is the reference architecture denoted architecture A. The box "Radix–2 Butterfly" is the structure from figure 3.6.

The butterfly datapath is combinational, and surrounded by pipeline registers for performance and power consumption reasons. Registers marked with an "E" have an enable signal. The content of the E-register is only updated when the enable signal is active, otherwise it keeps its old value. The datapath has two inputs and two outputs, twiddle factor not counted. The twiddle factors are stored in a ROM, which for clarity is not shown in the pictures.

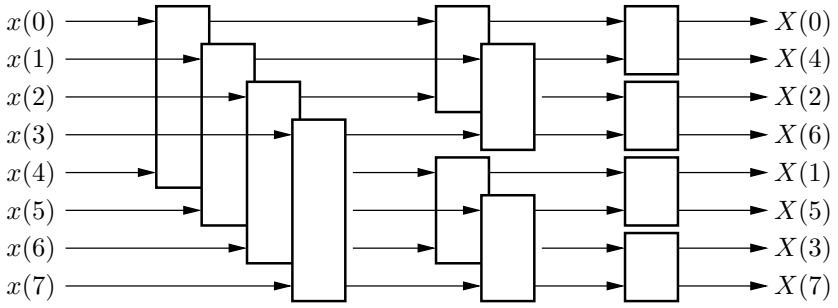


Figure 3.5: Eight-bin radix–2 FFT.

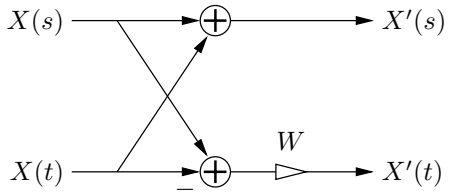


Figure 3.6: A radix–2 butterfly datapath. One butterfly operation requires six clock cycles. The datapath is used during the cycle marked "Calc" only.

At maximum pace, the datapath inputs two words and outputs two words of data each clock cycle. Since the memory can either read or write one word per clock cycle, the datapath has four times the possible throughput of the memory. Furthermore, as one clock cycle is required to perform the butterfly operation, and one clock cycle is lost due to read/write turnaround, the single-memory FFT requires a total of six clock cycles per butterfly, see the waveform diagram in figure 3.8.

An obvious drawback with architecture A is that memory and datapath operate in a serial fashion. By adding two more buffer registers, operation is parallelized. This architecture, denoted B, is shown in figure 3.9, and the corresponding waveform in 3.10. Note the analogy to software pipelining [42]. With this architecture, the theoretical lower bound is reached, since a butterfly operation is performed on average every fourth clock cycle, using a datapath instance having fourfold the throughput of the memory. In order to have a faster architecture, memories with higher throughput are required. Thus, when reaching the optimal solution, memory throughput is the limiting factor.

In order to increase throughput using single port memories, parallel memory

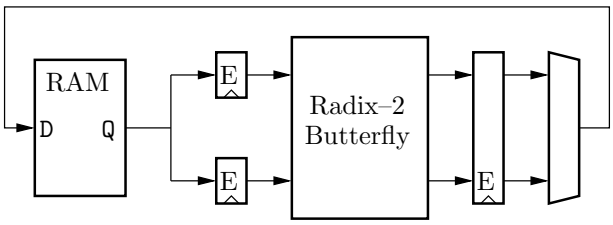


Figure 3.7: Architecture A, radix–2 FFT. The twiddle factor ROM is hidden in the butterfly datapath.

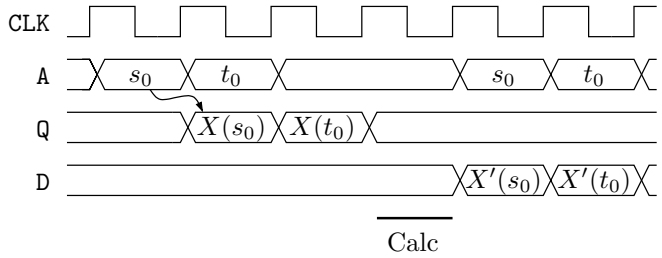


Figure 3.8: Waveform for architecture A.

units have to be instantiated. Using two instances each with half the original storage capacity results in two new architectures, A′ and B′. The idea is that an N bin radix–2 FFT can be split into two $N/2$-sized FFTs plus a pre or post stage. This follows directly from the derivation of the radix–2 FFT [20]. Data for the two smaller FFTs is stored disjointly in the two memories, and the transforms are calculated independently. Due to the low utilization of the datapath, the two $N/2$-sized FFTs can be skewed a few clock cycles apart, and use the same butterfly datapath without collision.

A better way to utilize parallel memories for FFT calculation is to use Cohen's scheme [30]. It is based on the observation by Pease that s and t always differ in parity [70]. Therefore, $x(s)$ and $X(t)$ can be stored in different memories that are accessed in parallel. More advanced schemes to reduce power consumption by further separating the memories into source and destination banks have been proposed, see for example [54].

Two architectures, C and C′, based on the observation by Pease [70], are depicted in figure 3.11. The difference between the architectures is that C′ has an extra set of flip-flops in the datapath. Data associated with odd-parity bins are stored in one memory, while data with even-parity address is stored in the

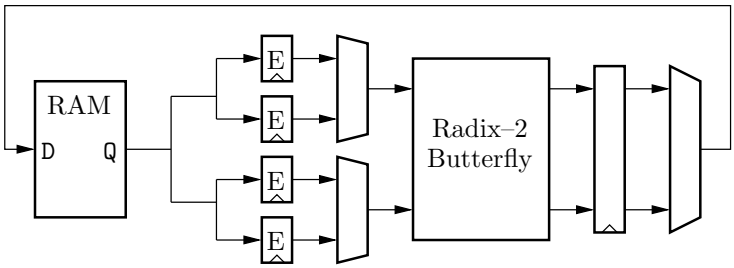


Figure 3.9: Architecture B. By adding two registers for intermediate storage, throughput can be increased by 50%.

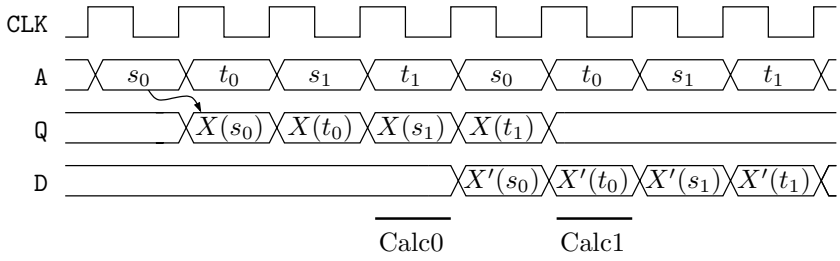


Figure 3.10: Waveform for architecture B. Two butterfly operations are executed in eight clock cycles.

other. Depending on the parity of s, the contents of the odd or even memory should be connected to the top or bottom input of the butterfly. The switching of the input signals is implemented as two switches, each composed of double multiplexers, at the input and output of the memories.

The number of clock cycles for a read, compute, and write back operation is three clock cycles, including synchronous memory and pipeline flip-flops. The throughput of the architecture is improved by retiming [69] the operation to four clock cycles by extending the path by an extra set of registers. Retiming results in the final architecture, C′ to be covered in this section. A wave graph for architecture C′ is shown in figure 3.12.

To summarize, the number of clock cycles for a given architecture is shown in table 3.1. The performance differs a factor three between the worst (A) and best (C′) architecture. It might be surprising to see that the initial architecture based on Cohen's scheme (C) is not as fast as B′. However, it is fair to compare B′ to the retimed version using Cohen's scheme (C′), where the latter

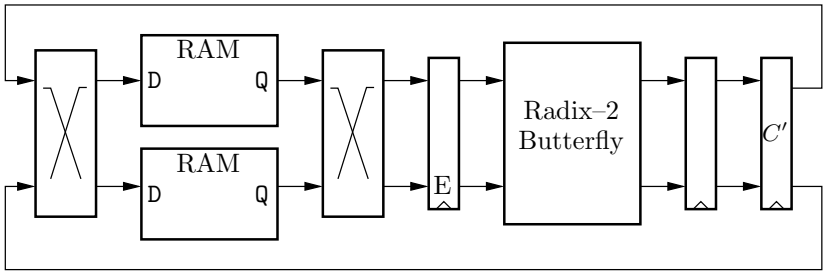


Figure 3.11: Architectures C and C′. The architecture C lacks the extra flip-flop at the output of the datapath, only present in Cprim. For the C′ architecture, the location of the flip-flop could be anywhere in the datapath.

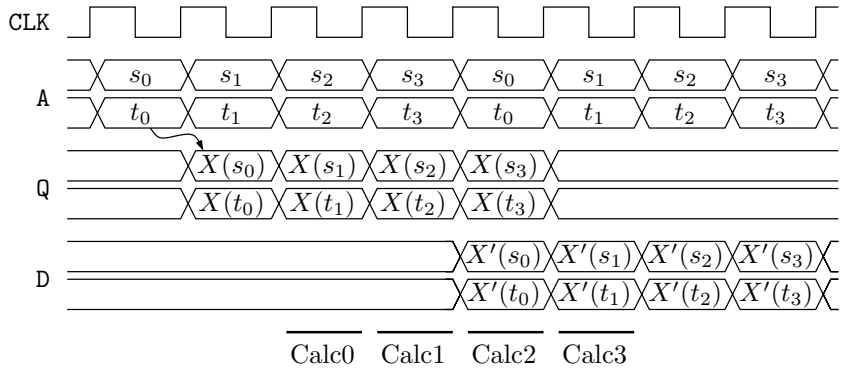


Figure 3.12: Waveform for architecture C′.

is the fastest.

It is interesting to note what happens when migrating from radix–2 to radix–4 using a time-shared architecture. The number of butterfly computations decrease from $N/2 \log_2 N$ to $N/4 \log_4 N$, or to one fourth. This indicates that a time-shared radix–4 architecture will be four times faster than its radix–2 counterpart, since the architecture is based on iterative use of one butterfly datapath. However, the radix–4 butterfly datapath requires a throughput of four reads and four writes per operation, that is, at least eight memory accesses per butterfly, whereas the radix–2 architecture required only four. As the radix–4 butterfly requires twice the number of memory accesses, total in-

Table 3.1: Clock cycles for an N bin FFT for architectures A to C′. A and B are based on one N-word memory instance, while B to C′ have two $N/2$ word instances.

N	A	B	A′	B′	C	C′
16	192	128	120	80	80	64
64	1152	768	672	448	480	384
256	6144	4096	3456	2304	2560	2048
1024	30720	20480	16896	11264	12800	10240
4096	147456	98304	79872	53248	61440	49152

crease in performance is not a factor four, but only two, depending on memory bandwidth. A further increase of the radix results in non-trivial twiddle multiplications within the butterfly structure and the gain of such an implementation is in general not worth the additional complexity introduced by these multiplications. Furthermore, with a higher radix, there are less choices of N, making the architecture less flexible.

The split-radix–2/4 algorithm outperforms both radix–2 and radix–4 in the number of multiplications per FFT operation [35]. The algorithm is commonly mentioned in the context of high performance FFT calculations, and is therefore an important implementation candidate. Unfortunately, the split-radix algorithm is implemented as a radix–2 or radix–4 FFT, with the same number of butterflies. Therefore, there will be no gain in the number of clock cycles per FFT for the split-radix algorithm. Power consumption may be lower though, if the trivial multiplications by j are implemented efficiently.

In conclusion, looking at the radix–2 case, a performance increase of a factor three is gained by building a few models to explore the design space. The cost associated with the performance gain is a few extra registers and a small overhead from using dual RAMs described in section 3.3.2. More complicated algorithms are faster, but maybe not as fast as expected when implemented using a limited memory bandwidth.

3.6 Low Power Optimizing Strategies

Power consumption is affected by decisions on all levels of the design process, from choice of algorithm to logic style and layout. For a low overall power consumption, optimizations have to be applied on as many design levels as possible [27].

In a CMOS circuit, the dynamic contribution to the total power consump-

tion dominates in general. Since dynamic power is energy per time, the power measurement is dependent of the execution rate. A more accurate measurement to minimize is the energy per operation. The theory behind CMOS power dissipation is discussed further in chapter 4.

Focusing on the energy per operation, there are two variables affecting the power consumption. First, the amount of energy spent for an operation. Second, the number of operations per time unit. The amount of energy per operation depend on supply voltage and load capacitance. The load capacitance depends on the layout level of wire lengths and transistor size. On a higher design level, factors affecting the load capacitance are logic style, arithmetic implementation, and memory size. The number of operations per time unit depend on the clock rate of the system, see section 4.1.3, and the average number of transitions in the circuit. In the next two sections, two methods to reduce power consumption based on minimizing the energy per operation and number of operations per time are presented.

As will be seen in section 4.1.2, the supply voltage V_{DD} has a quadratic impact of the power consumption. Clearly, a low voltage is desired for low power dissipation. However, there are two main effects contradicting a reduction of the supply voltage. First, the device gets slower. For voltages well above the threshold voltage, speed decreases approximately proportional to $1/V_{DD}$. This is a result from equation (4.3). Second, as the supply voltage is lowered, there is a corresponding decrease of the signaling voltage. When the noise margin determined by the threshold voltage is reached, the logic will malfunction.

A well-known power optimization technique is to design the circuit at higher target clock frequency than it is intended to be used at. The circuit is then run at the lower, intended pace. The energy per operation remains the same, independent of the clock rate, but power is saved by lowering the supply voltage [27]. The speed margin created by the higher target clock frequency allows for operation at a lower voltage. On the other hand, a circuit optimized for speed is in general more flattened, consisting of fewer gates in series. This results in a larger share of high fan-out nets, requiring large drivers to maintain speed. Large drivers and long interconnection wires have a higher capacitance resulting in a higher power consumption, since power is proportional to the total switched capacitance, see section 4.1.4. Thus, by making the circuit fast does not guarantee a low power consumption. Designing for low power is different from designing for high speed.

3.6.1 ACTIVITY FACTOR OPTIMIZATION

Activity is defined as the average number of switching cycles per time unit for a node, see section 4.1.4. By reducing the switching activity, power consumption

is lowered. For example, submodules can be deactivated when not computing. Deactivation can be done to a certain level. On the first level, all inputs to the submodule could be frozen, and internal feedback loops cut. Keeping input signals constant reduces switching activity inside the submodule. Remains the clock signal. As the clock signal is always switching large capacitive loads, the second level of deactivation is reached by gating the clock, virtually freezing all signals in the module. As there is no switching activity in the frozen module, power dissipation is due to leakage currents. Leakage current can be reduced by increasing the threshold voltage, and this is a subject of section 4.1.1.

For circuits with constant operation, or in designs where clock gating is not feasible, dynamic power consumption is decreased by reducing the switching activity of operating circuits. On the higher design levels, factors like the choice of algorithm and memory addressing have a large impact. On lower design levels, low-switching implementations of arithmetic functions and wordlength optimization are important. Low-activity arithmetic circuit realization is a technique applicable to most designs. For example, a tree adder compressor has a shorter logic depth, and thus less switching, compared to an array compressor, see paper III [17]. Another example is the use of carry save arithmetic, where no high switching activity carry ripple is present.

The width of buses and signals directly sets the width of the computational units in the datapath. Therefore, a minimum wordlength is a powerful means to reduce switching activity. In general purpose processors and DSPs, data wordlength is fixed. This implies that the datapath is of fixed size, independent of the required or necessary accuracy of the data. Some processors allow for splitting the data path in halves or smaller parts, though, thereby allowing an alternative data format with lower accuracy. The main aim with this construct, however, is to increase throughput by parallelizing the data flow in a Single Instruction Multiple Data (SIMD) like fashion — and not to reach a low power operation.

The ASDSP design allows for optimal wordlengths on all signals in the design. Datapaths and memories are optimized towards the input and output data wordlengths, resulting in a minimal overhead in computation and storage. Thereby, switching activity is decreased.

3.6.2 MEMORY OPTIMIZATION

Memory accesses are costly operations that have a major contribution to the total power consumption of an ASDSP [25]. The larger the memory, the more energy is consumed per access. Furthermore, large off-chip memory devices require orders of magnitude more energy per access than a small on-chip memory. Off-chip signaling is also expensive in energy due to the large capacitances

associated with bonding wires, package pins, and Printed Circuit Board (PCB) wires. For the reasons given, the number of memory accesses should be kept at a minimum — especially accesses to external memory devices. A primary goal is to reduce the number of accesses to large memories. A memory management methodology, such as the Data Transfer and Storage Exploration (DTSE) methodology is a means to reach the goal [25]. The idea is as follows:

> first, by loop transformation techniques [3], the memory accesses are concentrated in time and memory range
>
> second, cache memories — small energy efficient memories for intermediate storage — are used to mirror the data in the larger memories

If the accesses can be arranged to be local in time and address range for a time period, most memory accesses are addressing the small on-chip cache memory instead of the off-chip memory device. The arrangement leads to a lower energy per operation in average. Furthermore, as cache memory is faster, throughput is increased and the application executes faster.

The memory methodology is illustrated by an example. Consider a two-dimensional time domain convolution of an image and a filter kernel. For simplicity, both the image and the kernel have a quadratic shape. The image has $N \times N$ elements, whereas the kernel has $M \times M$ elements. It is assumed that the image is large and that the kernel is significantly smaller than the image.

The convolution operation is performed by moving the kernel over the image. For each kernel position, overlapping image and kernel elements are multiplied and accumulated. The accumulated sum is the correlation output value for the image element corresponding to the center of the kernel. The operation lasts until the multiply and accumulate operation has been carried out for all possible positions of the kernel. The multiply and accumulate operation of the M^2 elements is performed in its entity at all positions, and the kernel visits every unique position only once.

Figure 3.13 illustrates the convolution process. The shape of path described by the kernel during the convolution operation has no significance to the calculated result. To simplify the example, kernel movement is restricted — no part of the kernel is allowed outside of the image bounds. The restriction implies that there are no output values computed for the outermost elements of the image.

As a consequence of the large image size, it has to be stored in an off-chip memory. Large memories have a higher energy dissipation per access and longer access times than the small counterparts. Furthermore, off-chip accesses in themselves are expensive in energy dissipation. In order to reduce memory

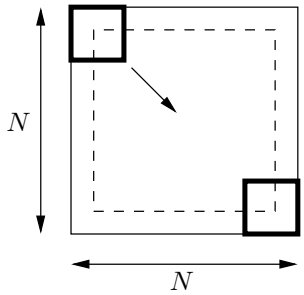


Figure 3.13: Two-dimensional convolution. Filtering of an image by a kernel.

access energy, a cache memory scheme is investigated. Since the kernel is $M \times M$ elements, and a multiply and accumulate operation is performed at $(N - M + 1) \times (N - M + 1)$ positions, there will be a total of $M^2(N - M + 1)^2 \approx M^2N^2$ read accesses to the large memory, resulting in a high energy consumption. Ideally, with a perfect cache scheme, the number of reads to the large memory is only N^2, or one access per element. If the M^2M^2 accesses are addressing a small cache memory instead, a significant reduction is expected.

Assume that the image is stored with address zero corresponding to the upper left element, and that the following elements are stored horizontal elements first, left to right. If the kernel moves in a top to down vertical movement first manner, starting at the top left position and working its way to the bottom right position, the memory access pattern will look like illustrated in figure 3.14(a). The thick line is not continuous — there are small leaps up and down the address range on the smaller scale. Every time the kernel moves one image element down, memory addressing will increase by N elements. When the kernel has reached the bottom line, it moves back to the top, one element to the right, and the procedure repeats. Thus, the majority of the memory address range is accessed continuously. If a small cache memory is to be used, its contents have to change rapidly. Therefore, it is difficult to apply a cache strategy with an access pattern as in figure 3.14(a).

From a cache point of view, the ideal access pattern for the convolution would look like in figure 3.14(b). The memory accesses are local in time and address space, and therefore cached data will be valid for a larger number of accesses. Again, this is on the larger scale, the curve is fluctuating on the smaller scale, but it is not significant in this context. An access pattern like in figure 3.14(b) can be achieved by applying loop transformations [3]. A simple modification to the algorithm — letting the kernel move horizontal before vertical, and not the other way around — will generate such a pattern.

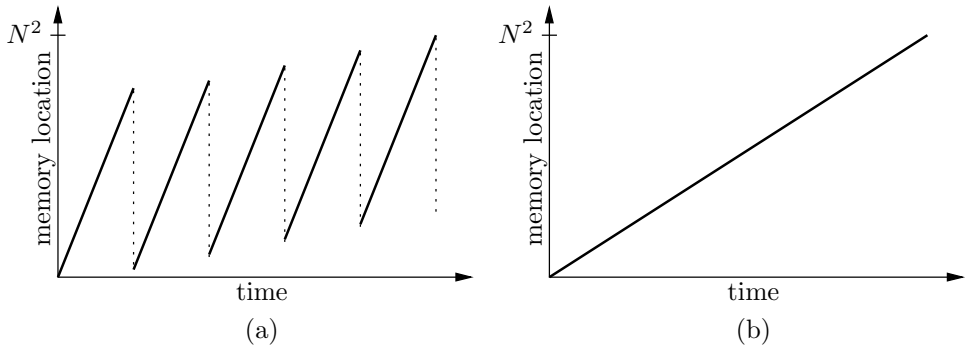


Figure 3.14: Memory access patterns for different loop ordering. (a) results from moving the kernel top to bottom, left to right; (b) when moving first left to right, then top to bottom. The thick lines are not continuous — on a lower scale addressing is less regular.

Now, as the kernel moves left to right, the memory addressing is incremented by one for each step to the right. When the rightmost operation is finished, the kernel jumps back to the left and begins at a new line, increasing the address counters by M only.

Loop transformations have to be applied with care. By rescheduling the order of read operations, output data will most likely be rescheduled too. Furthermore, if the loops of the kernel $M \times M$ multiply and accumulate operation is part of the loop transformation process, the calculation of each image element output will be distributed in time. If an output calculation is only performed in part, intermediate storage is required, and it is less likely that there will be any gain from the loop transformation. Instead, by keeping the multiply and accumulate operation intact, there are no dramatic changes to the algorithm.

There are a number of candidate cache memories for the convolution example. The more straightforward alternatives, found by examining the loop counter bounds after loop transformation, are depicted in figure 3.15.

Scheme A, based on the external memory M0 only, for reference.

Scheme B, using a combination of M0 and C0. The larger cache memory, C0, has capacity to store a kernel sized fraction plus the $M-1$ last visited lines of the image. For every right-step, only one new image element has to be read, while one old is discarded. For a right to left jump, M new elements are read. In total, N^2 reads from M0 are required for a complete convolution operation, which is optimal.

Table 3.2: Cache memory schemes and the corresponding number of accesses between each memory.

scheme	M0→	C0→	C1→
A: M0	$M^2(N-M+1)^2$		
B: M0→C0	N^2	$M^2(N-M+1)^2$	
C: M0→C1	$MN(N-M+1)$		$M^2(N-M+1)^2$
D: M0→C0→C1	N^2	$MN(N-M+1)$	$M^2(N-M+1)^2$

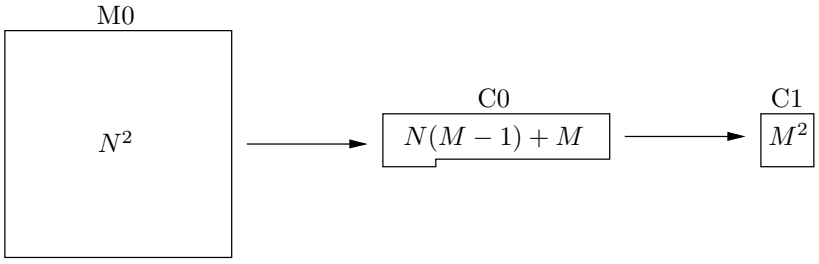


Figure 3.15: Possible cache memories for the two-dimensional convolution example. The number of stored elements is printed in each memory.

Scheme C, based on M0 and C1. The smallest memory, C1, is large enough to hold a kernel sized image fragment. For every step to the right, M new elements need to be read from the large memory M0, and M old values are discarded. When reaching the right border and jumping back to the left side, all M^2 values have to be replaced. This memory requires MN reads per row times $(N-M+1)$ rows, or $MN(N-M+1) \approx MN^2$ reads per convolution from M0.

Scheme D, combining M0, C0, and C1 in a hierarchy. The scheme is interesting, since C0 has the minimum number of accesses from the large M0, and C1 has minimum size and is target for most accesses.

The number of accesses between the large memory M0, the different cache memories (C0 and C1), and the target datapath are shown in table 3.2 for the four different cache hierarchies, A to D. As expected, the number of accesses to the datapath is constant for all four schemes, but the access count to the large memory varies.

The effect of the cache hierarchy exploration is best seen if real numbers are used. Let the size of the image memory be 1024×1024 elements, and the kernel 15×15 elements. These numbers are reasonable for some image

Table 3.3: Energy consumption for the external and internal memories. The off-chip signaling power dissipation due to accessing of M0 it not included in the table. See text for more information about the memory instances.

memory	words	wordlength	energy/access
M0	1048576	16	60 nJ
C0	15376	16	4 nJ
C1	225	16	1 nJ

applications [66]. Energy access cost are presented in table 3.3. The table is created using the same memory generator as used for the echo canceller implementation [1]. Data for the large off-chip memory is taken from a low power static RAM [58]. It should be noted that the external RAM is a state-of-the-art component, fabricated in a 0.18 μm process, while the embedded memories are implemented in a slower and more energy consuming 0.35 μm process. The presented number are nominal, expected to be valid under "normal" conditions.

Total cost of the different cache schemes A to D is shown in table 3.4. The scheme based on no cache memories (A) consumes 30 times more energy compared to the best scheme (D), for the same operation. The energy cost for off-chip communication is not included in the table, and neither are adjustments for the different manufacturing processes. If off-chip access cost and process scaling was included, the differences would be even more dramatic. Especially, comparing B and C, the former has about M times less accesses to M0, and is therefore a better choice when the external memory addressing is more expensive.

Cache memories add to the total chip area. The scheme having a 30 times reduction (D) in energy requires about 30 kilobyte internal memory. Scheme C, with an energy saving of twelve times, requires only 450 bytes memory, which comes at a negligible cost in most implementations.

Furthermore, as internal memories are faster, the operation will be carried out in a shorter time using cache memories, if the internal clock rate is higher. The external low power memory runs at a maximum frequency of 18 MHz [58], while on-chip circuitry in a corresponding process typically runs at one to two decades faster. Assuming the on-chip convolution processor and cache memory runs at 180 MHz, ten times faster than the external RAM, speedup will be in the range of six to nine times depending on the cache scheme.

In summary, all proposed cache schemes reduce energy consumption considerably while increasing the throughput. Which scheme to choose depends on a number of implementation constraints, such as required throughput, chip

Table 3.4: Energy dissipation for the different cache schemes.

cache configuration	energy cost
A: M0	13.82 J
B: M0→C0	1.03 J
C: M0→C1	1.21 J
D: M0→C0→C1	0.40 J

area, and control resources.

In [65], a convolution processor for image enhancement is presented. The architecture employs a cache hierarchy corresponding to scheme B in this section.

3.7 ASDSP Design Flow

A design flow covers the important levels and tools used for going from initial problem description to a hardware implementation. For digital circuits, the design process is carried out at a number of levels: initial problem description, algorithm, architecture, arithmetic, logic, and layout. Highest performance is achieved if all levels are considered during the design phase. For today's multi million transistor designs, however, this is not possible within a reasonable time perspective. Instead, synthesis tools and cell libraries are used to raise the abstraction level of the design process. Synthesis tools and cell libraries are the subjects of section 3.4. Still, for designs with extraordinary requirements on speed or power, low level full-custom design is the only way.

The ASDSP design flow is based on synthesis and automatic layout tools. The flowgraph is depicted in figure 3.16. The design process evolves vertically in the diagram, from initial description at the top, down to a fabricated chip at the bottom. From the initial specification, constraints that determine performance of every part of the design are derived. At each level, the design is compared to the initial specifications and constraints, and incremental optimization is performed as required. Further constraints are added at lower design levels. Sometimes, a larger leap up in the design flow might be necessary to meet the design constraints at a lower level. The initial step of going from a DSP problem to an algorithm is not within the scope of this thesis.

The design flow depends on a standard cell library and a synthesis tool, by which the lowest design levels are removed. Instead, the design is modeled using a Hardware Description Language (HDL), see section 3.4. The HDL description is mapped to *nets* and *cells* in the synthesis process. A cell is a component

that reside in a standard-cell library, and the nets are connections, or wires, between cells. The cell library has to contain the most necessary combinational and sequential logic functions for the process to work. The resulting circuit is sub-optimal, in the sense that all cells are pre-defined and no optimization on the transistor level is done, but the advantages are several:

A hardware description language allows for parameterization and flexibility of a design. Different wordlengths and coefficients can be elaborated with a short turn-around time.

The design does not depend on a specific hardware technology. Therefore, the design can be resynthesized to any cell-library.

Design time is shorter than for full-custom. Modifications and corrections can be included with a short turnaround time. Time to market is an important factor.

There are standardized, non-propriety languages available, and support from several independent software companies.

The main parts of the ASDSP design flow in figure 3.16 are

Specification. For a given a DSP problem, a specification regarding the operation of the implementation is elaborated. Initial constraints in terms of timing budget, cost, and accuracy are approximated. From these constraints, together with target implementation technology data, detailed design constraints are derived regarding area, power consumption, minimum clock frequency, and so on.

Algorithm An algorithm to solve the problem is developed in a high level simulation language. On the algorithm level, simulations are performed to find higher bounds on accuracy; and the number of arithmetic operations per second can be estimated. This simulation model is later used to compare lower level descriptions.

Architecture and arithmetic An architecture of hardware functional units is developed. The architecture is designed to make mapping of the algorithm to the functional units easy. Furthermore, the implementation of the arithmetic operations of the functional units is investigated. Standard architectures for the most common arithmetic operations are available in a library. If special arithmetic is to be used, it has to be designed separately. Modeling of the architecture is done in a high level hardware description language.

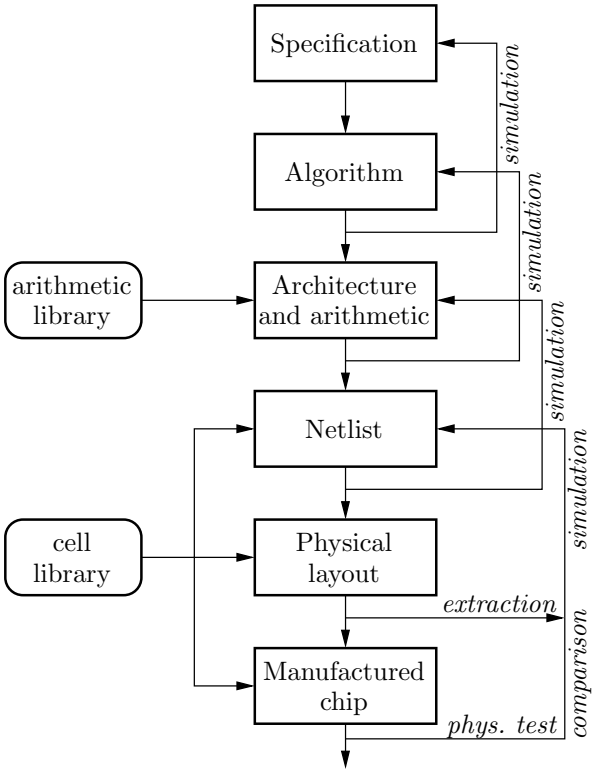


Figure 3.16: A typical design flow for ASDSP design using hardware description languages and synthesis tools.

Netlist When the algorithm has turned into an architecture, proven to be consistent with initial specifications by simulation, it is synthesized using a synthesis tool. The generated netlist consists of instances of components from the target cell library, memory instances created by memory generator, and interconnecting wires. The netlist is by simulation compared to the architecture model it originated from. It is also possible to do formal verification. Analysis of the netlist gives approximate information on area and timing of the final chip. After layout, timing goals might be violated, and it is necessary to satisfy design constraints with a sufficient margin at the netlist level. Small performance improvements can be achieved by exploring the parameter space of the synthesis tool, but larger improvements force re-design at a higher level, for example by

modifying the architecture or by using faster arithmetic.

Physical layout Using place and route tools, the netlist, together with information from the target cell library, is transformed into a layout description of the design. Cells are placed according to a floorplan, and interconnecting wires are routed accordingly. The floorplan effects the total wire-length of the layout, and is designed with care. Furthermore, power wires and clock signals are routed using special strategies. Most steps on the physical level are done by design tools. Design of the floorplan is an exception, designed by hand, as it has great impact of the performance of the circuit. Extraction of capacitances and wire lengths of the physical layout are fed back to the netlist simulation to get a more accurate timing of the placed and routed design. When the design is completed and verified it is sent for fabrication.

Manufactured chip When the chip has returned from fabrication, it is compared to the design specifications. Functional verification as well as performance in terms of maximum clock rate and power dissipation are investigated. For a mass-produced circuit, fast correctness tests are developed to find faulty chips [37].

As a DSP problem converges from initial specification to an ASIC, it is modeled on several abstraction levels. For each level, there is an accompanying language that is well-suited for the description. However, coupling between the different models and languages is a necessity, since design iteration depend on the possibility of going "back" one or more levels of the design flow, to make corrections to problems that becomes evident first on lower description levels. Furthermore, consistency between the different levels of the design flow is important to guarantee that the design complies to the initial specification. The design has to be validated at all design levels. Ideally, each model is by simulation compared to the initial algorithm. Since several modeling languages are involved in the design process, it is complicated to achieve such a comparison in practice. Instead, every design level is at least compared to the level immediately above.

3.8 A Design Methodology

As mentioned at the end of the previous section, consistency of the design flow is a problem due to the different modeling languages involved. Here, a method for keeping consistency from algorithm description to a low level fixed point `C` description is presented.

Initially, an algorithm is modeled in a high level language, such as Matlab [56] or Octave [36]. High level languages makes design entry fast, and comes with extensive debug capabilities to shorten design iteration time. Unfortunately, expressing the algorithm in terms of high level operations such as matrix operations and FFTs results in a description that is far from the architecture level. A description in for example the `C` Programming language is closer to the architecture level, since the basic data types are scalars, and arithmetic operations are performed on scalar data only. The method proposed in this section transforms a Matlab or Octave description into `C`, with a guaranteed consistency of the descriptions. The methodology is used in this thesis work and is summarized as follows:

1. The algorithm is developed in a high level language such as Octave [36] or Matlab [56]. On this design level evaluation of different algorithms and transformations is fast. The languages comes with libraries of standard DSP functions that can be called from the program, and arithmetic is performed using double floating point precision. Furthermore, powerful visual aids speed up debugging and assists in finding errors.

   The first description is typically non-causal, accessing input data for time n, $-\infty < n < \infty$. This makes it difficult to estimate important implementation properties, such as in which order things have to happen and how much memory that is required.

2. Sub-algorithms are rewritten one at a time into the `C` programming language. By linking the `C` code dynamically into Octave or Matlab, it can be run together with the original high level code. Therefore, the debugging process uses the same development environment with advanced graphical capabilities as used for the initial algorithm model. If every part of the algorithm is rewritten and debugged consecutively in this fashion, consistency with the original description is kept.

   Eventually, all code is transfered to `C` and debugged in this fashion, and at the same time it is verified to the original description. The `C` model of the algorithm does not, as opposed to the high level description, rely on matrix operations. Instead, it is constructed based on additions, multiplications, and `for`-loops. With such a description, it is easy to understand the complexity of each and every part of the algorithm.

3. All sub-algorithms are merged into one `C` function block. This block can still be run from inside the high level environment, keeping the same debug functionality, but having significantly lower simulation times. For the echo canceller implementation, a twenty times faster simulation time

was observed. The new description will contain several loops over time ranging from a low index to infinity.

```
for(n=nmin_0; n<∞; n++) {
  ··· loop body 0  ···
}
for(n=nmin_1; n<∞; n++) {
  ··· loop body 1  ···
}
for(n=nmin_2; n<∞; n++) {
  ··· loop body 2  ···
}
···
```

It is straightforward to move all loop bodies into one large `for`-loop. The largest loop initialization value is

$$\texttt{initial\_n} = \max_{\mathrm{i}}(\texttt{nmin}_{\mathrm{i}}).$$

By splitting all `for`-loops into two parts, one ranging from $\texttt{nmin}_{\mathrm{i}}$ to $\texttt{initial\_n}-1$, and one from `initial_n` to infinity, as

```
for(n=nmin_0; n<initial_n−1; n++) {
  ··· loop body 0  ···
}
for(n=initial_n; n<∞; n++) {
  ··· loop body 0  ···
}
```

all loops ranging to infinity are merged into one main loop. The residual loops ranging from $\texttt{nmin}_{\mathrm{i}}$ to $\texttt{initial\_n}-1$ describe transient behavior, and all variables with index within the range can be set to zero, as the reset sequence.

The main loop is translation invariant, and can begin at any time unit. For comprehension, the loop is translated to start at time unit zero.

4. The resulting `C`-code is prettified, pruned and subject to various low-level optimizations. Loop transformation [3, 25] is a powerful tool used to explore the design space of the algorithm. Memory size requirements are found by studying loop bounds. To verify the bounds and to simulate memories of limited size, memory addressing is performed integer modulo

the memory size. Cache hierarchies can be found and exploited to reduce memory power by utilizing the Data Transfer and Storage Exploration (DTSE) memory management methodology [25].

Signal wordlengths are optimized using a suitable tool. A `C++` library overloading the arithmetic operations has been found to be efficient [45].

The resulting code contains all necessary information for a hardware implementation: memory sizes, signal wordlengths, data operations, and the order in which operations are to be executed. However, the memory sizes found by this method are maximum sizes, and the description is subject to further optimizations.

The presented scheme is only part of the way towards an architecture. Every operation present in the resulting code can be realized in a more or less parallel fashion in the architecture. For a more parallel solution, requirements on intermediate data storage is reduced. In order to reach an efficient architecture, implementation constraints are added to the description. Typically, these constraints include sample rate, clock frequency, area limitations, and power consumption. Constraints for every functional unit can be derived from this information. With known constraints, functionality, and memory requirements, an architecture for each functional unit can be constructed.

A design project will consist of a large number of lines of code. In the design methodology, all code is written according to a number of rules regarding simplicity, clarity, and generality [48, 49]. To minimize the risk of old code turning obsolete, the methodology is based on standard, de-facto or otherwise, formats and programs, such as the `C` programming language [50], VHDL or Verilog, and common UNIX [48] tools.

3.9 Summary

A thorough methodology is necessary to complete large ASDSP designs. The design space is explored to find local cost minima for algorithms and architectures. By dividing the design into a hierarchy, design, verification, and maintenance becomes easier. Each submodule in the hierarchy contains one or more of the following components: memory, datapath, and controller. The different phases of the design process is further simplified by only letting the controller manage memory accesses, while the datapath computes operations on data only. To speed up the design process, the lower level of CMOS design are replaced by standard-cell libraries and synthesis tools. Memory throughput is often a bottleneck in digital designs. The use of cache memories is an efficient means to increase throughput and reduce energy per operation in architectures with large or frequently addressed memories.

Chapter 4

CMOS Technology

Complementary Metal Oxide Semiconductor (CMOS) technology is by far the most used digital integrated device technology today. The development of CMOS circuits follows Moore's law, stating that the number of components on a chip giving the lowest manufacturing cost grows exponentially over time. In his original paper, Moore stated that the number of devices on a chip will double every year [59]. The assumption was based on linear extrapolation in a logarithmic diagram of data from designs fabricated between 1959 and 1965. In 1959, the most cost-effective design had about 50 components. Later, Moore revised the prediction to a doubling every second year.

Today, four decades later, chips containing a hundred million transistors are realistic to fabricate. To make this dramatic increase in the number of transistors per chip possible, the size of each transistor has shrunk tremendously. A smaller transistor is faster, and this effect has in turn given the devices even higher performance. The corollary to Moore's law appears to be that microprocessor performance doubles even faster than the number of devices per chip. The rate of performance increase of microprocessors is as fast as a doubling every eighteen months[1] [87].

CMOS technology is under constant development, and Moore's law still seems to be relatively accurate. A common question is how long can CMOS technology development keep up with Moore's law. Most likely, the pace will decrease as device sizes approach the atom scale [87].

[1]There is a slight confusion about the term "Moore's law" in the literature. Transistor count or performance doubling every eighteen months is a common interpretation.

4.1 Power Dissipation in CMOS

A major reason that contemporary high performance devices are implemented in CMOS technology, except for low cost, reliability, and high density is the almost complete absence of static power consumption in steady-state mode [73]. For circuits not constantly occupied by computational tasks, this is beneficial, since they can be turned off when idle to save energy. Low power consumption is important in for example handheld devices, where battery capacity limits time of operation. Furthermore, low power consumption is also attractive in stationary, mains-powered applications. Low power consumption implies reduced requirements on cooling, which is expensive, and increases the reliability of the device since it will operate at a lower temperature.

To design power efficient hardware, it is important to understand the basic theory of CMOS power dissipation, even if the design process is focused on the architectural level. The total power consumption of a digital CMOS gate can be divided into a static and a dynamic contribution [26], as

$$P_{total} = P_{static} + P_{dynamic}. \tag{4.1}$$

The two parts consists in turn of a number of sources, of which the major ones will be discussed in more detail.

4.1.1 static power dissipation

Static power dissipation is due to leakage currents, and is modeled with a simple equation,

$$P_{static} = I_{leakage} V_{DD}, \tag{4.2}$$

where V_{DD} is the supply voltage. The leakage current is primarily determined by fabrication technology considerations [22]. There are two main sources of leakage current, reverse-biased diode junction current and sub-threshold current. The first is due to thermally generated carriers. These carriers flow through the reverse-biased diode junctions of the transistors located between the source or drain and the substrate. The reverse-biased diode junction current is generally negligible at room temperature, but increases with junction temperature in an exponential fashion.

The sub-threshold, or weak-inversion, current is potentially a bigger issue when considering low-power methodologies. When a CMOS transistor is "logically off", there is still a small drain-source current present. The transistor experiences an exponential decrease in drain-source current as the gate-source voltage V_{GS} decreases below the threshold voltage V_T. Figure 4.1 illustrates the

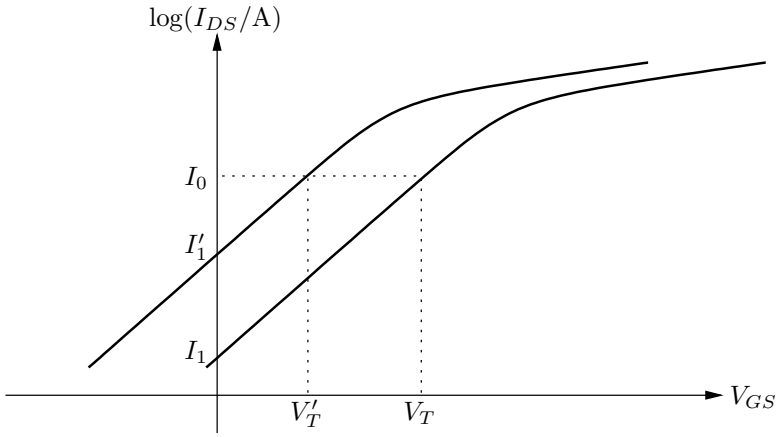


Figure 4.1: Logarithm of I_{DS} versus V_{GS} for two different threshold voltages. The exponential relation between gate voltage and the drain current is a straight line in a log plot. See text.

(logarithm of the) weak-inversion current versus V_{GS} for two different threshold voltages. Moving from V_T to V'_T results in a change in leakage current from I_1 to I'_1. Thus, a large V_T gives a low leakage current, which is desired for applications that utilize standby modes of operation. A small V_T, on the other hand, is desired for applications where speed is an important factor. The reason is the dependence of V_T to the propagation delay of a CMOS gate. The propagation time T can be written as

$$T = \frac{C_L}{k_p} \frac{V_{DD}}{(V_{DD} - V_T)^2}, \tag{4.3}$$

where C_L is the load capacitance and k_p is a process transconductance parameter [73]. Equation (4.3) states that an increase in V_T implies longer propagation times — the device gets slower. Thus, there is a trade-off between speed and static power consumption [75]. A CMOS process with dynamic threshold voltage is beneficial in low power applications where requirements on performance vary with time. For maximum performance, V_T is at its minimum, and otherwise it is adjusted to make the logic "fast enough" for the momentary performance requirement.

Design of dynamic circuits is another issue where static leakage is an important aspect. Dynamic implementations depend on the storage of charge in capacitors, and therefore leakage must be reasonable low. A firm lower bound on the value of the threshold voltage is set by this sub-threshold effect [73].

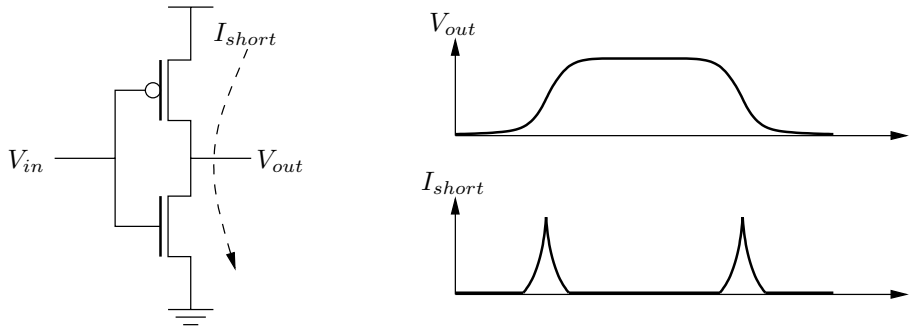


Figure 4.2: Direct path current in a CMOS inverter.

4.1.2 DYNAMIC POWER CONSUMPTION

The dynamic power consumption for one node switching a complete cycle, from ZERO to ONE and back to ZERO again, at a frequency f_{clk} is

$$P_{dynamic} = C_L \cdot V \cdot V_{DD} \cdot f_{clk}, \tag{4.4}$$

where V is the voltage swing at the output, V_{DD} the supply voltage, and C_L the capacitive load of the circuit [27]. The voltage swing V is in most cases equal to V_{DD}, resulting in the more commonly used expression

$$P_{dynamic} = C_L V_{DD}^2 f_{clk}. \tag{4.5}$$

However, some circuits rely on a voltage swing on some internal nodes to be less than V_{DD}. Examples include low power memory design [75] and low swing buses [43]. Lower voltage swing also appears in pass-transistor logic implementations [88],

There is a second source of dynamic power consumption, appearing at the moment of transition when both N and P type transistors are active, denoted direct path or short-circuit current, see figure 4.2. A first order approximation of the short-circuit direct path current is

$$P_{dp} = \frac{t_r + t_f}{2} V_{DD} I_{peak} f_{clk}, \tag{4.6}$$

where t_r and t_f is the rise and fall time of the transition respectively, and I_{peak} is the peak short circuit current, proportional to the size of the transistors [73]. The direct path current can be reduced by carefully sizing transistors for equal rise and fall times for the input and the output signals of a gate. Then, short-circuit currents contribute less than 20 percent to the dynamic dissipation [83].

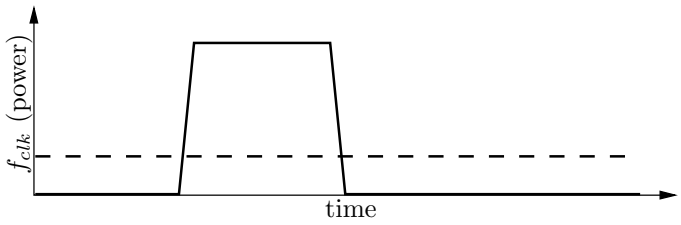


Figure 4.3: Operation of two different designs. The dotted line represents a design with a constantly low clock frequency. The solid line is a design with a faster clock frequency and power down mode.

4.1.3 ENERGY PER OPERATION

Examining the equations describing the dynamic power consumption, (4.5) and (4.6), it is apparent that both are proportional to the clock frequency f_{clk}. Thus, the higher clock frequency, the more power consumption. Is it then true that a slower design is more efficient?

Consider figure 4.3. The horizontal axis represents the time axis. On the vertical axis is the clock frequency. The vertical axis also represent execution speed and power consumption, since they are proportional to the clock frequency. The figure illustrates operation of two different designs. The solid curve is a fast design with a high clock frequency. To save energy, operation is halted when not performing any computations. The other design, represented by the dotted line, has no "power down" mode of operation, but runs at a significantly lower clock frequency.

The area under each curve represents both energy and total number of operations, and is equal for both designs. Thus, both designs consume the same amount of energy for the same number of operations. The energy per operation is constant.

When analyzing the energy per operation, it is common to only take into account the effect due to the capacitive load. Energy per operation is thus written as

$$E = C_L V_{DD}^2. \tag{4.7}$$

It is interesting to note that the energy per operation of a CMOS gate only depend on the capacitive load and supply voltage. In equation (4.7), the supply voltage has a quadratic impact on the power delay product, assuming the nodes swing to V_{DD}. As the energy per operation is independent of the clock frequency, it is clear that a reduced clock frequency does not affect the energy spent per operation. Power consumption will be lower, but the operation takes

longer to complete.

The key is found from equation (4.3), stating that the speed of a circuit decreases with a reduced supply voltage. Thus, by lowering the clock frequency of a device, supply voltage can be reduced as well. From (4.7), it is apparent that the supply voltage has a large impact of the energy per operation. By reducing the supply voltage, speed can be traded for energy [26]. For battery powered devices, the energy per operation, and not the momentary power consumption, determines the longevity.

4.1.4 CIRCUIT LEVEL POWER CONSUMPTION

To model power consumption on the circuit level, a useful expression is derived from equation (4.4). Assuming each node has a switching capacitance C_{Li}; a switching activity, α_i, defined as the average number of switching cycles per clock cycle of the node; and a swing voltage V_i. The circuit level power consumption is written [28]

$$P = \left(\sum_{\text{all nodes}} \alpha_i C_{Li} V_i \right) V_{DD} f_{CLK}. \tag{4.8}$$

The value of α_i is not bounded by an upper limit. Consider the example of a ripple carry adder with all inputs ZERO. Switching one of the input words plus the carry in bit to ONE forces all the outputs immediately to ONE. In parallel, a carry ripple chain runs through all the full adders and toggles all outputs back to ZERO again, resulting in a high activity of several transitions per adder cell. For this example, the number of transitions is only limited by the wordlength of the ripple carry adder.

The amount of switching activity in a circuit depends on the input signal statistics, circuit architecture, and logic style. Tools for determining switching activity are often based on statistical methods. A highly correlated data set causes less switching in for example an adder circuit, since only a few bits change per input word. On the logic level, switching can be decreased by applying the precharged, or dynamic, logic style. In this technique, a gate has a maximum of two transition per clock cycle. However, it turns out to be difficult to design generic circuits based on precharged gates. One example of where it is applicable is in design of memories.

4.2 MEMORY TECHNOLOGY

Most DSP algorithm implementations rely on some kind of memory for intermediate data storage. For example, adaptive filters need storage for coeffi-

cients, and autocorrelation devices need storage of the input signal. Therefore, memory technology is equally important as the implementation of arithmetic circuits.

The most common memory technologies are either Read Only Memory (ROM) or Random Access Memory (RAM). Data in a ROM is determined in the design process, and cannot be changed after fabrication. An increasingly popular memory component is the FLASH memory [73]. A FLASH behaves as a ROM, but it can be reprogrammed completely or partially. Efficient ROM implementations are based on two-dimensional arrays of programmed bit elements, requiring only one transistor per bit of storage. Alternatively, a ROM storing a small number of information bits can be implemented using random logic, realizing a multi-dimensional logic function. Implementation of RAMs are more delicate, and the subject of this chapter.

There exists two major techniques for on-chip RAM implementation, either static or dynamic. The dynamic memory will loose its contents after a short time period, depending on the implementation. In order to keep the data valid, it needs constant refresh accesses. The static memory, on the other hand, stores written data until power off, without any dedicated refresh cycles. Behavior of static and dynamic memories can be compared to static and dynamic logic. Small intermediate storage can also be implemented using flip-flops. This technique results in extremely fast memories, at the cost of a larger area and higher load on the clock net.

A static RAM implementation typically requires six transistors per stored data bit, arranged in an area-efficient scheme [2]. A six transistor static RAM cell is shown on the left hand part of figure 4.4. The implementation is similar to a flip-flop based on two cross-coupled inverters. A read operation is performed by activating the wordline WL, and detecting the stored state on the bitlines BL and $\overline{\texttt{BL}}$. Benefits of static memories are fast access times and non-volatile behavior as long as the supply voltage is present. To write a ZERO/ONE to the cell (forcing Q =ZERO/ONE), the wordline WL is set to ONE, and the bitline BL is set to ZERO/ONE (while $\overline{\texttt{BL}}$ is the opposite, ONE/ZERO).

The main advantage of dynamic memories is area efficiency. The most efficient implementations are based on only one transistor per bit of storage. Thus, they are significantly smaller than their static counterpart. However, the dynamic RAM is based on the principle of storing charge in capacitors. These capacitors are drained by the leakage current, and need to be refreshed intermittently [73]. A one transistor dynamic RAM cell is depicted at the right of figure 4.4. Data is written by setting the wordline WL to ONE, and putting the data on the bitline BL. A ONE will be stored in the cell capacitance, C_S, while a ZERO will drain C_S. A readout is performed similarly. First, the bitline is precharged to a value in-between ground and V_{DD}. The wordline

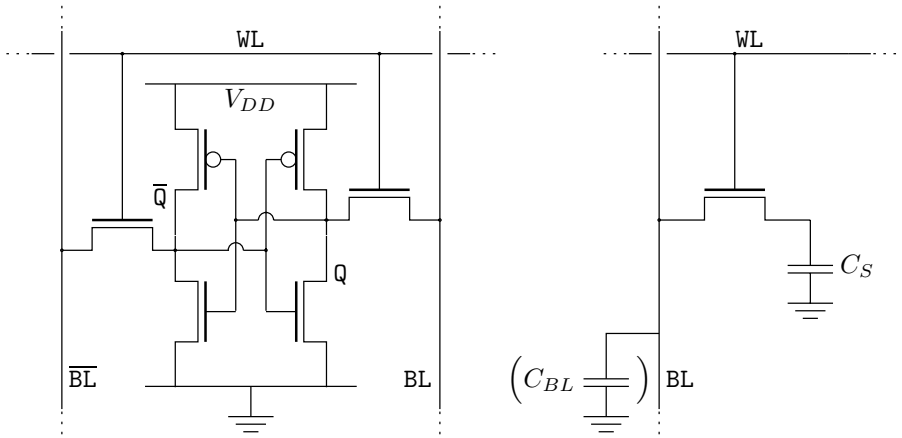


Figure 4.4: Static and dynamic random access memory cells [73]. See text.

is then activated and the charge in the cell capacitor will be redistributed to a equilibrium between C_S and the bitline wire capacitance C_{BL}, forcing the voltage on the bitline to either raise or fall by a small amount. The difference between initial precharge voltage and readout voltage is amplified in a *sense amplifier*, enhancing the difference into a logic ZERO or ONE. The readout of a dynamic RAM cell is destructive, that is, the originally stored value is affected by the read operation. Thus, the readout has to be followed by a write operation, either with the same or new data. Furthermore, if data is not accessed for some time, it has to be read out and written back again, since leakage current will drain the cell capacitance. The process of updating the memory cell information is referred to as a *refresh* operation.

Typically, dynamic RAMs are implemented in special processes, making it possible to design compact bit-cells with reasonable refresh-rates [46]. Processes suitable for both logic and dynamic RAM design are not yet standard. Instead, on-chip memories are static for several reasons: reliability, absence of refresh cycles, and IDDQ-test[2]. Unfortunately, the six-transistor static cell limits effective storage on a chip significantly. If a large storage is required, it has to be added externally.

[2]The IDDQ test is a fast method to find errors in manufactured devices. The device is set to a pre-determined state by test vector stimuli, and the static supply current is measured. With a high probability, short-circuits will be revealed in the static power consumption.

One factor setting the higher bound on throughput in a digital design is the memory bandwidth. As long as the design is within the area limits, datapaths can in general be parallelized to any degree of throughput. Memories can be split and parallelized, but smaller memories have a large overhead in address decoders and data multiplexers, making the total area grow considerably. Furthermore, to split a large memory into a number of smaller ones, partitioning of the contents has to be done in accordance to the access pattern of the data. If data is accessed locally during a time period, probably only one of the smaller memories is accessed continuously, and there will be no gain in throughput using the split memory approach, for a constant clock frequency. Keeping datapaths constantly busy sets the same throughput constraints on the memories as on the datapath itself. There is no use of a high performance parallel datapath if it can not constantly be fed with new data from a memory — the datapath will be unnecessarily fast.

As only relatively small memories can be implemented on chip, large sets of data have to be stored externally in dedicated memory integrated circuits. Unfortunately, external connections are limited by the number of IO-pads and their speed. Using a Low-Voltage Differential Signal (LVDS) scheme, or similar, IO-bandwidth is increased dramatically [43]. However, rapid off-chip transmission through pads, bonding wires, and Printed Circuit Board (PCB) is expensive in power consumption. Also, large and fast external memories consume a significant amount of energy per operation. Thus, off-chip memory will be slower and more energy consuming than an internal memory.

A popular scheme to reduce external memory accessing is to use internal cache memories, see section 3.6.2. In this scheme, the cache memory ideally mirrors the most commonly accessed data, and most accesses target the small internal cache memory instead of doing a power expensive external access. Cache techniques are widely used in microprocessors and DSPs, but as DSP algorithms becomes more complex and memory requiring, it is more common in ASDSP designs as well. In fact, for some systems several levels of cache memories are beneficial, and systems with three cache levels are common today [61].

4.3 Summary

CMOS technology delivers high performance digital signal processing at a low cost. A low power implementation is important both for battery powered handheld devices and stationary computers. The handheld device will run longer between charging, and a stationary computer will be less expensive and more reliable, due to less cooling equipment and lower temperatures of operation.

The subjects of static and dynamic power consumption are introduced, and it is shown that running at a reduced clock frequency just makes things go slower — the same amount of energy is still consumed per operation, and the application takes longer to finish. The key is to lower the clock frequency and adjust the supply voltage in parallel.

CMOS random access memories can be either implemented using a static or dynamic technology. The dynamic memory is significantly smaller in size, but requires constant refresh cycles and special fabrication processes for reliable operation. Therefore, most on-chip memories are static. These memories keep their contents for as long as the supply voltage is sufficiently high, but due to their size only a limited memory storage fits on a chip.

Chapter 5

Arithmetic

In digital CMOS technology, circuits conforming to the laws of Boolean logic are straightforward to implement [72, 73, 84, 86]. The logic values TRUE and FALSE are commonly represented by a low and high voltage level respectively.

Additionally, letting the low and high voltage levels represent the arithmetic numbers 0 and 1, binary arithmetic components are designed using logic gates. A binary digit, or "bit" for short, is a number with domain 0 and 1. Any number can be represented with a limited precision using binary digits [68]. It is common to use the two's complement binary number system for representing numbers, but other representations are used as well [68]. For digital signal processing applications, fixed point and floating point number representations are common.

5.1 Fixed and Floating Point Number Systems

Numbers are conveniently represented in a positional numbering system [51]. In such a system, there are several ways to represent negative numbers. A common representation of numbers using binary digits is the two's complement fixed point binary representation. An N-bit number x is written as

$$x_0.x_1x_2\cdots x_{N-2}x_{N-1}, \tag{5.1}$$

where each x_i is a binary digit. The leftmost bit, called the most significant bit, has negative weight, and the value of the number x is calculated by

$$x = -x_0 + \sum_{1}^{N-1} x_i 2^{-i}. \tag{5.2}$$

The position of the point separating integer from fraction bits is fixed, making it a "fixed point" representation. In equation (5.1), there is only one integer bit, and the value of the number is limited to the range $-1 \leq x < 1$. However, there can be any number of integer and fraction bits in a fixed point number. For a given sequence of bits, the fraction point is implicit, and has to be decided before the value of the number can be calculated.

Basic arithmetic operations does not depend on the position of the fraction point. Addition of two numbers, for example, can be performed as long as the fraction point is at the same position in both input numbers. A multiplication, on the contrary, is totally independent of what is fraction bits and what is integer bits. As already mentioned, the position of the fraction point does only make sense when the value of a fixed point number is to be calculated.

Floating point number systems consists of a significand and an explicit exponent [51]. To calculate the value of the number, the significand has to be multiplied by a base raised to the power of the exponent,

$$x = \pm \text{significand} \cdot \text{base}^{\text{exponent}}. \tag{5.3}$$

Comparing floating and fixed point numbers represented with the same number of bits, the floating point number has higher dynamics, whereas the fixed point number has a higher resolution. The floating point number gains dynamics from the exponent, but as the significand is represented with fewer bits, resolution will be lower. Floating point is therefore to be preferred when large dynamic range is more important than resolution, or when signal dynamics is unknown. For fixed point number representation, signal dynamics has to be known in order to determine how many integer and/or fractional bits that are required.

Today's high-end microprocessors and DSPs have floating point units of large wordlength, giving high dynamics. Large floating point number arithmetic has a higher cost, both in area and number of clock cycles per operation, but makes the processor more flexible. Since most signals are represented with a sufficiently high resolution, no time has to be spent solving overflow or accuracy problems. However, due to the overhead in the more costly floating point arithmetic circuits, and the excessively high dynamics, such a solution is always sub-optimal in terms of chip area and power consumption, and should be avoided for an application specific design.

5.2 Bit Serial and Bit Parallel Arithmetic

The most basic form of arithmetic uses binary digits, or a radix–2 representation, of numbers, while more advanced arithmetic circuits operate on digits of higher radices by combining several bits to represent one digit. The advantage

of low radix circuits is the ease of implementation, while higher radices gives advantages in certain cases.

In order to carry out for example an addition of two N-bit numbers, there are a number of way to organize the arithmetic circuits. One solution is to place N bit-adders, or *full-adders*, in parallel, feeding the carry signal from one full-adder to the other in a serial fashion [68]. In this way, the full-adders are working concurrently on all input bits in parallel. Therefore, this is called a bit-parallel adder. Although the adder has access to all input signals in parallel, worst case execution time of the N-bit adder is N times the delay of a single full-adder unit, from carry in signal to carry out. There are more efficient solutions to speed up adder circuits, for example carry-skip, carry-select, and carry-lookahead adders [68].

Using pipeline registers and feedback, the bit-serial solution is constructed using only one bit-adder working iteratively on all input bits. The bit-adder feeds the carry signal back to itself though a delay. This approach has minimal area, while requiring an N times higher clock rate. A solution in-between is the digit-serial approach, grouping bits to digits of a higher radix, and operating on one digit per clock cycle. Three different adder schemes are shown in figure 5.1.

The bit-serial and digit-serial architectures require less hardware to implement at the expense of a higher clock rate, when compared to a bit-parallel operation. Wave-digital filters are efficiently realized using bit-serial arithmetic [63, 84]. Operation on bit-serial data, input and output to the circuit must be in a serial format. However, high performance A/D and D/A converters[1] often have a parallel interface. In such cases, parallel to serial, and serial to parallel converters have to be used if arithmetic is carried out in a bit- or digit-serial fashion.

5.3 Arithmetic for Digital Signal Processing

Digital signal processing algorithms are expressed using the basic arithmetic operations addition, subtraction, multiplication, and division. The MAC is a common operation consisting of one multiplier and one adder connected as an accumulator. MAC, multiply, addition and subtraction are standard in most fixed and floating point DSPs, whereas division is not.

If the numbers are encoded in two's complement representation, the addition and subtraction operations are realized using the same hardware circuit. Multiplication and division are more expensive to implement compared to addition and subtraction. Arithmetic and logic integer shift operations, however,

[1] A/D (Analog to Digital) and D/A (Digital to Analog) converters are used to interface digital signal processing hardware to signals in the analog domain.

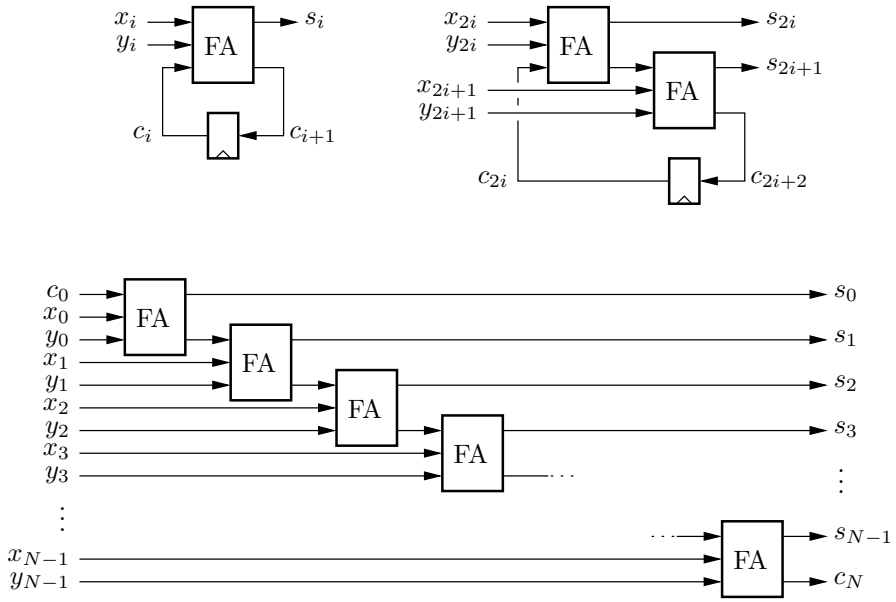


Figure 5.1: Bit-serial, digit-serial and bit-parallel adders. The FA module denotes a Full Adder. Top left is a bit-serial adder, processing one bit per clock cycle. Top right is a radix–4 digit-serial adder. Bottom is a bit-parallel adder, adding two numbers of wordlength N bits.

are of low or insignificant cost, and used to multiply or divide by powers of two. Multiplication of numbers is carried out as a series of conditional additions and small trivial digit multiplications.

Algorithms for division are implemented based on subtraction, or a combination of multiplications and subtractions. The main difference between multiplication and division of two's complement numbers is that the first is implemented without data dependency, while the latter is data dependent and requires decisions to be made for each digit. Data dependency makes a parallel implementation harder to realize. Division is often avoided, or limited in usage, and is not always implemented in a processor. For some applications, though, implementation of division is necessary.

For fixed point numbers, all four basic arithmetic functions increase data wordlength from input to output. Therefore, overflow detection, downscaling, or truncation has to be implemented as part of a datapath. Consider the operation on two numbers of N bits each. If the operation is an addition or subtraction, the output wordlength is maximum $N+1$ bits. For multiplication,

the output wordlength is $2N-1$ for two's complement numbers. Division is the most problematic operation. If the two input numbers are relatively prime, the division operation may go on forever, resulting in an infinite number of output digits. Thus, the division iteration has to be halted after a sufficient number of generated output bits.

5.4 Distributed Arithmetic

As an example of what kind of optimizations that are possible on the arithmetic level, this chapter gives an introduction to the concept of distributed arithmetic. Distributed arithmetic has been used in paper III and IV, and is an optimization technique on the arithmetic level, that generic processors or DSPs can not take advantage of.

The distributed arithmetic can be seen as a scheme to implement vector multiplication efficiently on a bit level [32, 69, 71], and is used in tailored architectures for digital filters and transforms, see for example [57, 81, 84], but it is not limited to such applications. Vector multiplication is a common operation in all areas of signal processing, and is therefore assigned many names: vector multiplication, vector scalar product, inner product, multiply and accumulate, sum of products, etcetera. Vector multiplication of length N can be written mathematically as

$$y = \sum_{n=0}^{N-1} h(n)x(n), \tag{5.4}$$

where $h(n)$ and $x(n)$ are variables of wordlengths W_h and W_x bits respectively. Assuming that the signal x is limited to $-1 \leq x < 1$, the value of x can be expressed in two's complement as

$$x = -x_0 + \sum_{i=1}^{W_x-1} x_i 2^{-i}, \tag{5.5}$$

and the corresponding $-x$ as

$$-x = -\overline{x}_0 + \sum_{i=1}^{W_x-1} \overline{x}_i 2^{-i} + 2^{-(W_x-1)}, \tag{5.6}$$

where each x_i is a one-bit variable with the value ZERO or ONE, that is $x_i \in \{0, 1\}$. The bar over $\overline{x}$ denotes the logic-NOT operation. Inserting (5.5) into

(5.4) results in

$$y = -x_0(n) \sum_{n=0}^{N-1} h(n) + \sum_{n=0}^{N-1} \sum_{i=1}^{W_x-1} h(n) x_i(n) 2^{-i}, \tag{5.7}$$

or by rearranging the order of summation,

$$y = -\underbrace{\sum_{n=0}^{N-1} h(n) x_0(n)}_{\text{I}} + \sum_{i=1}^{W_x-1} \underbrace{\sum_{n=0}^{N-1} h(n) x_i(n)}_{\text{II}} 2^{-i}. \tag{5.8}$$

This equation is close to the definition of a two's complement number from (5.5), but with more complicated terms, I and II. The braced sum I is an index $i = 0$ version of II, but can not be included in the sum over i since it is preceded by a minus sign. The observation of similarities leads to a simpler way to write equation (5.8). By defining a function F of N one-bit variables

$$F\big(x_i(0), x_i(1), \ldots, x_i(N-1)\big) = \begin{cases} -\sum\limits_{n=0}^{N-1} h(n) x_0(n) & : \quad i = 0 \\ \sum\limits_{n=0}^{N-1} h(n) x_i(n) & : \quad i \neq 0, \end{cases} \tag{5.9}$$

equation (5.8) is conveniently reduced to

$$y = \sum_{i=0}^{W_x-1} F\big(x_i(0), x_i(1), \ldots, x_i(N-1)\big) 2^{-i}. \tag{5.10}$$

The initial convolution of (5.4) has been turned into a sum. The function F can be pre-calculated and implemented as a Read Only Memory (ROM) containing 2^N words, one for each combination of the N input bits. ROM contents for $N = 3$ is shown in table 5.1.

Comparing the original equation (5.4) to (5.10), all multiplications are removed and replaced by lookups. Instead of a summation of N multiplications of size $W_x \times W_h$, the calculation is performed by summation of W_x lookup values of approximate length $W_h + \log_2 N$.

If only one ROM is used, the calculation of y has to be carried out in a serial fashion. A typical serial implementation uses an accumulator, adding one ROM lookup per clock cycle for W_x cycles [44]. A parallel implementation depend on W_x ROMs connected in parallel to an adder compressor with W_x inputs [84]. A hybrid serial-parallel solution is possible by having an accumulator with K parallel inputs from K ROMs, and iterating for W_x/K clock cycles.

Table 5.1: The F function for $N = 3$.

$x_i(0)$	$x_i(1)$	$x_i(2)$	$F\big(x_i(0), x_i(1), x_i(2)\big)$
0	0	0	0
0	0	1	$h(2)$
0	1	0	$h(1)$
0	1	1	$h(1) + h(2)$
1	0	0	$h(0)$
1	0	1	$h(0) + h(2)$
1	1	0	$h(0) + h(1)$
1	1	1	$h(0) + h(1) + h(2)$

For large values of N, implementation of the coefficient ROM becomes difficult, since the required coefficient storage grows exponentially as $\mathcal{O}(2^N)$. From a look at table 5.1, it becomes evident that each $h(n)$ is only present in half of the rows. For example, $h(0)$ is only used in the lower four rows. This suggests that it is possible to split F into two ROMs, one containing the permutations of $h(1)$ and $h(2)$, and one containing the value $h(0)$ only. The output of the latter is then added conditionally to the first depending on $x_i(0)$. In this example, ROM usage is decreased from eight words down to five.

A more formal method can be developed by splitting the function F of N variables into a set of functions of fewer variables by factoring $N = N_1 N_2$. Divide the input variables $\{x_i(n)\}_n$ into N_1 disjoint sets

$$\{x_i(n)\}_{n=0}^{N_2-1}, \quad \{x_i(n)\}_{n=N_2}^{2N_2-1}, \quad \dots \quad , \quad \{x_i(n)\}_{n=(N_1-1)N_2}^{N-1}. \tag{5.11}$$

Construct the corresponding functions $F_0 \dots F_{N_1-1}$ and add prior to accumulation

$$F = \sum_{k=0}^{N_1-1} F_k\big(x_i(kN_2+0), \dots, x_i(kN_2+N_2-1)\big). \tag{5.12}$$

Effective storage is reduced to $\frac{N}{N_2}2^{N_2}$, which is of $\mathcal{O}(N)$. The drawback is the extra additions represented by the summation in (5.12). These additions can be realized with carry-save adders at relatively low cost if the ROM is split in few parts, but with increasing N_1, the solution becomes less attractive. Consider the case with $N_2 = 1$. There are N ROMs of one word of storage each, or actually N ROMs storing zero and $h(n)$, but the zeroes are not implemented in the ROMs. All N words have to be added together before accumulation. This scheme corresponds to the straightforward implementation of (5.4), without the advantage gained by distributing the arithmetic.

5.5 Offset Binary Coding

Offset binary coding [24, 32] can further reduce coefficient storage. Write x as

$$2x = x - (-x), \tag{5.13}$$

and replace x by its value in two's complement from (5.5)

$$\begin{aligned} 2x &= -x_0 + \sum_{i=1}^{W_x-1} x_i 2^{-i} - (-x_0 + \sum_{i=1}^{W_x-1} x_i 2^{-i}) \\ &= -x_0 + \sum_{i=1}^{W_x-1} x_i 2^{-i} + (-\overline{x}_0 + \sum_{i=1}^{W_x-1} \overline{x}_i 2^{-i} + 2^{-(W_x-1)}) \\ &= -(x_0 - \overline{x}_0) + \sum_{i=1}^{W_x-1} (x_i - \overline{x}_i) 2^{-i} - 2^{-(W_x-1)}. \end{aligned} \tag{5.14}$$

The expression can be written more compact, by defining

$$\xi_i = \begin{cases} -(x_0 - \overline{x}_0) & : \quad i = 0 \\ (x_i - \overline{x}_i) & : \quad i \neq 0. \end{cases} \tag{5.15}$$

The definition or x_i is motivated by the domain

$$\xi_i \in \{-1, +1\}. \tag{5.16}$$

Now, equation (5.14) is written using ξ_i as

$$2x = \sum_{i=0}^{W_x-1} \xi_i 2^{-i} \quad - \quad 2^{-(W_x-1)}, \tag{5.17}$$

and the initial vector multiplication (5.4) becomes

$$y = \frac{1}{2} \sum_{n=0}^{N-1} \sum_{i=0}^{W_x-1} h(n) \xi_i(n) 2^{-i} \quad - \quad \underbrace{2^{-W_x} \sum_{n=0}^{N-1} h(n)}_{\text{extra}}. \tag{5.18}$$

The part marked "extra" is the sum of the coefficients multiplied by a weight corresponding to half the least significant weight of x. Neglecting this part makes the result slightly too big, but for applications where full accuracy is not required it can be disregarded in order to simplify hardware implementation [39].

Table 5.2: The F' function for $N = 3$.

$x_i(0)$	$x_i(1)$	$x_i(2)$	$\xi_i(0)$	$\xi_i(1)$	$\xi_i(2)$	$F'\big(\xi_i(0), \xi_i(1), \xi_i(2)\big)$
0	0	0	−1	−1	−1	$-(\,h(0) + h(1) + h(2)\,)$
0	0	1	−1	−1	1	$-(\,h(0) + h(1) - h(2)\,)$
0	1	0	−1	1	−1	$-(\,h(0) - h(1) + h(2)\,)$
0	1	1	−1	1	1	$-(\,h(0) - h(1) - h(2)\,)$
1	0	0	1	−1	−1	$(\,h(0) - h(1) - h(2)\,)$
1	0	1	1	−1	1	$(\,h(0) - h(1) + h(2)\,)$
1	1	0	1	1	−1	$(\,h(0) + h(1) - h(2)\,)$
1	1	1	1	1	1	$(\,h(0) + h(1) + h(2)\,)$

In accordance to equation (5.9), a function F' is introduced, replacing the innermost summation,

$$F'\big(\xi_i(0), \xi_i(1), \ldots, \xi_i(N-1)\big) = \sum_{i=0}^{W_x-1} h(n)\xi_i(n)2^{-i}. \tag{5.19}$$

As before, F' is stored in a ROM, and addressed when accumulating equation (5.18). The relation between ξ_i and x_i is straightforward. In table 5.2, F' is shown for $N = 3$. From the table it is clear that the values of F' are mirrored along the line between fourth and fifth rows, except for the sign. Therefore, using an external negation circuit, only half of F' is unique and need to be stored in a ROM. Thus, offset binary encoding reduce the coefficient storage by one half. Furthermore, the scheme of splitting the ROM in parts described earlier is also applicable on offset binary coded coefficients.

In some applications, distributed arithmetic and offset binary coding can be used without the need of ROM lookups. This is the case when inputs to the function already exists in the same format as would be stored in the ROM. Paper IV is an example where an FIR filter operate on data generated by an FFT. If the last butterfly stage of the FFT is removed, signals that connects directly to an distributed arithmetic multiplier are revealed. Thereby, complexity is reduced, both in the FIR filter and in the FFT units.

5.6 Summary

In CMOS, arithmetic operation units are straightforward to implement. For DSP applications, numbers are typically represented using two's complement

fixed point or floating point. The latter is easier for development, since the dynamic range for a fixed number of bits is larger, and problems of overflow or saturation are less frequent. However, fixed point representation comes at a lower cost, and is the most common choice for ASDSP implementations, especially where low power consumption is a concern.

Distributed arithmetic and offset binary coding is a means for efficient calculation of inner products. The inner product is common in DSP algorithms, for example as the basis of FIR filters. To gain from distributed arithmetic, datapaths have to be custom designed. Therefore, it can only be used in custom implementations such as ASIC ASDSPs, and not in standard DSP implementations.

Chapter 6

Conclusion and Summary of Papers

This thesis covers design methodology for Application Specific Digital Signal Processors (ASDSPs), mainly focusing on low power hardware implementations. Comparing ASDSPs to general purpose DSPs, high throughput and low power consumption are traded for flexibility of the final circuit. A delayless acoustic echo canceller is chosen as a target algorithm for custom hardware implementation, and subject of the developed design methodology proposed in the thesis.

A design methodology is presented to keep consistency between higher and middle level descriptions. The problem of controlling the dataflow is emphasized, and it is found that a hierarchical design approach reduces control complexity. Also, by strictly dividing processing elements into three sets: memory, datapath, and control, the problems of design, verification, and maintenance are reduced.

The fabricated echo canceller chip consists of more than two million transistors and twelve dedicated memory blocks. The design is successfully verified for functionality, throughput, and power consumption.

Paper I is an initial analysis of the chosen delayless echo canceller algorithm by Morgan and Thi [60]. The main parameters affecting the overall computational complexity of the algorithm are identified, and it is found that the number of subbands has the most impact on the number of operations per sample. Selecting the number of subbands for a low total complexity, the dominating part of the algorithm is the fullband FIR cancellation filter. A method to reduce power consumption of the filter is proposed, based on dividing the

filter architecture into a number of accumulators with different dynamics. The paper is equivalent to the presented paper [15] with a modified typesetting.

Paper II describes the custom hardware implementation of the echo canceller together with trade-offs and optimizations. The implemented architecture is composed of a hierarchy of autonomous submodules. This simplifies design of each module, and by using a simple inter-module communication protocol, operation of the complete design is controlled from the top-level. Memory addressing is investigated, and two cache memory candidates are identified. The fullband FIR filter is the most data requiring component, setting the lower limit on system clock frequency, and a redundant cache memory is inserted to have the filter operating at maximum pace. The paper is in preparation [5], and partial results are presented in [7].

Paper III presents the work on complex multipliers based on distributed arithmetic. To reduce power consumption of a complex multiplier, the number of partial products are reduced by distributed arithmetic and offset binary coding, and an adder tree is used for partial product addition. The original paper [17] is printed in its entity with typographic modifications. The multipliers were originally designed for an FFT chip for Terrestrial Digital Video Broadcast (DVB-T) transceivers developed in the Digital ASIC group [62, 41].

Paper IV presents a co-optimization between two digital signal processing algorithms, the FFT and the FIR filter, using a special arithmetic component. The component is based on distributed arithmetic, and the paper is an extension of the work on complex multipliers. The echo canceller algorithm is used as an example of where the optimization can be applied. Initially, the work was presented in [16], and the included paper is based on [6] that is submitted.

Paper V describes a flexible architecture for an efficient divider circuit. Division is common in digital signal processing applications, but the constraints on the divider derived from different application differ significantly. Thus, having only one fixed divider architecture for all ASDSP implementations results in sub-optimal performance. The proposed divider is configurable, and it is possible to explore a wide range of the design space using the parameter space of the divider description. The divider is used in the echo canceller implementation. The work is submitted as a paper [4].

Bibliography

[1] *ADS Documentation: SPS2 RAM generator for AMI Semiconductor 0.35 μm CMOS*, Alcatel Microelectronics.

[2] S. Asai, "Semiconductor memory trends," *Proceedings of the IEEE*, vol. 74, no. 12, pp. 1623–1635, Dec. 1986.

[3] U. Banerjee, *Loop Transformations for restructuring compilers: The foundations*. Kluwer Academic Publishers, 1993.

[4] A. Berkeman and V. Öwall, "A configurable divider using digit recurrence," submitted to *IEEE International Symposium on Circuits and Systems*, 2003.

[5] ——, "Custom silicon implementation of a delayless acoustic echo canceller algorithm," in preparation.

[6] ——, "Efficient implementation of an FFT-FIR structure using a distributed arithmetic multiplier," submitted to *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

[7] ——, "Architectural tradeoffs for a custom implementation of an acoustic echo canceller," in *Proceedings of the Nordic Signal Processing Symposium*, Hurtigruten from Tromsø to Trondheim, Norway, Oct. 2002.

[8] A. Berkeman, V. Öwall, P. Nilsson, P. Åström, and M. Torkelson, "A bit-serial implementation of a wavelet filter-bank," in *Proceedings of the Nordic Signal Processing Symposium*, Espoo, Finland, Sept. 1996.

[9] A. Berkeman, V. Öwall, and M. Torkelson, "A fast complex tree multiplier using distributed arithmetic," in *Proceedings of the NORCHIP Conference*, Tallin, Estonia, Nov. 1997.

[10] ——, "A complex multiplier with low logic depth," in *Proceedings of the 5th IEEE International Conference on Electronics, Circuits, and systems*, Lisbon, Portugal, Sept. 1998.

[11] ——, "A low logic depth complex multiplier," in *Proceedings of the 24th IEEE European Solid-State Circuits Conference*, The Hague, The Netherlands, Sept. 1998.

[12] ——, "An adder tree based complex multiplier," in *Proceedings of RVK Radio Science Conference*, Karlskrona, Sweden, June 1999.

[13] ——, "Implementation issues for acoustic echo cancellers," in *Proceedings of the 42th Midwest Symposium on Circuits and Systems*, Las Cruses NM, USA, Aug. 1999.

[14] ——, "Implementation of delayless echo cancellers for acoustic echoes," in *Proceedings of RVK Radio Science Conference*, Karlskrona, Sweden, June 1999.

[15] ——, "A prestudy of an echo canceler implementation," in *Proceedings of the International Conference on Signal Processing Applications and Technology*, Orlando FL, USA, Nov. 1999.

[16] ——, "Co-optimization of FFT and FIR in a delayless acoustic echo canceller implementation," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Geneva, Switzerland, May 2000.

[17] ——, "A low logic depth complex multiplier using distributed arithmetic," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 4, pp. 656–659, Apr. 2000.

[18] J. Bhasker, *A VHDL Primer*, 2nd ed. Prentice-Hall, 1995.

[19] ——, *A Verilog HDL Primer*, 2nd ed. Star Galaxy, 1999.

[20] G. M. Blair, "A review of the discrete Fourier transform. Part 1: Manipulating the powers of two," *IEE Electronics and Communication Engineering Journal*, vol. 7, no. 4, pp. 169–177, Aug. 1995.

[21] C. Breining, P. Dreiscitel, E. Hansler, A. Mader, B. Nitsch, H. Puder, T. Schertler, G. Schmidt, and J. Tilp, "Acoustic echo control: An application of very-high-order adaptive filters," *IEEE Signal Processing Magazine*, vol. 16, no. 4, pp. 42–69, July 1999.

[22] J. R. Brews, K. K. Ng, and R. K. Watts, "The submicrometer silicon MOSFET," in *Submicron Integrated Circuits*, R. K. Watts, Ed. John Wiley & Sons, 1989.

[23] F. P. Brooks, Jr, *The Mythical Man Month: 20th Anniversary Edition.* Addison-Wesley, 1995, ch. 1,2,3,4,8.

[24] M. Büttner and H.-W. Schüßler, "On structures for the implementation of the distributed arithmetic," *Nachrichtentechnik*, pp. 472–477, 1976.

[25] F. Catthoor, S. Wuytack, E. D. Greef, F. Balasa, L. Nachtergaele, and A. Vandecapelle, *Custom Memory Management Methodology.* Kluwer Academic Publishers, 1998.

[26] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. SC-27, no. 4, pp. 1082–1087, Apr. 1992.

[27] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, Apr. 1995.

[28] ——, *Low Power Digital CMOS Design.* Kluwer Academic Publishers, 1995.

[29] P. M. Clarkson, *Optimal and Adaptive Signal Processing.* CRC Press, 1993.

[30] D. Cohen, "Simplified control of FFT hardware," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-24, pp. 577–579, Dec. 1976.

[31] R. E. Crochiere and A. V. Oppenheim, "Analysis of linear digital networks," *Proceedings of the IEEE*, vol. 63, no. 4, pp. 581–595, Apr. 1975.

[32] A. Croisier, D. J. Esteban, M. E. Levilion, and V. Riso, "Digital filter for PCM encoded signals," US patent 3777130, Dec. 1973.

[33] J. M. de Haan, N. Grbić, I. Claesson, and S. Nordholm, "Design of oversampled uniform DFT filter banks with delay specification using quadratic optimization," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, May 2001.

[34] M. Dörbecker and P. Vary, "Reducing the delay of an acoustic echo canceller with subband adaption," in *4th International Workshop on Acoustic Echo and Noise Control*, 1995.

[35] P. Duhamel, "Implementation of "split-radix" FFT algorithms for complex, real, and real-symmetric data," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-34, no. 2, pp. 285–295, Apr. 1986.

[36] J. W. Eaton, *GNU Octave: A high-level interactive language for numerical computations: edition 3 for Octave version 2.0.5*, Free Software Foundation, `http://www.octave.org`, Feb. 1997.

[37] E. B. Eichelberger, E. Lindbloom, J. A. Waicukauski, and T. W. Williams, *Structured Logic Testing*. Prentice-Hall, 1991.

[38] T. Gänsler, M. Hansson, C.-J. Ivarsson, and G. Salomonsson, "A double-talk detector based on coherence," *IEEE Transactions on Communications*, vol. 44, no. 11, pp. 1421–1427, Nov. 1996.

[39] S. He and M. Torkelson, "A complex array multiplier using distributed arithmetic," in *Proceedings of IEEE Custom Integrated Circuits Conference*, 1996, pp. 71–74.

[40] ——, "A new approach to pipeline FFT processor," in *Proceedings of the International Parallel Processing Symposium*, 1996.

[41] ——, "Design and implementation of a 1024-point pipeline FFT processor," in *Proceedings of IEEE Custom Integrated Circuits Conference*, 1998.

[42] J. L. Hennesey and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 1990.

[43] *IEEE Standard for Low-Voltage Differential Signals (LVDS) for Scalable Coherent Interface (SCI)*, IEEE Standards Board, Mar. 1996.

[44] P. Ingelhag, B. Johnsson, B. Sikström, and L. Wanhammar, "A high-speed bit-serial processing element," in *Proceedings of ECCTD'89*, Brighton, UK, Sept. 1989, pp. 162–165.

[45] *OCAPI/RT User Manual version 0.81*, Interuniversity Micro-Electronics Centre, Kapeldreef 75, B-3001 Leuven, Belgium.

[46] K. Itoh, "Trends in megabit DRAM circuit design," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 3, June 1990.

[47] W. Kellerman, "Analysis and design of multirate systems for cancellation of acoustical echoes," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1988.

[48] B. W. Kernighan and R. Pike, *The UNIX programming environment*. Prentice-Hall, 1984.

[49] B. W. Kernighan and P. J. Plauger, *The elements of programming style*. McGraw-Hill, 1978.

[50] B. W. Kernighan and D. M. Ritchie, *The C programming language*. Prentice-Hall, 1978.

[51] D. E. Knuth, *The Art of Computer Programming*, 2nd ed. Addison-Wesley, 1981, vol. 2 / Seminumerical Algorithms.

[52] F. Kristensen, P. Nilsson, and A. Olsson, "A flexible FFT processor," in *Proceedings of the NORCHIP Conference*, 2002.

[53] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals: Architectures and Features*. Berkeley Design Technology, 1996.

[54] Y. Ma and L. Wanhammar, "A hardware efficient control of memory addressing for high-performance FFT processors," *IEEE Transactions on Signal Processing*, vol. 48, no. 3, pp. 917–921, Mar. 2000.

[55] D. Mansour and A. Gray, "Unconstrained frequency-domain adaptive filter," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 30, no. 5, pp. 726–734, 1982.

[56] *Using Matlab*, The Math works.

[57] M. Matsui *et al.*, "A 200 MHz 13 mm^2 2-D DCT macrocell using sense-amplifying pipeline flip-flop scheme," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 12, pp. 1482–1490, Dec. 1994.

[58] *M5M5W816TP-55HI Datasheet*, 6th ed., Mitsubichi Electric, Apr. 2002.

[59] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, Apr. 1965.

[60] D. R. Morgan and J. C. Thi, "A delayless subband adaptice filter architecture," *IEEE Transactions on Signal Processing*, vol. 43, no. 8, Aug. 1995.

[61] S. D. Naffziger and G. Hammond, "The implementation of the next generation 64b Itanium$^{\text{TM}}$ microprocessor," in *Digest of Technical Papers, IEEE International Solid-State Circuits Conference*, 2002.

[62] P. Nilsson, S. He, V. Öwall, A. Berkeman, S. Johansson, P. Åström, and M. Torkelson, "An FFT/IFFT chip for the european DVB system," Department of Applied Electronics, Lund University, Tech. Rep., Aug. 1997.

[63] P. Nilsson, M. Torkelson, M. Vesterbacka, and L. Wanhammar, "A bit-serial realization of a lattice wave digital intermediate frequency filter for mobile radio systems," in *Sixth Annual IEEE International ASIC Conference and Exhibit*, Sept. 1993, pp. 108–111.

[64] Z. Ning and R. W. Brodersen, "Architectural evaluation of flexible digital signal processing for wireless receivers," in *Conference Record of the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers*, vol. 1, 2000, pp. 78–83.

[65] V. Öwall *et al.*, "Custom DSP design of a GSM speech coder," *Journal of VLSI Signal Processing*, vol. 11, no. 3, pp. 213–228, 1995.

[66] V. Öwall, M. Torkelson, and P. Egelberg, "A custom image convolution dsp with a sustained calculation capacity of $>$ 1 GMAC/s and low I/O bandwidth," *Journal of VLSI Signal Processing*, pp. 335–349, Nov. 1999.

[67] V. Öwall, "Synthesis of controllers from a range of controller architectures," Ph.D. dissertation, Lund University, Department of Applied Electronics, 1994.

[68] B. Parhami, *Computer Arithmetic: Algorithms and hardware designs*. Oxford University Press, 2000.

[69] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. John Wiley & Sons, 1999.

[70] M. C. Pease, "Organization of large scale Fourier processors," *Journal of Association for Computing Machinery*, vol. 16, pp. 474–482, July 1969.

[71] A. Peled and B. Liu, "A new hardware realization of digital filter," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-22, no. 6, Dec. 1974.

[72] P. Pirsch, *Architectures for Digital Signal Processing*. John Wiley & Sons, 1996.

[73] J. M. Rabaey, *Digital Integrated Circuits: a design perspective*. Prentice-Hall, 1996.

[74] E. S. Raymond, Ed., *The New Hackers Dictionary*. MIT Press, 1996.

[75] K. Roy and S. C. Prasad, *Low-Power CMOS VLSI Circuit Design*. John Wiley & Sons, 2000.

[76] D. M. Samani, J. Ellinger, E. J. Powers, and E. E. S. Jr., "Implementation of several RLS nonlinear adaptive algorithms using a commercial floating point digital signal processor," in *Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, 1993, pp. 1574–1578.

[77] C. B. Shung, R. Jain, K. Rimey, E. Wang, M. B. Srivastava, B. C. Richards, E. Lettang, S. K. Azim, L. Thon, P. N. Hilfinger, J. M. Rabaey, and R. W. Brodersen, "An integrated CAD system for algorithm-specific IC design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. Vol. CAD-10, pp. 447–463, Apr. 1991.

[78] M. M. Sondhi, "An adaptive echo canceller," *Bell Systems Technical Journal*, vol. XLVI, no. 3, pp. 497–510, Mar. 1967.

[79] M. M. Sondhi and W. Kellerman, "Adaptive echo cancellation for speech signals," in *Advances in Speech and Signal Processing*, S. Furui and M. M. Sondhi, Eds. New York: Marcel Dekker, 1992, ch. 11.

[80] S. Theodoridis and M. G. Bellanger, "Adaptive filters and acoustic echo control," *IEEE Signal Processing Magazine*, vol. 16, no. 4, pp. 42–69, July 1999.

[81] S. I. Uramoto *et al.*, "A 100-MHz discrete cosine transform core processor," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, Apr. 1992.

[82] P. P. Vaidyanathan, *Multirate Systems and Filterbanks*. Prentice-Hall, 1993.

[83] H. J. M. Veendrick, "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits," *IEEE Journal of Solid-State Circuits*, vol. 19, no. 4, pp. 468–473, 1984.

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

[85] B. Widrow and S. D. Sterns, *Adaptive Signal Processing*. Prentice-Hall, 1985.

[86] W. Wolf, *Modern VLSI Design: A Systems Approach*. Prentice-Hall, 1994.

[87] D. J. Yang, "On Moore's law and fishing: Gordon Moore speaks out," *U.S. News*, Oct. 2000.

[88] K. Yano *et al.*, "A 3.8-ns CMOS 16×16-b multiplier using complementary pass-transistor logic," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, Apr. 1990.

Part II

Included Papers

Paper I

Paper I

A Prestudy of an Echo Canceller Implementation

The high computational complexity of acoustic echo cancellation algorithms requires application specific implementations to sustain real time signal processing with affordable power consumption. This is especially true for systems where a delayless approach is considered important, e.g. wireless communication systems. The proposed paper presents architectural considerations to reach a feasible hardware solution.

Based on: A. Berkeman, Viktor Öwall, and Mats Torkelson, "A Prestudy of an Echo Canceller Implementation," in *Proceedings of the International Conference on Signal Processing Applications and Technology (ICSPAT)*, Orlando FL, USA, Nov. 1999.

1 Introduction

This paper presents implementation aspects of an Application Specific Digital Signal Processor (ASDSP) designed to perform acoustic echo cancellation. The need for acoustic echo cancellers arises in systems where two or more people positioned at different locations are having a conversation using loudspeakers and microphones, for example a teleconferencing system. The problem is the acoustic path from loudspeaker to microphone. When the far end talker is speaking, the speech signal is fed to the near end loudspeaker, it enters the room and goes back to the far end through the microphone. Due to the various delays in the signal path, such as the acoustic delay from loudspeaker to microphone, coding delay, transmission delay etc, this returned signal is percepted as an annoying echo. The acoustic impulse responses considered have a duration in the order of hundreds of milliseconds. With a sample rate of 16 kHz this corresponds to roughly 2000-4000 samples. Such long impulse responses make a fullband approach unattractive for both convergence and complexity considerations and a subband approach is investigated [1].

There are two major candidates for implementation, where the first one is performing cancellation in the subband domain [2]. This algorithm is depicted in figure 1.1. The signals from the far end, $x(n)$, and from the microphone,

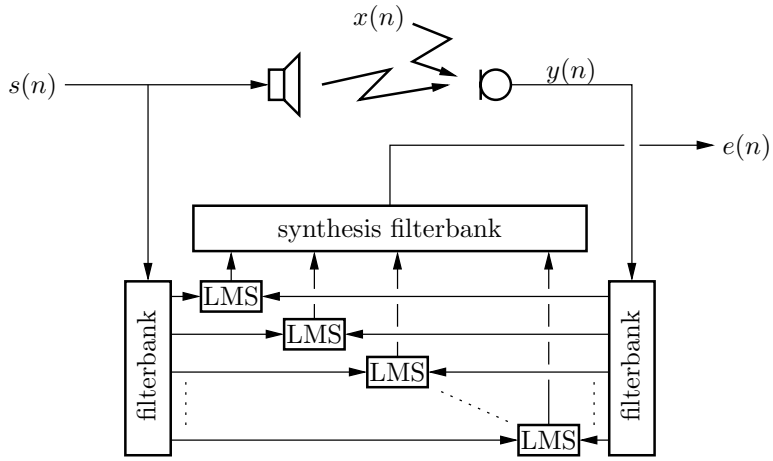


Figure 1.1: The echo cancellation algorithm from [1]. The signals to the loudspeaker, $x(n)$, and from the microphone, $y(n)$, are split into subbands, and cancellation is performed in each subband prior to fullband reconstruction.

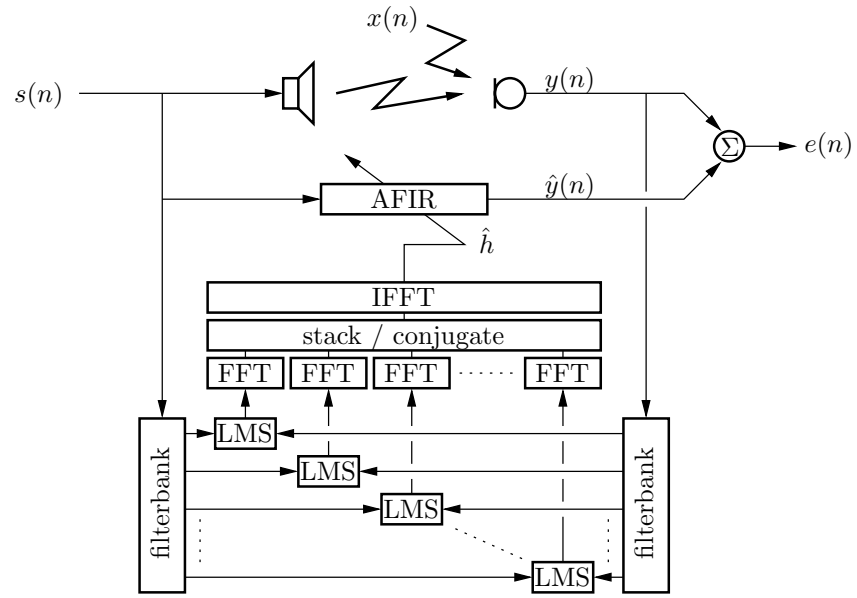


Figure 1.2: The subband echo cancellation algorithm, from [2]. Echo cancellation is done in the time domain.

$y(n)$, are filtered through filterbanks. Impulse response estimation is done on the corresponding subband signals, and the error signals for each subband adaptive filter are put together to a fullband signal using a synthesis filterbank. In this algorithm, there is a delay in the signal path of at least one filterbank delays.

The second candidate performs the actual echo cancellation in the time domain [1], see figure 1.2. This is achieved by a fullband finite impulse response (FIR) filter that convolves the input signal with the estimate of the room acoustic impulse response. In this algorithm, there is no delay in the signal path, but the estimate of the room acoustic impulse response is lagged. Due to the extra fullband FIR filter, this solution has higher complexity than the algorithm in figure 1.1. There is also a third approach which is a combination between the two [3].

In wireless and multimedia applications, delay in the signal path is a serious obstacle. Therefore, the delayless cancellation algorithm will be investigated. The computational complexity of the algorithms makes a standard digital signal processor implementation difficult, especially in a mobile terminal scenario were power consumption is a key parameter. Therefore, an application specific

solution is investigated to increase the throughput at the same time as the power consumption is reduced.

2 The Cancellation Algorithm

The main parts of the algorithm are shown in figure 1.2. To the upper left the far end signal $x(n)$ enters and connects to the canceller and to the loudspeaker. The signal from the microphone $y(n)$ is the other input to the canceller. Between the loudspeaker and the microphone is the acoustic signal path, where echo and a near end talker signal $v(n)$ is added. After the microphone, the far end signal with estimated echoes are subtracted, optimally leaving nothing but the near end talker. This signal $e(n)$ is then fed back to the far end.

The heart of the algorithm is a number of adaptive Least-Mean-Square (LMS) filters that track the frequency response from speaker to microphone. Each adaptive filter acts on a small frequency band of the x and y signals. If the number of subbands is large, each subband has a narrow bandwidth with near flat frequency response, and therefore convergence of the adaptive filters is fast.

The estimated filter taps from one adaptive filter represent a part of the total impulse response in a certain frequency band. To make a fullband impulse response, these taps have to be Fourier transformed, stacked in frequency and inverse transformed. This fullband impulse response is, due to the delays in the filterbanks, adaptive filters and FFTs, a delayed estimate of the room acoustics. It is used to filter the far end signal to simulate the effect of the sound traveling through the room. The difference between this filtered signal and the microphone signal should be close to zero, except for the sound added in the near end.

For the echo cancellation algorithm to work properly, the adaptive filters should try to make an estimate of the room acoustics only when there is energy in the far end signal, and when there is no double-talk. Double-talk is when the near and far end speakers are talking simultaneously, a case when it is impossible to estimate a transfer function from speaker to microphone. This situation is monitored by a signal coherence detector that turns off estimation as long as the double-talk situation lasts. The problem with silent far end is solved by an energy detector that switch off the adaption when the far end signal energy goes below a certain threshold.

3 Analysis of the Algorithm

3.1 the filterbanks

An M band filterbank consists of M finite impulse response (FIR) filters of length K, where each filter selects a part of the signal spectrum. For every input sample, KM multiplications are executed in the filters, and M outputs are calculated, one for each subband. This can be seen as an upsampling by a factor M in each band. All these extra samples have to be processed by the following parts of the algorithm, giving a large overhead in the number of operations per sample. To reduce the amount of data, the outputs from the filterbank are downsampled by a factor M/α.

Applying downsampling at the output decreases the computational burden in the filterbanks, since every filter only has to be updated every M/α sample. Furthermore, if the M frequency functions of the filters are chosen as shifted versions of one prototype filter H, the polyphase filterbank approach can be used [1], [4]. The number of taps in each filter can then be decimated from K to K/M. The price for this reduction is an extra IFFT of size M at the outputs of the filters, updated every α/M input sample period, but the total number of operations in the filterbank is reduced considerably.

3.2 the lms filters

For each input sample, the output from one band of the filterbank is α/M samples. Due to symmetry in the frequency plane, only $M/2+1$ bands contain unique information [1], and thus there are $M/2+1$ LMS filters to be updated, each every α/M input sample period. This results in a total of

$$\frac{\alpha}{M}(\frac{M}{2}+1) = \alpha(\frac{1}{2}+\frac{1}{M}) \tag{3.1}$$

LMS filters to be updated every sample period. If M is reasonably large, the expression is close to $\alpha/2$ and does not vary much with M.

3.3 generation of fullband filter taps

The adaptive weights calculated in the $M/2+1$ LMS filters are combined into a fullband impulse response of length N. This is achieved by Fourier transforming the LMS weights, stacking them in frequency into a fullband frequency function, and inverse Fourier transform to get an impulse response. If each adaptive filter is of length T, T/α weights are taken from the Fourier transform of the middle $M/2-1$ filters, and $T/2\alpha$ from the Fourier transform

of the left- and rightmost filters. Together this gives

$$(\frac{M}{2}-1)\frac{T}{\alpha}+2\frac{T}{2\alpha}=\frac{M}{2}\frac{T}{\alpha}=\frac{N}{2} \tag{3.2}$$

bins that are combined into a fullband frequency function of length N. The bins are stacked from position 0 to $N/2-1$, and then the complex conjugate is repeated in reversed order from $N/2$ to $N-1$. From equation (3.2) the length of each LMS filter can be calculated as

$$T=\alpha\frac{N}{M} \tag{3.3}$$

3.4 COMPLEXITY ANALYSIS

Combining equation (3.1) and (3.3) gives to hand that the number of LMS weights to be processed per input sample clock period is

$$\alpha^2\frac{N}{M}(\frac{1}{2}+\frac{1}{M}), \tag{3.4}$$

which is inverse proportional to M.

How M affects the total complexity of the algorithm is shown in table 3.1. The table is created assuming a fullband filter length N of 2048 taps and a constant α of two. Coherence and power detectors are not included in these numbers.

The first column is the number of subbands, ranging from two to above a thousand. The second column is an approximation of the required number of real multiplications per second in millions. The number of additions in the LMS and FIR filters are about the same as the number of multiplications, due to the intensive use of the multiply-accumulate (MAC) operation. A multiplication has several times higher complexity than an addition, and therefore only multiplications are considered in this table. It is assumed that a complex multiplier corresponds to four real number multipliers, but their implementation complexity can be reduced to about half that number [5]. Divisions have higher complexity, but are not used as frequent as multiplications in this algorithm.

Column three to five show how much of the total number of calculations that are carried out by the filterbanks, adaptive filters and the fullband FIR filter respectively. In the simulations the filter was updated every 128th sample [1]. This corresponds to the additional percents in table 3.1 and goes from 2% for $M = 2$ to 17% for $M = 1024$. If a higher update rate is desired, this will increase the total complexity considerably making an ASDSP solution even more advantageous.

Table 3.1: The number of operations for the echo canceller as a function of the number of subbands, M, in millions of multiplications per second. The three columns to the right shows the percentage of the multiplications spent by the filterbanks (FB), adaptive filters (LMS) and fullband time domain filter (FIR) respectively. The remaining percents are due to the fullband impulse response reconstruction.

M	Mmul/s	FB(%)	LMS(%)	FIR(%)
2	889	4	90	4
4	366	5	82	9
8	179	5	69	18
16	105	4	53	32
32	72	4	36	46
64	57	3	22	58
128	49	3	13	67
256	45	3	7	72
512	43	3	4	76
1024	42	3	2	78

A large number of subbands M gives a low overall complexity of the implementation. It can be seen from the table that as the number of subbands increase the dominating factor becomes the time domain fullband filter as contrary to the adaptation algorithm for a lower number of subbands. The drawback is the increased requirements on the prototype filter in the subband filterbanks, since the frequency response of the filter must have a more high defined selectivity. As the signal is never reconstructed from the subband decomposition, there are no "perfect reconstruction" criteria. Since the frequency response of the filters in a filterbank are abutted, care has to be taken in the transition part of the prototype filter to make the group delays between two filters equal.

For large values of M, the passband region for each filter in the filterbank becomes very narrow. Therefore, a signal filtered by such a filterbank can be considered to have a near flat power spectrum, see figure 3.1. This is beneficial for the convergence of the adaptive filters. So, not only does a large M reduce the number of operations, but the echo canceller will also have a faster convergence time. Simulations have shown that for a fullband filter length of $N = 2048$ taps, an M value of 256 to 512 is suitable.

As shown in table 3.1, for M greater than 32 the heaviest computational burden is in the fullband FIR filter. For every input sample, this filter calculates

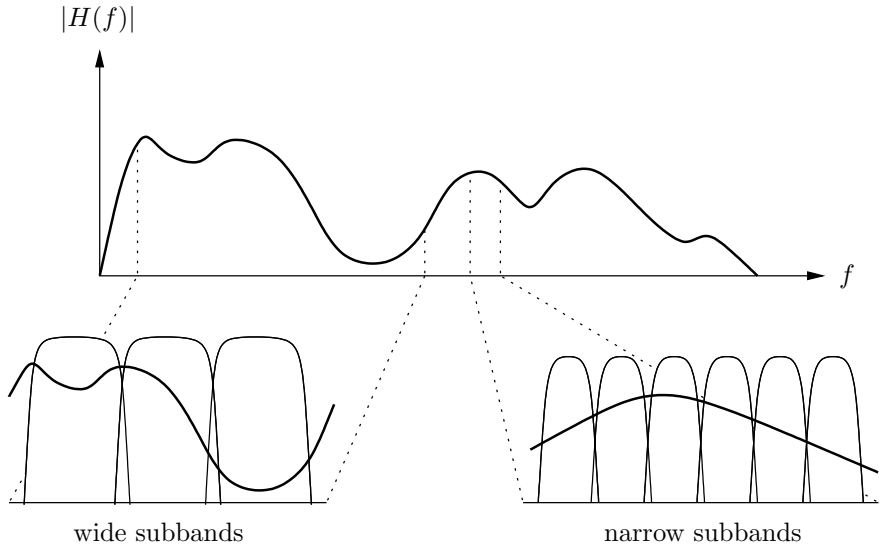


Figure 3.1: Having many subbands result in a near flat spectrum in each subband.

2000-4000 multiplies and accumulations. For a sample rate of 16 kHz, this corresponds to 32 to 64 million multiplies and accumulates per second. Also, the taps in this filter are updated hundreds of times per second to keep track of changes in the room acoustics.

4 Optimizations

4.1 wordlengths

Wordlength affects performance in several ways. Small lengths consume less power and chip area, i.e. buses, computational units and memories should have minimum widths. On the other hand, the wordlengths also determine the performance in terms of resolution and dynamic range. Therefore, it is important to keep signals wide enough to keep overflow and rounding errors at a minimum. These optimizations are performed on each bus, arithmetic unit and memory, so that different hardware units can have different wordlengths. The effects of this truncation into fixed point are investigated using simulations.

4.2 MEMORY MANAGEMENT

Memories are often a bottle-neck in digital signal processors (DSPs), since a memory has a certain number of ports and a limited access frequency. Furthermore, every access has a cost in energy. The larger the memory, the larger energy consumption. In a typical ASDSP the power consumed by memory accesses is a substantial part of the total consumption. By memory management the number of accesses to large memories is reduced by rescheduling the arithmetic operations and utilizing memory hierarchies [6].

4.3 THE SUBBAND FILTERBANK

There are two filterbanks in the algorithm, one filtering the signal x that comes from the far end, and one filtering the signal y from the near end. These two filterbanks are equal and consists of a bank of FIR filters followed by an IFFT. The coefficients of the FIR filters are real valued in the application, making the inputs to the IFFT real. Fourier transforms of real sequences can be reduced in the number of operation as compared to complex transforms. Since there are two real valued transforms in the algorithm, one solution is to feed the outputs of one FIR bank to the real input of a complex IFFT, and the outputs from the other FIR bank to the imaginary IFFT input. The output of the IFFT can then be separated into the transform of the two real data sequences by two additions and one complex conjugation operation per output bin [7]. This reduces the number of computations in the IFFTs by slightly less than a half.

4.4 UPDATING THE LMS FILTERS

The subband structure of the algorithm is well-suited for the necessary energy and coherence detectors. These detectors could work independently in each subband, giving the freedom to turn off adaption at only those frequencies where the far end signal does not contain any energy or has no coherence to the near end signal. This gives better convergence of the LMS filters and a lower number of operations in average.

4.5 THE FULLBAND FIR FILTER

Since the taps are an estimate of a room acoustic impulse response, they have a certain shape. By exploring the properties of typical impulse responses, some optimizations of the fullband FIR filter can be made. Figure 4.1 shows a recorded impulse response. It begins with zeros for a time corresponding to the shortest distance from speaker to microphone, and then a set of peaks with

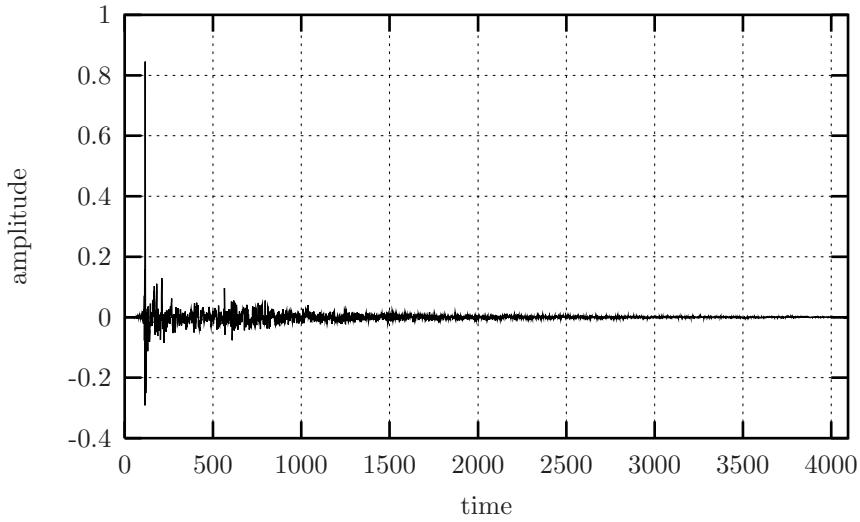


Figure 4.1: A typical acoustic impulse response of a conference room. The numbers on the horizontal axis is sample number at a sample rate of 16384 Hz. Note the different amplitude characteristics along the horizontal axis.

high amplitude, created by the direct wave and the first main reflections. From there on, the amplitude is decreasing.

To cover the full dynamics of the impulse response, a multiply and accumulate unit with very large dynamic is required. If, on the other hand, the amplitude pattern of the impulse response is explored the filter can be split in parts, where each part has its own multiply and accumulate unit with certain dynamics. This gives a significant decrease in the number of bit transitions per sample, and a corresponding decrease in power consumption.

5 Hardware Mapping

One extreme implementation would be to map the algorithm directly onto hardware, i.e. each operation is directly mapped onto a dedicated hardware module. Due to the high complexity of the algorithm the number of computational units would be very large at the same time as the clock frequency would be extremely low, and therefore this is not an attractive solution.

One solution is to reuse hardware and have a few components that perform

several different operations for each input sample. For example one adaptive filter circuit can be used to calculate the filter coefficients for all the subbands. The extreme here is a standard signal processor with only a single multiplier and adder, performing all computations sequentially. The challenge is to find a trade-off between speed, power and flexibility in order to find a solution with good characteristics.

A modest clock rate should be used, since a low clock rate gives the opportunity to lower the supply voltage and thereby reduce power [8]. A sample rate of 16 kHz and a clocking frequency of the processor of 16 MHz gives 1000 clock cycles per sample, which is reasonable for the algorithm. Thus, the application specific architecture enabling the low clock frequency and parallel processing reduce power consumption.

When hardware is reused for different tasks, there is a need for a controller to schedule the operations to the correct computation unit at the correct moment. For this scheduling to be as effective as possible, careful investigation of the memory access pattern for various algorithmic transformations is done [6]. There is also a need for a microprogrammed approach where the controller is synthesized from a high level description will facilitate flexibility during the implementation process [9].

6 Summary

For a hardware implementation of an acoustic echo canceller, an application specific digital signal processor solution is chosen. This solution gives a streamlined architecture in terms of hardware utilization and costly memory accesses, yielding realtime signal processing at an affordable overall power consumption. This is especially important in wireless multimedia systems.

Important aspects for the implementation are optimization of wordlength for buses and arithmetic units, determining of crucial parameters such as the number of subbands and construction of the control unit for steering the dataflow such that a good memory management is achieved.

References

[1] D. R. Morgan and J. C. Thi, "A delayless subband adaptice filter architecture," *IEEE Transactions on Signal Processing*, vol. 43, no. 8, Aug. 1995.

[2] M. M. Sondhi and W. Kellerman, "Adaptive echo cancellation for speech signals," in *Advances in Speech and Signal Processing*, S. Furui and M. M. Sondhi, Eds. New York: Marcel Dekker, 1992, ch. 11.

[3] M. Dörbecker and P. Vary, "Reducing the delay of an acoustic echo canceller with subband adaption," in *4th International Workshop on Acoustic Echo and Noise Control*, 1995.

[4] J. E. R. Ferrara, "Frequency-domain adaptive filtering," in *Adaptive Filters*, C. F. N. Cowan and R. M. Grant, Eds. Prentice-Hall, 1985, ch. 6.

[5] A. Berkeman, V. Öwall, and M. Torkelson, "A low logic depth complex multiplier," in *Proceedings of the 24th IEEE European Solid-State Circuits Conference*, The Hague, The Netherlands, Sept. 1998.

[6] F. Catthoor, S. Wuytack, E. D. Greef, F. Balasa, L. Nachtergaele, and A. Vandecapelle, *Custom Memory Management Methodology.* Kluwer Academic Publishers, 1998.

[7] H. V. Sorensen, D. L. Jones, M. T. Heideman, and S. Burrows, "Real-valued fast Fourier transform algorithms," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-35, 1987.

[8] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, Apr. 1995.

[9] V. Öwall *et al.*, "Custom DSP design of a GSM speech coder," *Journal of VLSI Signal Processing*, vol. 11, no. 3, pp. 213–228, 1995.

Paper II

Paper II

Custom Silicon Implementation of a Delayless Acoustic Echo Canceller Algorithm

An acoustic echo canceller is a necessary device in handsfree telecommunication systems. This paper presents a hardware implementation of an acoustic echo canceller for high quality applications using handheld devices. The implementation is based on an algorithm with no delay in the signal path, attractive for telecommunication systems where low signal path delay is crucial. However, the zero delay comes with the price of increased complexity. A custom silicon implementation fulfills quality and realtime operation while sustaining a low power consumption. The fabricated chip contains two million transistors, and occupies 29 mm^2 in a 0.35 μm CMOS process. At 16 MHz clock frequency, the chip processes 16 bit samples at a rate of 16 kHz, while consuming 55 mW for uncorrelated input data. The main algorithm as well as optimizations on the architecture and arithmetic level are presented together with measurements on the fabricated chip. The custom implementation is compared to two standard DSPs, and it is estimated that power consumption is reduced more than one order of magnitude using the dedicated echo canceller chip.

Based on: A. Berkeman and Viktor Öwall, "Custom Silicon Implementation of a Delayless Acoustic Echo Canceller Algorithm," in preparation.

1 Introduction

Acoustic echo cancellers are necessary for communication systems where there is a direct acoustic path from the loudspeaker to the microphone of the communication device. Examples of such applications are conference telephones, hands-free sets, and future applications such as communication using stationary or handheld computers, where earphones, cables and external microphones will be considered circumstantial.

This paper presents an Application Specific Digital Signal Processor (ASDSP) implementation of an acoustic echo canceller algorithm. The implementation is targeted towards high quality communication applications, as it has high dynamics, allows high sample rates, offers a low power consumption, and has no delay in the signal path. Signal path delay is a serious problem in telecommunications, since the turnaround time of such systems is large, due to delay in speech codecs, channel codecs, etcetera. The algorithm chosen for the implementation is based on subband estimation and time domain cancellation, providing a zero delay in the signal path at a higher implementation complexity compared to other canceller solutions.

The increased complexity leads to a high power consumption and device usage in a general purpose DSP implementation. An important aspect for handheld, battery powered devices, is the power consumption — the lower power consumption, the more longevity. High performance and low power consumption are major advantages of an Application Specific Integrated Circuit (ASIC), and is therefore favored.

Section 1.1 gives an initial description of the chosen algorithm, together with a complexity analysis. In section 2, more detailed descriptions of the operations and optimizations of the involved sub-modules are given. Thereafter, the architecture and implementation of the sub-modules is presented in section 3. Here, implementation issues regarding memory and controllers are explained. Section 4 contains details of the actual hardware implementation, design methodology, and results. Furthermore, the verification strategy used for the design is described. Finally, conclusions appear in section 5.

1.1 The echo canceller algorithm

The echo canceller algorithm chosen for implementation is presented in [1], and a block diagram is shown in figure 1.1. To the upper left the far end signal $x(n)$ enters and connects to both the canceller and loudspeaker. The signal from the microphone $y(n)$ is the other input to the canceller. An estimate, $\hat{h}$, of the impulse response of the acoustic path between loudspeaker and microphone is computed by the canceller. In the acoustic path, a near end talker signal $s(n)$

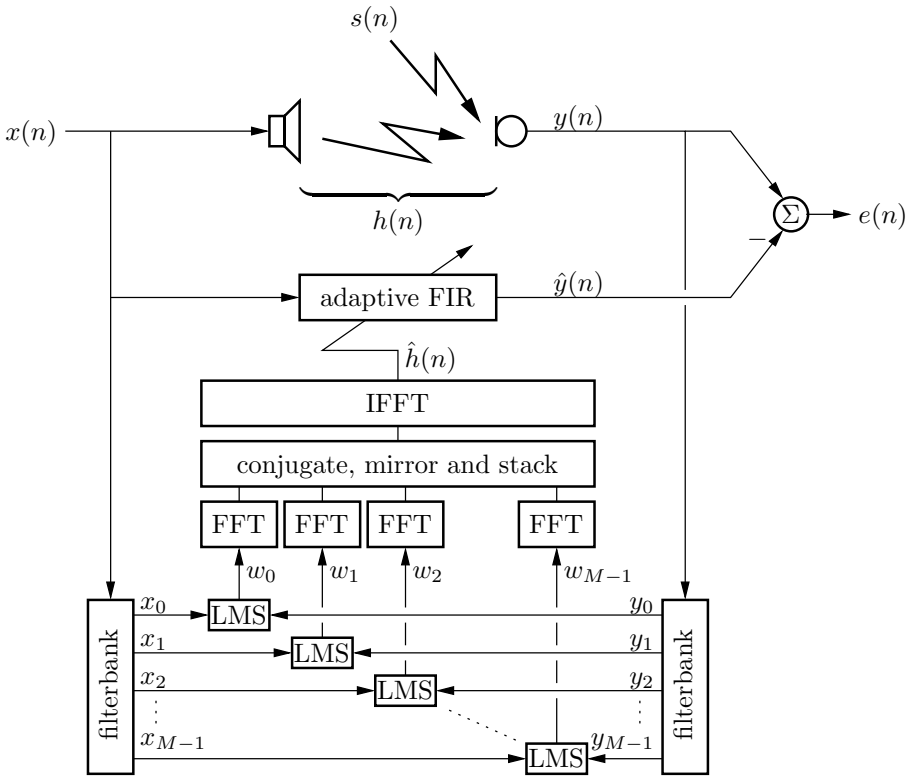


Figure 1.1: Schematic diagram of the delayless acoustic echo canceller.

plus noise is added. After the microphone, the far end signal with estimated echoes are subtracted, optimally leaving nothing but the near end talker. The cancelled signal $e(n)$ is then fed back to the far end. The subtraction operation is the only operation in the signal path, and does not contribute to any output sample delay.

Two analysis subband filterbanks are used to divide the input signals x and y into uniform subbands. The subband signals are input to a set of adaptive Least Mean Squares (LMS) filters, one for each subband, tracking the frequency response from speaker to microphone. If the subband filterbanks are designed with care, adaption rate of the LMS filters will be high, because the spectral range is reduced in each subband [1, 2].

The time domain impulse response is created by transforming all M subband impulse responses to the frequency domain, and assembling a frequency

function from parts of all the subbands. Thereafter, the assembled estimate is transformed back to the time domain. Transformation to the frequency domain and back again is done by the Fourier transform. Assembling of the transformed filter taps is performed in a way that assures the estimate of the time domain impulse response is real valued.

In order for the canceller to estimate the acoustic path from loudspeaker to microphone, it is important that update of the LMS filters is performed only when the near end talker is silent. Furthermore, for the adaption algorithm to work, there has to be energy in the loudspeaker signal. To solve the first problem, denoted double-talk, a double-talk detector is used. The detector can be implemented as a coherence detector between the near and far end signals [3], with the ability to freeze the LMS update when the far and near end speaker talks at the same time [3]. An energy detector halting adaption when the loudspeaker signal energy is too low solves the second issue.

Compared to traditional subband cancellers based on synthesis and analysis filterbanks [4], the complexity added by the fullband impulse response update and adaptive filter is substantial. In the compared algorithms, cancellation is performed in the subband domain, and the error signal $e(n)$ is generated by a synthesis filterbank, collecting the estimation error from each subband into a time domain signal. However, the chosen algorithm provides a zero signal path delay, whereas the other algorithms have a significant delay from one analysis and one synthesis filterbank connected in series.

Another canceller approach with no delay in the signal path is proposed in [5]. Here, part of the cancellation is performed in the subband domain, and part in the time domain. For a hardware implementation, this algorithm requires both a time domain and a subband domain cancellation unit plus the additional conversion of some taps from subband to time domain, and is considered to be unnecessary complicated and irregular.

1.2 COMPLEXITY ANALYSIS

The parameter having most impact on the overall computational complexity of the algorithm is the number of subbands, M [6]. The affect of M on the total complexity of the algorithm is shown in table 1.1. The table is created assuming a time domain adaptive filter length $N = 1024$, a length of the filterbank impulse response $K = 512$, a sample rate $F_S = 16384\,\mathrm{Hz}$, and an update rate of the fullband filter of every 32nd sample. Furthermore, operations related to double-talk and power detectors are not included in the analysis. The parameters are the same as chosen for the hardware implementation. As will be discussed later, the implementation is written in a hardware description language, with all parameters generic. A canceller with any set of parameters

Table 1.1: Number of million multiplications per second for the echo canceller, assuming a fullband filter length $N = 1024$ taps, a filter update rate of once every 32nd sample, and a sample rate of 16384 Hz. The four columns to the right shows the relative number of multiplications spent in the filterbanks (FB), estimation filters (LMS), fullband filter update (UPDATE), and fullband time domain filter (FIR) respectively. The FIR operation is not a function of M, and sets a lower bound on complexity. Note that the numbers for the filterbanks are calculated given a polyphase implementation, see text.

		relative #muls/second (%)			
M	Mmul/s	FB	LMS	UPDATE	FIR
2	474	7	85	4	4
4	205	8	74	10	8
8	107	8	59	18	16
16	68	7	42	26	25
32	50	6	27	34	34
64	41	4	16	39	41
128	36	4	9	41	46
256	33	4	5	41	50
512	31	4	3	40	53

can be elaborated and manufactured from the same description. Having a filter length of 1024 taps is sufficient to prove the concept while keeping the test chip to a reasonable size in the current silicon manufacturing process.

The table data was created by making two assumptions regarding the complexity of the filterbanks. First, "half critical downsampling" is applied, affecting the overall computational rate of the algorithm [1]. Second, the filterbanks are implemented using a polyphase technique, which significantly reduces the number of multiplications per second [7]. Downsampling and polyphase implementation are further discussed in section 2.1.

Since a multiplication has several times higher implementation cost than an addition, only multiplications are considered in table 1.1. Due to the nature of the involved signal processing algorithms, the multiply and accumulate (MAC) operation is frequently used and the number of additions is in the same order of magnitude as the number of multiplications. It is assumed that a complex multiplier corresponds to four real number multipliers, but their implementation complexity can be reduced to about half that number using special arithmetic operations possible to realize in an application specific device [8]. The division operation used by the normalized LMS has high complexity, but is not used as

frequent as multiplication in the algorithm, and is therefore not considered in the analysis.

The first column of table 1.1 is the number of subbands, ranging from two to five hundred. The second column is an approximation of the required number of real multiplications per second in millions. The third to sixth show the relative number of multiplications (in percent) performed in the filterbanks, LMS filters, update of the time domain impulse response estimate, and fullband FIR filter respectively. As seen in the table, for a large number of subbands, the dominating factor is the adaptive fullband FIR filter. To the contrary, for few subbands, the least mean squares filters dominate. The larger the number of subbands, M, the lower the total complexity.

A disadvantage of having many subbands is the increased requirements on the prototype filter in the subband filterbanks [2]. A division into more narrow subbands requires the filter to have well defined selectivity characteristics. Since the frequency response of the filters in a filterbank are abutted, care has to be taken in the transition part of the prototype filter to make the group delays between two filters equal [1]. However, as the signal is never reconstructed from the subband decomposition, there are no "perfect reconstruction" criteria to meet. Since only a fraction of the computations are due to the filterbanks, the prototype filter can be realized using a large number of taps with a high resolution, without having much impact of the total implementation complexity. The limit of the filter length is instead the tolerable amount of filterbank delay.

For M greater or equal to 32 the heaviest computational burden is in the fullband FIR filter. In an echo canceller implementation running at a sample frequency of 16 kHz, a sufficient value of N would be in the range of 1000 to 4000 taps [2]. For every input sample, this fullband FIR filter calculates 1000-4000 multiplies and accumulations. Also, the taps in this filter are updated hundreds of times per second to keep track of changes in the room acoustics. Considering the number of multiplications per second only, an implementation on a standard DSP would require a rather high performance processor, having correspondingly high power consumption.

2 Sub-Module Analysis

The echo canceller algorithm consists of a number of sub-algorithms, or sub-modules. In this section, four different sub-algorithms are described in more detail: the filterbanks, LMS filter, the impulse response update, and the time domain adaptive filter.

2.1 FILTERBANKS

A uniform M band filterbank consists in its most basic form of M finite impulse response (FIR) filters of length K, where each filter selects a part of the signal spectrum. For every input sample, KM multiplications are executed to calculate one output sample for each of the M subbands. The operation can be seen as an upsampling by a factor of M, since M outputs are generated for each input sample. All these new output samples have a large redundancy, and have to be processed by the algorithm parts succeeding the filterbanks, giving a large overhead in the number of operations per sample. In order to reduce the amount of data and computations, the outputs from the filterbank are downsampled by a factor M/α, where $\alpha = 1$ corresponds to critical downsampling. Downsampling decreases the computational burden in the adaptive LMS filters operating on filterbank output data, since every LMS filter has to be updated only every α/M sample after downsampling. For the implementation, "half-critical" downsampling, corresponding to $\alpha = 2$, is applied [1].

There are two M-band filterbanks in the algorithm, filtering the input signal $x(n)$ that comes from the far end, and the input signal $y(n)$ from the near end. The characteristics of the two filterbanks are equal, and therefore the same filter kernel coefficients can be used for both banks. Actually, as will be shown in section 3.1, both the x and the y filterbanks can be computed using the same hardware unit.

Initial studies of the computational burden for the different parts of the algorithm revealed that the two filterbanks required a large amount of computations per input sample. A way to significantly reduce the number of arithmetic operations per input sample for the filterbanks is to implement them using a polyphase approach [7]. The polyphase filterbank can be used if the M frequency functions of the subband filters are chosen as frequency-shifted versions of one prototype filter H [1, 7]. Assuming that filter H consists of K filter taps, total number of FIR multiplications to update all filterbank outputs can then be decimated from KM to K. The reduction comes at the expense of a size-M inverse discrete Fourier transform at the outputs of the FIR filters. Therefore, the algorithm is particularly efficient when M is a power of 2, and a fast Fourier transform algorithm can be used. For a radix–2 implementation, the total number of multiplications for the filterbank operation is reduced from KM to $K + M/2\ \log(M)$. Thus, by the polyphase technique, the total number of operations in the filterbank is reduced considerably. Note that the number of multiplications per sample is even lower, since the filterbanks are downsampled, and updated only every α/M input sample.

The coefficients of the prototype filter H are real valued in the application, making the inputs to the IFFTs real. Fourier transforms of real sequences

can be reduced in the number of operation compared to complex valued transforms [9]. Since there are two real valued transforms in the algorithm, one for the x and one for the y data, one solution is to feed the outputs of the two FIR banks to the real respectively imaginary data input of one complex IFFT. The output of the IFFT can then be separated into the transform of the two real data sequences by two additions and one complex conjugation operation per output bin [9]. This reduces further the number of computations in the IFFTs by slightly less than a half.

2.2 LMS FILTERS

Adaption is implemented using the Least Mean Squares (LMS) algorithm, which has reasonable complexity and accuracy, and is known to be stable. Rate of convergence is an important performance issue. The LMS filters suffer from non-uniform convergence for correlated input data, such as speech signals [10]. The problem is reduced by letting the LMS filters operate on subband domain signals [2, 10]. To further increase adaption rate, a common enhancement to the LMS algorithm is to normalize the input signal by its energy, yielding the Normalized LMS (NLMS) algorithm, which has a dramatically improved convergence rate.

Since the inputs to the filterbanks are real valued, only $M/2 + 1$ bands contain unique information [1], and thus there are $M/2 + 1$ LMS filters to be updated, each every α/M input sample period. This results in a total of

$$\frac{\alpha}{M}(\frac{M}{2} + 1) = \alpha(\frac{1}{2} + \frac{1}{M}) \tag{2.1}$$

LMS filters to be updated every sample period. If M is reasonably large, the expression is close to $\alpha/2$.

The subband structure of the algorithm is well-suited for the necessary energy and double-talk detectors. These detectors could work independently in each subband, making it possible to turn off adaption in frequency bands where the far end signal does not contain any energy or has no coherence to the near end signal. This gives better convergence of the LMS filters and a lower number of operations in average. However, the hardware implementation has to be designed for the worst case, leaving enough time to update all the LMS filter if necessary.

For an implementation of the LMS algorithm using a fixed point number representation with limited dynamics, special care has to be taken. With a strong input signal, convergence will be fast and precise. With a weaker signal however, adaption will likely proceed in the wrong direction, and performance will be poor. Therefore, an energy threshold circuit is implemented as part of

the NLMS unit. For every subband, the energy is checked against a threshold. If the energy is too low for adaption, the LMS filter will be turned off for that subband.

2.3 OBTAINING THE FULLBAND FILTER COEFFICIENTS

The adaptive weights calculated in the $M/2+1$ LMS filters are combined into a fullband impulse response of length N. This is achieved by Fourier transforming the LMS weights to the frequency domain, assembling them in frequency into a fullband frequency function, and inverse Fourier transform to get a time domain impulse response [1].

The time domain impulse response estimate consists of N taps, and thus N unique bins in the frequency domain. Due to the downsampling of a factor M/α, each subband need $\alpha/M \cdot N$ unique taps. This number is denoted T, thus

$$T = \frac{\alpha}{M} N. \tag{2.2}$$

According to [1], T/α frequency bins are taken from the Fourier transform of the middle $M/2-1$ filters, and $T/2\alpha$ from the Fourier transform of the left- and rightmost filters. Together this gives that

$$(\frac{M}{2}-1)\frac{T}{\alpha} + 2\frac{T}{2\alpha} = \frac{M}{2}\frac{T}{\alpha} = \frac{N}{2} \tag{2.3}$$

unique bins that are combined into a fullband frequency function of length N. The bins are assembled from position 0 to $N/2-1$, and then the complex conjugate of the bins is repeated in reversed order from $N/2+1$ to $N-1$. Bin $N/2$ is set to zero.

Figure 2.1 depicts how the assembly is implemented. Diagrams (a) to (c) show the frequency spectrum of the three first subbands, while diagram (d) illustrates the fullband spectrum, assembled from frequency contents from subband 0 to $M/2$. Each subband spectrum is downsampled a factor M/α, which is equal to N/T according to equation (2.2). Therefore, the Fourier transform of the subband impulse responses will have T unique taps only, since the fullband impulse response is of length N. In the diagrams (a) to (c), these T taps are repeated periodically to fill the range from 0 to $N/2$. The assembling of the fullband response is implemented for $\alpha = 2$ by taking half the number of unique taps from each subband, except the first and last, where only a quarter of the information is used.

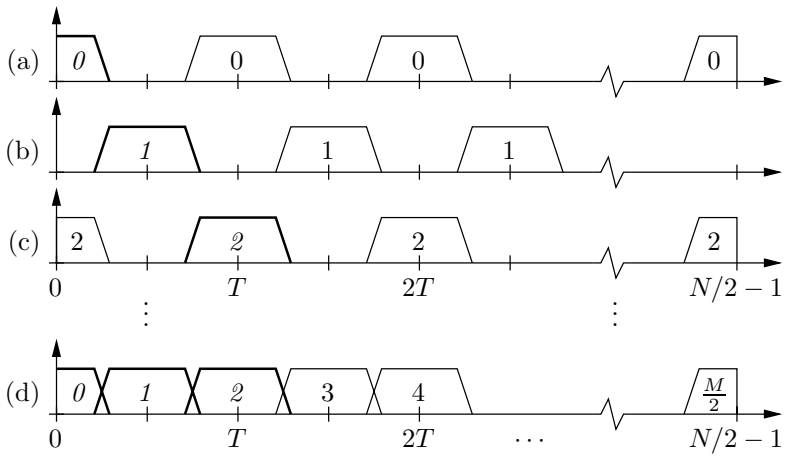


Figure 2.1: Assembly of the fullband impulse response estimate from a set of FFT transformed subband estimates. (a) to (c) show subbands zero, one, and two, with periodic aliases. Due to downsampling, the spectrum of the subbands have T unique taps. The bottom graph (d) shows the assembled spectrum of the impulse response. There are N unique taps in this spectrum, and the $N/2$ taps not shown in the image are mirrored conjugate copies of taps 0 to $N/2$. The thicker bands show what parts that are copied from each subband to the fullband spectrum.

2.4 ADAPTIVE TIME DOMAIN FIR FILTER

The adaptive time domain FIR filter performs a time domain convolution of the input signal $x(n)$ and the estimated impulse response $\hat{h}$. As shown in table 1.1, the dominating computational complexity is in the adaptive filter. Therefore, it is reasonable to assume that a large quantity of the dynamic power consumption is due to the arithmetic operations performed in the filter. For a low power implementation, it needs to be carefully designed. Implementation is further discussed in section 3.4.

3 ARCHITECTURE

As part of the ASDSP design process, an efficient hardware architecture is developed. The architecture has to be designed to make hardware mapping of the algorithm straightforward. In one end of the design space of possible architectures, is the hardware-mapped approach, where the signal-flow graph of the algorithm is mapped directly to arithmetic hardware units and wires.

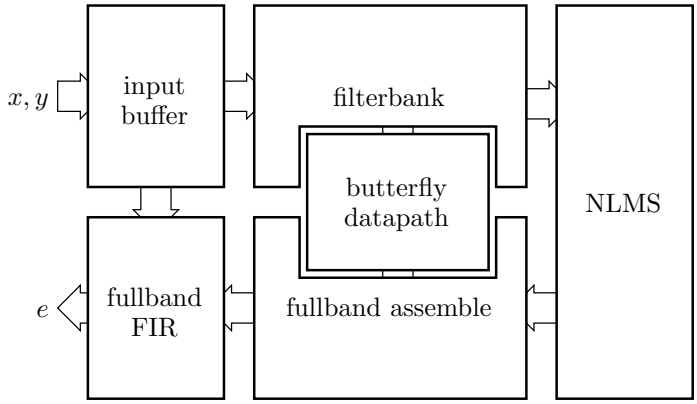


Figure 3.1: The top-level block diagram of the implementation.

In the other end of the design space is the generic datapath approach, where a programmable controller is used to execute operations on data using one flexible datapath. The datapath is designed to cover all possible arithmetic operations of the algorithm, and is time-shared between the operations.

For the echo canceller algorithm, operating on speech data with a relatively low sample rate, the hardware-mapped approach is infeasible. Consider for example the adaptive fullband filter, having a thousand taps. If each filter tap is implemented as a unique hardware multiplier, the implementation will be extremely parallel while operating at a low clock frequency. On the other hand, the generic datapath is required to run at a high clock frequency, to be able to execute all operations necessary per input sample using only one time-shared datapath.

The best solution lies somewhere in-between. A suitable system clock frequency can be found by looking at the fullband adaptive filter. The adaptive filter is, from a computational complexity perspective, the dominating part of the algorithm. If the filter is implemented using a single filter-tap processor, computing one filter-tap per clock cycle, the system clock speed is required to be at least N cycles per input sample. For a sample rate of 16 kHz, this corresponds to a clock frequency of 16 MHz, which is easily attained in the used silicon manufacturing process. All datapaths are then designed based on the constraint of N clock cycles per sample.

The architecture developed for the echo canceller implementation is shown as a block diagram in figure 3.1. A more detailed image is shown in figure 3.4. The architecture is composed of six major sub-modules, discussed in detail next.

3.1 FILTERBANK

The filterbank sub-module performs filterbank operations on both the $x(n)$ and $y(n)$ input signals. As mentioned in section 2.1, both filterbanks are identical in functionality and can therefore be computed using one shared hardware unit. The filterbank unit is implemented using three sub-blocks, an FIR filter, an IFFT, and a real-imaginary transform splitter. Convolution of the input signals and the prototype filter kernel is performed in the FIR filter. The $x(n)$ and $y(n)$ input signals are alternatingly fed in bursts of K/M samples to the FIR filter, generating two output signals, one written to the real part input of the IFFT processor, and the other at the imaginary part input as described in section 2.1. When all samples are convolved, the IFFT performs only one transformation, and the real-imaginary transform splitter reads the complex IFFT output, and converts it to the independent IFFTs of two real valued input signals. The IFFT processor takes advantage of a central butterfly processor sub-module, described further in section 3.5, in order to reduce the amount of redundant hardware.

3.2 NLMS MODULE

The NLMS module consists in turn of four sub-modules. Two sub-modules for the LMS algorithm itself, a FIR filter and a LMS filter coefficient update module; and two sub-modules for energy calculation and energy normalization.

The energy calculator keeps track of the energy of the last T taps in each subband, and sets a flag for all subbands having an energy too low for adaption, used to temporary stall estimation in a subband. This efficiently prevents the LMS from doing an iteration step in the wrong direction due to a too low or quantized signal. This is also the location of the double-talk detectors. However, there are no double-talk detectors implementation in the first version of the silicon chip.

The normalizer is implemented as a fixed point divider, using an efficient implementation of the SRT algorithm [11], dividing the input samples to the adaptive circuit by their energy.

3.3 FULLBAND ASSEMBLE

The fullband assemble module updates the filter kernel of the adaptive FIR filter using information from all subband LMS filter taps. The module consists of three parts: a small T sized FFT, a data reorder and conjugate module, and a large N point IFFT. The data reorder module performs the frequency stacking and data conjugation described in section 2.3. The FFT and IFFT

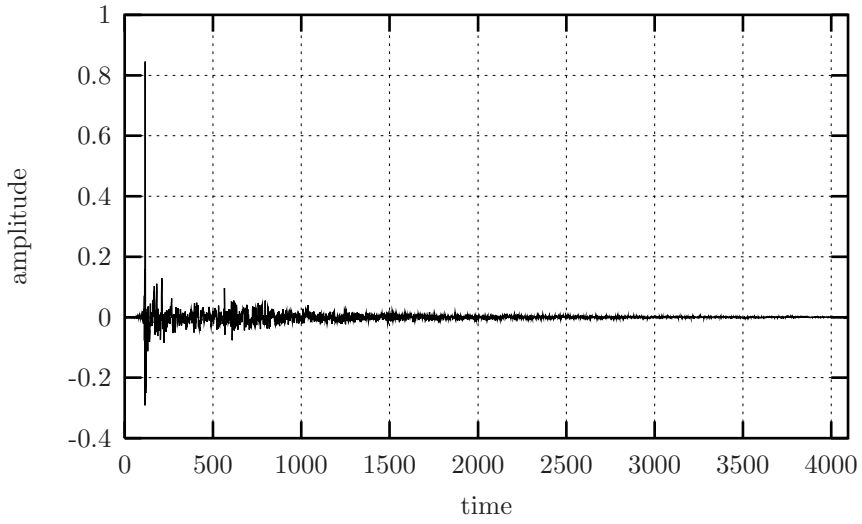


Figure 3.2: A typical acoustic impulse response of a conference room. Note the different amplitude characteristics along the time axis.

modules only contain the necessary transform controllers, since the modules rely on the central butterfly processor resource, which it further shared with the filterbank IFFT processor, and the topic of section 3.5. Since the fullband assemble module can not be run more often than the filterbanks, there is no danger of collision when sharing the butterfly processor with the filterbank sub-module.

3.4 FULLBAND FIR FILTER

As shown previously in section 1.2, the output time domain adaptive FIR filter alone consumes about 50% of the total number of multiplications of the canceller algorithm when M is reasonably large, i.e. $M \geq 32$ [6]. Therefore, it is beneficial to implement the FIR filter in a way that has a low switching activity and thus a low power consumption. This is achieved by using a tree compressor for the multiplier, and a carry save adder for the accumulator. The tree compressor has a low logic depth, and the carry save adder performs addition using a redundant number format, with no power consuming carry ripple transitions. To have a non-redundant output of the filter, the result has to be carry-rippled only once. Only one carry ripple operation for every Nth addition in the FIR filter is a significant reduction in signal activity.

Since the filter taps are an estimate of a room acoustic impulse response, they have a certain shape. By exploring the properties of typical impulse responses, some optimizations of the fullband FIR filter can be made. Figure 3.2 shows a recorded impulse response. It begins with zeros for a time corresponding to the shortest distance from speaker to microphone, and then a set of peaks with high amplitude, created by the direct wave and the first main reflections. From there on, the amplitude is decreasing.

To cover the full dynamics of the impulse response, a multiply and accumulate unit with large wordlength is required. If, on the other hand, the amplitude pattern of the impulse response is explored the filter can be split in parts, where each part has its own multiply and accumulate unit with unique dynamics. This gives a significant decrease in the number of bit transitions per sample, and a corresponding decrease in power consumption. This has not been implemented in the test chip, though.

By utilizing the input-output characteristics of a second order distributed arithmetic multiplier, the fullband FIR and IFFT can be optimized, resulting in a solution with less power consumption. The idea is to replace the N multipliers in the fullband FIR filter with $N/2$ distributed arithmetic multipliers, at the same time making the last butterfly stage of the IFFT unnecessary [12]. This reduces the arithmetic switching in the FIR and IFFT, as well as the number of accesses to the large memory addressed by the IFFT, thereby lowering the power consumption. However, this co-optimization has not been included in the first version of the chip.

An FIR filter with a high number of operations per sample, as the fullband filter, has a high requirement on data throughput. For every filter tap, one coefficient and one sample value has to be read. This sets special constraints on how the coefficients and sample data are stored. For the echo canceller chip, based on single-port memory technology, the fullband filter memory access count sets a lower bound on the clock rate of the design, as long as only one memory instance is used for storage of the estimate.

The location of the FIR filter in the canceller architecture is depicted in figure 3.3. The FIR filter convolves data read form the input buffer and the fullband assemble sub-modules. The situation is complicated for two reasons:

> The sample data in the input buffer is also read and used by the filterbank. Every M/α input sample, when the filterbank is executing, there is a possible collision situation.
>
> The filter coefficients are generated by the fullband assemble sub-module, intermittently writing new coefficients to the memory that the fullband FIR filter is reading from. The impulse response estimate used in the

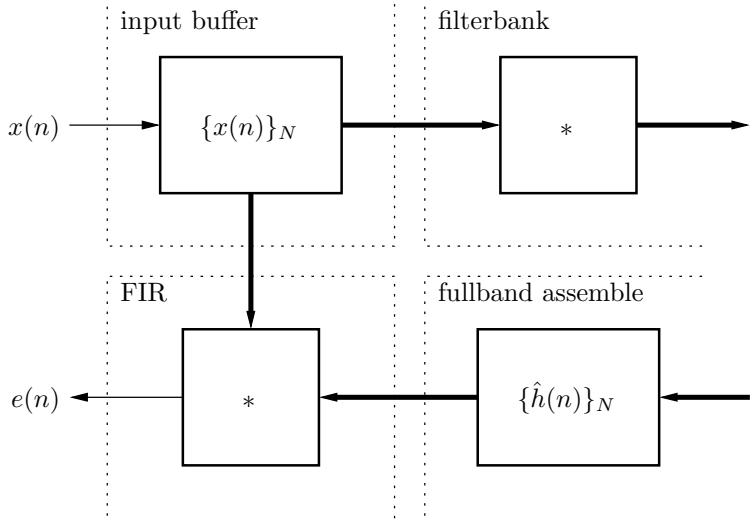


Figure 3.3: The fullband FIR filter convolves data from the input buffer and from the update sub-block. The blocks with an asterisk denote FIR filters, and $\{x(n)\}_N$ refers to a memory containing the last N samples of $x(n)$.

filter has to be updated between filter operations, to avoid "clicking" sounds in the output.

The system clock rate could be increased to have more memory accesses per input sample, thereby time-sharing the memory accesses between modules. However, this solution is not attractive for power consumption reasons. Instead, there are other efficient solutions to the problem. The first problem is solved as follows. The data in the input buffer is accessed on every new input sample by the fullband FIR filter. When the filterbank is executing, it listens to the data stream between the input buffer and the FIR filter. This can be arranged if the filterbank and the FIR filter process the same data at the same pace.

The second problem has to be solved by inserting an extra memory instance, since the impulse response estimate is generated by an IFFT processor, computing output data at a significantly slower pace than processed by the FIR filter. Without the extra memory, there would be a loss in sound quality when the filter coefficients are updated simultaneously to a filter computation. The solution is presented in more detail when investigating cache memory possibilities in section 3.6.

3.5 FFT BUTTERFLY

There are two modules in the architecture that use either the FFT or IFFT: the fullband impulse response reconstruction described in section 2.3, actually containing both an FFT and an IFFT, and the polyphase filterbank sub-module. These three transforms are all of different size and wordlength, one is forward transform while the other two are inverse transforms. Implementing the transform processors with small and efficient controllers, the dominating hardware in terms of area and power is the butterfly datapath. The overall timing of the design allows for the three transforms to share the same butterfly hardware, thereby reducing chip area. Resource sharing is achieved by scaling the butterfly datapath to the largest used wordlength, and by implementing the ability to do the required IFFT integer downscaling. Furthermore, the transforms can also share the same coefficient ROM if the twiddle factors can be conditionally conjugated when performing inverse transformations.

3.6 MEMORY CONSIDERATIONS

Memories are often a bottle-neck in digital signal processors (DSPs), since a memory has a certain number of ports and a limited access frequency. One example is the fullband FIR filter described in section 3.4. Every access has a cost in energy, the larger the memory, the larger energy consumption. In a typical application specific DSP, the power consumed by memory accesses is a substantial part of the total consumption [13].

To have a more power efficient solution, a memory management methodology has to be applied. A goal of the methodology is to have the most number of accesses to smaller memories, giving a higher throughput and a lower overall power consumption. This is achieved by rescheduling arithmetic operations and utilizing memory hierarchies [13]. Memory instantiation was investigated during the implementation process. There are three ways to instantiate a memory:

Global, or accessible to any sub-module in the design.

Local, that is, accessible only to one block. This is equal to embedding the memory instance inside a sub-module.

Partially global, accessible to a few sub-modules only.

The design strategy for the echo canceller implementation has been to keep independent data separated in different memories, and to keep data as close to the data path as possible. This strategy has several benefits: it increases memory bandwidth, since all disjoint memories can be accessed independent of

each other; it simplifies control circuitry, since bus arbitration can be neglected; it reduced power consumption since most accesses are to small memories and follow localized access patterns since all data in a memory is related. Furthermore, a division into several smaller memory instances does not affect the total amount of data storage in the design.

Cache memories or cache memory hierarchies is a means to increase data bandwidth and reduce power [13]. The idea is to have one or a set of smaller memories that mirrors a part of the contents of a large memory. If the large memory is accessed frequently in a certain address range, power is saved if this range is copied to a smaller memory, and read from there instead. The total number of reads to the memory hierarchy becomes unchanged, but most accesses target the smaller memory only.

Possible cache memory instantiation has been investigated, and two memories have been identified as candidates for caching. The first is the memory where the subband information of $x(n)$ are stored after the filterbank. This memory has T words of complex data for $M/2$ subbands. The memory is read twice in a short address range of T words, once by the NLMS FIR filter, and once by the NLMS update circuit. A cache memory of T words would decrease the number of accesses to the larger memory by 50%, while increasing throughput of the NLMS with 100%, since the FIR and update circuitry could work in parallel for consecutive subbands. However, hardware for the NLMS is "fast enough", and the gain in throughput is negligible in this application.

The second candidate for caching is the memory that contains the time-domain impulse response estimate. This memory is read continuously by the convolution unit of the fullband FIR filter. For every new input sample, N read accesses are targeting the memory. Every time a new impulse response estimate is calculated, the same memory is written to by the fullband assemble unit. Thus, there are two units addressing the same memory. The scheme is introduced in section 3.4, and depicted in figure 3.3.

Furthermore, the update of the estimate is generated by an IFFT processor, implemented using a time-shared architecture, executing the transform in-place. The IFFT processor requires approximately $2N \log_2 N$ clock cycles to perform one transform. Thus, if the IFFT and FFT share one memory instance, clock rate has to be at least $N + 2N \log_2 N$ times the input sample rate. Since the IFFT is operating only once every M/α input sample, the overhead in clock frequency is unacceptable.

Instead, the solution is to use an additional cache memory from which the FIR filter reads the impulse response estimate. When a new estimate is updated and written by the IFFT, the FIR filter instead address the IFFT memory for N clock cycles, performing the convolution while simultaneously updating the cache memory. This extra memory is redundant and increases the total chip

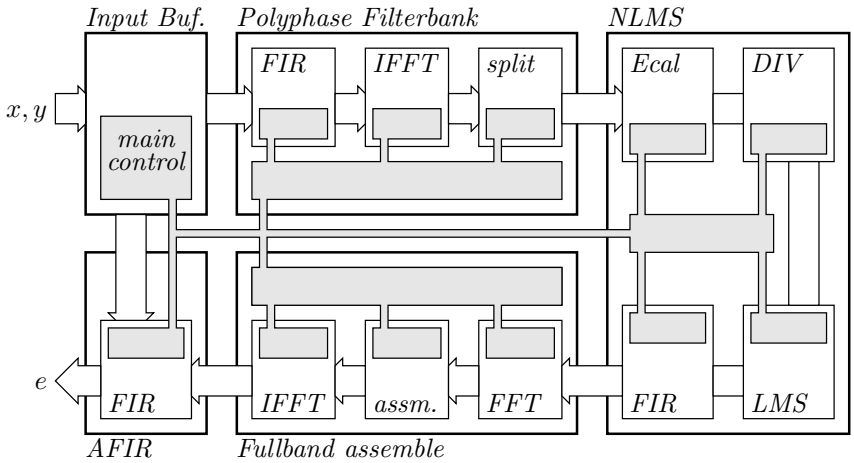


Figure 3.4: The echo canceller hardware architecture. Parts belonging to the controller hierarchy are shaded gray.

area by approximately 5%.

3.7 CONTROLLER HIERARCHY

With an increasing number of components in a datapath, control becomes more difficult. A way to keep the complexity of the control circuitry low is to distribute the controller locally in the design, that is, a specific data path is steered from a local controller unit close by. However, with several controllers interacting, intercommunication problems such as deadlocks may arise.

Although the echo canceller consists of a number of different functional modules realizing more or less complicated algorithms, data flows in a "left to right" manner between modules, which simplifies inter-controller communications. In particular, most modules in the design have only two input control signals, input signal `DO` and output `IDLE`. If a module is signaling `IDLE`, it can be trigged to run by asserting the `DO` signal synchronously for one clock cycle.

In figure 3.4, a more detailed structure of the architecture is revealed. The controllers are arranged in a three level hierarchy, and is visualized as the shaded boxed and wires in the image. The master controller is located in the input buffer module. When the input buffer receives a new pair of input signals, the master controller sends a sequence of `DO` signals to the other four main modules. After a number of clock cycles, all involved modules have computed their part of the calculation, and the master controller is notified that all modules are idle and ready for a new input sample. In this fashion, the system

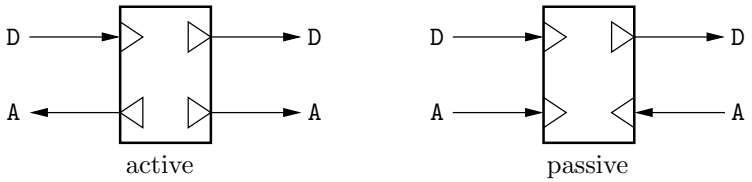


Figure 3.5: The two kinds of modules. The active module contains an addressing controller that access the neighboring blocks when reading and writing input and output data. The passive module operates on data stored in its internal memory, and relies on its neighbors to read and write data to the memory.

clock rate is arbitrary, as long as it provides enough clock cycles for each input sample.

Sub-modules containing a datapath, a controller, and possibly a RAM memory are divided into two disjoint groups based on behavior: active and passive modules, see figure 3.5. The active module contains a controller that upon activation reads its input from a neighboring module or memory. It performs some computation, either on the fly or on buffered data, and writes out the result in another neighboring module or memory. The passive module, on the other hand, looks to its neighboring modules similar to a memory. When activated it performs some computation on the contents of its internal memory, and goes back idle while setting the `IDLE` signal.

An FFT processor module is depicted in figure 3.6. The module is passive, and has two interfaces to its internal memory, write interface A and read interface B. The interfaces look like any other memory interface in the design, and can be used to read and write data to and from the internal memory. To avoid collision, both interfaces can not be used simultaneously, and a controlling device operates a set of multiplexers, connecting the memory to one of the interfaces or to the internal datapath. Communication with the controller from outside the module is done by the two signals, `DO` and `IDLE`. The first initializes a series of operations of the module, in this case, performs an in-place FFT of the data in the internal memory. The `IDLE` is inactive during the FFT computation, but goes active upon completion, signaling that the transformed data is ready to be read out.

3.8 WORDLENGTH OPTIMIZATION

Unnecessarily large signal wordlengths result in larger arithmetic components and larger memories for data storage. Additionally, such extra hardware con-

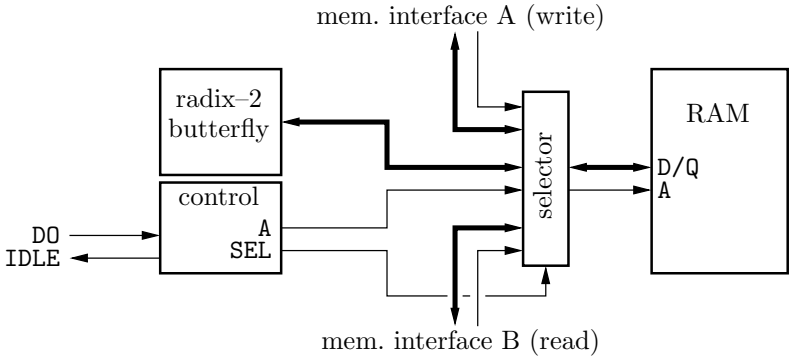


Figure 3.6: Example of a hardware accelerator. This is an FFT processor with two memory bus interfaces to neighboring modules. The bold lines represent data in and out, and the thin lines represent address bus, select and read write signals. To the left is the external control interface.

sume power without any gain in performance. Therefore, all signals should have a minimum wordlength. Since wordlengths also determine resolution and dynamic range, it is important to keep signals wide enough to avoid overflow and rounding errors, and maintain acceptable signal to noise ratios.

Wordlength optimizations are performed on each bus, arithmetic unit and memory instance, so that different hardware units can have different wordlengths. The effects of this truncation into fixed point are investigated using simulations. Input and output speech sample wordlengths are set to 16 bits. The internal signal wordlength ranges from 16 to 32 bits, where the highest dynamics is used for intermediate frequency domain variables in the fullband assemble unit. The taps of the NLMS filters are stored with an accuracy of 29 bits, and the adaptive FIR filter accumulator is 31 bits wide. The implementation of the adaptive FIR filter can be reduced in complexity as explained in 3.4.

4 The Chip

This section is divided into four parts: a description of the verification interface of the chip, short summary of the design methodology used, results, and a comparison to implementation on existing generic DSP processors.

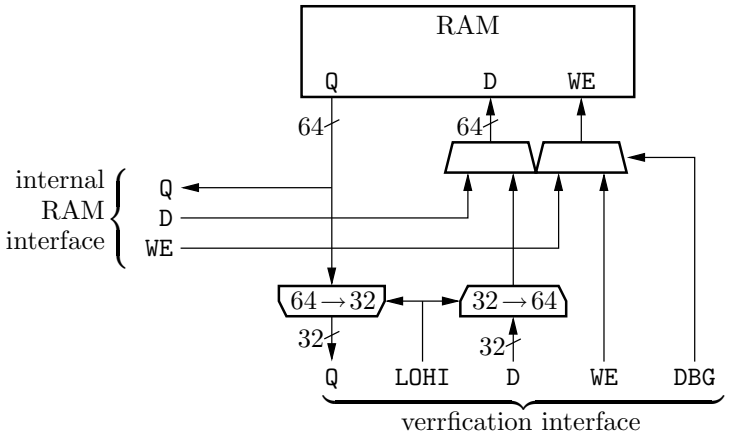


Figure 4.1: Verification encapsulation of the memories.

4.1 VERIFICATION INTERFACE

The fabricated chip is a research design, requiring extensive verification possibilities. A test strategy aiming at a high fault coverage indicating if the design has a failure is not sufficient. The failure should be locateable. Also, if one part of the design has a failure, testing of the remaining design should still be possible.

Thus, if parts of the chip malfunction, it should be possible to *observe* internal signals in the design, and additionally to *stimulate* some internal circuitry form the outside, giving a chance to verify different modules in the design independent of each other. The ability to observe and stimulate internal circuitry in the design requires extra pads on the chip. The total number of pads in a design is limited by the available chip packages and the associated cost. Thus, facilities for observability and controllability have to share pads with the inputs, outputs and power supply required by the design.

Scan chains are popular for reading and writing the contents of the on chip flip-flops, but were discarded for the test chip for two reasons: it is difficult to include memory in the scan chain and the chains have a limited data rate. Instead, the test chip relies on a verification bus that connects to all RAM instances and other interesting signals. Each RAM is encapsulated in a small verification bus interface unit, see figure 4.1. With select signals, the contents of the RAM could be read or written at a high data rate using the verification bus.

In total there are ten RAMs in the design, ranging from 16 to 1024 words, with wordlengths of 16 to 64 bits. The largest wordlengths are used to store

complex numbers with 32 real and 32 imaginary bits of data. A parallel interface, consisting of 10 bits address bus, 32 bit input and output data bus, and signals for read/write and real/imaginary select was implemented. Additionally, four bits input is used to select which of the RAMs that is currently being accessed.

The verification bus is also used for transferring speech sample data to and from the chip. Thereby, the audio sample interface is hidden in the verification interface, and requires no pads of its own.

Using the verification interface, it is possible to read and write to every RAM in the design. Every sub-module can be tested independently by writing data to its source RAM, and after operation reading back the result from the destination RAM. Other interesting possibilities is realtime readout of the energy in each subband, or realtime readout of the time domain impulse response estimate, for the purpose of making a visual presentation of how the impulse response changes with time and acoustic path. Another feature is that it is possible to eavesdrop on the output of each RAM in realtime without affecting the operation of the chip.

4.2 METHODOLOGY

As the design is written in a hardware description language, all parameters are kept generic. As a consequence, a chip with any configuration of wordlengths, number of subbands, or FIR filter length can be elaborated and manufactured from the same source description.

Initial modeling was performed using Matlab. When the algorithm was developed, add sub-algorithm were one at a time rewritten in the `C` programming language. By linking the `C` code dynamically into Matlab, it could be verified together with the original high level code.

The `C` description of the algorithm was then subject to loop transforms and memory access pattern investigations [13] to find efficient memory addressing schemes. A `C++` class overloading arithmetic operations was used to convert the description into a fixed point representation, with a minimal wordlength on all signals. Finally, an architecture was developed, and the design rewritten using a hardware description language.

During the lower level design process, a strict hierarchical design methodology was used. The complete design was modeled on a high level top-down, and sub-modules were designed and thoroughly verified bottom-up. The resulting description of the design consist of a top level, and a number of instantiated functional sub-modules, which in turn may be composed of further levels of sub-modules in the hierarchy. The hierarchical methodology simplifies design, verification, and maintenance.

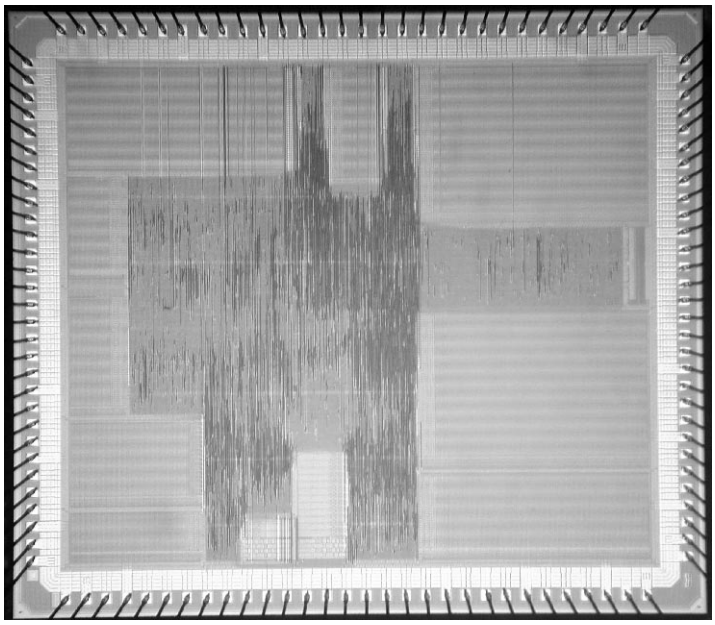


Figure 4.2: Micro photograph of the manufactured echo canceller chip. The size is 5.02×5.76 square millimeters.

4.3 RESULTS

The echo canceller is implemented using a standard cell library and synthesis tools. The chip is manufactured in a $0.35\,\mu$m CMOS process with five metal layers. It contains 120 pads and twelve memory instances, two ROMs and ten RAMs. In total, about 250 kbit RAM and 30 kbit ROM is instantiated. The chip has an advanced interface for verification purposes, where the content of every RAM can be read or altered at full clock rate. The size of the chip is about $29\,\text{mm}^2$ in area, consisting of more than two million transistors. A micro photograph is shown in figure 4.2.

The chip is targeted towards a clock frequency of 16 MHz, operating on 16 kHz audio samples. Due to the pipelined architecture of the datapaths, the design was synthesized to a maximum frequency of 50 MHz without difficulties. This speed margin is used to save power consumption by lowering the supply voltage [14].

The manufactured chip has full operating functionality, and power measurements during operation has been measured for a number of clock frequencies. The power measurement numbers are presented in table 4.1. The numbers

Table 4.1: Power Consumption versus frequency.

Audio Sample Rate	Clock Frequency	Power Consumption
8 kHz	8 MHz	23 mW
16 kHz	16 MHz	55 mW
32 kHz	32 MHz	168 mW
50 kHz	50 MHz	493 mW

are measured using white noise input signals, resulting in a higher switching activity than for more correlated speech signals, and a correspondingly higher power consumption.

4.4 COMPARE WITH STANDARD DSPS

In order to compare the application specific implementation with an implementation on a standard processor, a number of issues have to be investigated. Since the semiconductor manufacture process has a great impact on performance, it has to be established what processes the compared circuits are fabricated in. Also, for a fair comparison, the standard DSP should be able to hold the intermediate data in its internal memory, since external accesses are costly, and difficult to estimate. Furthermore, the wordlength of the datapaths must be investigated. The presented chip operates on 16 bit sample data, making 16 bits a minimum internal wordlength. As shown in section 3.8, most signals have a significantly larger wordlength. Since most fixed point DSPs operate on a 16 bit wordlength, implementation on such a processor will be complicated.

The first comparison is made to the TMS320LC54x low power fixed point processor family. Implemented in a 0.35 μm process, power consumption is claimed to be 1.95 mW/MIPS [15]. Counting the multiplications only, the power consumption would be about 60 mW for 16 kHz sample rate, but a large number of instructions is required to perform Fourier transforms etcetera. In the benchmarks, it is claimed that more than 8500 instructions are required for a 256 point FFT. The fullband update has to perform an 1024 point every 32 samples, which will translate to more than 17 MIPS ($8500 \times 4 \times 16384/32$), or 34 mW for the IFFT only at 16 kHz sample rate. Also, the processor has a 16 bit data wordlength, witch is far from enough for the 16 bit sample input/output canceller with intermediate wordlengths of 32 bits. For implementation on the fixed-point processor with a 16×16 bits multiplier, this would require iterated use of the multiplier for every multiplication operation, significantly increasing the number of MIPS.

Instead, comparing to the floating point TMS320VC33, with a stated power consumption of less than 200 mW@75 MIPS [16], the algorithm might just fit. But as the DSP is implemented in a high performance 0.18 μm CMOS process, power consumption has to be adjusted to be compared to the 0.35 μm process used of the presented chip. It is difficult to present exact numbers on the performance difference between the two processes. By applying general scaling or full scaling [17], energy per operation grow cubic with the scale factor. Thus, the custom ASDSP echo canceller has more than a decade less power consumption than the standard DSP.

5 Conclusions

A delayless acoustic echo canceller chip has been implemented and fabricated in a 0.35 μm silicon CMOS process with five metal layers. Optimizations on the algorithm and architecture levels are presented, reducing implementation complexity and power consumption.

The manufactured chip is verified for functionality, and at a audio sample rate of 16 kHz, it consumes 55 mW power for white noise input. The chip contains more than two million transistors and occupy an area of 29 mm^2. On-chip memory storage is 250 kbit RAM and 30 kbit ROM, divided into ten RAMs and two coefficient ROMs. An advanced verification interface is built-in, making it possible to read and alter contents of any RAM at full clock rate.

Compared to a standard DSP, it is shown that the custom wordlengths of the design are difficult to apply to a fixed point DSP with a reasonable number of MIPS. Comparing to a floating point DSP, power consumption is estimated to be more than a decade lower for the custom implementation.

References

[1] D. R. Morgan and J. C. Thi, "A delayless subband adaptice filter architecture," *IEEE Transactions on Signal Processing*, vol. 43, no. 8, Aug. 1995.

[2] M. M. Sondhi and W. Kellerman, "Adaptive echo cancellation for speech signals," in *Advances in Speech and Signal Processing*, S. Furui and M. M. Sondhi, Eds. New York: Marcel Dekker, 1992, ch. 11.

[3] T. Gänsler, M. Hansson, C.-J. Ivarsson, and G. Salomonsson, "A double-talk detector based on coherence," *IEEE Transactions on Communications*, vol. 44, no. 11, pp. 1421–1427, Nov. 1996.

[4] W. Kellerman, "Analysis and design of multirate systems for cancellation of acoustical echoes," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1988.

[5] M. Dörbecker and P. Vary, "Reducing the delay of an acoustic echo canceller with subband adaption," in *4th International Workshop on Acoustic Echo and Noise Control*, 1995.

[6] A. Berkeman, V. Öwall, and M. Torkelson, "A prestudy of an echo canceler implementation," in *Proceedings of the International Conference on Signal Processing Applications and Technology*, Orlando FL, USA, Nov. 1999.

[7] P. P. Vaidyanathan, *Multirate Systems and Filterbanks*. Prentice-Hall, 1993.

[8] A. Berkeman, V. Öwall, and M. Torkelson, "A low logic depth complex multiplier," in *Proceedings of the 24th IEEE European Solid-State Circuits Conference*, The Hague, The Netherlands, Sept. 1998.

[9] H. V. Sorensen, D. L. Jones, M. T. Heideman, and S. Burrows, "Real-valued fast Fourier transform algorithms," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-35, 1987.

[10] C. Breining, P. Dreiscitel, E. Hansler, A. Mader, B. Nitsch, H. Puder, T. Schertler, G. Schmidt, and J. Tilp, "Acoustic echo control: An application of very-high-order adaptive filters," *IEEE Signal Processing Magazine*, vol. 16, no. 4, pp. 42–69, July 1999.

[11] M. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, 1994.

[12] A. Berkeman, V. Öwall, and M. Torkelson, "Co-optimization of FFT and FIR in a delayless acoustic echo canceller implementation," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Geneva, Switzerland, May 2000.

[13] F. Catthoor, S. Wuytack, E. D. Greef, F. Balasa, L. Nachtergaele, and A. Vandecapelle, *Custom Memory Management Methodology*. Kluwer Academic Publishers, 1998.

[14] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, Apr. 1995.

[15] A. Fishman and P. Rowland, "Designing low-power applications with the 320LC54x," Texas Instruments, Tech. Rep., 1997.

[16] *320VC33 Datasheet*, Texas Instruments, July 2002.

[17] J. M. Rabaey, *Digital Integrated Circuits: a design perspective*. Prentice-Hall, 1996.

Paper III

Paper III

A Low Logic Depth Complex Multiplier using Distributed Arithmetic

A combinatorial complex multiplier has been designed for use in a pipelined fast Fourier transform processor. The performance in terms of throughput of the processor is limited by the multiplication. Therefore, the multiplier is optimized to make the input to output delay as short as possible. A new architecture based on distributed arithmetic, Wallace-trees and carry-lookahead adders has been developed. The multiplier has been fabricated using standard cells in a 0.5 μm process and verified for functionality, speed and power consumption. Running at 40 MHz, a multiplier with input wordlengths of 16+16 times 10+10 bits consumes 54% less power compared to an distributed arithmetic array multiplier fabricated under equal conditions.

Based on: A. Berkeman, Viktor Öwall, and Mats Torkelson, "A Low Logic Depth Complex Multiplier using Distributed Arithmetic," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 4, pp. 656-659, Apr. 2000.

1 Introduction

Complex multiplication is one of the most time-critical and area consuming operation in a digital signal processor. Therefore, effort has to be made to decrease the number of multipliers and to increase their speed. A pipelined Fast Fourier Transform (FFT) processor has been designed for use in an Orthogonal Frequency Division Multiplex (OFDM) system. In the designed FFT processor the critical path consists of a complex multiplier in series with a butterfly unit performing addition and subtraction. A part of the FFT pipeline is shown in figure 2.1. Since the butterfly processors are much faster than the complex multiplier, the maximum clock frequency of the processor strongly depends of the multiplier delay. This paper present a novel multiplier architecture based on distributed arithmetic and Wallace trees. The multiplier is fully parameterized, so any configuration of input and output wordlengths could be elaborated.

2 The FFT Processor

In the early versions of the FFT-processor, a complex array multiplier was used [1]. The array multiplier is a highly regular structure resulting in a minimal wire-length, which is important for high-speed design in sub-micron processes where wiring delay gives a significant contribution to the overall delay. However, in a process where cell delay dominates wire delay, the logic depth of the design is more important than regularity. In the complex array multiplier the logic depth is $\mathcal{O}(N)$, where N is the input wordlength. In the adder tree multiplier, on the other hand, the depth is $\mathcal{O}(\log N)$ [2]. Even for short wordlengths this leads to a substantial reduction in delay.

A way to decrease the critical path of the FFT processor would be to pipeline the multiplier into two or more stages. However, due to the pipelined structure of the FFT processor, complexity of the controlling hardware would increase [3]. Furthermore, the wordlengths of the datapaths are wide, due to the application of the processor, and all operators use complex arithmetic. A multiplier in this application has between 44 and 52 input bits, and a pipeline register inserted somewhere in the middle of the multiplier would need a word length of more than a hundred bits, due to the internal "carry save" number representation. This would increase area, routing and clock load and is not a preferable solution. Instead, the multiply operation is entirely combinatorial.

The FFT processor is implemented using the R2^2DIF FFT-algorithm [3]. In this algorithm, every second multiplication can be exchanged to a multiply by $-j$, which for an 8192-point FFT leaves only six complex multipliers. This is to be compared to twelve using a straightforward radix-2 implementation.

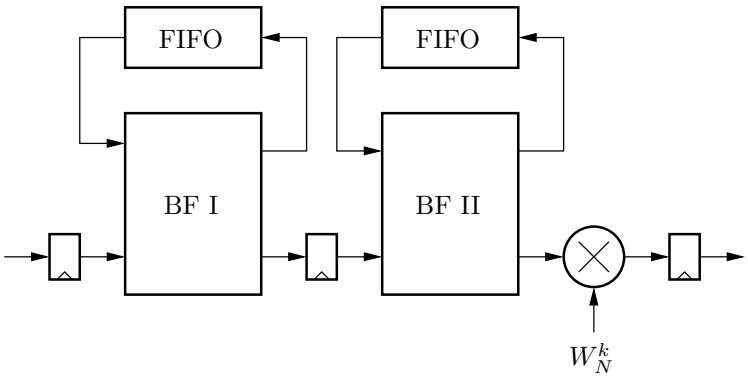


Figure 2.1: Part of the R2^{2}DIF FFT processor pipeline. The butterfly processors are combinational and named "BF I" and "BF II". The FIFOs are synchronous.

The multiplication by $-j$ is realized without a multiply by real-imaginary swap and negation of the imaginary part. This is the reason for the two different butterfly processors, "BF I" and "BF II", in figure 2.1. By using this algorithm, the number of instantiated multipliers is minimized compared to an ordinary radix-2 FFT without any loss in throughput.

3 Multiplier Algorithm

A complex multiplier calculates two inner products,

$$\begin{cases} Z_R = A_R W_R - A_I W_I \\ Z_I = A_R W_I + A_I W_R. \end{cases} \tag{3.1}$$

In the case of the FFT-processor, $W = W_R + jW_I$ are the twiddle-factors stored in a ROM. The wordlength of W_R and W_I is denoted M. According to equation (3.1), four real multiplications and two additions are required. With the exception of logic minimization, there are two methods to decrease multiplication logic depth if it is assumed that multiplication is performed by summation of partial products. The first is to reduce the number of partial products, and the second is to use a faster adder strategy to sum all the partial products together [2]. Both methods have been combined in the presented architecture. Distributed arithmetic [4, 5, 6] was chosen as a means to reduce the number of partial products. A Wallace tree adder was selected for adding the partial products together, since it has a low logic depth resulting in fast addition.

Table 3.1: Relation between the bits a_{R_i}, a_{I_i}, and the partial products.

α_{R_i}	α_{I_i}	a_{R_i}	a_{I_i}	$W_I\alpha_{R_i} + W_R\alpha_{I_i}$	$W_R\alpha_{R_i} - W_I\alpha_{I_i}$
-1	-1	0	0	$-W_\Sigma$	$-W_\Delta$
-1	1	0	1	W_Δ	$-W_\Sigma$
1	-1	1	0	$-W_\Delta$	W_Σ
1	1	1	1	W_Σ	W_Δ

By using distributed arithmetic the complex multiplication is treated as two independent inner products Z_R and Z_I. Each of the two inner products will be calculated using one distributed arithmetic multiplier, as compared to a direct realization of equation (3.1) which requires four real multipliers. In the equations that follow, a bit-variable is treated as an integer variable holding the arithmetic value of ZERO or ONE. In this way, bits can be used together with arithmetic variables and operators. If A is an N-bit fractional number in two's complement, the value of A is calculated according to

$$A = -a_0 + \sum_{i=1}^{N-1} a_i\, 2^{-i}. \tag{3.2}$$

By using the identity

$$A = \frac{1}{2}\big(A - (-A)\big) \tag{3.3}$$

and the rule for negating a two's complement number, $-A = \overline{A} + 2^{-(N-1)}$, equation (3.2) can be written as

$$A = -(a_0 - \overline{a}_0)\, 2^{-1} + \sum_{i=1}^{N-1} (a_i - \overline{a}_i)\, 2^{-i-1} - 2^{-N}. \tag{3.4}$$

Introduce $\alpha_0 = (\overline{a}_0 - a_0)$, and for $k \neq 0$, $\alpha_k = (a_k - \overline{a}_k)$, note that all $\alpha_k \in \{-1, +1\}$. Using this notation, A can be written as

$$A = \sum_{i=0}^{N-1} \alpha_i\, 2^{-i-1} - 2^{-N}. \tag{3.5}$$

The relationship between a_i and α_i is shown in table 3.1. Using this encoding

the complex product can be expressed as

$$Z_R = \sum_{i=0}^{N-1} (W_R\alpha_{R_i} - W_I\alpha_{I_i})\, 2^{-i-1} - (W_R - W_I)\, 2^{-N} \tag{3.6}$$

$$Z_I = \sum_{i=0}^{N-1} (W_I\alpha_{R_i} + W_R\alpha_{I_i})\, 2^{-i-1} - (W_I + W_R)\, 2^{-N}, \tag{3.7}$$

where $\alpha_i = \alpha_{R_i} + j\alpha_{I_i}$. The expressions $W_R\alpha_{R_i} - W_I\alpha_{I_i}$ and $W_I\alpha_{R_i} + W_R\alpha_{I_i}$ are for $i \neq 0$ examined in table 3.1, where the twiddle-factors have been transformed from W_R and W_I to W_Σ and W_Δ according to

$$\begin{cases} W_\Sigma = W_R + W_I \\ W_\Delta = W_R - W_I. \end{cases} \tag{3.8}$$

This transformation does not cause any problems in the implementation, since the twiddle-factors are pre-calculated in the W_Σ and W_Δ format before realization. From table 3.1 it is clear that $p_i = (a_{R_i} \oplus a_{I_i})$ can be used to select W_Σ or W_Δ. Treating p_i as integers holding the values 0 or 1 and $\vee$ as a bitwise inclusive-or operator, equation (3.7) and (3.7) can be written as

$$Z_R = \sum_{i=0}^{N-1} \left(\overline{a}_{R_i} \oplus (p_i W_\Sigma \vee \overline{p}_i W_\Delta) + \overline{a}_{R_i}\right) 2^{-i-1} - W_\Delta\, 2^{-N} \tag{3.9}$$

$$Z_I = \sum_{i=0}^{N-1} \left(\overline{a}_{I_i} \oplus (p_i W_\Delta \vee \overline{p}_i W_\Sigma) + \overline{a}_{I_i}\right) 2^{-i-1} - W_\Sigma\, 2^{-N}. \tag{3.10}$$

When evaluating the sums, the powers $\overline{a}_{R_i}$ and $\overline{a}_{I_i}$ should be replaced with a_{R_i} and a_{I_i} for the case $i = 0$, since these bits have negative weight in two's complement representation. The partial inner products

$$\overline{a}_{I_i} \oplus (p_i W_\Delta \vee \overline{p}_i W_\Sigma) + \overline{a}_{I_i} \tag{3.11}$$

and

$$\overline{a}_{R_i} \oplus (p_i W_\Sigma \vee \overline{p}_i W_\Delta) + \overline{a}_{R_i} \tag{3.12}$$

are suitable for hardware mapping. They are realized as multiplexers selecting W_Σ or W_Δ, depending on the value of p_i. The bits a_{R_i} and a_{I_i} conditionally negates the outputs of the multiplexers by inverting and adding a ONE in the least significant position. Figure 3.1 shows all the partial product bits that has to be added to generate Z_R or Z_I. The wordlength for the twiddle factor, W, is

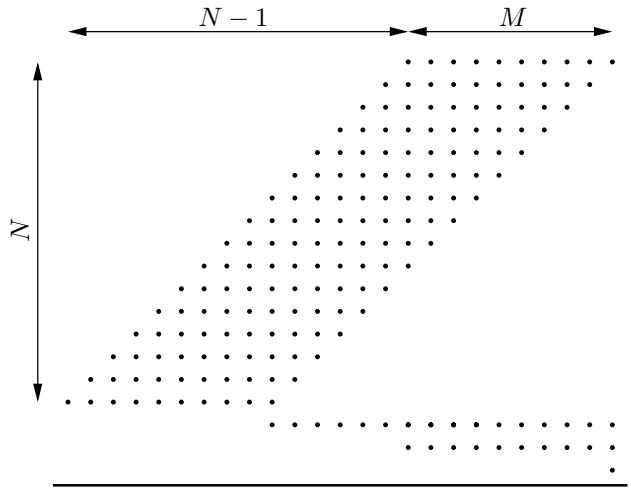


Figure 3.1: Partial product bits by significance for Z_R or Z_I. Input wordlength is N and coefficient wordlength is M. Each dot represents one partial product bit.

M bits and for the data, A, it is N bits, in this case 10 and 16 bits respectively. The top sixteen lines in the figure is the partial products generated inside the sum of equation (3.10) or (3.10), and the third line from bottom is the ones that form the corresponding two's complement of these products. The last two lines is the $-W_\Sigma$ or $-W_\Delta$ times 2^{-N} term.

As an alternative to distributed arithmetic, modified Booth-encoding was considered [7, 8]. However, as the number of partial products are about the same for both methods, modified Booth-encoding requires more logic gates to implement. This is due to that in the modified Booth algorithm, three variables have to be decoded to select the proper partial product. In a complex multiplier based on distributed arithmetic, a simple two-input XOR-gate implements the selection.

4 Implementation

The proposed multiplier consists of two distributed arithmetic blocks, one calculating Z_R, and the other Z_I. The two blocks are similar and the difference is basically the minus in equation (3.1). Each block is divided into three parts, partial inner product generator, adder tree and carry lookahead adder, see figure 4.1.

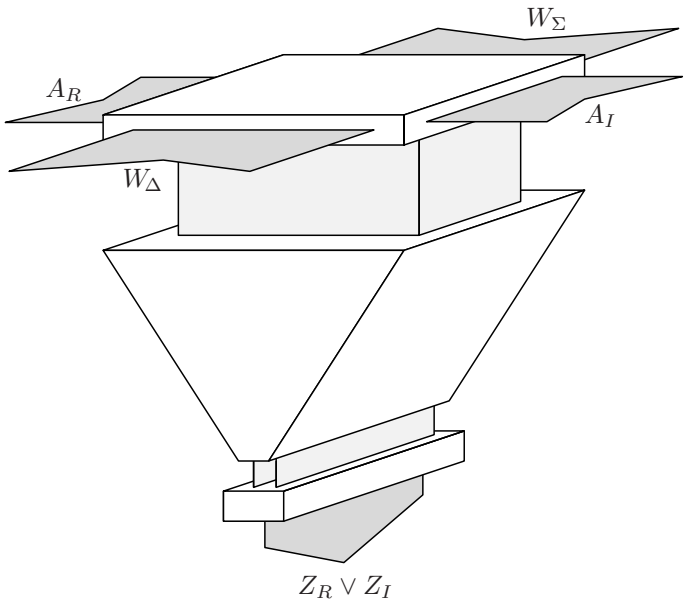


Figure 4.1: The multiplier for Z_R or Z_I, the complete complex multiplier consists of two of these. Partial inner product generator at the top, adder tree in the middle, and fast carry-lookahead adder at the bottom.

When designing the adder tree, a generic tree generator was used. This generator produces a tree with y inputs of wordlength x, that is a rectangle of x by y input bits. This rectangle has to be large enough to cover all the partial product bits of figure 3.1, i.e. $x = M + N - 1$ and $y = N + 3$. For certain sizes of N and M, the two last lines in figure 3.1 can be joined with two of the N first lines, minimizing y to $N + 1$. Unfortunately, almost 50% of the inputs to the adder tree are unused or used for sign extension only, and extra logic will be generated. Therefore, the area for the tree multiplier is approximately 75% larger than for the array multiplier. The number of gates for the array multiplier is 3000, while the tree multiplier uses 6200 gates, of which 4400 belongs to the two adder trees. Theoretically, the area for a dedicated tree generator should be only slightly larger than for the array multiplier.

When data flows through the pipeline of the FFT processor, the wordlength has to increase to keep accuracy in the calculations. For the current application the input wordlength is 12+12 bits (real + imaginary) and the output wordlength is 16+16 bits. The twiddle-factors are kept constant at 10+10 bits at all stages of the pipeline. Different wordlengths in the datapath means that a

set of multipliers of different wordlengths have to be instantiated if the longest wordlength is not to be used for all multipliers with a corresponding increase in area. Also, as FFT processors will be built for different applications the wordlength is subject to change. Therefore, the multiplier is fully parameterized and a multiplier of specific wordlength can be elaborated when needed.

For the FFT application, the output wordlength should equal the input wordlength, i.e. some of the least significant bits of the result are cut away. A simple rounding scheme is applied to lower the distortion when the output is truncated. A rounding bit is added to the right of the rightmost bit to be kept after truncation, causing a carry to propagate when the most significant position of the bits cut away is a one. A feature of the adder tree is that this bit can be inserted together with the partial inner products at the top of the tree, see figure 3.1. In the array multiplier, an additional row of half-adders had to be included to handle rounding. As rounding includes addition of a one with the product, arithmetic overflow at the output is possible. Therefore, a saturation unit is placed at the output of the carry-lookahead adder. This unit checks the most significant bits of the result and saturates the output if an overflow has occurred.

Both the proposed multiplier and a multiplier based on a regular adder array have been fabricated under equal conditions for comparison. Chip micro photograph for the tree multiplier is shown in figure 5.1. The multipliers were fabricated using a three metal layer 0.5 μm cell library that does not contain any dedicated half or full adder cells. Such cells could be beneficial in a multiplier architecture due to the large amount of additions in the algorithm. Instead, adders are realized using basic gates such as two-input XOR-gates.

Simulations show that 55% of the total delay in the critical path is due to the adder tree. The partial inner-product generator contributes with 20% and the carry-lookahead adder 25% of the total delay. Simulation results have been presented in [9]. Most of the delay is spent in the adder tree, and by using dedicated adder cells this delay can be decreased. However, the target cell library does not contain any such cells and such improvements have not been implemented, which is the case for both designs.

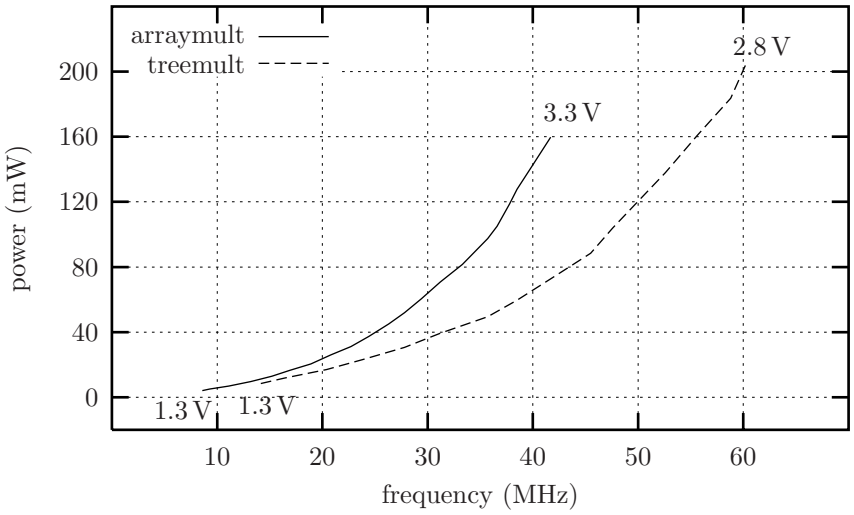


Figure 4.2: Power versus frequency for the tree and the array multipliers.

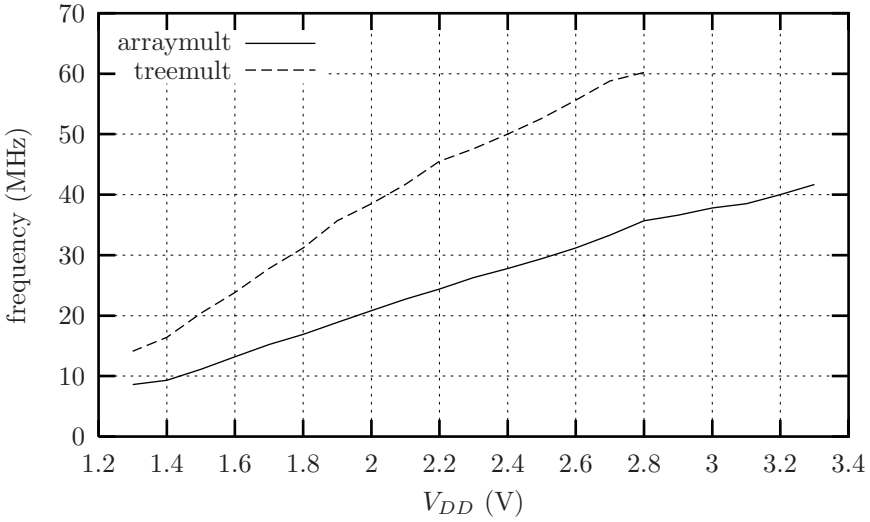


Figure 4.3: Frequency versus supply voltage for the tree and array multipliers.

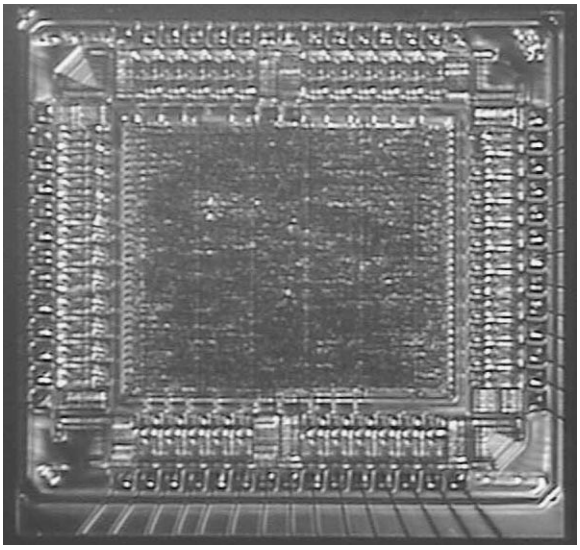


Figure 5.1: Chip micro photograph of the proposed multiplier. The pad-frame is 3.2x2.9 mm^2 and equal for both designs. Core size is 3.59 mm^2.

5 Results

Both the array multiplier [1] and tree multiplier were fabricated on the same wafer in a 3.3 V 0.5 μm three metal layer CMOS process using standard cells. This process is neither a dedicated low power or high speed process, but the enhanced performance is due to the architectural solution and should be applicable to more advanced processes as well. To keep the number of IO pads to a reasonable amount, the input was chosen to 16 real plus 16 imaginary times 10 real plus 10 imaginary bits. The most significant 16 real plus 16 imaginary product bits were output.

Power consumption and maximum operating frequency was measured over a supply voltage range from 1.3 to 3.3 volts in steps of 100 mV. None of the circuits worked below 1.3 V. Power consumption was measured as the power dissipated in the core when the chip ran at maximum operational frequency for a given voltage. The measured results are plotted in figure 4.2 and 4.3. Figure 4.2 shows power consumption versus operating frequency, and figure 4.3 shows maximum operating frequency versus supply voltage.

Maximum speed for the array multiplier was 41 MHz at 3.3 volts, while the tree multiplier exceeded the 60MHz limit of the test instrument when driven

with 2.8 volts. Due to the trade-off between power and speed [10], the proposed multiplier is either faster than the array multiplier, or, at equal speed, it is less power consuming.

6 Conclusion

A Wallace-tree based complex multiplier using distributed arithmetic has been designed, fabricated and verified for functionality, speed and power consumption. The multiplier is compared to a complex multiplier based on a regular array adder fabricated under equal conditions. The multipliers were fabricated in a three metal layer 0.5 μm process on the same wafer using a standard cell library. This library does not contain any full or half adder cells that could be beneficial in a multiplier architecture, but all adders are built with basic gates.

At a frequency of 40 MHz, the proposed tree multiplier consumes 66 mW, which is 54% less than the array multiplier. Operating at same voltage, the tree multiplier is 54% faster than the array multiplier. Maximum speed for the proposed multiplier is beyond 60 MHz at 3.3 volts. Enhanced performance comes from a novel architecture and is transferable to more advanced processes. Performance could be further improved by using a dedicated adder tree. Since the multiplier dominates the critical path, the delay contribution from the adder or subtracter can be ignored and throughput of the FFT-processor is expected to increase by approximately 80%. The multiplier is fully parameterized so any configuration of input and output wordlengths can be elaborated and synthesized.

References

[1] S. He and M. Torkelson, "A complex array multiplier using distributed arithmetic," in *Proceedings of IEEE Custom Integrated Circuits Conference*, 1991.

[2] C. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, Feb. 1964.

[3] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proceedings of the International Parallel Processing Symposium*, 1996.

[4] A. Croisier, D. J. Esteban, M. E. Levilion, and V. Riso, "Digital filter for PCM encoded signals," US patent 3777130, Dec. 1973.

[5] S. Smith and P. Denyer, "Efficient bit-serial complex multiplication and sum-of products computation using distributed arithmetic," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1986.

[6] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation.* John Wiley & Sons, 1999.

[7] A. D. Booth, "A signed binary multiplication technique," *Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, pp. 236–240, 1951.

[8] L. P. Rubinfield, "A proof of the modified booth algorithm for multiplication," *IEEE Transactions on Computers*, Oct. 1975.

[9] A. Berkeman, V. Öwall, and M. Torkelson, "A low logic depth complex multiplier," in *Proceedings of the 24th IEEE European Solid-State Circuits Conference*, The Hague, The Netherlands, Sept. 1998.

[10] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, Apr. 1995.

Paper IV

Paper IV

Efficient Implementation of an FFT-FIR Structure Using a Distributed Arithmetic Multiplier

When mapping algorithms to hardware, arithmetic optimizations traditionally deal with increasing performance on a block level and not in the connection between blocks. This paper present an arithmetic optimization applicable in situations where an FFT is used to compute coefficients for an FIR filter. The optimization is made possible by utilizing the properties of a distributed arithmetic multiplier. A zero signal path delay echo canceller is used as an application example. For the application, the number of arithmetic operations in the optimized part is reduced by more than 50%. Furthermore, if a time-shared architecture is used, all memory accesses in the last butterfly stage are removed. This reduced complexity will increase the throughput and/or lower the power consumption.

Based on: A. Berkeman and Viktor Öwall, "Efficient Implementation of an FFT-FIR Structure Using a Distributed Arithmetic Multiplier," Submitted to *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*
and A. Berkeman, V. Öwall, and M. Torkelson, "Co-Optimization of FFT and FIR in a Delayless Acoustic Echo Canceller Implementation," in *Proceedings of IEEE International Symposium on Circuits and Systems,* Geneva, Switzerland, May 2000.

1 Introduction

In order to achieve efficient implementation of an algorithm, it is important that hardware aspects are considered at all levels of the design process. Implementation style of arithmetic operations is such an aspect. Traditionally, arithmetic optimization primarily deals with the problem of decreasing the complexity on a block by block basis. This paper present a complexity reduction achieved by applying arithmetic optimization in the border between two blocks, a Fast Fourier Transform (FFT) and a Finite Impulse Response (FIR) filter.

A structure where the optimization is applicable is shown in figure 1.1, where an IFFT is used to calculate the coefficients of an FIR filter. One example of such a case is a delayless echo canceller [1] which has been used as an application example. The optimization can be applied to structures where a butterfly lattice precedes a multiply and accumulate unit, figure 1.2(a).

The optimization is based on a special arithmetic function, a combinational inner product multiplier calculating $P = AX + BY$. The implementation of this operation is particularly efficient if it is built using distributed arithmetic and offset binary coding [2, 3]. The distributed arithmetic multiplier requires one of the pairs (A, B) or (X, Y) to be coded in sum and difference form, i.e.

$$(A + B, A - B) \quad \text{or} \quad (X + Y, X - Y) \tag{1.1}$$

which might be regarded as a drawback in the general case. However, in the presented optimization process, the sum and difference form is utilized to reduce the algorithmic complexity. Distributed arithmetic in FIR filter applications is often considered to require extra memory storage of pre-calculated

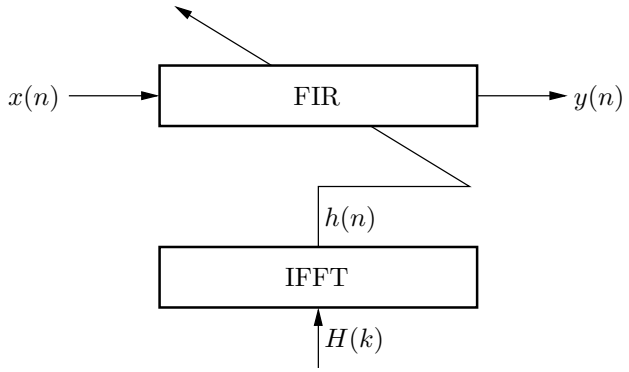


Figure 1.1: An IFFT transforms a frequency response $H(k)$ into an impulse response $h(n)$, used to filter a sequence $s(n)$.

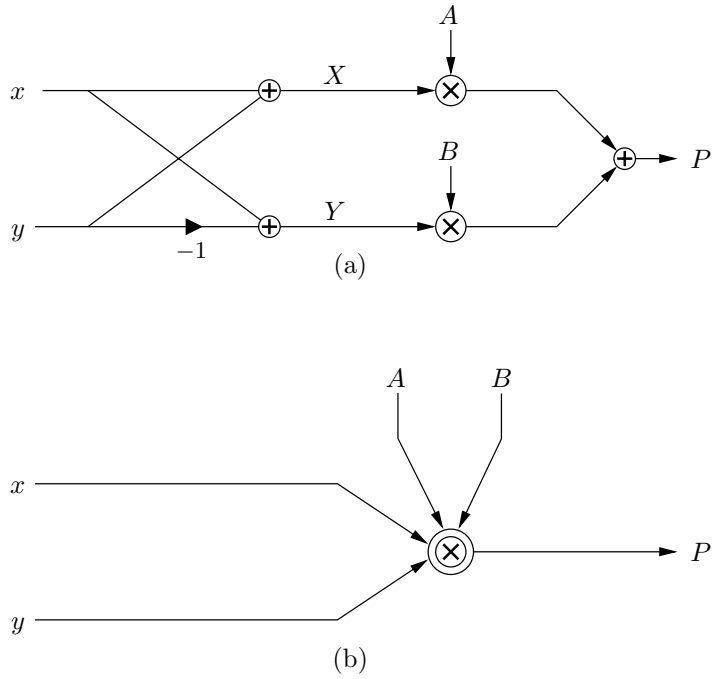


Figure 1.2: (a) A structure calculating $P = AX + BY = A(x + y) + B(x - y)$. (b) Equivalent functionality using a distributed arithmetic

coefficients [3]. However, in this application the multiplier is not used in a traditional distributed arithmetic FIR fashion and no extra memory is required.

2 Echo Canceller Application

In wireless communication systems, delay in the signal path is a serious obstacle and care has to be taken to reduce it. This motivates a canceller with no delay in the signal path, although such algorithms have a more complex realization in terms of operations per second [4]. An echo canceller fulfilling the zero delay criteria is introduced in [1]. This section gives a brief description of the algorithm.

A block diagram is shown in figure 2.1. The far end signal $s(n)$ is filtered by an estimate of the echo path impulse response $\hat{h}(n)$, and subtracted from the near end signal $y(n)$. The subtraction is the only operation performed in the signal path of the microphone to output signal, therefore, the algorithm is

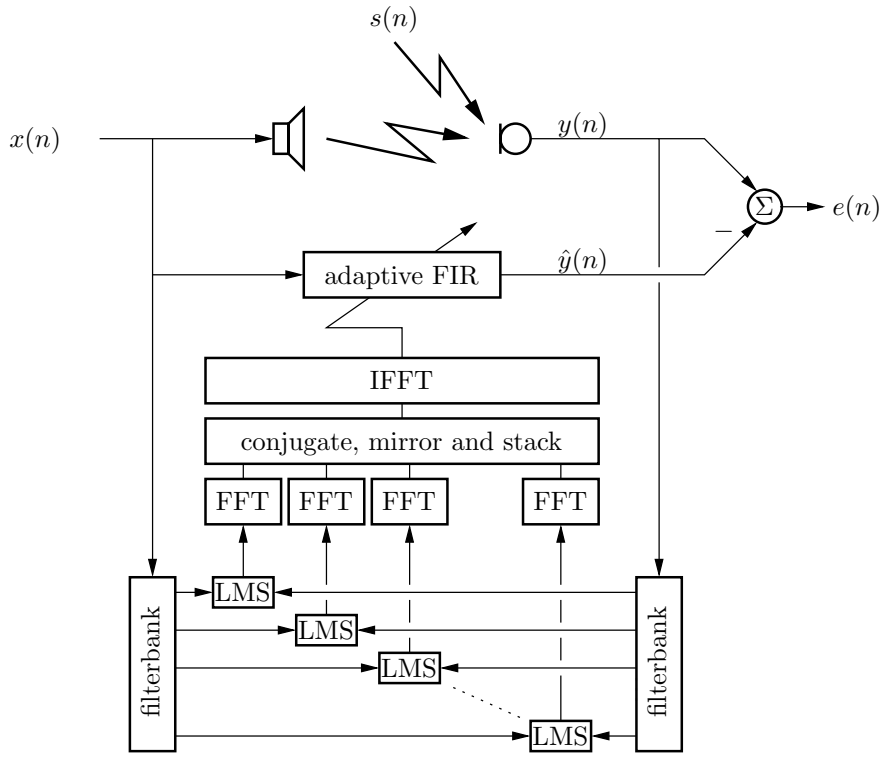


Figure 2.1: The subband echo cancellation algorithm [1].

a "zero signal path delay" echo canceller.

The estimated impulse response $\hat{h}$ is assembled from the output of a set of Least Mean Square (LMS) filters, operating on subbands of the near and far end signals. Two uniform filterbanks compute these subband signals. In order to create a full band time domain impulse response, the approximated filter taps from each LMS is transformed to the frequency domain by a set of FFTs. Information from each FFT output is permuted and stacked together according to a certain scheme, and transformed back to time domain by a large IFFT. The permutation of the transformed filter taps assures that the estimate of the time domain impulse response is real valued. The presented optimization is applicable on this IFFT together with the fullband FIR filter.

The taps of $\hat{h}$ are denoted $\hat{h}(0), \hat{h}(1), \ldots \hat{h}(N-1)$. These N taps are used to filter the audio signal from the far end speaker to estimate the impact of the acoustic signal path to the signal from the far end. The filtered signal $\hat{y}(n)$ is

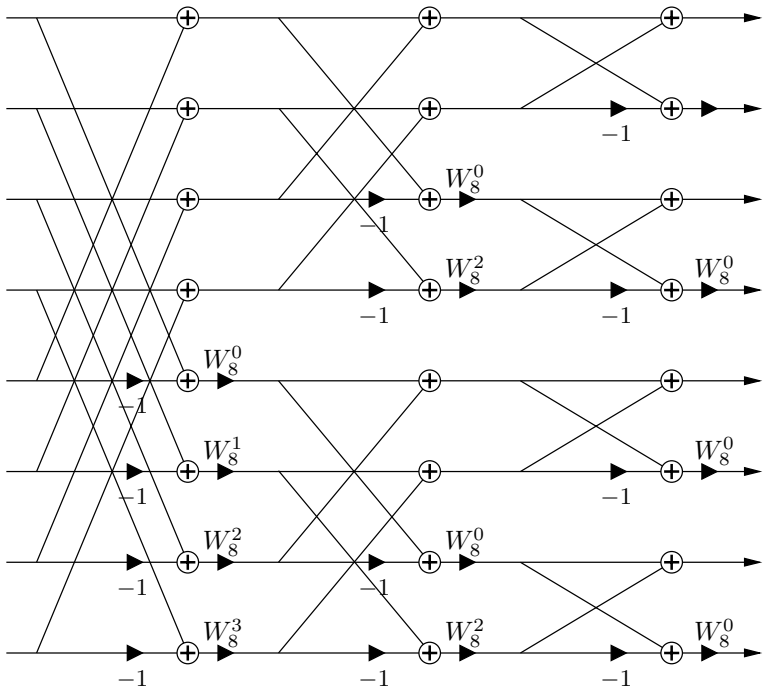


Figure 2.2: Eight bin decimation in frequency (DIF) FFT. W_N denotes $e^{-j2\pi/N}$. Note that $W_N^0 \equiv 1, \forall N$ and thus the last stage is without twiddle multiplication.

created according to the convolution

$$\hat{y}(n) = \sum_{k=0}^{N-1} \hat{h}(k)s(n-k). \tag{2.1}$$

This filter is run on every input sample, typically at a rate of 16kHz. To be able to cancel out long echoes associated with acoustic signals, the length N of the estimated impulse response has to be in the range of 500–4000 taps. The output FIR filter alone consumes about 50% of the total number of computations in the canceller algorithm when M is reasonably large, i.e. $M > 64$ [5].

2.1 THE FFT-FIR CONSTELLATION

The proposed optimization is intended for radix–2 FFTs or IFFTs, but also works on radix–4 or split radix–2/4. Generally, it works on any FFT where

there are no twiddle multiplications on the outputs of the last butterfly stage, independent of radix and decimation in time or frequency. As an example, twelve butterflies of a radix–2 FFT are depicted in figure 2.2. The last butterfly stage only consists in additions, subtractions and trivial multiplications, since the twiddle factors are $W_N^0 = e^{\mathrm{j}2\pi \cdot 0/N} = 1$.

Figure 2.3 shows the last stage of an FFT and the preceding FIR filter, both of size N. The signals $\hat{h}_{pre}$ are introduced as intermediate results inside the FFT, that is, before the last butterfly. The relationship between $\hat{h}$ and $\hat{h}_{pre}$ is

$$\begin{cases} \hat{h}(2k) &= \hat{h}_{pre}(2k) + \hat{h}_{pre}(2k+1) \\ \hat{h}(2k+1) &= \hat{h}_{pre}(2k) - \hat{h}_{pre}(2k+1) \end{cases} \tag{2.2}$$

or, the other way around

$$\begin{cases} \hat{h}_{pre}(2k) &= \frac{1}{2}\big(\hat{h}(2k) + \hat{h}(2k+1)\big) \\ \hat{h}_{pre}(2k+1) &= \frac{1}{2}\big(\hat{h}(2k) - \hat{h}(2k+1)\big). \end{cases} \tag{2.3}$$

This can be compared to the sum and difference form of equation (1.1). Due to the stacking at the input of the IFFT, mentioned early in this section, the signals $\hat{h}(k)$ are real. Accordingly, the signals $\hat{h}_{pre}(k)$ must be real.

The last stage of butterflies partitions the FIR filter into $N/2$ partial convolvers, each computing

$$\begin{aligned} \hat{y}_k(n) &= \hat{h}(2k)s(n-2k) \\ &+ \hat{h}(2k+1)s(n-(2k+1)), \end{aligned}$$

such that the complete convolution (2.1) can be written as

$$\hat{y}(n) = \sum_{k=0}^{\frac{N}{2}-1} \hat{y}_k(n), \tag{2.4}$$

see figure 2.3. In the following, it is shown how the calculations of equations (2.2) and (2.1) can be implemented in an efficient way using a combinational inner product multiplier using distributed arithmetic.

3 Utilizing the Distributed Arithmetic Multiplier

Distributed arithmetic was presented in [2]. This section will give a theoretical background to how a distributed arithmetic multiplier can be used to realize

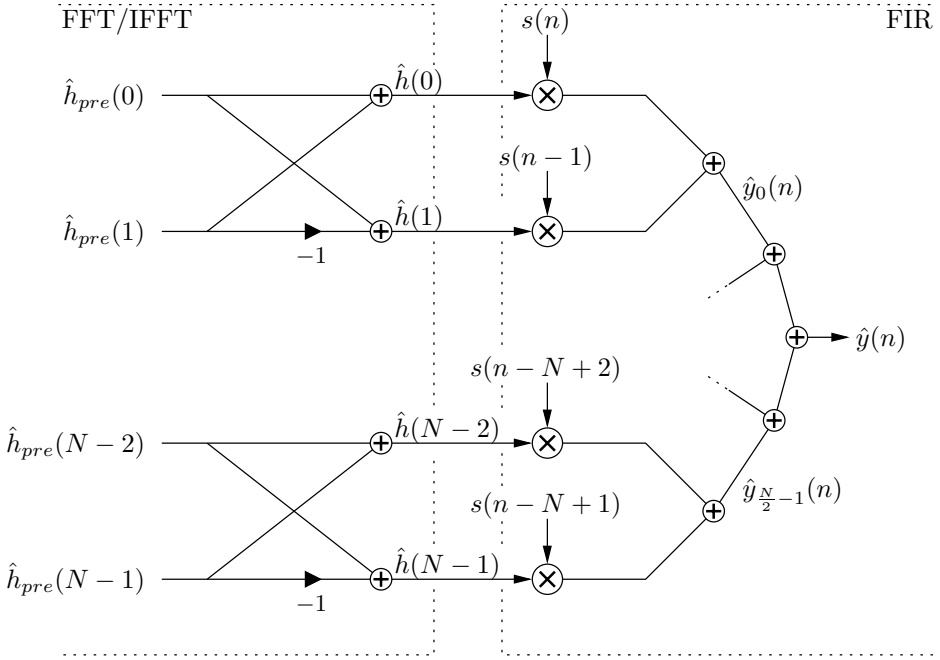


Figure 2.3: FIR filter preceded by the last stage of an FFT/IFFT.

equation (2.1) in a hardware efficient way. Why the algorithmic transformation in the following section is performed might seem unclear until the final point of showing the hardware implementation. The most basic butterfly structure, depicted in figure 1.2(a), will be used to present the proposed optimization. As stated previously it is assumed that all parameters are real valued, which is the case for the acoustic echo canceller. However, it is straightforward to extend the optimization to work on complex numbers. In figure 1.2(a) the output P is equal to

$$P = AX + BY. \tag{3.1}$$

Distributed arithmetic is working on a bit by bit level. Therefore, a transformation to a bit level representation is performed. If A is an L-bit fractional number in two's complement, the value of A can be expressed as

$$A = -a_0 + \sum_{\ell=1}^{L-1} a_\ell \, 2^{-\ell}, \tag{3.2}$$

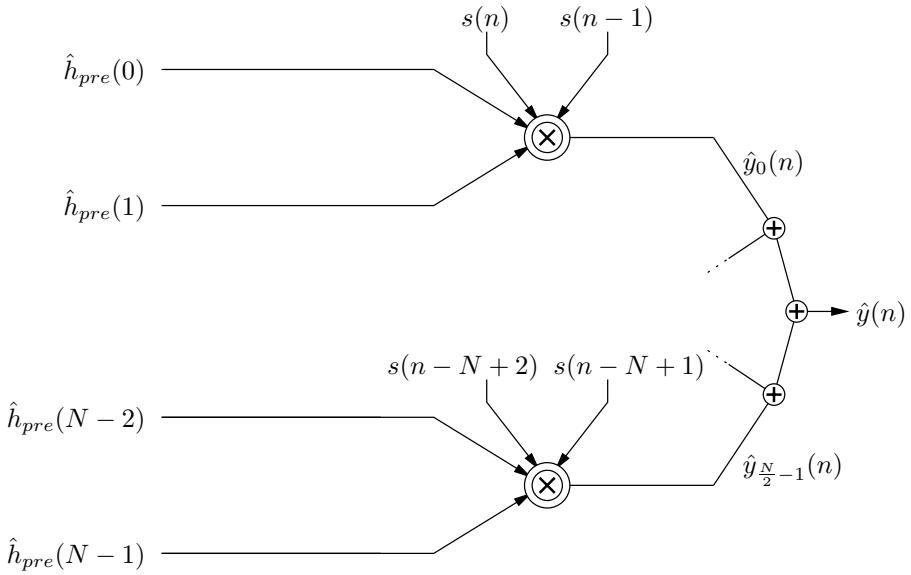


Figure 2.4: Same function as in figure 2.3, with distributed arithmetic multipliers.

where a_0 represents the sign bit. By using the identity

$$A = \frac{1}{2}\Big(A - (-A)\Big) \tag{3.3}$$

and the rule for negating a two's complement number

$$-A = \overline{A} + 2^{-(L-1)}, \tag{3.4}$$

equation (3.2) can be written as

$$A = -2^{-L} - (a_0 - \overline{a_0})\, 2^{-1} + \sum_{\ell=1}^{L-1} (a_\ell - \overline{a_\ell})\, 2^{-\ell-1}. \tag{3.5}$$

Introduce $\alpha_0 = (\overline{a_0} - a_0)$, and for $\ell \neq 0$, $\alpha_\ell = (a_\ell - \overline{a_\ell})$. Then $\alpha_\ell \in \{-1, +1\}$ [3]. The relationship between a_ℓ and α_ℓ becomes

$$\alpha_\ell = \begin{cases} +1 \text{ , if } a_{\ell \neq 0} = 1 \text{ or } a_0 = 0 \\ -1 \text{ , if } a_{\ell \neq 0} = 0 \text{ or } a_0 = 1. \end{cases} \tag{3.6}$$

Table 3.1: Expression $X\alpha_\ell + Y\beta_\ell$ as a function of the bits a_ℓ and b_ℓ.

a_ℓ	b_ℓ	α_ℓ	β_ℓ	p_ℓ	q_ℓ	$X\alpha_\ell + Y\beta_\ell$
0	0	-1	-1	0	1	$-(X+Y)$
0	1	-1	1	1	1	$-(X-Y)$
1	0	1	-1	1	0	$(X-Y)$
1	1	1	1	0	0	$(X+Y)$

Using this notation, A can be written as

$$A = -2^{-L} + \sum_{\ell=0}^{L-1} \alpha_\ell \, 2^{-\ell-1}. \tag{3.7}$$

Following the notation of equation (3.7), B can be expressed using a sum of β_ℓ. With this encoding of A and B, equation (3.1) can be written as

$$P = -(X+Y)\, 2^{-L} + \sum_{\ell=0}^{L-1} (X\alpha_\ell + Y\beta_\ell)\, 2^{-\ell-1}. \tag{3.8}$$

A hardware efficient implementation of equation (3.8) can be achieved by introducing p_ℓ and q_ℓ according to

$$\begin{cases} p_\ell = a_\ell \oplus b_\ell \\ q_\ell = \overline{a_\ell}, \end{cases} \tag{3.9}$$

where $\oplus$ is the boolean exclusive-or operator. The expression $X\alpha_\ell + Y\beta_\ell$ of equation (3.8) is examined in table 3.1 for $\ell \neq 0$. From the table it is clear that p_ℓ and q_ℓ can be used to control a multiplexer, i.e. p_ℓ is selecting either $(X+Y)$ or $(X-Y)$ and q_ℓ is selecting the preceding sign.

The variables X and Y can be expressed using the inputs x and y of figure 1.2(a) as

$$\begin{cases} X &= x + y \\ Y &= x - y. \end{cases} \tag{3.10}$$

Then it is clear that the expressions $(X+Y)$ and $(X-Y)$ of table 3.1 can be written as $2x$ and $2y$ respectively, since

$$\begin{cases} X+Y &= 2x \\ X-Y &= 2y. \end{cases} \tag{3.11}$$

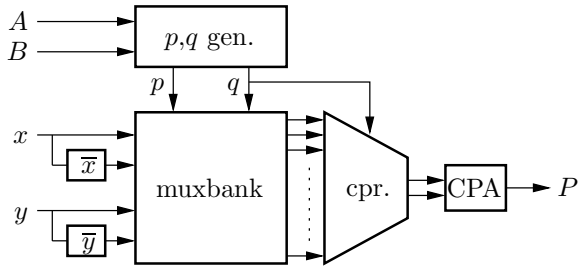


Figure 3.1: Structure view of the distributed arithmetic multiplier. Inputs A and B are fed to a p, q-generator. Signals x, y and their inverses are input to a bank of multiplexers, controlled by p and q. The selected signals are added together in a compressor (cpr.) to a redundant carry-save format, and finally carry propagate added (CPA).

Therefore, the core expression $X\alpha_\ell + Y\beta_\ell$ of equation (3.8) can be implemented by a four-input multiplexer selecting one of $\{x, -x, y, -y\}$. The negative versions of x and y are represented in two's complement according to equation (3.4). If the multiplexer is described mathematically as a function F, equation (3.8) can be expressed as

$$\begin{aligned} P \quad = \quad -2x\ 2^{-L} \quad & +2\sum_{\ell=0}^{L-1} F\big(x, y, \overline{x}, \overline{y}, p_\ell, q_\ell\big)2^{-\ell-1} \\ & +2\sum_{\ell=0}^{L-1} q_\ell 2^{-(K-1)}2^{-\ell-1}, \end{aligned} \tag{3.12}$$

where the one's complement numbers $\overline{x}$ and $\overline{y}$ are pre-calculated. The constant 2 from equation (3.11) has been moved outside of the summation. According to table 3.1, the function F is realized as a four-input multiplexer selecting x, y, $\overline{x}$, $\overline{y}$ depending of p_ℓ and q_ℓ. When q_ℓ is a one, negative x or y should be chosen. This is implemented by selecting their bitwise complements in the multiplexer, and adding a one to the carry-in of the least significant position of the sum, represented by the last summation term of equation (3.12). Here, q_ℓ is treated as a real variable having the value 0 or 1, and it is assumed that $2^{-(K-1)}$ is the least significant weight of x and y. An illustration of the structure of the multiplier can be seen in figure 3.1. The multiplier is described in more detail in [6].

Equation (3.12) describes a construct similar to an ordinary multiplier in terms of the "shift and add" structure, but with extra multiplexers and control logic. Specificly, one extra xor-gate to calculate p_ℓ and a conditional addition

Table 4.1: Optimization results for the last stage of the FFT/IFFT and the FIR filter.

Element	without optimization	with optimization
Butterfly adders and subtracters	N	0
FIR adders	$N-1$	$\frac{N}{2}-1$
FIR multipliers	N	0
Distributed arithmetic multipliers	0	$\frac{N}{2}$

of a least significant bit per index ℓ. When evaluating the sums, q_ℓ should be replaced with $\overline{q_\ell}$ for the case $\ell = 0$, since the most significant bit has negative weight in two's complement representation.

Since A and B maps to p_ℓ and q_ℓ, the inner product $P = AX + BY$ can be calculated according to equation (3.12) using x and y. Thus, the basic butterfly stage of figure 1.2(a) can be implemented by the introduced multiplier structure. This equivalent implementation is shown in figure 1.2(b).

The structure of figure 1.2(a) can be identified in figure 2.3. Thus, a distributed arithmetic multiplier can be used to calculate the partial convolutions $\hat{y}_k(n)$ of equation (2.1), if its inputs are $s(n-2k)$, $s(n-(2k+1))$, and the sums and differences of $\hat{h}(2k)$ and $\hat{h}(2k+1)$. These sums and differences are visible in equation (2.3), as $\hat{h}_{pre}(2k)$ and $\hat{h}_{pre}(2k+1)$. The factor of 1/2 is compensated by an arithmetic shift. Therefore, all butterflies in the last stage of the IFFT can be removed, and every pair of multipliers replaced by one distributed arithmetic multiplier, as depicted in figure 2.4.

4 Complexity Analysis

This section will compare the complexity of the two structures of figures 2.3 and 2.4, and the results are presented in table 4.1 and 4.2. The original implementation of the butterfly structure of figure 1.2(a) requires two multiplications and three add/subtract operations. Following the derivation of section 3, the optimized version requires only one distributed arithmetic multiplier. Accordingly, in figure 2.3, two adders, one subtracter and two multipliers are replaced by one distributed arithmetic multiplier. Assuming the FFT and FIR filter are of size N, this will reduce the number of butterfly adders and subtracters

Table 4.2: Number of additions and multiplications for an N-sized FFT/IFFT preceding an N tap FIR filter.

	no opt.		with opt.		reduction (%)	
N	add	mul	add	mul	add	mul
4	6	11	4	5	33.3	54.5
8	16	31	12	19	25.0	38.7
16	40	79	32	55	20.0	30.4
32	96	191	80	143	16.7	25.1
64	224	447	192	351	14.3	21.5
128	512	1023	448	831	12.5	18.8
256	1152	2303	1024	1919	11.1	16.7
512	2560	5119	2304	4351	10.0	15.0
1024	5632	11263	5120	9727	9.1	13.6
2048	12288	24575	11264	21503	8.3	12.5
4096	26624	53247	24576	47103	7.7	11.5

from N to zero and the number of products to be added in the filter is reduced from $N-1$ to $N/2-1$. Furthermore, the N multipliers are replaced by $N/2$ distributed arithmetic multipliers.

Table 4.1 summarizes how the number of arithmetic operations change in the combined last stage of the IFFT and the FIR filter when the optimization is applied. Total and relative reduction of the number of additions and multiplications for the complete IFFT and FIR filter instances are presented in table 4.2. In this table, a multiplier and distributed arithmetic multiplier are considered to be of the same implementation complexity. This will be further discussed in section 4.1. In table 4.2, it can be seen that the relative impact of the optimization is smaller with increasing N. For the echo canceller case with an N greater or equal to 1024, a reduction of at least 10% is gained.

In the echo canceller algorithm the length of the filter, N, has to be in the range of 500–4000 taps and is running at the relatively low sampling frequency of 16kHz. Therefore, it is advantageous to make a time-multiplexed architecture instead of a fully parallel implementation, that is, the IFFT is implemented by hardware reuse and one butterfly unit is used for all stages of the computation. Such a time-multiplexed architecture requires temporary storage of intermediate data. By removal of the last addition and subtraction stages of the IFFT, both the number of loads and stores are reduced by N, or for the whole IFFT by a factor of $1/\log_2 N$. Load and store operations consume energy for memory access, address calculation and bus driving. Reduction of

the number of memory accesses is a means to reduce power in an application specific integrated circuit [7].

4.1 HARDWARE IMPLEMENTATION COMPLEXITY

To compare the hardware complexity of the original scheme using standard multipliers and adders/subtracters to the one using distributed arithmetic multipliers, consider the case when A and B are L-bit fixed point representations of real valued numbers, and x and y are K-bit numbers, see figure 1.2(a). In the first case, X and Y needs to be created from x and y by one addition and one subtraction according to equation (3.11). In order to prevent overflow, a one bit extension is performed and the wordlengths of X and Y are set to $K+1$ bits. This results in

$$2L(K+1) \tag{4.1}$$

partial product bits to be added. These partial products can be added in one adder tree [6, 8] or one adder array structure, but it is common to use two partial product adders, one per multiplier, followed by a carry propagate adder to generate the output P.

The distributed arithmetic multiplier, on the other hand, is fed by x and y directly, together with A and B, figure 2.4. The number of partial inner product bits is in this case equal to

$$(L+1)(K+1). \tag{4.2}$$

Thus the number of partial bits to add compared to the two multiplier case is

$$\frac{(L+1)(K+1)}{2L(K+1)} = \frac{L+1}{2L}, \tag{4.3}$$

which is approximately equal to $1/2$ for large values of L.

The use of modified Booth-encoding on ordinary multipliers reduce the number of partial bits to about one half [9, 10] thus being comparable to the gain presented in equation (4.3). However, the complexity for implementing the selection logic in Booth-encoding is significantly larger than for the distributed arithmetic multiplier. Also, the presented reduction in arithmetic complexity, and a possible reduction in the number of memory accesses as mentioned in section 4, is made possible due to use of the distributed arithmetic multiplier.

5 Conclusion

An optimization of a combined FFT and FIR structure has been presented which reduce the number of arithmetic operations by more than 50% in the last butterfly stage. Furthermore, memory accesses are removed if a time-multiplexed architecture is used. The optimization is achieved by utilizing the properties of a distributed arithmetic multiplier and it adds no extra complexity or memory requirement.

A delayless acoustic echo canceller has been used as a target application of the optimization process. For the N tap IFFT-FIR structure of the canceller, the combined last stage of the IFFT and the FIR filter have been optimized. The number of adders and subtracters are reduced by approximately $1.5N$, and all FIR multipliers are removed at the cost of $N/2$ distributed arithmetic multipliers of comparable size. In the application N is in the range of 500 to 4000 taps, leading to a substantial overall complexity reduction. The presented optimization will save power and/or throughput at no extra overhead or loss in algorithmic performance.

References

[1] D. R. Morgan and J. C. Thi, "A delayless subband adaptice filter architecture," *IEEE Transactions on Signal Processing*, vol. 43, no. 8, Aug. 1995.

[2] A. Croisier, D. J. Esteban, M. E. Levilion, and V. Riso, "Digital filter for PCM encoded signals," US patent 3777130, Dec. 1973.

[3] A. Peled and B. Liu, "A new hardware realization of digital filter," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-22, no. 6, Dec. 1974.

[4] W. Kellerman, "Analysis and design of multirate systems for cancellation of acoustical echoes," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1988.

[5] A. Berkeman, V. Öwall, and M. Torkelson, "A prestudy of an echo canceler implementation," in *Proceedings of the International Conference on Signal Processing Applications and Technology*, Orlando FL, USA, Nov. 1999.

[6] A. Berkeman, V. Öwall, and M. Torkelson, "A low logic depth complex multiplier using distributed arithmetic," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 4, pp. 656–659, Apr. 2000.

[7] F. Catthoor, S. Wuytack, E. D. Greef, F. Balasa, L. Nachtergaele, and A. Vandecapelle, *Custom Memory Management Methodology.* Kluwer Academic Publishers, 1998.

[8] C. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, Feb. 1964.

[9] A. D. Booth, "A signed binary multiplication technique," *Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, pp. 236–240, 1951.

[10] L. P. Rubinfield, "A proof of the modified booth algorithm for multiplication," *IEEE Transactions on Computers*, Oct. 1975.

Paper V

Paper V

A Configurable Divider using Digit Recurrence

The division operation is essential in many digital signal processing algorithms. For a hardware implementation, the requirements and constraints on the divider circuit differ significantly with different applications. Therefore, it is not possible to design one divider component having optimal performance and cost for all target applications. Instead, the presented divider has a modular architecture, based on instantiation of small efficient divider sub-blocks. The configuration of the divider architecture is set by a number of parameters controlling wordlength, number of quotient bits, number of clock cycles per operation, and fixed or floating point operation. Digit recurrence algorithms with carry save arithmetic and on-the-fly two's complement output quotient conversion are used to make the sub-blocks small, fast and power efficient. The modularity gives the designer freedom to elaborate different parameters to explore the design space. Two applications using the proposed divider are presented. Furthermore, an example divider circuit has been fabricated and performance measurements are included.

Based on: A. Berkeman, Viktor Öwall, and Mats Torkelson "A Configurable Divider using Digit Recurrence," Submitted to *IEEE International Symposium on Circuits and Systems,* 2003.

1 Introduction

Division is a common operation in digital signal processing algorithms. A hardware implementation of the division operation is complicated, and depends on the environment of where the device is instantiated. For example, in some none time-critical applications, area can be saved by hardware reuse and the divider can be based on iterative execution. In other applications, the divider has maximum throughput, computing one result per clock cycle. Also, the quotient output of the divider has high dynamics, depending on the input data and wordlength, and an accuracy depending on how many recurrences or iterations the divider algorithm is run. For a fixed point implementation, it is likely that only a subrange of the quotient bits are used, while the others are discarded. Altogether, to get an optimal divider implementation, a thorough examination of the target application has to be performed. To cover the different cases, it would be necessary to have a set of divider implementations to choose from. This can be achieved by either having several implementations for various conditions or by having a reconfigurable divider with several configuration options.

The presented divider architecture is configurable and modular, written in a hardware description language, keeping parameters as word lengths and number of iterations generic until synthesis. This gives the designer freedom to explore the design space for each target application. Parameters can be set controlling important properties such as maximum clock cycle period, accuracy of the quotient, number of clock cycles per operation, and area. It is also possible to instantiate the divider with either a fixed or a floating point interface. Furthermore, effort has been made to achieve a power efficient solution.

The divider is based on several low complexity sub blocks, implemented using digit recurrence [1]. In this particular case the SRT algorithm [1, 2, 3] has been chosen. However, any digit recurrence algorithm could be used. The sub blocks are highly optimized, using carry save arithmetic to shorten the critical path and reducing power consuming switching. Each sub block has an on-the-fly output conversion unit, so that the quotient is always represented in two's complement.

The presented divider is currently used in two designs. One is an acoustic echo canceller where the divider is used in a Normalized Least Mean Squares (NLMS) implementation. The other application is a random number generator for a turbo codec interleaver. An example circuit has been fabricated as a stand-alone component to verify functionality and measure performance.

2 Divider Architecture

The flexibility of the divider architecture comes from instantiating small and efficient divider sub blocks. The sub block architecture is based on the SRT algorithm, but in fact any digit recurrence algorithm could be chosen. By connecting the sub blocks in different constellations, a set of different divider architectures conforming to various time and throughput constraints is constructed.

The architecture has two main levels of hierarchy. The lower level is the divider core, consisting of instantiations of the divider sub blocks. The higher level instantiates the divider core and the interfaces to the input and output signals. The interfaces can be either configured for fixed or floating point representation of numbers.

2.1 divider core

As will be shown in section 3, only one quotient bit is computed in the SRT recurrence. Thus, computation of several output bits require iterated use of the SRT operation. In a hardware implementation, this can be achieved either by hardware reuse or hardware duplication, that is, data can be looped through the SRT block in an iterative fashion, or several identical blocks can be connected in cascade for higher throughput. Due to the low complexity of the presented carry save SRT sub block, the implementation is built up of rather few gates. The carry save logic together with the low radix of the SRT algorithms assures a low gate depth and thereby a short critical path. Therefore, it is beneficial in most cases to connect several sub blocks in series. This increases the number of output quotient bits per clock cycle, while adapting the computational delay to the system clock speed. The architecture of a divider sub block is described in detail in section 3.1.

A parameter controls how many SRT sub blocks that will be connected in cascade, and how many iterations that will be performed on them. The number of iterations can be set so that no looping of the data is performed. In figure 2.1(a), a constellation optimized for maximal throughput is shown. It consists of P pipeline stages with N SRT blocks per stage, yielding PN output quotient bits per clock cycle. Figure 2.1(b), on the other hand, depicts a divider core with a loop over N SRT sub blocks, which in T clock cycles will produce NT output quotient bits.

Saving the decision of which constellation to use until the actual instantiation from a higher level of hierarchy ensures maximal flexibility for the designer. Also, it is easy to modify the parameters of the instantiated multiplier to explore the design space. A feature of the configuration in figure 2.1(b) is that

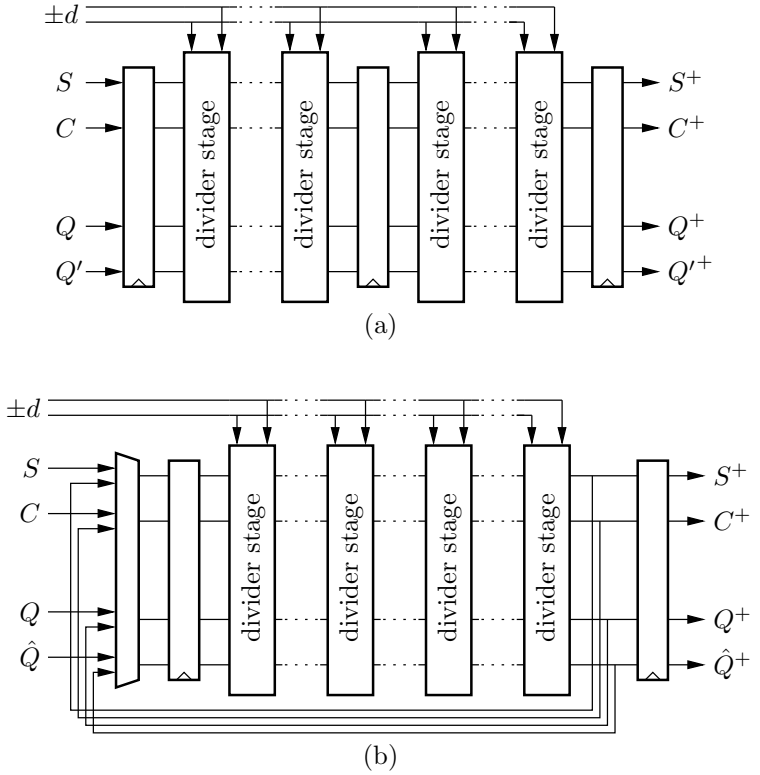


Figure 2.1: The two basic configurations of the divider.

it is possible to quit execution at run time when a pre-determined accuracy is attained. This results in a faster execution on average, while additionally lowering the power consumption.

2.2 TOP LEVEL ARCHITECTURE

The implemented SRT division algorithm requires the divisor and dividend to be initially normalized. Furthermore, the output is generated in a normalized fashion. With little extra logic, this is suitable for interfacing signals using a floating point number representation scheme, since the significand of a floating point number is in general normalized.

For fixed-point input numbers, however, additional logic has to be added at the input and output of the divider core. To be specific, normalization stages are added to the x and d inputs and a corresponding de-normalization

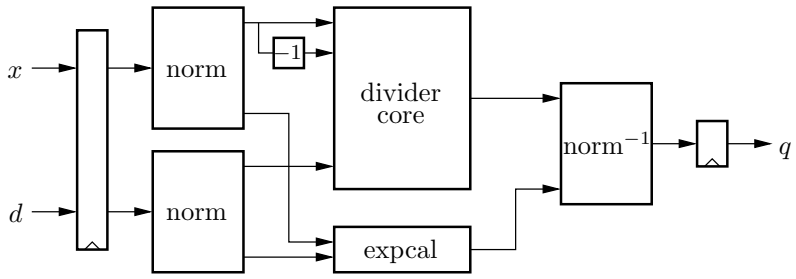


Figure 2.2: A top level of the fixed point divider architecture, having fixed to floating point normalizers to the left, and float to fixed point de-normalizer to the right. The *core* block contains the divider sub blocks and the *expcal* generates the correct number of shifts to the de-normalizer.

unit is added at the quotient output of the divider core, see figure 2.2. The normalization stage locates the least precision sign bit of the input words, and shifts left to put the sign bit in the most significant position. It also generates the corresponding exponents of the inputs before shifting. These exponents are subtracted and fed to the output de-normalizer after a delay that corresponds to the latency in clock cycles of the divider core. The inclusion of this normalization logic can be controlled by setting a parameter during synthesis.

3 Divider Sub Blocks

The divider sub blocks are based on the well-known SRT algorithm. This algorithm computes only one quotient bit per recurrence, but in a hardware implementation perspective, it is compact, straightforward, and efficient. Division of the number x by the number d is defined by

$$\begin{cases} x = qd + rem \\ |rem| < |d| \cdot ulp \end{cases}$$

where q is the quotient and rem is the remainder. The unit with least precision (ulp) is equal to 1 for integers and to r^{-N} for N-bit radix–r fractional numbers. Digit recurrence division is performed based on the equation

$$w_{j+1} = rw_j - q_j d, \tag{3.1}$$

where w_{j+1} is the next residual calculated from the current residual w_j, the radix r, current quotient digit q_j, and the dividend d. Initial residual $w_0 = x$.

It is assumed that x and d are normalized such that $1/2 \leq |x|, |d| < 1$. To ensure convergence of the algorithm, the quotient digit q_j has to be selected such that w_j conform to the error bound

$$|w_j| < d, \quad \forall j.$$

This selection is performed by a selection function SEL,

$$q_j = \mathrm{SEL}(rw_j, d).$$

Derivation of the selection function is based on selection intervals, defined by pairs $\{L_n(d), U_n(d)\}$ such that if

$$L_n(d) \leq rw_j \leq U_n(d),$$

it is possible to select $q_j = n$. The complexity of the selection function for a given radix depends strongly on the quotient digit set and the number system used to represent the residual. In order to make an efficient implementation of equation (3.1), there are three main points that have to be considered:

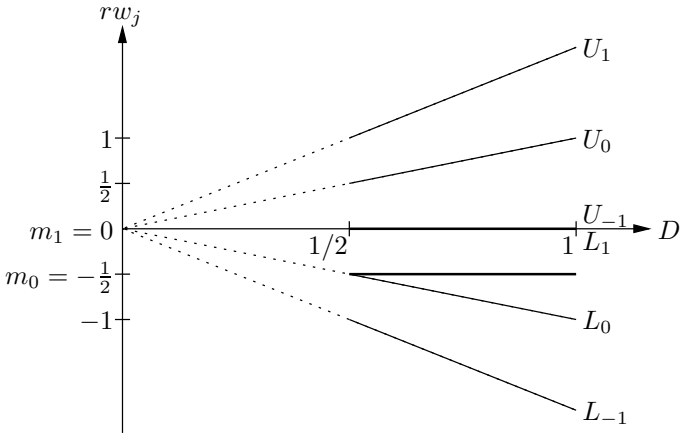
1. Calculation of the products rw_j and $q_j d$
2. The full scale subtraction $rw_j - q_j d$
3. Full scale comparison of rw_j and d in the selection function

The SRT algorithm gives an elaborate solution for the radix–2 case. Let q_j belong to the redundant radix–2 digit set

$$q_j \in \{-1, 0, 1\}. \tag{3.2}$$

For this digit set, selection intervals are derived [4], and presented in the table in figure 3.1. Due to the redundant digit set, there is always one or two choices for q_j. The situation is depicted in figure 3.1. Since d and x are normalized, the interesting range for d in the diagram is between $1/2$ and 1. For example, in the region limited horizontally by $d = 1/2$ and $d = 1$ and vertically by L_0 and U_0 it is legal to select $q_j = 0$. However, in the upper half of this area it is equally possible to select $q_j = 1$, and in the lower half $q_j = -1$ due to the redundancy.

The digit set in (3.2) implies a solution to point 1 mentioned earlier. The product $q_j d$ becomes trivial, $q_j d \in \{-d, 0, d\}$, where the $-d$ has to be calculated only once. Furthermore, the product rw_j corresponds to a logic shift left operation since $r = 2$. The subtraction mentioned in point 2 is simplified by



Upper limit	Lower limit	q_j
$U_1 = 2d$	$L_1 = 0$	1
$U_0 = d$	$L_0 = -d$	0
$U_{-1} = 0$	$L_{-1} = -2d$	-1

Figure 3.1: Selection intervals for $q_j \in \{-1, 0, 1\}$. Selection interval i is defined by the tuple (L_i, U_i).

utilizing carry save (CS) arithmetic, by representing the residual implicitly in a redundant format as a carry and a sum signal, that is,

$$rw_j = S + C. \tag{3.3}$$

Now, subtraction latency and power consumption can be significantly reduced. However, as the original selection function relies on the exact residual as stated in point 3, it has to be simplified. It can be proven that S and C can be stored with three integer bits plus a number of fraction bits depending on the word length of the inputs x and d [4]. If $\hat{y}_j$ is an approximation of rw_j, calculated by adding the t most significant bits of S and C, it is shown that the minimal but sufficient value of t is four. That is, three integer bits and the most significant fractional bit of $\hat{y}_j$ are enough input to the selection function. This solves the issue of point 3. The range of $\hat{y}_j$ can be calculated to

$$-\frac{5}{2} \leq \hat{y}_j \leq \frac{3}{2}, \tag{3.4}$$

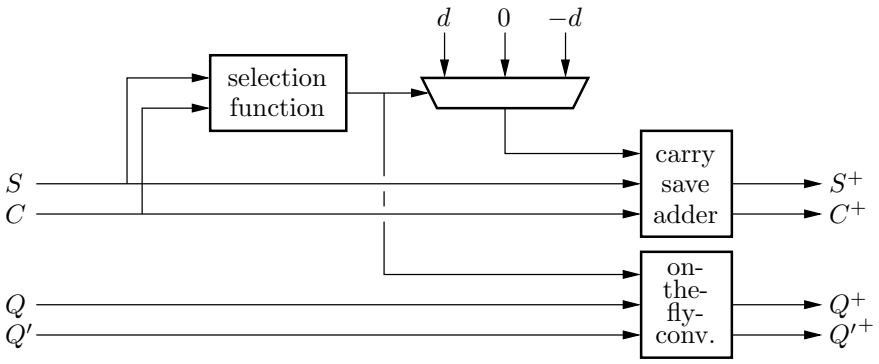


Figure 3.2: Schematic of an SRT sub block.

and the associated error

$$\epsilon_j = rw_j - \hat{y}_j \tag{3.5}$$

is always positive, and less than a least significant bit of $\hat{y}$, that is

$$0 \leq \epsilon_j < 1/2. \tag{3.6}$$

The corresponding selection function is

$$q_{j+1} = \begin{cases} 1 & : \quad 0 \leq \hat{y}_j \leq 3/2 \\ 0 & : \quad \hat{y}_j = -1/2 \\ -1 & : \quad -5/2 \leq \hat{y}_j \leq -1. \end{cases}$$

The selection intervals are visible in figure 3.1 as the thick lines $m_0 = -1/2$ and $m_1 = 0$, and can be explained as follows. If the approximation $\hat{y_j}$ is positive (or zero), y is also positive, following equations (3.5) and (3.6). Then it is safe to select $q_j = 1$ according to the selection interval $[L_1(d), U_1(d)]$. If $\hat{y_j} = -1/2$, on the other hand, y is larger or equal to $-1/2$, but less than 0. Then q_j can be selected to 0 as shown in figure 3.1. Similarly, if $\hat{y_j} \leq -1$, q_j can be selected to -1. Note that this selection function is independent of d, i.e. SEL = SEL(rw_j).

3.1 ARCHITECTURE OF THE SRT SUB BLOCK

A block diagram of the resulting SRT divider sub block is shown in figure 3.2. The inputs are the residual in S, C format, a positive and negated version of the dividend, and a partial quotient Q in two's complement representation.

The quotient is calculated on-the-fly from the $\{-1, 0, 1\}$ set into two's complement given the current quotient digit q_j, current partial quotient Q, and

Table 3.1: Power consumption versus clock frequency.

Frequency	V_{core}	Power
73 MHz	3.3 V	27 mW
47 MHz	2.0 V	4.8 mW
37 MHz	1.8 V	3.6 mW

a complementary signal Q'. This on-the-fly conversion is further described in [5]. Outputs from the core block are the new residual S^+, C^+ and quotient Q^+, Q'^+. The main blocks of the divider circuit are:

The selection function, operating on a total of eight bits from S and C, as explained in the previous section

Selection of d, $-d$ or 0 by a multiplexer ($q_j d$ multiplier)

Carry save adder to subtract $q_j d$ from the residual

On-the-fly converter. Consists of a number of fast multiplexers and low logic depth lookup tables [5]

As the sub block is small and has a low complexity, it can be realized using full-custom techniques. While a number of cascade connected sub blocks are used for most applications, the layout of the divider core will be regular. Performance in power and speed of such a regular full custom implementation will be significant at a rather low additional design cost.

4 Application Examples

The divider is currently instantiated in two different designs, an acoustic echo canceller and a turbo decoder. The adaptation part of the echo canceller is implemented using the Least Mean Squares (LMS) algorithm, which has reasonable complexity and accuracy, and is known to be stable. Convergence speed of the adaption is crucial to get a high signal quality. A common enhancement to the LMS algorithm is to normalize the input signal by its energy, yielding the Normalized LMS (NLMS), which has a dramatically improved convergence rate. A divider with seven SRT sub blocks looped thrice was instantiated in the echo canceller design. Seven sub blocks cope well with the clock frequency used for the rest of the chip, and the necessary accuracy was about 20 bits.

The second application is for the interleaver of a turbo channel coder. In this system, a large number of interleaver schemes are evaluated, and a lookup

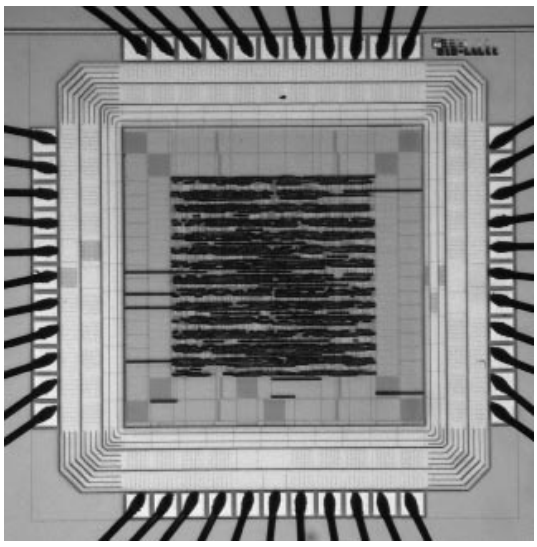


Figure 4.1: Chip micro photograph of the divider test chip.

table solution is infeasible. Instead, table data is calculated on-the-fly using a controllable random number series generator. This generator requires the modulo, or integer remainder, of two integers. The remainder is extracted from the divider in integer S, C-format, and is carry ripple added to get a non-redundant format. If the number is less than zero, a value of d is added to make the remainder positive.

5 Measured Results

A test chip has been fabricated to verify the operation of the divider in a real time environment. The divider operates on 16 bit integer inputs and generates a 16 bit quotient in four clock cycles by the use of a loop over four SRT sub blocks in cascade. The circuit was fabricated in a $0.35\,\mu$m three metal layer standard cell CMOS process. Initial measurements have been performed and power consumption figures for the divider core are presented in table 3.1.

6 Conclusion

A flexible divider architecture based on efficient digit recurrence sub blocks is presented. The divider is written in a hardware description language, and

is configurable at synthesis so that a wide range of the design space can be explored, including both iterated and pipelined architectures, and fixed or floating point operation. A test chip has been fabricated and performance measurements are presented. The design based on efficient sub blocks results in a small, regular, fast and power efficient solution, with a wide range of configuration options that makes it attractive for a number of target applications with different requirements.

References

[1] J. E. Robertson, "On the design of very high speed computer," Computer Science Department, University of Illinois at Urbana-Champaign, Tech. Rep. 80, 1957.

[2] J. Cocke and D. W. Sweeney, "High speed arithmetic in a parallel device," IBM, Tech. Rep., 1957.

[3] T. Tocher, "Techniques of multiplication and division for binary computers," *Quarterly Journal of Applied Math*, vol. 2, pp. 364–384, 1958.

[4] M. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, 1994.

[5] M. Ercegovac and T. Lang, "On-the-fly conversion of redundant into conventional representations," *IEEE Transactions on Computers*, 1987.