Modulation schemes for the audio beam system : fixed-point analysis, simulation and implementation

Seth, Abhishek

2012

Seth, A. (2012). Modulation schemes for the audio beam system : fixed-point analysis, simulation and implementation. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/48016>

<https://doi.org/10.32657/10356/48016>

NANYANG
TECHNOLOGICAL
UNIVERSITY

**MODULATION SCHEMES FOR THE AUDIO BEAM SYSTEM:
FIXED-POINT ANALYSIS, SIMULATION AND
IMPLEMENTATION**

ABHISHEK SETH

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

2012

**MODULATION SCHEMES FOR THE AUDIO BEAM SYSTEM:
FIXED-POINT ANALYSIS, SIMULATION AND
IMPLEMENTATION**

ABHISHEK SETH

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

A THESIS SUBMITTED TO THE NANYANG TECHNOLOGICAL UNIVERSITY

IN FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING

2012

Acknowledgments

I would like to express my deep and sincere gratitude to my supervisor, Associate Professor Dr. Gan Woon Seng for his continual advice and support. His wide knowledge and logical way of thinking have been of great value for me. His guidance helped me in all the time of research and writing of this thesis. I am extremely grateful for the financial support provided by Dr. Gan Woon Seng during my tenure as Masters student.

During this work I have collaborated with many colleagues for whom I have great regard. I wish to extend my special thanks to Mr. Tan Ee Leng who has advised me on several topics extending from the principles of nonlinear acoustics to the architecture of DSP processors. I am very grateful to him for his advice on the matters of thesis writing. I want to thank other past and present DSP Lab. members: Shi Chuang, Ji Peifeng, Ji Wei, Reuben Johannes, Wang Tong Wei, Nguyen Dinh Quy, Wang Liang, Nay Oo, Xing Jin, Jin Feng, Mu Hao, Zhu Caiyi, Kumar Dileep and Kushal Anand for the innumerable fruitful discussions I have with them.

I am grateful to both the technicians of DSP Lab.: Mr. Yeo Sung Kheng and Mr. Ong Say Cheng for their support and help in logistic and administrative matters.

My special gratitude is due to my family members for their loving support throughout my life.

Table of Contents

Acknowledgements.....	i
Table of Contents.....	ii
List of Figures.....	v
List of Tables.....	viii
List of Abbreviations.....	x
List of Symbols.....	xiii
Summary.....	xvii
 Chapter 1.....	 1
Introduction.....	1
1.1 Directional sound.....	1
1.2 Motivation.....	5
1.3 Major contributions of the Thesis.....	9
1.4 Organization of the Thesis.....	10
 Chapter 2.....	 12
Signal processing for audio beam system.....	12
2.1 Modulation schemes for audio beam system.....	14
2.1.1 Double-sideband amplitude modulation.....	14
2.1.2 Square root amplitude modulation.....	16
2.1.3 Single-sideband amplitude modulation.....	18
2.1.4 Modified amplitude modulation.....	21
2.1.5 Distortion analysis of modulation schemes.....	29
2.2 Fixed-point signal processing.....	33
2.2.1 Fixed-point number format and arithmetic.....	34
2.2.2 Finite wordlength effects.....	37

2.3	Performance metrics.....	40
2.3.1	Signal-to-quantization-noise ratio.....	40
2.3.2	Computational workload.....	42
2.4	Real-time implementation platform.....	46
2.5	Conclusion.....	49
Chapter 3		50
Double-sideband and modified amplitude modulation		50
3.1	Double-sideband amplitude modulation.....	51
3.2	Modified amplitude modulation.....	53
3.3	Q format selection for MAM implementation.....	57
3.4	Adaptive MAM.....	61
3.5	Real-time implementation.....	65
3.6	Conclusion.....	66
Chapter 4		67
Square root amplitude modulation		67
4.1	Fixed-point square root amplitude modulation.....	68
4.2	Fixed-point square rooting methods.....	70
4.3	Square root approximation by real functions.....	71
4.3.1	Taylor's series expansion.....	72
4.3.2	Chebyshev expansion.....	75
4.4	Square rooting algorithms based on Newton-Raphson formula.....	76
4.4.1	Direct Newton-Raphson variant 1.....	77
4.4.1.1	DNR-1 using PEs for SQRT and ISQRT initialization.....	78
4.4.1.2	DNR-1 using LUTs for SQRT and ISQRT initialization.....	81
4.4.2	Nonlinear infinite-impulse response filter method.....	88
4.4.3	Direct Newton-Raphson variant 2.....	90
4.4.4	Inverse Newton-Raphson.....	92
4.5	Performance comparison of square rooting methods.....	93

4.6	Real-time implementation.....	95
4.7	Conclusion.....	98
Chapter 5.....		99
Single-sideband amplitude modulation.....		99
5.1	Single-sideband amplitude modulation for the ABS.....	100
5.2	Truncated DSBAM.....	101
5.3	SSBAM using Hilbert transformer.....	104
5.3.1	Hilbert transformer with finite-impulse response.....	106
5.3.2	FIR-HT based single-rate SSBAM.....	108
5.3.3	FIR-HT based multi-rate SSBAM.....	109
5.3.4	Performance comparison of FIR-HT based single-rate and multi-rate SSBAM.....	124
5.3.5	SSBAM using IIR filters.....	125
5.4	SSBAM using Weaver's method.....	126
5.4.1	Multi-rate SSBAM using Weaver's method.....	132
5.4.2	Performance comparison of single-rate and multi-rate Weaver's method.....	138
5.5	Performance comparison of SSBAM generation methods.....	139
5.6	Real-time implementation.....	141
5.7	Conclusion.....	142
Chapter 6.....		143
Conclusions and future works.....		143
6.1	Conclusions.....	143
6.2	Future works.....	147
Appendix A.....		152
Appendix B.....		157
Appendix C.....		167
Appendix D.....		170
Author's publications.....		174
References.....		175

List of Figures

Figure 1.1	Private listening zones created by directional sound beams.....	2
Figure 1.2	Application of directional sound beams: (a) at poster/art gallery and (b) multi-lingual teleconferencing.....	3
Figure 1.3	Formation of parametric array in air.....	4
Figure 1.4	Self-demodulation of ultrasonic frequencies in air.....	6
Figure 1.5	Block diagram of ABS.....	8
Figure 2.1	ABS with fixed-point processor.....	12
Figure 2.2	Block diagram of DSBAM.....	15
Figure 2.3	DSBAM: (a) time domain and (b) frequency spectrum plots.....	16
Figure 2.4	Block diagram of SRAM.....	17
Figure 2.5	SRAM: (a) time domain and (b) frequency spectrum plots.....	18
Figure 2.6	Block diagram of SSBAM.....	19
Figure 2.7	SSBAM with lower-sideband: (a) time domain and (b) frequency spectrum plots..	20
Figure 2.8	Block diagram of MAM.....	21
Figure 2.9	Block diagram of MAM1.....	23
Figure 2.10	MAM1: (a) time domain and frequency spectrum plots.....	24
Figure 2.11	Block diagram of MAM2.....	25
Figure 2.12	MAM2: (a) time domain and frequency spectrum plots.....	26
Figure 2.13	Block diagram of MAM3.....	27
Figure 2.14	MAM3: (a) time domain and frequency spectrum plots.....	28
Figure 2.15	Transducer and air modeling for ABS.....	29
Figure 2.16	(a) THD and (b) IMD plots for modulation schemes.....	32
Figure 2.17	Different $Q_{m,n}$ fixed-point formats.....	35
Figure 2.18	Quantization noise model.....	39
Figure 2.19	Roundoff operation modeled as the addition of white noise.....	39
Figure 2.20	Modulation schemes implemented on finite precision fixed-point system generates quantization errors.....	41
Figure 2.21	Generating error values for N -bit fixed-point processor simulated on MATLAB..	42

Figure 2.22	Multi-rate implementation of DSP algorithm.....	43
Figure 2.23	Multi-rate implementation of DSP algorithm using block processing.....	44
Figure 2.24	TMS320C5505 functional-block diagram.....	47
Figure 2.25	TMSC5505 eZDSP USB stick development board.....	48
Figure 3.1	Single Q format implementation of fixed-point DSBAM.....	51
Figure 3.2	Mixed Q format implementation of fixed-point DSBAM.....	52
Figure 3.3	Block diagram of MAM1, MAM2 and MAM3.....	54
Figure 3.4	Single Q format implementation of fixed-point MAM1, MAM2 and MAM3.....	55
Figure 3.5	Mixed Q format implementation of fixed-point MAM1, MAM2 and MAM3.....	56
Figure 3.6	$SQNR_{MAMq, Diff}$ for (a) MAM1, (b) MAM2 and (c) MAM3.....	59
Figure 3.7	Operating regions of MAM3, MAM2, MAM1 and SRAM for single Q format implementation of MAM3 on (a) 8-bit, (b) 16-bit and (c) 32-bit fixed-point processors.....	64
Figure 3.8	Block diagram of adaptive MAM.....	65
Figure 4.1	Fixed-point SRAM implemented in $Q1.N-1$ format.....	68
Figure 4.2	Effect of SQRT operator on performance of SRAM.....	69
Figure 4.3	Histograms showing output bit precision of TSA on 8-bit fixed-point processor...	73
Figure 4.4	Histograms showing output bit precision of TSA on 16-bit fixed-point processor...	74
Figure 4.5	Histograms showing output bit precision of TSA on 32-bit fixed-point processor...	74
Figure 4.6	Block diagram of DNR-1.....	78
Figure 4.7	$DNR_T(n)$ as a cascade of non-repetitive and repetitive structures.....	83
Figure 4.8	Block diagram of $DNR_T(1)$, $DNR_T(3)$, $DNR_T(5)$ and $DNR_T(7)$	84
Figure 4.9	Block diagram of $DNR_T(2)$	85
Figure 4.10	EP LUT: (a) 8-bit entry stored in EP ISQRT/2 LUT and (b) EP ISQRT/2 LUT decoder.....	86
Figure 5.1	Single-rate (a) LSBAM and (b) USBAM for floating-point implementation.....	101
Figure 5.2	Single-rate (a) LSBAM and (b) USBAM for fixed-point implementation.....	102
Figure 5.3	SSBAM using HT for floating-point implementation.....	104
Figure 5.4	SSBAM using HT for fixed-point implementation.....	105
Figure 5.5	Time alignment for FIR-HT based fixed-point SSBAM.....	107

Figure 5.6	FIR-HT based multi-rate SSBAM.....	110
Figure 5.7	Multi-rate SSBAM with (a) single-stage and (b) multi-stage interpolation design for audio signal.....	112
Figure 5.8	Interpolation by factor of L using M stages.....	113
Figure 5.9	Normalized signal spectrum for multi-stage interpolation design: (a) normalized spectrum of base-band signal, (b) normalized signal spectrum after interpolation by a factor of L_1 , (c) normalized signal spectrum at output of first interpolation stage, (d) normalized signal spectrum after interpolation by a factor of L_2 , and (e) normalized signal spectrum at output of second interpolation stage.....	115
Figure 5.10	Polyphase implementation of factor of L LPIF using sub-filters.....	120
Figure 5.11	SSBAM using all-pass IIR filters.....	126
Figure 5.12	Weaver's method for generation of SSBAM signals.....	127
Figure 5.13	Weaver's method for ABS.....	127
Figure 5.14	Signal spectrums for Weaver's method: (a) spectrum of input signal, (b) signal spectrum at output of HPF, (c) signal spectrum after first DSBAM, (d) signal spectrum at output of LPF and (e) spectrum of output signal.....	130
Figure 5.15	Multi-rate Weaver's method: (a) multi-rate Weaver's method with 3 stages and (b) polyphase implementation of multi-rate Weaver's method.....	135
Figure 6.1	IMD plot for modulation schemes for (a) $f_{\text{trans}}=6$ kHz and (b) $f_{\text{trans}}=12$ kHz.....	145
Figure 6.2	ABS with phase equalization.....	148
Figure 6.3	ABS with PM.....	149
Figure 6.4	ABS with Class D amplifier.....	150

List of Tables

Table 2.1	Dynamic range and precision for a 16-bit number using different Q formats.....	36
Table 3.1	$AVG_SQNR_{MAMq,Diff}$ for MAM1, MAM2 and MAM3.....	60
Table 3.2	Relationship defining operating regions for MAM3, MAM2, MAM1 and region in which MAM becomes SRAM.....	62
Table 3.3	MIPS results for DSBAM, MAM1, MAM2, MAM3 and adaptive MAM.....	66
Table 4.1	Performance of TSA on 8-bit, 16-bit, and 32-bit fixed-point processors.....	75
Table 4.2	Performance of CPA on 8-bit, 16-bit and 32-bit fixed-point processors.....	76
Table 4.3	Naming convention used for DNR-1(SQRT(x) PE: ISQRT(x)/2 PE).....	79
Table 4.4	Performance comparison for DNR-1 using PEs on 8-bit, 16-bit, and 32-bit fixed-point processors.....	80
Table 4.5	Normal and EP LUT size and performance comparison.....	87
Table 4.6	Performance of DNR _T (n) algorithms.....	88
Table 4.7	Performance comparison for NLIIRF using PEs on 8-bit, 16-bit, and 32-bit fixed-point processors.....	90
Table 4.8	Performance comparison for DNR-2 using PEs on 8-bit, 16-bit, and 32-bit fixed-point processors.....	91
Table 4.9	Performance comparison for INR on 8-bit, 16-bit, and 32-bit fixed-point processors.....	93
Table 4.10	Performance comparison of square rooting methods on 8-bit, 16-bit and 32-bit fixed-point processors.....	94
Table 4.11	MIPS comparison of proposed square rooting method and DSPLIB SQRT.....	96
Table 4.12	MIPS and SQNR comparison of SRAM implemented with proposed square rooting method and DSPLIB SQRT.....	97
Table 5.1	Filter specification for LSBAM and USBAM.....	104
Table 5.2	f_{IN} , f_{OUT} and L values for audio and speech signals.....	111
Table 5.3	Specifications of FIR-HT based multi-rate SSBAM.....	111
Table 5.4	Different configurations of multi-stage interpolator for audio and speech signal...	121
Table 5.5	Performance comparison for various configurations of multi-stage interpolator.....	122

Table 5.6	Performance comparison for FIR-HT based multi-rate SSBAM with multi-stage interpolator.....	124
Table 5.7	Computational workload comparison for FIR-HT based single-rate and multi-rate SSBAM.....	125
Table 5.8	Parameters for single-rate Weaver's method.....	132
Table 5.9	Parameters for multi-rate Weaver's method.....	136
Table 5.10	Filter specification and MPOS values for multi-rate Weaver's method.....	137
Table 5.11	MPOS comparison for single-rate and multi-rate Weaver's method.....	138
Table 5.12	Computational workload comparison of SSBAM generation methods.....	139
Table 5.13	MPOS and SQNR values for 8-bit, 16-bit and 32-bit fixed-point processors for audio and speech signals.....	140
Table 5.14	MIPS results for 16-bit fixed-point implementation of FIR-HT based SSBAM...	142
Table 6.1	Computational workload and THD comparison for modulation schemes simulated and implemented on 16-bit fixed-point processor.....	144
Table 6.2	Computational workload and IMD comparison for modulations schemes.....	146
Table 6.3	Efficiency of amplifier topologies	150

List of Abbreviations

ABS	Audio beam system
ADC	Analog-to-digital converter
AGC	Automatic gain controller
APOS	Additions per output sample
CCS	Code composer studio
CPA	Chebyshev polynomial approximation
CPE	Cubic polynomial expansion
DAC	Digital-to-analog converter
DNR-1	Direct Newton-Raphson variant 1
DNR-2	Direct Newton-Raphson variant 2
DNR _T (n)	DNR-1 based tabular method for calculating square root of fixed-point numbers
DP	Double-precision
DSBAM	Double-sideband amplitude modulation
DSP	Digital signal processing
DSPLIB SQRT	C55x DSP library square root algorithm
EP	Enhance precision
FIR	Finite-impulse response
HPF	High-pass filter
HT	Hilbert transformer
HWFFT	FFT hardware accelerator
IDE	Integrate development environment
IIR	Infinite-impulse response
IMD	Inter-modulation distortion
INR	Inverse Newton-Raphson
ISQRT	Inverse square root
KZK	Khokhlov-Zabolotskaya-Kuznetsov

LPE	Linear polynomial expansion
LPF	Low-pass filter
LPIF	Low-pass interpolation filter
LSB	Lower-sideband
LSBAM	Lower-sideband amplitude modulation
LUT	Look-up table
MAM	Modified amplitude modulation
MAM _q	Modified AM _q
MIPS	Million instructions per second
MPOS	Multiplications per output sample
MSB	Most significant bit
N.A.	Not applicable
NLIIRF	Nonlinear infinite-impulse response filter
NPP	Non-polyphase implementation
NR	Newton-Raphson
PE	Polynomial expansion
PM	Phase modulation
PP	Polyphase implementation
PWM	Pulse width modulation
QAM	Quadrature amplitude modulation
QPE	Quadratic polynomial expansion
SPL	Sound pressure level
SQRT	Square root
SQNR	Signal-to-quantization-noise ratio
SR-SSBAM	Single-rate single-sideband amplitude modulation
SRAM	Square root amplitude modulation
SSBAM	Single-sideband amplitude modulation
TDSBAM	Truncated double-sideband amplitude modulation
THD	Total harmonic distortion
TI	Texas Instruments

TSA	Taylor's series approximation
USB	Upper-sideband
USBAM	Upper-sideband amplitude modulation

List of Symbols

Only the main symbols used most frequently throughout the thesis are listed here. Occasionally, some of the symbols listed below are assigned alternative definitions.

Δ	Quantization interval
ϕ	Phase in radians
ρ_0	Ambient density
α	Absorption coefficient of air
β	Coefficient of nonlinearity for air
δ_p	Pass-band ripple of LPF/HPF in dB
δ_s	Stop-band attenuation of LPF/HPF in dB
$g\omega_{t,IFi}$	Additional normalized transition bandwidth gained at i^{th} interpolation stage
$\omega_{p,IFi}$	Normalized pass-band edge of i^{th} interpolation filter
ω_{sg}	Normalized signal bandwidth
$\omega_{s,IFi}$	Normalized stop-band edge of i^{th} interpolation filter
ω_t	Normalized transition bandwidth of FIR based HT
$\omega_{t,IFi}$	Normalized transition bandwidth of i^{th} interpolation filter
a	Radius of parametric loudspeaker
b_i	Bit value of the i^{th} bit of an N -bit fixed-point number
c_0	Small signal sound speed in air
$e[n]$	Quantization noise
f_c	Ultrasonic carrier frequency in Hz
f_{DSBAM1}	Modulating frequency of the first double-sideband amplitude modulator used in Weaver's method in Hz
f_{DSBAM2}	Modulating frequency of the second double-sideband amplitude

	modulator used in Weaver's method in Hz
f_{IN}	Input sampling frequency in Hz
f_{lt}	Lower transition band of FIR based HT in Hz
f_{OUT}	Output sampling frequency in Hz
f_{p}	Pass-band edge of LPF/HPF in Hz
f_{s}	Stop-band edge of LPF/HPF in Hz
f_{sg}	Signal bandwidth in Hz
f_{trans}	3 dB bandwidth of the ultrasonic transducer in Hz
f_{ut}	Upper transition band of FIR based HT in Hz
$h_{\infty}[n]$	Infinitely long impulse response of the HT
$h[n]$	Finite length impulse response of the HT
m	Modulation index
q	Variable representing MAM order
r	Axial distance from ultrasonic source
$x[n]$	Input signal
$x_N[n]$	$x[n]$ quantized to N bits
$\hat{x}[n]$	Hilbert transform of $x[n]$
$\text{AVG_SQNR}_{\text{MAM}q, \text{Diff}}$	Average SQNR difference between single Q and mixed Q format implementation of $\text{MAM}q$
B	Sample block size for block processing
CC	Number of CPU clock cycles required to process B samples
$D_{if_1 \pm jf_2}$	Inter-modulation harmonics at $if_1 \pm jf_2$
$H_{\infty}(e^{j\omega})$	Frequency response of the HT with infinitely long impulse response
I_{DX}	LUT index
$\text{ISQRT}(x)$	$\frac{1}{\sqrt{x}}$
L	Upsampling factor
$\text{LUT}_{\text{ISQRT}}(i)$	i^{th} entry of the ISQRT LUT
$\text{LUT}_{\text{SQRT}}(i)$	i^{th} entry of the SQRT LUT

$MIPS_{\text{Multi-rate}}$	MIPS for multi-rate processing
$MIPS_{\text{Single-rate}}$	MIPS for single-rate processing
$MPOS_{FIR\ HT, MR}$	MPOS for multi-rate SSBAM (using FIR based HT) with multi-stage interpolator
$MPOS_{FIR\ HT, MR, PP}$	MPOS for multi-rate SSBAM (using FIR based HT) with polyphase implementation of multi-stage interpolator
$MPOS_{FIR\ HT, SR}$	MPOS for single-rate SSBAM using FIR based HT
$MPOS_{IF}$	MPOS for multi-stage interpolator
$MPOS_{IF, PP}$	MPOS for multi-stage interpolator with polyphase implementation
$MPOS_{TDSBAM, SR}$	MPOS for single-rate TDSBAM
$MPOS_{\text{Weaver}, MR}$	MPOS for multi-rate Weaver's method
$MPOS_{\text{Weaver}, MR, PP}$	MPOS for multi-rate Weaver's method with polyphase implementation
$MPOS_{\text{Weaver}, SR}$	MPOS for single-rate Weaver's method
N	Wordlength of a fixed-point number
P_0	Wave pressure at the ultrasonic source
P_2	Demodulated wave pressure at distance r along the axis of propagation
P_e	Quantization noise power
P_y	Unquantized signal power
$Q_{m.n}$	Fixed-point number representation where m bits are used to represent the integer portion, and n bits are used to represent the fractional portion
$SQRT(x)$	$\sqrt{x}$
$SQNR_{DSBAM, Diff}$	SQNR (in dB) difference between single Q and mixed Q format implementation of DSBAM
$SQNR_{DSBAM, Mixed\ Q}$	SQNR (in dB) for mixed Q format implementation of DSBAM in dB
$SQNR_{DSBAM, Single\ Q}$	SQNR (in dB) for single Q format implementation of DSBAM
$SQNR_{MAMq, Diff}$	SQNR (in dB) difference between single Q and mixed Q format implementation of MAM q
$SQNR_{MAMq, Mixed\ Q}$	SQNR (in dB) for mixed Q format implementation of MAM q
$SQNR_{MAMq, Single\ Q}$	SQNR (in dB) for single Q format implementation of MAM q

T_a	Attack time in seconds
T_r	Release time in seconds
T_1	Amplitude of fundamental frequency component at f_1
T_i	Amplitude of higher harmonics at if_1
V_{FS}	Full scale (peak to peak) voltage in Volts
$X[n]$	Logarithmic value (base 10) of $x[n]$

Summary

Directional sound finds application in many areas such as entertainment, gaming, communication, and personal messaging. Directional sound can be created by modulating an audible signal onto an ultrasonic carrier and projecting the modulated signal via special types of ultrasonic transducer to generate the parametric array through air, in such a way that audible sound can travel in a column of sound beams. The device that generates this phenomenon is generally referred to as the parametric loudspeaker. Parametric loudspeaker uses several proprietary modulation schemes to generate audible sound with lesser distortion. These modulation schemes constitute the major signal processing tasks in the parametric loudspeaker system. Due to several advantages, such as size, speed, flexibility, accuracy, and ease of programming, digital implementation of the modulation schemes is preferred over analog approach.

Fixed-point programmable digital signal processing (DSP) processor is a common embedded processor used in many low power and cost-efficient consumer electronic designs. Compared to floating-point DSP processors, fixed-point DSP processors are low cost, high speed, and power-efficient processors, but with lower precision and signal-to-quantization-noise ratio (SQNR). SQNR is the most commonly used criterion for evaluating the system accuracy in the digital domain. As we are interested in the real-time implementation of the modulation schemes used in parametric loudspeaker, we have the dual objectives of maximizing the SQNR and minimizing the computational workload of modulation schemes. For each modulation

scheme, there exist several algorithms. Therefore, careful analysis on each algorithm can help in achieving the dual objectives. In addition, new algorithms and implementation techniques are devised to further achieve a more efficient real-time implementation. For each modulation scheme, the algorithm with maximum SQNR and minimum computational workload is selected for real-time implementation. Real-time profiling of the selected algorithms is carried out on a 16-bit fixed-point DSP processor in terms of their processing MIPS, memory usage and accuracies.

Chapter 1

Introduction

Sound is an oscillation of pressure transmitted through solid, liquid, or gas, and consists of frequencies within the range of hearing and of a level sufficiently strong to be heard [1]. Sound propagates through air as longitudinal wave and its speed varies with temperature. Frequencies that can be perceived by human ears are known as audible frequencies, which range from 20 Hz to 20 kHz. In the past, when there was no communicating device, sound was mainly used for communication. E. W. Siemens in 1874 [2] developed the first sound reproduction device based on moving coil transducer. In 1934, E. W. Kellogg patented an improved version of the electrostatic loudspeaker [3]. With advancement in sound reproduction technology, many sound related systems were developed for commercial and public use, for instance radio, telephone, surround sound systems are in common use today.

1.1 Directional sound

The ability to control sound radiation patterns in entertainment, gaming, communication and personal messaging is becoming an important differentiating feature in many commercial products. A common feature in these systems is to create a highly directional sound field to targeted audiences by forming a private listening zone, as shown in Fig. 1.1.

Figure 1.1 Private listening zones created by directional sound beams.

Creation of private listening zone is useful in many scenarios, as shown in Fig. 1.2(a) and (b). Private listening zone is desirable at places like art galleries, museums, etc. It can be used with vending and dispensing machines, automatic teller machine, exhibition booths, and billboards, etc. Alternatively, directional sound can be useful in projecting different languages to different listening zones in a multi-lingual teleconferencing, as shown in Fig. 1.2(b).

(a)

(b)

Figure 1.2 Application of directional sound beams: (a) at poster/art gallery and (b) multi-lingual teleconferencing (part of the picture is extracted from [4]).

Several techniques are available to accomplish directional sound projection, and each has its benefits and drawbacks. In 1995, Brown patented an acoustic imaging sound dome [5]. He used an acoustically reflective dome to transmit audible frequencies to the interior of dome by reflection forming a small listening area under the dome. In 2003, 1 Limited and Dolby Laboratories collaborated to study the digital sound projector technology [6] for the commercial cinema market. By using array signal processing technique, the digital sound projector is able to generate multiple high directivity beams. This approach creates virtual speakers for surround sound system. Both acoustic imaging sound dome and digital sound projector are transmitting audible frequencies directly from the transducer. As audible frequencies cover about 10 octaves, array signal processing is not very efficient in generating directional audio beams. Even with a broadband beamformer, it requires a very large transducer array (few meters) to beamform the low frequency (bass). This approach of creating a focused sound beam incurs a high cost and cannot be generated by small loudspeaker. Researchers have also been experimenting on the

creation of directional sound through the nonlinear interaction of collimated ultrasound waves in air since the early 1960s [7]. Ultrasound has wavelengths much smaller than audible sound, and thus can form a much narrower beamwidth than any traditional audible loudspeaker system. An ultrasonic transducer can be used to project a narrow beam of modulated ultrasound that is powerful enough to substantially change the speed of sound in the air that it passes through. These high frequency modulated beams are commonly referred as primary waves. The air within the beam behaves nonlinearly and demodulates the audible modulating signal from the ultrasound, resulting in sound that can be heard only along the path of the beam, or that appears to radiate from any surface that the beam strikes. The nonlinear interaction of primary waves in air gives rise to an end-fire array of virtual sources, which is referred to as parametric array. An illustration of parametric array [8] is given in Fig. 1.3.

Figure 1.3 Formation of parametric array in air [8].

The ultrasonic transducer emits a finite-amplitude (very high intensity) modulated sound beam into air. This produces an end-fire virtual array along the axis of propagation. The generation of these virtual sources extends to the point where the sound (ultrasonic beam) ceases to be of high enough intensity. In this way, audible sound can be generated in a tight column as shown in Fig. 1.3. This column of virtual audible sources forms an end-fire array of audible sources that add up in phase along the axis of propagation. Therefore, along the propagation axis, the audible sources get louder at greater distances from the transducer. Outside the propagation axis, the virtual audible sources are out of phase and interfere destructively. In Fig. 1.3, the audible sound waves take on the directivity of the ultrasonic wave and the generation of these virtual sources extends to an effective array distance (or absorption length) within the near field. This effective distance is defined as the distance whereby the sound pressure level (SPL) of the ultrasound falls (or acoustic energy of ultrasound is absorbed by the atmosphere) below the utilization limit of nonlinearities. Outside the effective array distance, the virtual end-fire array ceases to exist and audible sound will not be generated within a tight beam. Instead, the audible sound starts to attenuate at -6 dB per doubling distance outside this effective array distance. Unlike the omnidirectional sound radiation generated by a conventional loudspeaker that is attenuated by -6 dB for every doubling of the distance starting from the loudspeaker surface; the audible sound generated by a parametric loudspeaker becomes louder as the distance increases, until the ultrasound no longer causes nonlinear effects in air in the far field.

1.2 Motivation

Parametric array was first discovered and explained by Westervelt [7] in 1963. It was first used in underwater, and subsequently, the use of parametric array in air was first verified

experimentally by Bennet and Blackstock [9] in 1975. They showed that an additional difference frequency of 5 kHz was generated from 18.6 kHz and 23.6 kHz frequencies. This experiment showed that it is possible to generate an audible frequency from inaudible ultrasonic frequencies. Since then, the parametric array in air has been developed, and the device that generates this phenomenon is generally referred to as the parametric loudspeaker. Audible frequencies are produced due to nonlinear interaction of ultrasonic frequencies in air as shown in Fig.1.4. Nonlinear interaction of finite-amplitude ultrasonic frequencies in air produces sum and difference of the fundamental frequencies and their harmonics. The sum frequencies lie above the human audible frequency range, while the difference frequencies fall within the human audible frequency range.

Figure 1.4 Self-demodulation of ultrasonic frequencies in air.

In 1982, Yoneyama et al. used the parametric loudspeaker to generate broadband audio signal [10]. Their experiments revealed that the demodulated sound wave generated by the nonlinear acoustic phenomenon has a very sharp directivity pattern. However, the demodulated signal suffers from high harmonic distortion. In 1984, Kamakura et al. [11] reduced the distortion from double-sideband amplitude modulation by pre-processing the modulating signal. Similarly,

Pompei introduced a practical device in 1998 [12], which adopted the pre-processing technique proposed by Kite et al. [13]. About the same time, Croft and Norris [14] commercialized the hypersonic system by using proprietary algorithms and transducers.

Audio beam system (ABS) is the name given to the parametric loudspeaker system developed by the School of Electrical and Electronic Engineering at Nanyang Technological University [15]. The ABS is based on the above nonlinear acoustics principle in air and uses proprietary modulation schemes (known as the modified amplitude modulation) to create audible signal in air with low audible distortion. Beside the modified amplitude modulation (MAM) technique, our ABS can also be programmed to implement other types of modulation techniques, such as double-sideband amplitude modulation (DSBAM), square root amplitude modulation (SRAM), and single-sideband amplitude modulation (SSBAM). The modulated signal is amplified using analog amplifier before passing to a special type of ultrasonic transducer that can project sound in a highly directional manner. In this thesis, the ABS serves as an experimenting platform in implementing different types of fixed-point modulation techniques, and examines their digital noise performance and computational loads. The block diagram of the ABS is shown in Fig. 1.5.

Figure 1.5 Block diagram of ABS.

From Fig. 1.5, it is noted that modulation schemes play an important role in ABS. Due to several advantages, such as size, speed, flexibility, accuracy, and ease of programming, digital implementation of the modulation schemes is preferred over analog approach. A fixed-point digital signal processing (DSP) processor caters to all these needs is used in this work. A previous implementation platform based on the FPGA was reported in [16]. However, in this work, we look into the digital implementation of modulation techniques based on a programmable fixed-point DSP processor.

Fixed-point DSP processors are low cost, high speed, power-efficient processors but with lower precision and lower signal-to-quantization-noise ratio (SQNR) compared to the floating-point DSP processors. Fixed-point programmable DSP processors have now become a common embedded processor in many high-volume designs in which time to market is critical. Developing optimum software for the designed algorithm on a given DSP processor requires an understanding of the processor architecture, instruction set, addressing modes, and software development techniques. However, fixed-point programming added extra challenge like

quantization and roundoff errors considerations compared to floating-point programming. These digital errors introduce noise in the system and reduce SQNR.

Real-time implementation of ABS is bound to meet strict deadlines. Since ABS operates at sampling frequencies lying in the ultrasonic range, therefore, meeting strict deadlines at high sampling rates necessitates efficient programming, and minimizing the computational workload of signal processing algorithms. Each modulation scheme used in the ABS can be realized by more than one algorithm. However, the SQNR performance and computational workload of each algorithm differs. Therefore, the objective of this thesis is to investigate and formulate algorithms, which maximize SQNR and minimize computational workload for each modulation scheme on a fixed-point DSP processor.

1.3 Major contributions of the Thesis

In this thesis, analysis, simulation, and implementation of different modulation schemes are carried out on a fixed-point DSP processor. Major tasks that are undertaken in this thesis can be summarized as follows:

1. Several algorithms to realize each modulation scheme are evaluated on a fixed-point DSP processor. Fixed-point analysis of each algorithm is performed to maximize its SQNR performance. The analysis performed is generic in nature and is applicable to fixed-point processor of any wordlength. To reduce the computational workload, several well-known techniques, like multi-rate signal processing and polyphase filter structures are studied and applied to these algorithms. For each modulation scheme, the algorithm with maximum SQNR performance and minimum computational workload is selected.

2. The modulation schemes are simulated on fixed-point DSP processor. The SQNR performance and computational workload of modulation schemes are evaluated for 8-bit, 16-bit, and 32-bit fixed-point DSP processors.
3. For real-time implementation of each modulation scheme, the algorithm with the maximum SQNR and minimum computational workload is implemented on a 16-bit fixed-point DSP processor. Processing million instruction per second (MIPS) and memory usage of these algorithms are profiled in the real-time processor.
4. A method is proposed to find the square root of fixed-point numbers. The proposed square rooting method outperforms all existing methods in terms of SQNR performance and computational workload. The proposed method is used for the real-time implementation of SRAM on a 16-bit fixed-point DSP processor.

1.4 Organization of the Thesis

The thesis is organized as follows. Chapter 2 discusses different modulation schemes used by the ABS. This chapter introduces fixed-point arithmetic format, DSP architecture, and finite wordlength effects. In addition, performance metrics to quantify SQNR performance and computational workload are defined. The real-time fixed-point DSP processor used in this thesis for implementation purpose is also described. Chapter 3 investigates DSBAM and MAM modulation schemes. Two algorithms to realize these modulation schemes are discussed, and evaluated for SQNR performance and computational workload. The algorithm with maximum SQNR performance and minimum computational workload is selected, and its real-time DSP implementation is performed. Chapter 4 investigates the implementation of SRAM. Several existing algorithms and a proposed algorithm to find the square root (SQRT) of a fixed-point

number are outlined in detail. The proposed square rooting algorithm outperforms existing algorithms in terms of SQNR performance and computational workload, and is used for implementing SRAM on a real-time fixed-point DSP processor. Chapter 5 investigates three different algorithms for generating SSBAM signal. Existing techniques to reduce computational workload are applied to each of these algorithms. The algorithm with the least computational workload is selected and its real-time implementation is performed. Chapter 6 concludes the thesis and provides some recommendations for future work.

Chapter 2

Signal processing for audio beam system

The ABS uses the self-demodulation property of finite-amplitude ultrasonic waves in air to generate directional sound. There are several modulation schemes that can be employed in the ABS to generate self-demodulating ultrasonic waves. These modulation schemes can be implemented in either analog or digital platform. We have explored the possibility of the digital-analog implementation of the ABS. The digital-analog implementation has its own limitations and implementation issues which are discussed in Appendix A. In the present study, we are interested in the real-time implementation of these modulation schemes on a fixed-point DSP processor. The block diagram of the ABS with modulation schemes implemented on a fixed-point DSP processor is shown in Fig. 2.1.

Figure 2.1 ABS with fixed-point processor.

The finite wordlength effect in fixed-point processing introduces quantization errors at the output signal. The sources of these quantization errors include input quantization, coefficient quantization and product (roundoff) quantization. These quantization errors are well studied in literature [17]-[20], and they can be modeled to analyze their effect at the system's output. The effect of quantization errors on output can be quantified by calculating the SQNR. Beside quantization errors, we are also interested in the computational workload incurred by the arithmetic operations used by different modulation schemes. For real-time implementation, the computational workload incurred by modulation schemes should be significantly lesser than the computational rating of the DSP processor.

This chapter is organized as follows. Section 2.1 describes various modulation schemes used in the ABS, and presents a distortion analysis of these modulation schemes. Section 2.2 introduces fixed-point number format and arithmetic used in the fixed-point DSP processor. The quantization errors introduced by finite wordlength and fixed-point processing are also described in this section. Section 2.3 defines the SQNR as the performance metric to quantify the quantization errors introduced in the system. A performance metric is also defined to quantify the computational workload of different modulation schemes. These two performance metrics are used as evaluating criteria in subsequent chapters. Section 2.4 describes the fixed-point DSP processor used for real-time implementation. Finally, Section 2.5 summarizes and concludes this chapter.

2.1 Modulation schemes for audio beam system

ABS amplitude-modulates the ultrasonic carrier with an audio signal to generate broadband audio. ABS uses several proprietary amplitude modulation schemes for this purpose. DSBAM is the simplest modulation scheme used in ABS. The demodulated sound from DSBAM has high directivity and harmonic distortion. To reduce the harmonic distortion of DSBAM two approaches are adopted. In the first approach, a pre-distortion function is introduced to reduce the distortion. SRAM adopts this approach, and uses square root function to reduce the distortion. The second approach introduces pre-distortion terms for distortion reduction. MAM and SSBAM are the quadrature modulation schemes that introduce the pre-distortion terms through quadrature carrier. These modulation schemes are discussed briefly in the following subsections.

2.1.1 Double-sideband amplitude modulation

Yoneyama et al. [10] developed the first parametric array in 1983 using DSBAM. His experimental results showed that the demodulated secondary sound wave has high directivity, as well as high harmonic distortion. The output of the DSBAM with input signal $x[n]$ and modulation index m is given as

$$y[n] = \underbrace{(1 + mx[n])}_{E[n]} \cos[2\pi f_c n], \quad (2.1)$$

where $E[n]$ is the envelope of the DSBAM. The block diagram of DSBAM is shown in Fig. 2.2.

Figure 2.2 Block diagram of DSBAM.

The time domain and frequency spectrum plots for DSBAM are generated for a single tone input of 1 kHz and a carrier of 40 kHz. Figures 2.3(a) and (b) show the time domain and frequency domain plots, respectively. The two sidebands of DSBAM are marked in Fig. 2.3(b).

(a)

(b)

Figure 2.3 DSBAM: (a) time domain and (b) frequency spectrum plots.

2.1.2 Square root amplitude modulation

To reduce distortion in the demodulated signal, Kamakura et al. [11] proposed a pre-distortion function, which uses SQRT before modulation. This pre-processing technique is now widely known as the SRAM. The distortion in the demodulated signal can be totally removed if and only if all pre-distorted harmonics generated from SQRT are reproduced. However, this requires a very wide bandwidth ultrasonic transducer which is still a practical challenge to manufacture.

The output of the SRAM is stated as

$$y[n] = \underbrace{\left(\sqrt{1 + mx[n]} \right)}_{E[n]} \cos[2\pi f_c n]. \quad (2.2)$$

The block diagram of the SRAM is shown in Fig. 2.4.

Figure 2.4 Block diagram of SRAM.

The time and frequency domain plots for SRAM are shown in Fig. 2.5(a) and (b), respectively.

The frequency spectrum of SRAM has infinite harmonics (we have marked up to 6th harmonic in Fig. 2.5(b)), which requires infinite transducer bandwidth.

(a)

(b)

Figure 2.5 SRAM: (a) time domain and (b) frequency spectrum plots.

In order to further reduce the harmonic distortions, Kite et al. [13] proposed to integrate the modulating signal twice before taking SQRT as shown:

$$y[n] = \underbrace{\left(\sqrt{1 + m \iint x[n] dn^2} \right)}_{E[n]} \cos[2\pi f_c n]. \quad (2.3)$$

This pre-processing technique was later called “processed AM” and implemented by Pompei [12], [21]. Digital integrators [22]-[24] can be used to implement the double integration of (2.3) on DSP processors.

2.1.3 Single-sideband amplitude modulation

Nonlinear operators, like SQRT and double integration, extend the modulated signal to infinite bandwidth in frequency domain. However, ultrasonic transducer generally has limited

bandwidth. Therefore, the performance of both SRAM and processed AM are limited by the bandwidth of ultrasonic transducer. To overcome this difficulty, SSBAM was examined by many researchers [25]-[27]. The block diagram of SSBAM is shown in Fig. 2.6.

Figure 2.6 Block diagram of SSBAM.

There are three methods of SSBAM signal generation [28], [29]. These methods are described in detail in Chapter 5. Hilbert transform [28]-[31] is the most commonly used method for SSBAM signal generation. The output of SSBAM using Hilbert transform is stated as

$$\begin{aligned}
 y[n] &= \underbrace{(1 + mx[n])\cos[2\pi f_c n]}_{\text{Inphase component}} \pm \underbrace{m\hat{x}[n]\sin[2\pi f_c n]}_{\text{Quadrature phase component}} \\
 &= \underbrace{\sqrt{(1 + mx[n])^2 + (m\hat{x}[n])^2}}_{E[n]} \cos[2\pi f_c n + \phi],
 \end{aligned} \tag{2.4}$$

where $\hat{x}[n]$ is the Hilbert transform of the input signal $x[n]$, and $\phi = \tan^{-1}\left(\frac{m\hat{x}[n]}{1 + mx[n]}\right)$. In (2.4),

the addition or subtraction sign of combining the inphase component with the quadrature phase component determines the use of lower-sideband (LSB) modulation or upper-sideband (USB) modulation. The time and frequency domain plots for SSBAM are shown in Fig. 2.7(a) and (b),

respectively. SSBAM only requires finite transducer bandwidth as its spectrum consists of a carrier and one sideband (LSB is used in our case).

(a)

(b)

Figure 2.7 SSBAM with lower-sideband: (a) time domain and (b) frequency spectrum plots.

2.1.4 Modified amplitude modulation

The modified AM is a quadrature modulation scheme proposed by Liew et al. [32] in 2002. The block diagram of the modified AM scheme is shown in Fig. 2.8.

Figure 2.8 Block diagram of MAM.

The advantage of using quadrature AM (QAM) is the flexibility of introducing pre-distortion term via quadrature carrier. If the inphase term is $(1 + mx[n])\cos[2\pi f_c n]$ and quadrature term is $(\sqrt{1 - m^2 x^2[n]})\sin[2\pi f_c n]$, this quadrature modulation scheme can be viewed as a DSBAM with a quadrature carrier modulating a pre-distorted term. For this scheme,

$$y[n] = (1 + mx[n])\cos[2\pi f_c n] + (\sqrt{1 - m^2 x^2[n]})\sin[2\pi f_c n]. \quad (2.5)$$

Addition of pre-distorted term $\sqrt{1 - m^2 x^2[n]}$ into DSBAM, completely removes the distortion in it [33]. However, this is only true if and only if the ultrasonic transducer has infinite bandwidth. As this is not the case, $\sqrt{1 - m^2 x^2[n]}$ is approximated using Taylor's series expansion, which is written as

$$\sqrt{1-m^2x^2[n]} = \sum_{q=0}^{\infty} \left(\frac{(2q)!}{(1-2q)(q!)^2 4^q} m^{2q} x^{2q}[n] \right), \quad \text{for } |m^2x^2[n]| < 1. \quad (2.6)$$

Hence, (2.5) can be approximated as

$$\tilde{y}[n] = (1 + mx[n]) \cos[2\pi f_c n] + \left(\sum_{q=0}^{\infty} \frac{(2q)!}{(1-2q)(q!)^2 4^q} m^{2q} x^{2q}[n] \right) \sin[2\pi f_c n], \quad (2.7)$$

where q is the order of the QAM scheme. This special type of quadrature modulation scheme is referred as the modified AM q (MAM q), where q is the order of the expansion. MAM can approach the theoretical total harmonic distortion (THD) level of SSBAM and can adapt to the bandwidth of ultrasonic transducer [33]. Compared to other pre-processing methods, intermodulation distortion (IMD) of the MAM is reported [26], [33] to be greatly reduced. The performance of MAM q can also be tuned to suit the bandwidth of an ultrasonic transducer. Unlike SRAM whose performance degrades significantly if narrow bandwidth ultrasonic transducer is used, MAM q (with different order) offers bandwidth flexibility [33]. However, the THD and IMD values for MAM q does not decrease significantly for $q > 3$ [33]. Therefore in our study, we do not go beyond $q=3$. The following subsections show the equations and block diagrams for different orders of MAM.

A MAM1

The MAM1 scheme is derived by setting $q=1$ in (2.7). It generates frequencies up to second harmonics of the input signal $x[n]$. Since MAM1 is the lowest order of all MAM q algorithm, it requires the minimum bandwidth. However, the depth of distortion cancellation is also minimal. MAM1 is given by

$$\begin{aligned}
y[n] &= (1 + mx[n])\cos[2\pi f_c n] + \left(1 - \frac{m^2 x^2[n]}{2}\right)\sin[2\pi f_c n] \\
&= \underbrace{\sqrt{(1 + mx[n])^2 + \left(1 - \frac{m^2 x^2[n]}{2}\right)^2}}_{E[n]} \cos[2\pi f_c n + \phi],
\end{aligned} \tag{2.8}$$

where $\phi = \tan^{-1} \left(\frac{1 - \frac{m^2 x^2[n]}{2}}{1 + mx[n]} \right)$. MAM1 can be represented by the block diagram shown in Fig.

2.9.

Figure 2.9 Block diagram of MAM1.

The time and frequency plots of MAM1 are shown in Fig. 2.10(a) and (b), respectively. The frequency spectrum of MAM1 contains only the first and second harmonics of the input signal, and the second harmonic is used for distortion cancellation in the demodulated signal.

(a)

(b)

Figure 2.10 MAM1: (a) time domain and (b) frequency spectrum plots.

B MAM2

MAM2 is the second order of the MAM q algorithm. It generates frequencies up to the fourth harmonics of the input signal $x[n]$. As compared to MAM1, MAM2 requires more transducer bandwidth and is more effective in cancelling distortion. MAM2 is given by

$$y[n] = (1 + mx[n])\cos[2\pi f_c n] + \left(1 - \frac{m^2 x^2[n]}{2} - \frac{m^4 x^4[n]}{8}\right)\sin[2\pi f_c n] \quad (2.9)$$

$$= \underbrace{\sqrt{(1 + mx[n])^2 + \left(1 - \frac{m^2 x^2[n]}{2} - \frac{m^4 x^4[n]}{8}\right)^2}}_{E[n]} \cos[2\pi f_c n + \phi],$$

where $\phi = \tan^{-1} \left(\frac{1 - \frac{m^2 x^2[n]}{2} - \frac{m^4 x^4[n]}{8}}{1 + mx[n]} \right)$. The block diagram of MAM2 is shown in Fig. 2.11.

Figure 2.11 Block diagram of MAM2.

The time and frequency plots of MAM2 are shown in Fig. 2.12(a) and (b), respectively. MAM2 generates first, second and fourth harmonics as shown in Fig. 2.12(b). Due to additional harmonics in the modulated signal, MAM2 needs larger transducer bandwidth as compared to MAM1.

(a)

(b)

Figure 2.12 MAM2: (a) time domain and (b) frequency spectrum plots.

C MAM3

MAM3 is the third order of the MAM q family. It generates frequencies up to the sixth harmonics of $x[n]$. MAM3 requires more transducer bandwidth than MAM2, and is more effective in cancelling distortion. MAM3 is given by

$$y[n] = (1 + mx[n])\cos[2\pi f_c n] + \left(1 - \frac{m^2 x^2[n]}{2} - \frac{m^4 x^4[n]}{8} - \frac{m^6 x^6[n]}{16}\right)\sin[2\pi f_c n]$$

$$= \underbrace{\sqrt{(1 + mx[n])^2 + \left(1 - \frac{m^2 x^2[n]}{2} - \frac{m^4 x^4[n]}{8} - \frac{m^6 x^6[n]}{16}\right)^2}}_{E[n]} \cos[2\pi f_c n + \phi], \quad (2.10)$$

where $\phi = \tan^{-1} \left(\frac{1 - \frac{m^2 x^2[n]}{2} - \frac{m^4 x^4[n]}{8} - \frac{m^6 x^6[n]}{16}}{1 + mx[n]} \right)$. The block diagram of MAM3 is shown in Fig.

2.13.

Figure 2.13 Block diagram of MAM3.

The time and frequency plots of MAM3 are shown in Fig. 2.14(a) and (b), respectively. MAM3 generates first, second, fourth and sixth harmonics as shown in Fig. 2.14(b). These additional harmonic results in more distortion cancellation but also requires larger transducer bandwidth as compared to MAM1 and MAM2.

(a)

(b)

Figure 2.14 MAM3: (a) time domain and (b) frequency spectrum plots.

The modulation schemes described in this section are used in the ABS for generating directional sound. The self-demodulated audible signal from different modulation schemes contains different amounts of distortion. A distortion analysis of the modulation schemes used in ABS is presented in the next section.

2.1.5 Distortion analysis of modulation schemes

In previous section, we discussed several proprietary modulation schemes used in the ABS. The modulated signal generated by these modulation schemes is fed to an ultrasonic transducer, as shown in Fig. 2.15. The ultrasonic transducer has finite bandwidth and is therefore, modeled as a band-pass filter. The 3 dB bandwidth of the ultrasonic transducer is denoted by f_{trans} . This transducer projects the modulated ultrasonic waves in air, which self-demodulates and generates audible sound at a targeted region in the far field. The demodulated signal can also be obtained by modeling the self-demodulation process in air. Bertkay's far field solution [34] and the Khokhlov-Zabolotskaya-Kuznetsov (KZK) solution [35] are some of the nonlinear acoustic models, commonly used to model the self-demodulation process in air.

Figure 2.15 Transducer and air modeling for ABS.

The KZK equation [35] can accurately describe the self-demodulation process and can be used to perform accurate analysis of the distortion present in the demodulated signal. The distortion analysis can be simplified by employing Bertkay's far field solution [34], which adequately reveals the effects of the limited bandwidth of the ultrasonic transducer. Bertkay stated that the demodulated wave pressure at a distance from the ultrasonic source is proportional to the second time derivative of the squared envelope of the primary wave. Using Bertkay's solution, the demodulated wave pressure P_2 at distance r from ultrasonic source is given by

$$P_2 = \frac{\beta P_0^2 a^2 m}{8 \rho_0 c_0^4 \alpha r} \frac{\partial^2}{\partial t^2} E^2(t), \quad (2.11)$$

where β is the coefficient of nonlinearity for air, P_0 is pressure amplitude at the ultrasonic source, a is the radius of the parametric loudspeaker, ρ_0 is ambient density, and c_0 is the small signal sound speed, and α is the absorption coefficient in air. The demodulated wave consists of the desired audible sound and the distortion components. THD and IMD can be used to measure the distortion in the demodulated signal. For a single tone input, THD measures the amount of distortion in the demodulated ultrasonic signal. The definition of THD is given by

$$THD = \sqrt{\frac{T_2^2 + T_3^2 + \dots + T_{n-1}^2 + T_n^2}{T_1^2 + T_2^2 + T_3^2 + \dots + T_{n-1}^2 + T_n^2}} \times 100\%, \quad (2.12)$$

where T_1 and T_i are the amplitudes of the fundamental frequency (f_1) component and the higher harmonics at if_1 (for $i=2, 3, \dots, n$), respectively. For a dual tone input (f_1 and f_2), distortion is due to a series of inter-modulation harmonics given by $if_1 \pm jf_2$, where i and j are positive integers larger than zero. Distortion due to inter-modulation harmonics is known as the IMD and is given by

$$IMD = \frac{\sqrt{\sum_{i=1}^{\infty} \sum_{j=1}^{\infty} D_{if_1 \pm jf_2}^2}}{\sqrt{D_1^2 + D_2^2 + \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} D_{if_1 \pm jf_2}^2}} \times 100\%, \quad (2.13)$$

where D_1 , D_2 and $D_{if_1 \pm jf_2}$ are the amplitudes of f_1 , f_2 and inter-modulation harmonics at $if_1 \pm jf_2$, respectively. The amount of distortion in the demodulated signal from different modulation schemes is shown in [26], [33]. In [33], the distortion analysis of each modulation schemes is performed by using Betkay's solution to model the self-demodulation process. We use the model of [33] to calculate the THD and IMD values for different modulation schemes. The modulation index $m=0.70$ is used for THD and IMD calculations for different modulation schemes, and is shown in Fig. 2.16.

(a)

Figure 2.16 (a) THD and (b) IMD plots for modulation schemes.

For THD (%) calculation, a single tone input of 1 kHz is used. Fig 2.16(a) shows the THD values for different modulation schemes over a range of f_{trans} . DSBAM has the highest THD and remains almost constant for all values of f_{trans} . SSBAM has the lowest THD value of 0 for all values of f_{trans} . The THD values for SRAM and MAM q are non zero and decrease monotonically as f_{trans} increases. The IMD (%) computation is derived from multi-tone input. Figure 2.16(b) shows IMD values for different modulation schemes, for the input signal having two tones at 700 Hz and 1 kHz. Similar to THD, DSBAM has the highest IMD which remains almost constant for different values of f_{trans} . The IMD (%) values for SRAM, SSBAM and MAM q decrease monotonically as f_{trans} increases. It can be observed that except DSBAM, the THD and IMD performance of other modulation schemes improve as transducer bandwidth increases. As compared to other modulation schemes, the THD and IMD values of MAM2 and

MAM3 decrease at a faster rate and attain lesser values when more bandwidth ($f_{trans} \geq 3$ kHz) is available. For all practical purposes, f_{trans} depends on the ultrasonic transducer used, and distortion analysis can be used to select the modulation scheme with minimum THD and IMD for the given ultrasonic transducer. It is possible that for the given ultrasonic transducer, two or more modulation schemes have comparable THD and IMD performance but significantly different computational workload requirements. In such case, the modulation scheme with the least computational workload should be selected.

Apart from the modulation schemes discussed in this section, ABS uses algorithms to enhance the listening experience by amplifying or attenuating different sections of the modulating signal automatically. This is achieved through an automatic gain control algorithm which is described in Appendix B. As specified earlier, all signal processing blocks are implemented on a fixed-point DSP processor. The fixed-point DSP processor uses fixed-point number format and arithmetic operations, which are described in the next section.

2.2 Fixed-point signal processing

Fixed-point processor's data format determines its ability to handle signals of different precisions, dynamic ranges, and SQNRs. The fixed-point data is processed with fixed-point arithmetic operations. In practical DSP systems, we have to use digital hardware with finite wordlength to represent and process digital signals, which introduce errors in the system. This is commonly known as the finite wordlength effect. Two of the major sources of errors in practical implementation are due to the quantization of input signals and the roundoff of the products during the execution of the algorithms. To analyze the deviation of practical system response

from ideal system response, these error sources can be modeled as sources of additive white noise. In this section, we review the fixed-point data representation and arithmetic followed by error sources introduced as a consequence of finite wordlength processing. We shall start with the fixed-point number format and arithmetic.

2.2.1 Fixed-point number format and arithmetic

In fixed-point processors, a number is represented with a series of binary digits (1s and 0s). Binary numbers can be represented as sign magnitude, one's complement or two's complement. In most fixed-point systems, signed numbers are represented using two's complement format. In binary representation, different weights are associated with each bit. Consider a binary number with N bits. In $Qm.n$ fixed-point number representation, m bits are used to represent integer portion of the number, which also includes the sign bit (the most significant bit). n bits are used to represent the fractional portion. Therefore, the total number of bits,

$$N = m + n. \quad (2.14)$$

The binary values of N -bit in the $Qm.n$ representation can be computed using the equation

$$\begin{aligned} x &= (-b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + b_{N-3}2^{N-3} + \dots + b_0)2^{-n} \\ &= -b_{N-1}2^{m-1} + \sum_{k=0}^{N-2} b_k 2^{k-n}. \end{aligned} \quad (2.15)$$

For example, different $Qm.n$ representations for $N=16$ are shown in Fig. 2.17. Note that there is an imaginary binary-point that determines the number of bits, m for the sign integer number. The remaining n bits are the fractional number.

Figure 2.17 Different $Qm.n$ fixed-point formats.

The dynamic range is the range of numbers that can be represented in the available wordlength. Dynamic range (in dB) for a fixed-point number with wordlength N can be calculated using the equation

$$\text{Dynamic Range (dB)} = 20 \times \log_{10} \left(\frac{\text{Max.}}{\text{Min.}} \right). \quad (2.16)$$

Similarly, precision is defined as the smallest step (or difference) between two consecutive numbers that can be obtained for a given number of bits. Table 2.1 shows a summary of dynamic range and precision for a 16-bit number using different formats.

Table 2.1 Dynamic range and precision for a 16-bit number using different Q formats.

Format	Largest +ive value (Max.)	Least -ive value	Precision (Min.)	Dynamic Range (dB)
Q1.15	0.999969482421875	-1	0.00003051757813	90
Q2.14	1.99993896484375	-2	0.00006103515625	90
Q3.13	3.9998779296875	-4	0.0001220703125	90
Q4.12	7.999755859375	-8	0.000244140625	90
Q5.11	15.99951171875	-16	0.00048828125	90
Q6.10	31.9990234375	-32	0.0009765625	90
Q7.9	63.998046875	-64	0.001953125	90
Q8.8	127.99609375	-128	0.00390625	90
Q9.7	255.9921875	-256	0.0078125	90
Q10.6	511.984375	-512	0.015625	90
Q11.5	1,023.96875	-1,024	0.03125	90
Q12.4	2,047.9375	-2,048	0.0625	90
Q13.3	4,095.875	-4,096	0.125	90
Q14.2	8,191.5	-8,192	0.25	90
Q15.1	16,383.5	-16,384	0.5	90
Q16.0	32,767	-32,768	1	90

Although different Q formats have same dynamic range (in dB), there is a tradeoff between the range and the precision of the numbers when using different Q formats. Q1.15 has the best precision (2^{-15}), but the narrowest range from -1 to $(1 - 2^{-15})$. The integer Q16.0 has the worst precision (1), but with the widest range from $-32,768$ to $+32,767$.

Addition and multiplication are the basic arithmetic operations of a fixed-point processor. Therefore, many fixed-point processors have a built-in arithmetic logic unit that can perform addition/subtraction, and hardware multiplier that can perform fast N -bit multiplication. Addition and multiplication are performed on binary numbers represented using two's complement format [36], [37]. The addition of two fixed-point numbers in different Q formats requires radix point alignment. For this, the binary numbers to be added should have identical format. When two

numbers are added, result may overflow. Generally, an extra bit is provided to prevent this overflow. The radix alignment is not required for binary multiplication. Suppose two binary numbers A and B in Q format $Q_{m_1.n_1}$ and $Q_{m_2.n_2}$ with N_1 and N_2 bits, respectively, are multiplied together. The resultant, C requires (N_1+N_2-1) bits to accommodate the result without any loss of precision. The most significant bit (MSB) consists of sign bit. Integer and fractional parts of C requires m_1+m_2 and n_1+n_2 bits, respectively. This shows that when two binary numbers are multiplied, the storage requirement of the resultant is much larger than those of the operands. In order to prevent continuous growth of bit length in consecutive multiplications, bit truncation (or rounding) of C to smaller bits is always required.

Fixed-point processors generally do not have a hardware divisor. Instead, division can be realized by repeated subtraction. For an N -bit fractional number, fractional division can be realized by repeating the conditional subtraction instruction $(N-1)$ times. In the case of integer division, conditional subtraction must be conducted N times. Alternatively, fixed-point division can be replaced with right shift if the divisor is a power-of-two.

2.2.2 Finite wordlength effects

Finite wordlength effects occur when the wordlength of the memory is less than the required precision needed to store the actual values. These effects introduce noise into finite wordlength digital systems and create non-ideal system responses. Real world signals, such as audio from an analog microphone are continuous in time and amplitude. An analog-to-digital converter (ADC) produces a digital signal by sampling and quantizing analog signals. In our study, we have

considered only linear quantizers. For an N -bit ADC with full-scale voltage of V_{FS} (peak to peak), the physical quantization interval (step) is defined as

$$\Delta = \frac{V_{FS}}{2^N}. \quad (2.17)$$

Difference between the analog-valued discrete-time signal and the corresponding discrete-valued digital signal results in quantization error. Digital systems are designed based on assumption that parameters have infinite precision. But any digital system requires finite number of bits for data storage and processing. So, the DSP system's parameters need to be quantized before they can be implemented. This is called coefficient quantization, which is another source of error. Similarly, the multiplication operation also introduces errors in the system. When two numbers (of equal wordlengths) are multiplied, the wordlength required to accommodate the result is doubled. Multiplication of result with another number further increases the wordlength requirement. In order to prevent the growth of wordlength for subsequent multiplications, rounding (or truncation) must be carried out. This rounding (or truncation) introduces roundoff errors in the system.

These quantization and roundoff errors have been widely studied [38]-[40]. It is found that if certain conditions are fulfilled, the quantization and roundoff operations can be modeled as processes with additive white noise in the system [17], [18]. For signal quantization, if the dynamic range of input signal is sufficiently greater than quantum step size Δ and signal bandwidth is large enough, quantization of a signal can be modeled as addition of white noise to the signal [17]-[19]. Quantization modeled as a process of adding uniformly distributed noise $e[n]$ to the system is shown in Fig. 2.18.

Figure 2.18 Quantization noise model.

The noise added is white, uniform and uncorrelated with the signal [17]. Similarly, roundoff errors can be modeled as addition of white noise if signal dynamic range is sufficiently greater than the quantum step size, signal bandwidth is large enough, and the number of discarded bits is sufficiently high [20]. Roundoff operation modeled as the addition of white noise $e[n]$ is shown in Fig. 2.19.

Figure 2.19 Roundoff operation modeled as the addition of white noise.

Errors generated from different quantization and roundoff sources contribute to noise in the system. SQNR is a measure used to quantify how much a signal has been corrupted by quantization noise. SQNR is used as one of the key performance metrics to evaluate the digital noise level introduced by the fixed-point algorithm. The performance metrics used in our study are described in the next section.

2.3 Performance metrics

For a given modulation scheme, there can be more than one algorithm to generate the desired output required by the ABS. For fixed-point algorithms, algorithm accuracy and execution speed are the two main evaluation criteria. Ideally, the algorithm with the highest bit accuracy and lowest execution speed should be chosen. The performance metrics used to quantify the bit accuracy and execution speed of the algorithm are described in the next section.

2.3.1 Signal-to-quantization-noise ratio

In digital domain, SQNR is the most commonly used criterion for evaluating the system accuracy [41]-[43]. SQNR is computed by simulating modulation schemes on a fixed-point DSP processor. Implementing modulation schemes on an N -bit fixed-point processor generates quantization noise along with modulated signal at its output, as shown in Fig. 2.20.

Figure 2.20 Modulation schemes implemented on finite precision fixed-point system generates quantization errors.

Suppose, $x[n]$ is the unquantized discrete-time digital input signal, and $y_N[n]$ is the digital output signal. $y_N[n]$ is a combination of desired output $y[n]$ and error signal $e[n]$. The desired signal power at output is defined as

$$P_y = \frac{\sum_{i=1}^L y^2[i]}{L}, \quad (2.18)$$

where L is the length of the signal. Quantization noise power at output is defined as

$$P_e = \frac{\sum_{i=1}^L e^2[i]}{L}. \quad (2.19)$$

SQNR is defined as the ratio of signal power to quantization noise power, and is calculated as

$$SQNR = 10 \times \log_{10} \left(\frac{P_y}{P_e} \right). \quad (2.20)$$

For SQNR analysis, the modulation schemes are simulated in MATLAB. In MATLAB simulation, a double-precision (DP) floating-point (64-bit) format is used to represent the input signal, $x[n]$, and performs arithmetic operations to derive the output signal $y[n]$. These variables are treated as the golden reference when comparing the fixed-point representation and arithmetic. The MATLAB fixed-point toolbox [44] is used to simulate the computational operations in an N -bit fixed-point processor. Modulation schemes simulated in MATLAB using N -bit fixed-point

arithmetic takes $x[n]$ as input and generates $y_N[n]$ as output. Subtracting $y_N[n]$ from $y[n]$ (golden reference output) results in a quantization error signal of $e[n]$ as shown in Fig. 2.21.

Figure 2.21 Generating error values for N -bit fixed-point processor simulated on MATLAB.

To calculate SQNR, P_y is calculated from $y[n]$ using (2.18), P_e is calculated from $e[n]$ using (2.19), and SQNR is calculated from P_y and P_e using (2.20). In subsequent chapters, the SQNR values for modulation schemes are calculated in this way. To choose among the various algorithms to implement a given modulation scheme, the algorithm with the largest SQNR value is preferred. But the final decision also takes into account of the computational workload of different algorithms.

2.3.2 Computational workload

A DSP operation can be performed by a sequence of additions and multiplications. Filtering is the most commonly used DSP operation, which can be performed by multiplication and accumulation. Similarly, mathematical operations such as division, SQRT, tangent, etc. are signal processing algorithms that can be computed by a specified sequence of additions and multiplications. This sequence accounts for the computational workload of algorithm. For each

modulation scheme, our objective is to maximize the system SQNR and at the same time, minimizes its computational workload.

The simplest way to quantify the computational workload is to count the number of additions and multiplications per output sample required by the DSP operation. For a filtering block, the number of additions per output sample (APOS) is equal to the number of multiplications per output sample (MPOS). Therefore, if the DSP algorithm is a sequence of filtering blocks, then MPOS alone is sufficient to indicate its computational workload. Multi-rate processing is widely used to decrease the computational workload of a signal processing algorithm. For multi-rate implementation [45], the number of input samples and output samples are different. For example, consider the multi-rate implementation of a generic DSP algorithm, named here (ALGO). Suppose ALGO consists of two sections: ALGO1 and ALGO2. The first section ALGO1 operates at an input sampling frequency of f_{IN} , and the second section ALGO2 operates at an output sampling frequency of f_{OUT} , which is L times the input sampling frequency. In multi-rate implementation, L is called the upsampling factor. The block diagram of the multi-rate implementation of ALGO is shown in Fig. 2.22.

Figure 2.22 Multi-rate implementation of DSP algorithm.

The upsampling operation is always followed by an interpolation filter [36]-[38], [45]. In Fig. 2.22, the interpolation filter is included in ALGO2. If ALGO1 requires M_1 multiplications and A_1 additions, and ALGO2 requires M_2 multiplications and A_2 additions for processing one sample, then the MPOS and APOS for ALGO are calculated as

$$MPOS = \frac{M_1}{L} + M_2, \quad (2.21)$$

$$APOS = \frac{A_1}{L} + A_2. \quad (2.22)$$

The MPOS calculation is based on sample-by-sample processing, but for DSP implementation of algorithms, block processing is sometime preferred over sample-by-sample processing [36], [37]. In block processing, B samples of input signal are collected over a period of time. This set of input samples is referred to as a frame (or block) of length B . The whole block is processed to produce a set of output samples. The multi-rate implementation of ALGO with block processing is shown in Fig. 2.23.

Figure 2.23 Multi-rate implementation of DSP algorithm using block processing.

Apart from additions and multiplication, a DSP algorithm can have ‘if-then-else’ or ‘switch’ statements which are not reflected by MPOS value. But these conditional statements can significantly affect the computational workload performance in real-time. Similarly, execution time of code can be reduced by taking advantage of DSP architecture. For real-time implementation, MIPS count takes into account of these factors. MIPS of an algorithm is defined as the number of millions of CPU clock cycles per second required by the algorithm for its execution. The MIPS rating of a processor is related to the clock speed that the processor is running. Therefore, the MIPS rating of a processor must be larger than the MIPS required to run the algorithm in real-time. For sample processing, the number of CPU clock cycles required to process one sample is recorded whereas for block processing the number CPU clock cycles required to process B samples are recorded. For single-rate processing, MIPS (block processing) is calculated as

$$MIPS_{Single-rate} = \frac{CC f_{IN}}{B}, \quad (2.23)$$

where CC is the number of CPU clock cycles taken to process B samples, and f_{IN} is the sampling frequency. For multi-rate processing, total MIPS of the algorithm is the summation of MIPS consumed during base-band processing ($MIPS_1$), and MIPS consumed during processing at higher sampling rates ($MIPS_2$), i.e.,

$$MIPS_{Multi-rate} = MIPS_1 + MIPS_2. \quad (2.24)$$

If the real-time DSP implementation of ALGO1 takes CC_1 CPU cycles for processing B samples, and ALOG2 requires CC_2 CPU cycles for processing LB samples, then $MIPS_{TOTAL}$ for ALGO is calculated as

$$MIPS_{Multi-rate} = \underbrace{\frac{CC_1 f_{IN}}{B}}_{MIPS_1} + \underbrace{\frac{CC_2 f_{OUT}}{BL}}_{MIPS_2}. \quad (2.25)$$

2.4 Real-time implementation platform

The TMS320C55x [46] is a 16-bit fixed-point power-efficient DSP processor from Texas Instrument (TI). In the present study, TMS320C55x is used for the real-time implementation. The C55x processor's low power consumption and high performance support feature-rich miniaturized personal and portable applications, such as cell phones, programmable digital-audio devices, digital cameras, and digital hearing aids. The C55x architecture achieves high performance through parallelism by providing additional hardware, more built-in instructions; user programmed parallel functions, and additional instructions that take advantage of the expanded hardware to accomplish more operations in fewer clock cycles. Proper instructions are scheduled to run automatically in parallel by C55x assembly optimizer and C compiler. The processor provides advanced automatic power management for all peripherals, memory arrays, and individual CPUs. Figure 2.24 shows the functional block diagram of the C55x DSP processor [36], [37].

Figure 2.24 TMS320C5505 functional-block diagram [47].

A number of development boards with TMS320C55x processor on it are available in the market. In our study, we use the TMSC5505 eZDSP USB stick development board [47], [48] for the real-time implementation of modulation schemes. The processor has 120 MIPS performance and up to 320 kBytes onchip memory. Figure 2.25 shows the TMSC5505 eZDSP USB stick development board.

Figure 2.25 TMS5505 eZDSP USB stick development board [101].

The TMS5505 eZDSP USB stick development board comes with the development tool namely code composer studio (CCS) version 4 (v4) [49]. CCS is an integrated development environment (IDE) that incorporates the C compiler, assembler, linker, simulator, debugger etc., with additional features, such as a graphical display for developing DSP software. It provides an editor for programming and a project manager for building application programs. For software testing purposes, it provides steps, breakpoints, watch windows, graphic capabilities for data monitoring, and probe points for data-file I/O. The breakpoint feature of the CCS is used as a means to record the CPU cycle count required to execute a DSP routine. The CPU cycle count along with the block size (used for real-time implementation) and sampling frequency are substituted into (2.23) or (2.25) to estimate the MIPS of the algorithm. Appendix D shows the screenshot of CCS v4 with breakpoints inserted in a code segment, and the CPU cycle count for the code execution.

2.5 Conclusion

In this chapter, different modulation schemes used in ABS were introduced. Advantages and disadvantages of each modulation scheme were briefly discussed. The fixed-point number format and arithmetic used by fixed-point processors were also described. A finite wordlength operation used by a DSP processor introduces quantization errors and incurs computational workload. To quantify the amount of quantization error present and computational workload incurred, performance metrics were defined. These performance metrics will be used to select the best algorithm for each modulation scheme in the following chapters. The last section of this chapter described the fixed-point DSP processor used in our research work.

Besides examining the SQNR and computational workload of different modulation schemes, we are also interested to explore implementation tricks and efficient techniques in realizing the algorithm on a real-time processor. Therefore, in the following chapters, some of these novel works will also be explained in detail.

Chapter 3

Double-sideband and modified amplitude modulation

The ABS pre-processes the input signal before modulating it with ultrasonic carriers. There are many modulation schemes mentioned in Chapter 2 that can be used. In this chapter, we discuss two such modulation schemes namely DSBAM and MAM q . DSBAM is the simplest modulation scheme and is discussed first. MAM q uses a QAM technique, and is described after DSBAM. These modulation schemes are implemented on a fixed-point processor. For digital implementation, SQNR and computational workload are two important performance metrics. Higher SQNR and lower computational workload are always preferred for the real-time fixed-point implementation. In this chapter, we describe different methods to increase the SQNR and decrease the computational workload of these modulation schemes.

This chapter is organized as follows. Section 3.1 gives a brief description of the DSBAM and its SQNR performance with different Q formats. Section 3.2 describes the MAM q modulation scheme for $q=1, 2$ and 3 . The SQNR performance of MAM q is investigated in Section 3.3. In addition, we also select the optimal Q format for fixed-point MAM q implementation. In Section 3.4, we propose a new type of MAM algorithm, known as the adaptive MAM. Adaptive MAM reduces processing cycles (or MIPS) of MAM q and it is very suited for fixed-point processing. Real-time implementations of DSBAM, MAM q and adaptive MAM are carried out in Section 3.5. Finally, Section 3.6 summarizes and concludes this chapter.

3.1 Double-sideband amplitude modulation

DSBAM is the simplest pre-processing method, and is introduced in Section 2.1.1. The block diagram and mathematical expression for DSBAM is given by Fig. 2.2 and (2.1), respectively. Equation (2.1) can be easily implemented in floating-point format, which supports number representation over a wide range. But for fixed-point implementation, the range of representable numbers depends on arithmetic format used. For example, consider an N -bit fixed-point processor using $Q1.N-1$ arithmetic format. $Q1.N-1$ arithmetic format can represent numbers lying in the range $[-1, 1-2^{-(N-1)}]$. For $-1 \leq x[n] < 1-2^{-(N-1)}$, (2.1) generates $0 \leq y[n] < 2$. In order to prevent overflow or saturation at any stage of fixed-point DSBAM, (2.1) is scaled by a factor of 0.5, as shown

$$y[n] = \left(\frac{1}{2} + \frac{mx[n]}{2} \right) \cos[2\pi f_c n]. \quad (3.1)$$

This scaling factor of 0.5 can also be merged with modulation index m . The scaling attenuates $y[n]$, and it is compensated by applying an additional gain of 6 dB through analog amplifier. This leads to higher power consumption by analog amplifiers for single Q format implementation. The block diagram of DSBAM using $Q1.N-1$ format is shown in Fig. 3.1.

Figure 3.1 Single Q format implementation of fixed-point DSBAM.

The quantization errors, $e_A[n]$ and $e_B[n]$, which are introduced by different fixed-point arithmetic operations, are shown in Fig. 3.1. The output contains the desired signal $y[n]$ and quantization noise ($e[n]=e_A[n]+e_B[n]$). Similarly, $Q2.N-2$ arithmetic format supports the number range $[-2, 2-2^{-(N-2)}]$. This increased range is obtained at the expense of precision, which reduces from $2^{-(N-1)}$ for $Q2.N-1$ to $2^{-(N-2)}$ for $Q2.N-2$. If $Q2.N-2$ arithmetic format is used for input, input scaling is not required and (2.1) can be used directly. There is no need to use the $Q3.N-3$ format and beyond, as $Q2.N-2$ can provide the dynamic range required by (2.1). Figure 3.1 uses $Q1.N-1$ format for arithmetic operations at every step and is referred to as a single Q format implementation. A mixed Q format, which uses a mix of $Q1.N-1$ and $Q2.N-2$ formats, is also possible. Q format is selected (from $Q1.N-1$ and $Q2.N-2$) for each step depending on its required dynamic range. The block diagram of DSBAM using mixed Q format is shown in Fig. 3.2.

Figure 3.2 Mixed Q format implementation of fixed-point DSBAM.

Next, we examine the SQNR performance using single Q and mixed Q formats. SQNR is described in Section 2.3.1 and is given by (2.20). In single Q format implementation, multiplication by scaling factor of 0.5 reduces P_y , thereby reducing the SQNR. For mixed Q format implementation, P_y remains unchanged but P_e increases due to reduced precision at

various steps. SQNR of single Q and mixed Q format are denoted as $SQNR_{DSBAM, SingleQ}$ and $SQNR_{DSBAM, MixedQ}$, respectively. SQNR difference between single Q and mixed Q format is calculated as

$$SQNR_{DSBAM, Diff} = SQNR_{DSBAM, MixedQ} - SQNR_{DSBAM, SingleQ}. \quad (3.2)$$

The $SQNR_{AM, Diff}$ on 8-bit, 16-bit and 32-bit fixed-point processor for different values of m is zero. There is no change in SQNR whether single Q or mixed Q format is used for implementation. As mixed Q format implementation results in lesser power consumption by analog amplifiers than the single Q format implementation, it is recommended for DSBAM implementation. Next, we examine another type of amplitude modulation scheme, known as MAM.

3.2 Modified amplitude modulation

MAM was previously introduced in Section 2.1.4. In this section, we examine its detailed fixed-point implementation for different q^{th} order of MAM. The general q^{th} order MAM q is expressed as (2.7). There are two modulation paths in MAM q . The inphase modulation path is similar to DSBAM, and the quadrature modulation path consists of an infinite series of pre-distorted terms that can be used to remove distortion generated from the formation of parametric array effect in air. Instead of an infinite series, a finite order is used. It is found through simulations that for $q > 3$, THD and IMD performance do not improve significantly [33]. We therefore restrict ourselves up to $q=3$. Figure 3.3 shows a block diagram of MAM3, which also include MAM2 and MAM1, where certain paths are removed.

Figure 3.3 Block diagram of MAM1, MAM2 and MAM3.

If quadrature path for MAM3 (marked with **red lines**) is removed, the block diagram of Fig. 3.3 becomes MAM2 shown in Fig. 2.11. Similarly, if quadrature paths for MAM2 (marked with **purple lines**) and MAM3 (marked with **red lines**) are both removed, Fig. 3.3 becomes the block diagram of MAM1 shown in Fig. 2.9. Figure 3.3 is appropriate for floating-point implementation, but for fixed-point implementation, a scaling factor needs to be introduced. The scaling factor depends on the Q format used for the implementation. For single Q format, a scaling factor of 0.5 is used, and the inphase and quadrature paths are implemented in Q1.N–1 format. MAM3 implemented using single Q format is represented as follows.

$$y[n] = \left(\frac{1}{2} + \frac{mx[n]}{2} \right) \cos[2\pi f_c n] + \left(\frac{1}{2} - \frac{m^2 x^2[n]}{4} - \frac{m^4 x^4[n]}{16} - \frac{m^6 x^6[n]}{32} \right) \sin[2\pi f_c n]. \quad (3.3)$$

The scaling factor of 0.5 avoids saturation/overflow at every step of inphase and quadrature paths. As mentioned previously, the scaling attenuates $y[n]$ and it is compensated by applying an additional gain of 6 dB through analog amplifier. This leads to higher power consumption by analog amplifiers for single Q format implementation of MAM. If the modulation index m for (3.3) is less than 0.82, then $y[n]$ does not saturate/overflow when inphase and quadrature components are added. Moreover, $m > 0.80$ is seldom used for ABS as it increases THD [33]. Therefore, in the present discussion, we assume that $m \leq 0.80$. The block diagram of MAM3 implemented in single Q format is shown in Fig. 3.4.

Figure 3.4 Single Q format implementation of fixed-point MAM1, MAM2, and MAM3.

For mixed Q format implementation of MAM3, scaling by 0.5 is not required and (2.10) holds for fixed-point implementation also. Block diagram of MAM3 implemented in mixed Q format is shown in Fig. 3.5.

Figure 3.5 Mixed Q format implementation of fixed-point MAM1, MAM2 and MAM3.

For mixed Q format implementation of MAM3, the inphase path is implemented in mixed Q format, the quadrature path is implemented in $Q1.N-1$ format, and the output is in $Q2.N-2$ format. To choose the appropriate Q format, SQNR is calculated for MAM implemented in single Q format and mixed Q format. In the next section, we shall investigate the SQNR

performance of single Q and mixed Q format, and select the optimal Q format for MAM1, MAM2 and MAM3 algorithms.

3.3 Q format selection for MAM implementation

Single Q and mixed Q format implementations for MAM are shown in Fig. 3.4 and 3.5, respectively. It is shown in Section 3.1 that SQNRs for single Q and mixed Q formats DSBAM implementation are the same. We will examine whether there is any difference when implementing quadrature path in MAM. The SQNR of MAM_q implemented in single Q and mixed Q formats are denoted as $SQNR_{MAMq, SingleQ}$ and $SQNR_{MAMq, MixedQ}$, respectively. SQNR difference between single Q and mixed Q format is calculated as

$$SQNR_{MAMq, Diff} = SQNR_{MAMq, MixedQ} - SQNR_{MAMq, SingleQ}. \quad (3.4)$$

A positive $SQNR_{MAMq, Diff}$ value indicates that mixed Q format MAM implementation improves SQNR, and vice-versa. $SQNR_{MAMq, Diff}$ for 8-bit, 16-bit and 32-bit fixed-point processors for different modulating indices m are plotted in Fig. 3.6(a), (b), (c) for MAM1, MAM2 and MAM3, respectively.

(a)

(b)

(c)

Figure 3.6 $SQNR_{MAMq, Diff}$ for (a) MAM1, (b) MAM2 and (c) MAM3.

It can be observed from Fig. 3.6(a), (b) and (c), that for 8-bit fixed-point processor, smaller m values ($m < 0.3$) leads to negative $SQNR_{MAMq, Diff}$ value for MAM q . The $SQNR_{MAMq, Diff}$ values become positive as m values increase ($m \geq 0.4$). For 16-bit and 32-bit fixed-point processors, $SQNR_{MAMq, Diff}$ values are mainly positive for all values of m . For a particular fixed-point processor wordlength, the $SQNR_{MAMq, Diff}$ that corresponds to different m values can be averaged as

$$AVG_SQNR_{MAMq, Diff} = \frac{\sum_{i=1}^L SQNR_{MAMq, Diff}[i]}{L}. \quad (3.5)$$

In practice, $(m < 0.5)$ is seldom used. Instead, $AVG_SQNR_{MAMq, Diff}$ values for MAM q implemented in 8-bit, 16-bit and 32-bit fixed-point processor are averaged over $0.50 \leq m < 0.80$ (in step of 0.05), as shown in Table 3.1.

Table 3.1 $AVG_SQNR_{MAMq, Diff}$ for MAM1, MAM2 and MAM3 (for $0.50 \leq m \leq 0.80$ in step of 0.05).

MAM q	$AVG_SQNR_{MAMq, Diff}$ for N -bit fixed-point processor (dB)		
	8-bit	16-bit	32-bit
MAM1	0.36	0.31	0.46
MAM2	0.37	0.31	0.54
MAM3	0.37	0.34	0.52

Table 3.1 shows that a marginal increment of around 0.50 dB is achieved by mixed Q format implementation over single Q format implementation. The number of additions and multiplications required by single Q and mixed Q format implementations is similar, but mixed Q format implementation requires great care from programmer to interpret its results. It is the programmer's responsibility to take care of binary-point position, while performing arithmetic operations and generating output. The binary-point adjustment can be achieved by binary shifting. As mixed Q format implementation provides higher SQNR and lesser power consumption by analog amplifiers than the single Q format implementation, it is recommended for MAM q implementation. Next, we discuss an adaptive MAM algorithm to reduce the computational workload of MAM q in real-time.

3.4 Adaptive MAM

Adaptive MAM algorithm refers to a fixed-point MAM q implementation whose order depends on the input sample amplitude. For an N -bit fixed-point representation, any number below the significant bit level (or the smallest quantization step) is represented as zero. This is known as underflow. For MAM q , the range of input values whose outputs underflow can be determined offline. An adaptive MAM algorithm, which varies its order depending on input sample magnitude, can be used to reduce computational workload. We will show the conditions in which MAM q algorithm underflow for $q=3, 2$ and 1 .

Equation (2.10) shows the MAM3 algorithm output. For mixed Q format implementation of (2.10) on an N -bit fixed-point processor, $x^6[n]$ underflows if

$$x[n] < \frac{2^{-(N-5)/6}}{m}, \quad (3.6)$$

and MAM3 becomes MAM2. If both $x^6[n]$ and $x^4[n]$ underflow, MAM3 reduces to MAM1. The condition for underflow of $x^4[n]$ is given in Table 3.2. Similarly, if $x^6[n]$, $x^4[n]$ and $x^2[n]$ underflows, output of MAM3 is given by

$$y[n] = (1 + mx[n])\sin[2\pi f_c n] + \cos[2\pi f_c n]. \quad (3.7)$$

(3.7) can be re-written as

$$y[n] = \left(\sqrt{2 + 2mx[n] + m^2 x^2[n]} \right) \cos[2\pi f_c n + \phi]. \quad (3.8)$$

But since, $x^2[n]$ underflows, (3.8) becomes

$$y[n] = \sqrt{2(1 + mx[n])} \cos[2\pi f_c n + \phi]. \quad (3.9)$$

Since ϕ is small and can be neglected [33]. This implies that when $x^2[n]$ underflows, MAM3 becomes SRAM. Similarly, for single Q format implementation, MAM3 reduces to MAM2 when $x^6[n]$ underflows, and reduces to MAM1 when both $x^6[n]$ and $x^4[n]$ underflow. If $x^6[n]$, $x^4[n]$ and $x^2[n]$ underflow, output of MAM3 is given by

$$y[n] = \sqrt{\frac{1+mx[n]}{2}} \cos[2\pi f_c n + \phi]. \quad (3.10)$$

Therefore, for both single Q format and mixed Q format implementations, MAM3 becomes SRAM if $x^2[n]$ underflows. Table 3.2 shows the range of $x[n]$ for which MAM3 reduces to MAM2, MAM1 and SRAM.

Table 3.2 Relationship defining operating regions for MAM3, MAM2, MAM1 and region in which MAM becomes SRAM.

Number that underflows	Condition for underflow		MAM3 reduces to
	Single Q format implementation	Mixed Q format implementation	
$x^6[n]$	$ x[n] < \frac{2^{-(N-6)/6}}{m}$	$ x[n] < \frac{2^{-(N-5)/6}}{m}$	MAM2
$x^4[n]$	$ x[n] < \frac{2^{-(N-5)/4}}{m}$	$ x[n] < \frac{2^{-(N-4)/4}}{m}$	MAM1
$x^2[n]$	$ x[n] < \frac{2^{-(N-3)/2}}{m}$	$ x[n] < \frac{2^{-(N-2)/2}}{m}$	SRAM

Figure 3.7 shows the operating regions for MAM3, MAM2, MAM1, and SRAM under different bit precisions of 8-bit, 16-bit, and 32-bit, for single Q format implementation of MAM3 as a function of m .

(a)

(b)

(c)

Figure 3.7: Operating regions of MAM3, MAM2, MAM1 and SRAM for single Q format implementation of MAM3 on (a) 8-bit, (b) 16-bit, and (c) 32-bit fixed-point processors.

Figure 3.7(a) shows that for an 8-bit fixed-point implementation, MAM3 output underflows for all values of $|x[n]|$ when $m \leq 0.80$. This implies that 8-bit fixed-point processor does not support MAM3 for ABS. But for 16-bit and 32-bit fixed-point processors, all the 4 regions namely MAM3, MAM2, MAM1 and SRAM exists as shown in Fig. 3.7(b) and (c), respectively. Consider the example where adaptive MAM is implemented on 8-bit, 16-bit, and 32-bit fixed-point processors with $m=0.75$. For 8-bit fixed-point processor, MAM2 reduces to MAM1 when $|x[n]| < 0.79$, and further reduces to SRAM when $|x[n]| < 0.24$. For a 16-bit fixed-point processor, MAM3 exists for $0.42 < |x[n]| < 1$ and reduces to MAM2 for $0.20 < |x[n]| \leq 0.42$,

MAM1 for $0.01 < |x[n]| \leq 0.20$, and SRAM for $0 < |x[n]| \leq 0.01$. For 32-bit fixed-point processor, the range of $|x[n]|$ for which MAM3 exists widen and the range for which SRAM exists almost shrink to 0. For a 32-bit fixed-point processor, MAM3 exists for $0.07 < |x[n]| < 1$, reduces to MAM2 for $0.01 < |x[n]| \leq 0.07$, and MAM1 for $\approx 0 < |x[n]| \leq 0.01$. Similarly, for the given values of N and m , values of $x[n]$ which underflows can be determined from Table 3.2. Depending on the value of each input sample, a decision block is used to select a suitable pre-processing technique. The decision block can be easily implemented based on the input sample amplitude under a certain modulation index m , and does not incur significant computational workload. The block diagram of adaptive MAM algorithm is shown in Fig. 3.8. A real-time implementation of adaptive MAM is carried out in the following section.

Figure 3.8 Block diagram of adaptive MAM.

3.5 Real-time implementation

A real-time implementation of DSBAM, MAM1, MAM2, MAM3, and adaptive MAM are implemented on the C5505 eZDSP USB Stick, which is a 16-bit fixed-point DSP processor. MIPS performance of DSBAM, MAM1, MAM2, MAM3 and adaptive MAM are similar for

both single Q and mixed Q format implementations. Since adaptive MAM selects the appropriate MAM algorithm for each input sample, it can help in reducing the computational workload in real-time. The average MIPS results are shown in Table 3.3.

Table 3.3 MIPS results for DSBAM, MAM1, MAM2, MAM3 and adaptive MAM.

Modulation scheme	MIPS	Onchip memory (kBytes)
DSBAM	6.40	15
MAM1	8.90	34
MAM2	9.96	34
MAM3	10.25	34
Adaptive MAM	9.79	35

Table 3.3 shows that adaptive MAM outperforms MAM2 and MAM3 for real-time implementation. It is, therefore, recommended that MAM3 and MAM2 are implemented with a decision block, which decreases the MIPS count in real-time implementation.

3.6 Conclusion

In this chapter, we analyzed the DSBAM and MAM q modulation schemes in terms of their computational workload and SQNR performance. Single Q and mixed Q format implementations of DSBAM and MAM q are compared for SQNR performance. Our study showed that single Q and mixed Q format implementation results in similar SQNR for DSBAM, but mixed Q format applied to MAM q implementation performs marginally better. Mixed Q format implementation requires lesser gain setting for analog amplifier, and reduces power consumption. An adaptive MAM scheme is also described to reduce MIPS in real-time. Therefore, a mixed Q format implementation of adaptive MAM improves both SQNR and computational workload performance simultaneously.

Chapter 4

Square root amplitude modulation

In this chapter, we investigate the performance of the SRAM on a fixed-point processor. SRAM algorithm is first described in Section 2.1.2 of this thesis. It consists of pre-processing of input signal and modulation of pre-processed signal. As the name suggests, pre-processing envelope of the SRAM is generated by taking SQRT of the envelope of DSBAM signal. Using the basic operations, SQRT is computed using a series of multiplications, additions, and look-up tables (LUTs) over a few iterations. Since fixed-point SQRT is an important operator of the SRAM algorithm, it is investigated in detail in this chapter.

This chapter is organized as follows. Section 4.1 describes the fixed-point SRAM. Section 4.2 gives a brief introduction of various approaches described in literature to calculate SQRT of a number. Out of the various approaches, two approaches are specifically suitable for processors with a fast multiplying unit. Section 4.3 describes the first approach and presents simulation results for fixed-point processors using this approach. Subsequently, Section 4.4 describes the second approach and presents simulation results for fixed-point processors. A square rooting method for fixed-point numbers is also proposed in this section. Bit accuracy and computation workload performance of various square rooting methods are compared in Section 4.5. Based on bit accuracy and computational workload performance comparison, the best fixed-point square rooting algorithm is chosen. A real-time implementation of fixed-point SRAM,

which uses the chosen square rooting algorithm, is carried out in Section 4.6. Finally, Section 4.7 summarizes and concludes this chapter.

4.1 Fixed-point square root amplitude modulation

The fixed-point SRAM implementation uses $Q1.N-1$ fixed-point arithmetic format, where N is the wordlength of the fixed-point processor. The $Q1.N-1$ arithmetic format is chosen to ensure maximum bit accuracy. As mentioned in Section 2.2.1, $Q1.N-1$ arithmetic format can represent numbers between $[-1, 1-2^{-(N-1)}]$. Any number lying outside this range either overflows or saturates. The SRAM envelope is expressed in (2.2). The maximum and minimum values of $E[n]$ are 1 (for $m=0$) and 1.41 (for $m=1$), respectively. Therefore, in fixed-point implementation using $Q1.N-1$ format, $E[n]$ needs to be scaled to avoid saturation or wrapping. For the case of $m < 1$, we can scale the SRAM envelope in (2.2) as

$$E[n] = \sqrt{\frac{1 + mx[n]}{2}}. \quad (4.1)$$

Figure 4.1 shows the block diagram of the scaled version of the fixed-point SRAM implementation.

Figure 4.1 Fixed-point SRAM implemented in $Q1.N-1$ format.

The SRAM algorithm relies heavily on the SQRT operation to provide a good estimation of the SQRT envelope so as to achieve good demodulation performance in parametric array in air. The

SQRT operator approximates the SQRT of a number, which is correct up to certain number of bits. The difference between the actual SQRT and approximated SQRT results in error term (noise), as shown in Fig. 4.2.

Figure 4.2 Effect of SQRT operator on performance of SRAM.

The error term is given by

$$e_{SQRT}[n] = \left(\sqrt{\frac{1+mx[n]}{2}} \right)_{APPROXIMATED} - \sqrt{\frac{1+mx[n]}{2}}. \quad (4.2)$$

The magnitude of the error term depends on the bit accuracy of the SQRT operator. A highly precise SQRT operator generates error term (noise) with lower magnitude resulting in higher SQNR for SRAM algorithm.

The SQRT operator consists of a sequence of multiplications and additions. The bit precision of the SQRT operator depends on the number of iterations of this sequence used for SQRT computation. SQRT operator contributes significantly to the computational workload of

the SRAM. The computational workload of the SQRT operator is approximately 50% of the entire computational workload of the SRAM algorithm. A detailed breakdown of the computational complexity of the SRAM algorithm will be given in the later section. The SQRT operator, therefore, has significant influence on both performance metrics of computational workload and SQNR. SQRT of a number can be calculated using one of the several algorithms described in [50]. The algorithms described in [50] approximate the SQRT of a number with certain bit accuracy. With better bit precision offered by the SQRT algorithm, arithmetic error can be significantly reduced in the pre-processed signal which lowers the noise in the square root amplitude modulated signal.

4.2 Fixed-point square rooting methods

In this section, we investigate several square rooting methods described in literature [50], and evaluate their bit accuracies for fixed-point implementation. They can be broadly classified as:

- A. Direct methods.
- B. Normalization techniques.
- C. Approximation by real functions.
- D. Algorithms based on Newton-Raphson (NR) formula.

Direct methods [51]-[54] and normalization techniques [55]-[58] are suitable for hardware implementation, whereas approximation by real functions [59], [60], [61] and NR-based algorithms [62], [63] are suitable for software implementation on processors with a fast multiplying unit. As many fixed-point processors, including DSPs and FPGAs support fast multiplying unit, we investigate performance of (C) and (D) on fixed-point processors. The

SQRT methods are generally evaluated for an interval ($0.25 \leq x < 1$ in our case). Square rooting methods are simulated on 8-bit, 16-bit and 32-bit fixed-point processors using the MATLAB fixed-point toolbox [44]. Here, we compare the accuracy of the fixed-point SQRT algorithm against the golden standard of 64-bit DP floating-point SQRT algorithm, which is also computed from MATLAB. Negative $\log_2$ of magnitude of error values is taken to calculate the bit precision for each output sample. Denoting the DP floating-point result by y , and N -bit rounded result after k^{th} iteration (or k^{th} order) by $y_{k,N}$, bit precision of the output sample is calculated as

$$\text{Bit precision of } y_{k,N} = -\log_2 |y - y_{k,N}|. \quad (4.3)$$

4.3 Square root approximation by real functions

In this section, we examine the SQRT approximation by real functions to compute the SQRT of a number. The real functions are generated by taking the first few finite terms of an infinite series. It is reasonable to think that more terms used in the series will lead to better SQRT approximation. This is however not true in the case of fixed-point implementation, as we will show in subsequent sections. The two most popularly used functions for SQRT approximations are:

- A. Taylor's series expansion
- B. Chebyshev polynomial approximation

These functions are simulated using MATLAB fixed-point toolbox and compared against DP floating-point results to evaluate their bit accuracies. These functions are discussed next.

4.3.1 Taylor's series expansion

The infinite length Taylor's series expansion for SQRT is expressed as

$$\sqrt{1+t} = 1 + \frac{1}{2}t - \frac{1}{8}t^2 + \dots + (-1)^{n-1} \frac{1 \times 3 \dots (2n-3)}{2 \times 4 \dots 2n} t^n + \dots, \quad (4.4)$$

and it is valid for $|t| < 1$. For evaluating finite length Taylor's series approximation (TSA) for SQRT, polynomials up to 6th order are considered. MATLAB simulations for both DP floating-point and fixed-point (8-bit, 16-bit and 32-bit) are carried out. As mentioned earlier, MATLAB fixed-point toolbox is used to implement the algorithm in fixed-point arithmetic precision. For N -bit ($N=8, 16$ or 32) fixed-point simulation, it would be useful to find out what percentage of output samples has bit precision greater than $N-1$. Figure 4.3 shows the histogram of bit precision of output samples for an 8-bit fixed-point processor. The bit precision axis of the plot ranges from 1 to N , where N is the wordlength of the fixed-point processor. The amplitude of histogram at bit precision $i=N$ indicates the percentage of output having *bit precision* $> (N-1)$, and the histogram amplitude at any other position i , such that $1 \leq i < N$, indicates the percentage of output having bit precision lying in the range $(i-1, i]$. For example, in a 5th order Taylors series polynomial, 91.75% of output data samples have *bit precision* > 7 bits ($N=8$), and the remaining 8.25% of output data samples have bit precision lying in the range $(6, 7]$, as shown in Fig. 4.3.

Figure 4.3 Histograms showing output bit precision of TSA on 8-bit fixed-point processor.

The histogram plots indicate that the output bit precision on fixed-point processors do not improve after the 5th polynomial order, which is due to finite wordlength effect on Taylor's series polynomial. For an 8-bit fixed-point processor, the percentage of output samples with *bit precision* > 7 is 91.75% for 5th order polynomial, and there is no improvement in the percentage of the output samples even if the polynomial order is increased to 6 or higher. Similarly, Fig. 4.4 and 4.5 show histograms of bit precision of output samples for 16-bit and 32-bit fixed-point processors, respectively.

Figure 4.4 Histograms showing output bit precision of TSA on 16-bit fixed-point processor.

Figure 4.5 Histograms showing output bit precision of TSA on 32-bit fixed-point processor.

For the 16-bit fixed-point processor, the maximum attainable value of the percentage of output samples with *bit precision* > 15 is 40.48%, which is the maximum value attained by the 5th order Taylor's series polynomial. Similarly, for the 32-bit fixed-point processor, the maximum attainable value of the percentage of output samples with *bit precision* > 31 is 6.46% and it is also attained by the 5th order polynomial. As the order of polynomial increases, no improvement in performance is observed for 8-bit, 16-bit and 32-bit fixed-point processors. Table 4.1 summarizes the computational workload and bit accuracy performance of TSA for 8-bit, 16-bit, and 32-bit fixed-point processors.

Table 4.1 Performance of TSA on 8-bit, 16-bit, and 32-bit fixed-point processors.

Fixed-point processor wordlength (N)	MPOS, APOS	Total memory size (Bytes)	% of output samples with bit precision > $N-1$
8-bit	9, 5	6	91.75
16-bit	9, 5	12	40.48
32-bit	9, 5	24	6.46

4.3.2 Chebyshev expansion

A general n^{th} order Chebyshev expansion can be derived [59], [60], [64] from (4.4) by operating substitutions with Chebyshev polynomials. For simulation, Chebyshev polynomials up to the 5th order are considered for approximating SQRT. The 5th order Chebyshev polynomials is given by

$$\sqrt{1+t} = 0.9993591 + 0.5017624t - 0.1134644t^2 + 0.0484009t^3 - 0.0698242t^4 + 0.055542t^5, \quad (4.5)$$

and it is valid for $|t| < 1$. For 5th order Chebyshev polynomial approximation (CPA), the percentage of output samples with *bit precision* > 7 (for $N=8$) is 65.98%, the percentage of output samples with *bit precision* > 15 (for $N=16$) is 1.91%, and the percentage of output

samples with *bit precision* > 31 (for $N=32$) is 0. Table 4.2 summarizes the performance of the 5th order CPA for 8-bit, 16-bit and 32-bit fixed-point processors.

Table 4.2 Performance of CPA on 8-bit, 16-bit, and 32-bit fixed-point processors.

Fixed-point processor wordlength (N)	MPOS, APOS	Total memory size (Bytes)	% of output samples with bit precision $> N-1$
8-bit	9, 5	6	65.98
16-bit	9, 5	12	1.91
32-bit	9, 5	24	0

The performance of the 5th order CPA is poorer than the 5th order TSA. The higher order CPAs are, therefore, not evaluated. TSA is preferred over CPA for SQRT computation using real function approximation on fixed-point processors as they provides better bit accuracy.

4.4 Square rooting algorithms based on Newton-Raphson formula

SQRT approximation using NR formula is the second class of algorithm used to calculate the SQRT of a number. There are many variants of NR method. They are classified as

- A. Direct Newton-Raphson variant 1 (DNR-1).
 - (i) DNR-1 using polynomial expansions
 - (ii) DNR-1 using LUTs
- B. Nonlinear infinite-impulse response filter (NLIIRF) method
- C. Direct Newton-Raphson variant 2 (DNR-2).
- D. Inverse Newton-Raphson (INR).

Variants (A.i), (B), (C) and (D) are well-known studied methods [61], [65], [66]. Variant (A.ii) is the NR-based method proposed for approximating SQRT with fixed-point arithmetic. The performance of these methods is evaluated and compared in subsequent sections.

4.4.1 Direct Newton-Raphson variant 1

NR is an iterative method [61] to compute SQRT of a number given by the following equation

$$y_{k+1} = \frac{1}{2} \left(y_k + \frac{x}{y_k} \right), \quad k = 0, 1, 2, \dots, \quad (4.6)$$

where y_{k+1} is the estimated value of SQRT of x (or $\sqrt{x}$) obtained after $(k+1)$ iterations.

Equation (4.6) can be re-written as

$$y_{k+1} = y_k + \frac{1}{2y_k} (x - y_k^2). \quad (4.7)$$

DNR-1 is obtained from (4.7) by replacing $\frac{1}{2y_k}$ with $\frac{1}{2\sqrt{x}}$ and is expressed as

$$y_{k+1} = y_k + \frac{1}{2\sqrt{x}} (x - y_k^2). \quad (4.8)$$

Note that unlike (4.7), DNR-1 starts with an initial value of $\frac{1}{2\sqrt{x}}$ and this value is used for all

successive iterations. For each iteration, DNR-1 requires 2 multiplications and 2 additions. Here,

we will use the following notation:

$$\text{SQRT}(x) = \sqrt{x}, \text{ and } \text{ISQRT}(x)/2 = \frac{1}{2\sqrt{x}}. \quad (4.9)$$

DNR-1 requires initial approximation of $\text{SQRT}(x)$, and $\text{ISQRT}(x)/2$. The initial approximation for $\text{SQRT}(x)$ is represented by y_0 and this approximation is improved after every iteration. Initial approximation of $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ can be carried out using polynomial

expansions (PEs) or LUTs. The block diagram of the DNR-1 method using two iterations to find the SQRT of an N -bit fixed-point number is shown in Fig. 4.6. In our notation the subscripted number N in x_N and $y_{0,N}$, for example, in Fig. 4.6 denotes that those samples are N bits in width.

Figure 4.6 Block diagram of DNR-1.

Based on the methods used for $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ initialization, two variations of DNR-1 are considered:

- A. DNR-1 using PEs for $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ initialization,
- B. DNR-1 using LUTs for $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ initialization.

These two variations are discussed and their performance is investigated in subsequent sections.

4.4.1.1 DNR-1 using PEs for SQRT and ISQRT initialization

This is the first variation of the DNR-1 algorithm. In this variation, we use PEs for initial approximation of both $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$. We restrict ourselves to 3rd order polynomials

for both $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ initialization i.e., linear PE (LPE), quadratic PE (QPE) and cubic PE (CPE) are used for $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ initialization. For naming purpose, DNR-1 using LPE for both $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ initialization is abbreviated as DNR-1(LL). Similarly, there are 8 more such combinations as shown in Table 4.3. Number of multiplications, additions and total memory size required by different PE combinations for N -bit fixed-point processor are also shown in Table 4.3.

Table 4.3 Naming convention used for DNR-1($\text{SQRT}(x)$ PE: $\text{ISQRT}(x)/2$ PE).

$\text{SQRT}(x)$	$\text{ISQRT}(x)/2$	MPOS, APOS	Total memory size (Bytes)	Abbreviation
LPE	LPE	2, 2	$4 \times (N/8)$	DNR-1(LL)
LPE	QPE	4, 3	$5 \times (N/8)$	DNR-1(LQ)
LPE	CPE	6, 4	$6 \times (N/8)$	DNR-1(LC)
QPE	LPE	4, 3	$5 \times (N/8)$	DNR-1(QL)
QPE	QPE	6, 4	$6 \times (N/8)$	DNR-1(QQ)
QPE	CPE	8, 5	$7 \times (N/8)$	DNR-1(QC)
CPE	LPE	6, 4	$6 \times (N/8)$	DNR-1(CL)
CPE	QPE	8, 5	$7 \times (N/8)$	DNR-1(CQ)
CPE	CPE	10, 6	$8 \times (N/8)$	DNR-1(CC)

There are ways described in [67]-[69] to determine the best initializing polynomials for $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$. These polynomials are derived by minimizing relative errors [67]-[69]. The relative error, E_k after k^{th} iteration is defined as

$$E_k = \frac{|y_k - \sqrt{x}|}{\sqrt{x}}. \quad (4.10)$$

But our simulation results show that for fixed-point processors, polynomials that fit $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ in the least square sense perform better than those derived by minimizing relative errors. In the present work, polynomials that fit $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ in the least square sense are used for evaluation purpose. It is expected that the percentage of output samples with *bit precision* $> N - 1$ increases with number of iterations, and reaches 100% after few iterations.

But due to finite wordlength effects, this does not happen for some PE combinations. For example, on an 8-bit fixed-point processor, the percentage of output samples with *bit precision* > 7 never reaches 100% for DNR-1(LL), DNR-1(LQ), DNR-1(QQ) and DNR-1(CQ) regardless of number of iterations used. Similar observations are made for 16-bit and 32-bit fixed-point processors. For 16-bit fixed-point processors, it is observed that the number of output samples with *bit precision* > 15 never reaches 100% for DNR-1(LL). Regardless of the number of iterations, the maximum attainable value for DNR-1(LL) is 84.30%. For the 32-bit fixed-point processor, all PE combinations except DNR-1(LL) can produce 100% output samples with *bit precision* > 31. Table 4.4 shows number of iterations, additions, multiplications, total memory size in bytes and maximum percentage value that can be achieved by DNR-1 using different combinations of PEs. Last column of Table 4.4 shows the maximum percentage value that can be achieved, and no improvement in the percentage values is observed even after increasing the number of iterations.

Table 4.4 Performance comparison for DNR-1 using PEs on 8-bit, 16-bit, and 32-bit fixed-point processors.

DNR-1	Iterations			MPOS, APOS			Total memory size (Bytes)			% of output samples with bit precision > $N-1$		
	8	16	32	8	16	32	8	16	32	$N=8$	$N=16$	$N=32$
DNR-1(LL)	3	4	10	8,8	10,10	22,22	4	8	16	90.63	84.30	84.48
DNR-1(LQ)	4	3	7	12,11	10,9	18,17	5	10	20	87.5	100	100
DNR-1(LC)	2	2	5	10,8	10,8	16,14	6	12	24	100	100	100
DNR-1(QL)	4	3	8	12,11	10,9	20,19	5	10	20	100	100	100
DNR-1(QQ)	13	2	6	32,30	10,8	18,16	6	12	24	81.25	100	100
DNR-1(QC)	3	2	4	14,11	12,9	16,13	7	14	28	100	100	100
DNR-1(CL)	1	3	8	8,6	12,10	22,20	6	12	24	100	100	100
DNR-1(CQ)	4	2	6	16,13	12,9	20,17	7	14	28	89.58	100	100
DNR-1(CC)	1	2	4	12,8	14,10	18,14	8	16	32	100	100	100

Based on the MPOS, APOS and output accuracy achieved, DNR-1(CL) generates the best result for 8-bit fixed-point processor. For the 16-bit fixed-point processor, both DNR-1(LC) and DNR-1(QQ) require minimum MPOS to generate 100% output samples with *bit precision* > 15. Therefore, DNR-1(LC) and DNR-1(QQ) are preferred over other PE combinations for 16-bit fixed-point processor. Similarly, DNR-1(QC) is preferred for 32-bit fixed-point processor.

4.4.1.2 DNR-1 using LUTs for SQRT and ISQRT initialization

In previous section, we investigated the performance of DNR-1 using PEs for SQRT(x) and ISQRT(x)/2 initialization. In this section, we propose a DNR-1 based tabular method for calculating SQRT of fixed-point numbers. This DNR-1 based tabular method is denoted as DNR_T(n). The symbol “ n ” in DNR_T(n) denotes the number of multiplications used by the algorithm. DNR_T(n) uses LUTs to store initial approximate values of SQRT(x) and ISQRT(x)/2. A direct advantage of using LUTs over PEs is that it saves a number of additions and multiplications at the expense of memory consumed by the LUTs. The input range for the DNR_T(n) spans from $0.25 \leq x < 1$. To generate the SQRT(x) and ISQRT(x)/2 LUTs, the interval $0.25 \leq x < 1$ is divided into subintervals with equal width of $2^{-(L-1)}$ ($L < 16$). This gives $0.75 \times 2^{(L-1)}$ different values, denoted as x_L , obtained by rounding x_N to L bits (1 bit for sign and $(L-1)$ bits for magnitude). The SQRT(x_L) and ISQRT(x_L)/2 values are computed for each x_L , rounded to N bits, and stored in the SQRT(x) and ISQRT(x)/2 LUTs, respectively. So the SQRT(x) LUT (or SQRT LUT) and ISQRT(x)/2 LUT (or ISQRT LUT), each has $0.75 \times 2^{(L-1)}$ entries, where each entry is N bits in width. The index of each LUT memory location, integer I_{DX} , is in the range $0 \leq I_{DX} \leq (0.75 \times 2^{(L-1)} - 1)$ and is computed using

$$I_{DX} = (x_L - 0.25) \times 2^{(L-1)} \quad (4.11)$$

requiring one subtraction operation and a multiply implemented by an arithmetic left shift. The same I_{DX} value is used for both $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ LUTs. Sample x_N is rounded to x_L by adding 2^{-L} to x_N and truncating the lower $(N-L)$ bits of the result. Therefore, including the subtraction required by (4.11), one addition and one subtraction are required to generate index for LUTs.

In the first iteration of Fig. 4.6, a $\text{SQRT}(x)$ LUT value (assuming that LUT is used for $\text{SQRT}(x)$ initialization) is squared to produce an x_L data sample. That x_L is subsequently subtracted from x_N to generate the error term e as

$$e = x_N - x_L. \quad (4.12)$$

We can compute e without using (4.12). If x_N is truncated to L bits, the truncation error term, denoted by e_T , is obtained by extracting the least significant $(N-L)$ bits of x_{16} . Our desired error term e is then calculated from e_T as

$$e = (e_T \leftarrow L) \rightarrow L, \quad (4.13)$$

where ' $\leftarrow L$ ' (' $\rightarrow L$ ') means an arithmetic left (right) shift by L bits (examples to show that (4.12) and (4.13) generate same values are given in Appendix C). Therefore, our first trick is to replace the squaring and subtraction of the first iteration of Fig. 4.6 with L -bit truncation and arithmetic shifts as shown in Fig. 4.7. The subsequent iterations remain unchanged. Figure 4.7 represents $\text{DNR}_T(n)$ as a cascade of a non-repetitive and repetitive structures, where the non-repetitive structure performs the first iteration and the repetitive structure performs the subsequent iterations. The non-repetitive structure is followed by none or more than or equal to one repetitive structures depending on the n value of $\text{DNR}_T(n)$.

Figure 4.7 $\text{DNR}_T(n)$ as a cascade of repetitive and non-repetitive structures.

A Case study for 8-bit, 16-bit and 32-bit fixed-point processors

In this section, we present the $\text{DNR}_T(n)$ algorithm (for $n=1, 2, 3, 5$ and 7) that is suitable for 8-bit, 16-bit, and 32-bit fixed-point processors. Figure 4.8 shows the block diagram of $\text{DNR}_T(1)$, $\text{DNR}_T(3)$, $\text{DNR}_T(5)$ and $\text{DNR}_T(7)$ as a cascade of repetitive and non-repetitive structures. For an 8-bit fixed-point number, only the $\text{DNR}_T(1)$ is used; for a 16-bit fixed-point number, any one of the $\text{DNR}_T(1)$, $\text{DNR}_T(2)$ or $\text{DNR}_T(3)$ can be used; and for 32-bit fixed-point number $\text{DNR}_T(5)$ or $\text{DNR}_T(7)$ can be used.

Figure 4.8 Block diagram of $\text{DNR}_T(1)$, $\text{DNR}_T(3)$, $\text{DNR}_T(5)$ and $\text{DNR}_T(7)$.

Due to the range of data values at various nodes in Fig. 4.8, we can make a few reasonable assumptions about that data and approximate the $y_{2,N}$ output using only one processing iteration.

We call that single-iteration scheme the “ $\text{DNR}_T(2)$ method” and depict it in Fig. 4.9.

Figure 4.9 Block diagram of DNR_T(2).

The $y_{2,N}$ output in Fig. 4.8 is approximated by the $y_{1,N}$ output in Fig. 4.9, requiring only two multiplications, using

$$y_{1,N} \approx y_{2,N} \approx LUT_{SQRT}(i) + LUT_{ISQRT/2}(i) \times \left(e - \frac{e^2}{2} \right), \quad (4.14)$$

where $LUT_{SQRT}(i)$ and $LUT_{ISQRT/2}(i)$ are the i^{th} entries of $SQRT(x)$ and $ISQRT(x)/2$ LUT, respectively. The derivation of (4.14) is provided in Appendix C. If the $e^2/2$ term in (4.14) is neglected, we notice that (4.14) becomes the DNR_T(1) method of Fig. 4.8. The non-repetitive structure of Fig. 4.8 uses enhanced precision (EP) ISQRT and SQRT LUTs with decoders. Next, we show an improved DNR_T(n) scheme that uses EP LUTs.

B Enhanced precision LUTs

Because the $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ values are always positive, the sign bit of the binary representations of those values is always '0'. Now, for the input interval $0.25 \leq x_N < 1$, the contents of both LUTs are in the range of $0.50 \leq \text{tablevalue} < 1$. So the MSB (bit next to sign bit) is always '1' for each entry of the LUTs. Our next trick is to make use of these facts to generate EP $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ LUTs. Figure 4.10(a), for example, shows how an EP $\text{ISQRT}(x)/2$ LUT value having 10-bit precision can be stored in an 8-bit LUT memory location. When the 8-bit wide EP $\text{ISQRT}(x)/2$ LUT value is accessed for computation, the 8-bit value is converted to an EP 10-bit value, as shown in Fig. 4.10(b).

(a)

(b)

Figure 4.10 EP LUT: (a) 8-bit entry stored in EP $\text{ISQRT}/2$ LUT and (b) EP $\text{ISQRT}/2$ LUT decoder.

Similarly, we can generate 16-bit EP LUTs. Table 4.5 compares the performance of $\text{DNR}_T(n)$ with normal LUTs and EP LUTs. The bit-width of $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ LUT used in both cases is shown in Table 4.5. The L values used by $\text{DNR}_T(n)$ algorithms for L -bit truncation are also listed in Table 4.5. Those L values are a consequence of the chosen $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ LUT subinterval widths. For example, if the $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ LUT subinterval width is 2^{-4} , then $L = (1+4) = 5$. We interpret $(1+4)$ to be one sign bit and four magnitude bits (because processing is in $Q1.N-1$ binary arithmetic), so in this case x_N is rounded to five bits.

Table 4.5 Normal and EP LUT size and performance comparison.

N	Number of bits per LUT entry		$\text{DNR}_T(n)$	Subinterval width, L	Total EP LUT size (Bytes)		% of output samples with bit precision $> N-1$	
	Normal (SQRT, ISQRT/2)	EP (SQRT, ISQRT/2)			Normal	EP	Normal	EP
8	8,8	8,8	$[\text{DNR}_T(1)]_A$	$2^{-3}, 4$	12	12	98.96	100
16	16,16	16,8	$[\text{DNR}_T(1)]_B$	$2^{-6}, 7$	192	144	93.81	98.65
			$[\text{DNR}_T(1)]_C$	$2^{-7}, 8$	384	288	99.74	100
			$[\text{DNR}_T(2)]_A$	$2^{-6}, 7$	192	144	99.41	99.67
			$[\text{DNR}_T(2)]_B$	$2^{-7}, 8$	384	288	99.91	100
			$[\text{DNR}_T(3)]_A$	$2^{-4}, 5$	48	36	98.10	98.22
			$[\text{DNR}_T(3)]_B$	$2^{-5}, 6$	96	72	99.99	99.98
32	16,16	16,8	$\text{DNR}_T(5)$	$2^{-7}, 8$	384	288	98.47	98.86
			$\text{DNR}_T(7)$	$2^{-6}, 7$	192	144	99.96	99.98

We see from Table 4.5 that using EP LUTs not only reduces memory requirements, but also generates more accurate results than using normal LUTs. Table 4.6 shows results for different configurations of $\text{DNR}_T(1)$, $\text{DNR}_T(2)$, $\text{DNR}_T(3)$, $\text{DNR}_T(5)$ and $\text{DNR}_T(7)$ using EP LUTs. Total memory size is the number of bytes required for storing $\text{SQRT}(x)$ and $\text{ISQRT}(x)/2$ LUTs. Table

4.6 shows that for each $\text{DNR}_T(n)$, the bit precision can be increased/decreased by increasing/decreasing the size of the LUTs. Number of multiplications and additions used by $\text{DNR}_T(1)$, $\text{DNR}_T(2)$, $\text{DNR}_T(3)$, $\text{DNR}_T(5)$ and $\text{DNR}_T(7)$ are also shown in Table 4.6.

Table 4.6 Performance of $\text{DNR}_T(n)$ algorithms.

N	$\text{DNR}_T(n)$	MPOS, APOS	Total memory size (Bytes)	% of output samples with bit precision $> N-1$
8	$[\text{DNR}_T(1)]_A$	1,3	12	100
16	$[\text{DNR}_T(1)]_B$	1,3	144	98.65
	$[\text{DNR}_T(1)]_C$	1,3	288	100
	$[\text{DNR}_T(2)]_A$	2,4	144	99.67
	$[\text{DNR}_T(2)]_B$	2,4	288	100
	$[\text{DNR}_T(3)]_A$	3,5	36	98.22
	$[\text{DNR}_T(3)]_B$	3,5	72	99.98
32	$\text{DNR}_T(5)$	5,7	288	98.86
	$\text{DNR}_T(7)$	7,9	144	99.98

Apart from $\text{DNR-1}(\text{SQRT}(x)$ PE: $\text{ISQRT}(x)/2$ PE) and $\text{DNR}_T(n)$, there are two more variations of DNR-1. The first one uses LUT for $\text{SQRT}(x)$ initialization and PE for $\text{ISQRT}(x)/2$ initialization, and is denoted as $\text{DNR-1}(\text{SQRT}(x)$ LUT: $\text{ISQRT}(x)/2$ PE). $\text{DNR-1}(\text{SQRT}(x)$ LUT: $\text{ISQRT}(x)/2$ PE) does not generate acceptable results and is not investigated further. The DNR-1 variation using PE for $\text{SQRT}(x)$ and LUT for $\text{ISQRT}(x)/2$ initialization is considered in [65]. It is referred in [65] as the NLIIRF method. NLIIRF is described in the next section.

4.4.2 Nonlinear infinite-impulse response filter method

This method [65] uses an iterative technique to find the SQRT of a number. The input range of the NLIIRF method is $0.25 \leq x < 1$. Any value of x lying outside this range needs to be normalized to this range. The iterative equation used to determine the $\text{SQRT}(x)$ of x is given by

$$y_{k+1} = y_k + \beta(x - y_k^2), \quad k = 0, 1, 2, \dots \quad (4.15)$$

The nonlinear difference equation of (4.15) can be implemented by an infinite-impulse response (IIR) filter, and hence the name NLIIRF method. In (4.15), y_k is the approximation for $\text{SQRT}(x)$ at the k^{th} iteration. However, to start off (4.15), the starting approximation for $\text{SQRT}(x)$ (stated as y_0) needs to be found by using a linear PE. In addition, β can be considered as an approximation of $\text{ISQRT}(x)/2$ and stored in a LUT in [65]. Each iteration of NLIIRF method requires 2 multiplications and 2 additions. 1 addition and 1 subtraction are required to generate the LUT index to fetch β . In [66], the authors suggested that a LPE (NLIIRF(L)) of the form of

$$\beta = 1.0688 - 0.61951x, \quad (4.16)$$

and a QPE (NLIIRF(Q)) of the form of

$$\beta = 1.314 - 1.5688x + 0.763x^2 \quad (4.17)$$

can be used. For β initialization, NLIIRF(L) requires 1 multiplication and 1 addition whereas NLIIRF(Q) requires 3 multiplications and 2 additions. In [65], the author showed results for the 16-bit fixed-point processor only. We can use the same LUT to generate results for the 8-bit fixed-point processor, but not for the 32-bit fixed-point processor. As we do not have LUT for the 32-bit fixed-point processor, NLIIRF(LUT) is not evaluated for the 32-bit fixed-point processor. Table 4.7 shows the performance comparison for NLIIRF(LUT), NLIIRF(L) and NLIIRF(Q) on 8-bit, 16-bit and 32-bit fixed-point processors.

Table 4.7 Performance comparison for NLIIRF using PEs on 8-bit, 16-bit, and 32-bit fixed-point processors.

NLIIRF method	Iterations			MPOS, APOS			Total memory size (Bytes)			% of output samples with bit precision > $N-1$		
	8	16	32	8	16	32	8	16	32	$N=8$	$N=16$	$N=32$
NLIIRF (LUT)	1	3	N.A.	3,5	7,9	N.A.	14	28	N.A.	100	100	N.A.
NLIIRF (L)	3	5	9	8,8	12,12	20,20	4	8	16	92.71	83.77	85.10
NLIIRF (Q)	1	3	5	6,5	10,9	14,13	5	10	20	100	100	100

N.A. - Not Applicable

Except NLIIRF(L), both NLIIRF(LUT) and NLIIRF(Q) can produce 100% output samples with *bit precision* > $N-1$. For NLIIF methods, NLIIRF(LUT) is the preferred method for 8-bit and 16-bit fixed-point processors, and NLIIRF(Q) is the preferred method for the 32-bit fixed-point processor.

4.4.3 Direct Newton-Raphson variant 2

The DNR-2 [61] is the second variant of the direct NR formula and is given by

$$\left(\frac{1}{\sqrt{x}}\right)_{k+1} = \left(\frac{1}{\sqrt{x}}\right)_k \times \left(2 - (\sqrt{x})_k \times \left(\frac{1}{\sqrt{x}}\right)_k\right), \quad k = 0, 1, 2, \dots, \quad (4.18)$$

and

$$(\sqrt{x})_{k+1} = \frac{1}{2} \times \left((\sqrt{x})_k + x \left(\frac{1}{\sqrt{x}}\right)_{k+1} \right), \quad k = 0, 1, 2, \dots \quad (4.19)$$

Each iteration of DNR-2 requires 3 multiplications and 2 additions to calculate $(\sqrt{x})_{k+1}$ from

$(\sqrt{x})_k$ and $\left(\frac{1}{\sqrt{x}}\right)_k$ using (4.18) and (4.19). Similar to DNR-1, DNR-2 requires initial estimation

of SQRT(x) and ISQRT(x), and every iteration enhances the approximation of both SQRT(x) and

ISQRT(x). The improved approximations of SQRT(x) and ISQRT(x) are used in the next iteration. We first investigate the performance of DNR-2 when PEs are used for both SQRT(x) and ISQRT(x) initialization. As in DNR-1, DNR-2 also uses initializing polynomials (best fitting polynomials in least square sense) up to the 3rd order. Naming convention similar to DNR-1(SQRT(x) PE: ISQRT(x) PE) is used for DNR-2(SQRT(x) PE: ISQRT(x) PE). Table 4.8 compares the performance of DNR-2 on 8-bit, 16-bit, and 32-bit fixed-point processors.

Table 4.8 Performance comparison for DNR-2 using PEs on 8-bit, 16-bit, and 32-bit fixed-point processors.

DNR-2	Iterations			MPOS, APOS			Total memory size (bytes)			% of output samples with bit precision > $N-1$		
	8	16	32	8	16	32	8	16	32	$N=8$	$N=16$	$N=32$
DNR-2(LL)	1	4	6	5,4	14,10	20,14	4	8	16	50.00	49.02	51.83
DNR-2(LQ)	1	4	6	7,5	16,11	22,15	5	10	20	53.13	49.63	51.09
DNR-2(LC)	3	4	6	15,10	18,12	24,16	6	12	24	44.79	49.15	52.08
DNR-2(QL)	2	5	7	10,7	19,13	25,17	5	10	20	44.79	47.52	50.22
DNR-2(QQ)	1	3	5	9,6	15,10	21,14	6	12	24	46.88	47.97	49.16
DNR-2(QC)	2	3	7	14,9	17,11	29,19	7	14	28	43.75	46.83	49.53
DNR-2(CL)	N.A.	7	8	N.A.	27,18	30,20	N.A.	12	24	N.A.	47.56	49.60
DNR-2(CQ)	N.A.	2	5	N.A.	14,9	23,15	N.A.	14	28	N.A.	48.66	51.09
DNR-2(CC)	N.A.	2	4	N.A.	16,10	22,14	N.A.	16	32	N.A.	48.70	51.27

The finite wordlength effects are very prominent for DNR-2. Regardless of the number of iterations used, no combination of SQRT(x) PE and ISQRT(x) PE can produce 100% output samples with *bit precision* > $N-1$ for 8-bit, 16-bit or 32-bit fixed-point processors. Notice that for 16-bit fixed-point processor, the percentage of output samples with *bit precision* > 15 does not reach even 50% for any PE combination. Among different PE combinations, DNR-2(LQ) is preferred for 8-bit and 16-bit fixed-point processors, and DNR-2(LC) is preferred for the 32-bit

fixed-point processor. The percentage of output samples with *bit precision* $> N - 1$ for 8-bit, 16-bit and 32-bit fixed-point processors is very low as compared to any variation of DNR-1 or NLIIFR methods. Therefore, other variations of DNR-2 are not investigated.

4.4.4 Inverse Newton-Raphson

The INR method is the third variant of the NR method [61] and is given by

$$\left(\frac{1}{\sqrt{x}}\right)_{k+1} = \frac{1}{2} \times \left(\frac{1}{\sqrt{x}}\right)_k \times \left[3 - x \left(\frac{1}{\sqrt{x}}\right)_k^2\right], \quad k = 0, 1, 2, \dots \quad (4.20)$$

Each iteration of INR requires 3 multiplications and 1 addition to calculate $\left(\frac{1}{\sqrt{x}}\right)_{k+1}$ from

$\left(\frac{1}{\sqrt{x}}\right)_k$. Similar to the DNR-2 method, output bit precision of the INR method does not increase

continuously, and saturates after some iterations. This is again due to finite wordlength effect.

The iteration after which output bit precision stop increasing is referred here as final iteration.

After final iteration, $\text{SQRT}(x)$ is obtained by an additional multiplication, $x \left(\frac{1}{\sqrt{x}}\right)_k = (\sqrt{x})_k$.

INR requires an initial estimation of $\text{ISQRT}(x)$, which is successively improved using iterations of INR. We investigate the performance of the INR method using PE for initialization. INR using LPE, QPE and CPE for $\text{ISQRT}(x)$ initialization are denoted by INR(L), INR(Q) and INR(C), respectively. Similar to the DNR methods, our fixed-point simulation for the INR method shows that polynomials fitting $\text{ISQRT}(x)$ in least square sense perform better than those derived by minimizing relative error. In the present work, polynomials that fit $\text{ISQRT}(x)$ in the least square sense are used for evaluating INR method. As in DNR, here we restrict ourselves to initializing

polynomials up to the 3rd order. Table 4.9 compares the performance of INR using PE on 8-bit, 16-bit, and 32-bit fixed-point processors.

Table 4.9 Performance comparison for INR on 8-bit, 16-bit, and 32-bit fixed-point processors.

INR	Iterations			MPOS, APOS			Total memory size (bytes)			% of output samples with bit precision $> N-1$		
	8	16	32	8	16	32	8	16	32	$N=8$	$N=16$	$N=32$
INR(L)	2	4	5	8,3	14,5	17,6	2	4	8	72.92	84.05	86.96
INR(Q)	2	4	5	10,4	16,6	19,7	3	6	12	71.88	85.07	88.27
INR(C)	2	3	5	12,5	15,6	21,8	4	8	16	76.04	84.54	86.03

For an 8-bit fixed-point processor, INR(C) provides maximum percentage of output samples with *bit precision* > 7 , and CPE is the preferred PE to be used with the INR method on an 8-bit fixed-point processor. For both 16-bit and 32-bit fixed-point processors, INR(Q) generates best results, and QPE is the preferred QE to be used. The output bit precision generated by INR(ISQRT(x) PE) on 8-bit, 16-bit and 32-bit fixed-point processors is better than those generated by the DNR-2 method using PEs. But the INR method using PE is still inferior to the DNR-1 method (including DNR-1 using PEs, DNR-1 using LUTs and NLIIRF method). Therefore, the LUT variation of the INR method is not investigated.

4.5 Performance comparison of square rooting methods

To compare the performance of various square rooting methods, number of additions, multiplications, total memory size required and output accuracy for each square rooting method is calculated (for 8-bit, 16-bit, and 32-bit fixed-point processors). Having output accuracy as performance criterion, the best case results are compared for 8-bit, 16-bit and 32-bit fixed-point

processors in Table 4.10. For all methods, except $\text{DNR}_T(n)$, the last column of Table 4.10 shows the maximum percentage value that can be achieved, and no improvement in the percentage values is observed even after increasing the number of iterations (or polynomial order). The $\text{DNR}_T(n)$ method with different subinterval widths results in different memory consumption and output bit accuracy. Table 4.10 shows results for different configurations of the $\text{DNR}_T(1)$, $\text{DNR}_T(3)$, $\text{DNR}_T(5)$ and $\text{DNR}_T(7)$ methods, which are discussed previously.

Table 4.10 Performance comparison of square rooting methods on 8-bit, 16-bit and 32-bit fixed-point processors.

N	Method	MPOS,APOS	Memory (Bytes)	% of output samples with bit precision $> N-1$
8-bit	$[\text{DNR}_T(1)]_A$	1,3	12	100
	NLIIRF(LUT)	3,5	14	100
	NLIIRF(Q)	6,5	5	100
	DNR-1(CL)	8,6	6	100
	TSA	9,5	6	91.75
	INR(C)	12,5	4	76.04
	CPA	9,5	6	65.98
	DNR-2(LQ)	7,5	5	53.13
16-bit	$[\text{DNR}_T(1)]_C$	1,3	288	100
	$[\text{DNR}_T(2)]_B$	2,4	288	100
	NLIIRF(LUT)	7,9	28	100
	DNR-1(LC)/DNR-1(QQ)	10,8	12	100
	NLIIRF(Q)	10,9	10	100
	$[\text{DNR}_T(3)]_B$	3,5	72	99.98
	INR(Q)	16,6	6	85.07
	DNR-2(LQ)	16,11	10	49.63
	TSA	9,5	12	40.48
	CPA	9,5	12	1.91
32-bit	NLIIRF(Q)	14,13	20	100
	DNR-1(QC)	16,13	28	100
	$\text{DNR}_T(7)$	7,9	144	99.98
	$\text{DNR}_T(5)$	5,7	288	98.86
	INR(Q)	19,7	12	88.27
	DNR-2(LC)	24,16	24	52.08
	TSA	9,5	24	6.46
	CPA	9,5	24	0

As Table 4.10 shows, the NR-based methods outperform SQRT approximation by real functions for 8-bit, 16-bit and 32-bit fixed-point processors. Therefore, using real functions for SQRT approximation is not a good choice for fixed-point processors. Past works [50], [61], [65] have reported that the DNR-1 and NLIIRF methods have property of linear convergence, while the DNR-2 method has the property of convergence rate that lie between linear and quadratic. The INR method has property of quadratic convergence. These convergence rates are theoretical expressions for infinite wordlengths. But in practice, finite wordlength effects must be taken into consideration. According to theoretical convergence rates, the INR method should produce the best result followed by the DNR-2, and the DNR-1 and the NLIIRF methods. But fixed-point simulation results showed that DNR-1 and NLIIRF methods perform better than INR and DNR-2 methods. This shows that DNR-1 and NLIIRF methods are less sensitive to finite wordlength effect. In general, it can be said that for fixed-point processors, the DNR-1 method generates the best results followed by INR, DNR-2, TSA, and Chebyshev polynomials. Among the DNR-1 variants, $\text{DNR}_T(n)$ outperforms DNR-1 using PEs, and NLIIRF method and its variants in terms of bit accuracy and computational workload. Moreover, the $\text{DNR}_T(n)$ method provides options for reducing computational workload at the expense of memory. This scalability is not supported by any square rooting method, and can be exploited in various situations making $\text{DNR}_T(n)$ a versatile algorithm.

4.6 Real-time implementation

In this section, we present the implementation of $[\text{DNR}_T(1)]_C$, $[\text{DNR}_T(2)]_B$ and $[\text{DNR}_T(3)]_B$ on a 16-bit fixed-point processor. Most of the implementation platforms provide some sort of library support for calculating basic functions, like SQRT, division, arc tangent, etc. We use TI's C55x

fixed-point DSP processor [48] as our fixed-point implementation platform. TI C55x provides DSP library to compute SQRT of a 16-bit fixed-point number [70]. From now onwards, we refer to the C55x DSP library SQRT algorithm as ‘DSPLIB SQRT’. We are interested in evaluating the performance of the DSPLIB SQRT and compare it against the $\text{DNR}_T(n)$ based algorithms, in terms of bit accuracy and computational workload performance. For this, we implemented $[\text{DNR}_T(1)]_C$, $[\text{DNR}_T(2)]_B$ and $[\text{DNR}_T(3)]_B$ on the TI C55x processor. The square rooting algorithm discussed so far works in the range of $0.25 \leq x < 1$. Any number smaller than 0.25 is first normalized to lie in the range of $0.25 \leq x < 1$, then its SQRT is calculated and the result is denormalized. The normalization-denormalization process can be carried out using binary shifts. This normalization-denormalization process needs to be carried out for each sample before taking SQRT, and consumes significant amount of MIPS. However, in our ABS implementation discussed in Section 3.2, $m > 0.80$ is seldom used. This is because higher m values increase THD and degrade sound quality [33]. We therefore restrict ourselves to $m \leq 0.80$. In such a case, $0.10 \leq \frac{1+mx[n]}{2} < 0.90$, and the smallest input value that needs to be normalized is 0.10. This fact helps in reducing the complexity of normalization-denormalization process. Table 4.11 compares the bit accuracy and MIPS of DSPLIB SQRT, $[\text{DNR}_T(1)]_C$, $[\text{DNR}_T(2)]_B$ and $[\text{DNR}_T(3)]_B$.

Table 4.11 MIPS comparison of proposed square rooting method and DSPLIB SQRT.

Square rooting method	% of output samples with bit precision > 15	MIPS
DSPLIB SQRT	66.93	9.55
$[\text{DNR}_T(1)]_C$	100	3.71
$[\text{DNR}_T(2)]_B$	100	3.72
$[\text{DNR}_T(3)]_B$	99.98	9.49

Table 4.11 shows that percentage of output samples with *bit precision* > 15 is 66.93% for DSPLIB SQRT. The percentage of output samples with *bit precision* > 15 is greater than 99.97% for $[\text{DNR}_T(1)]_C$, $[\text{DNR}_T(2)]_B$ and $[\text{DNR}_T(3)]_B$. In terms of bit accuracy, $\text{DNR}_T(n)$ algorithms outperform DSPLIB SQRT by significant margins.

The $[\text{DNR}_T(1)]_C$, $[\text{DNR}_T(2)]_B$ and $[\text{DNR}_T(3)]_B$ are implemented as C codes, and are not optimized for the C55x DSP architecture. On the other hand, DSPLIB SQRT is assembly coded and optimized for the C55x DSP architecture. $[\text{DNR}_T(1)]_C$ and $[\text{DNR}_T(2)]_B$ has significantly reduced MIPS requirement as compared to DSPLIB SQRT. The MIPS requirement of $[\text{DNR}_T(3)]_B$ is comparable to DSPLIB SQRT. $[\text{DNR}_T(1)]_C$, $[\text{DNR}_T(2)]_B$ and $[\text{DNR}_T(3)]_B$ outperform DSPLIB SQRT both in terms of MIPS and output accuracy. Next, we compare the bit accuracy and computational workload performance of SRAM with $[\text{DNR}_T(1)]_B$, $[\text{DNR}_T(2)]_B$, $[\text{DNR}_T(3)]_B$ and DSPLIB SQRT as square rooting methods. MIPS and SQNR comparison for SRAM using different square rooting methods are shown in Table 4.12.

Table 4.12 MIPS and SQNR comparison of SRAM implemented with proposed square rooting method and DSPLIB SQRT.

SRAM	SQNR	MIPS	Onchip memory (kBytes)
DSPLIB SQRT	85.88	17.47	17
$[\text{DNR}_T(1)]_C$	91.59	11.21	38
$[\text{DNR}_T(2)]_B$	91.59	12.22	38
$[\text{DNR}_T(3)]_B$	91.58	16.98	38

The MIPS results of Table 4.12 follow a similar trend as the MIPS results in Table 4.11. The MIPS requirement of $[\text{DNR}_T(1)]_C$, $[\text{DNR}_T(2)]_B$ and $[\text{DNR}_T(3)]_B$ is lesser than the MIPS

requirement of DSPLIB SQRT. But the memory requirement of $[\text{DNR}_T(1)]_C$, $[\text{DNR}_T(2)]_B$ and $[\text{DNR}_T(3)]_B$ is considerably higher than the memory requirement of DSPLIB SQRT in real-time. This is because the DSPLIB SQRT function is a pre-compiled assembly code optimized for memory, whereas $[\text{DNR}_T(1)]_C$, $[\text{DNR}_T(2)]_B$ and $[\text{DNR}_T(3)]_B$ are unoptimized codes written in C. Table 4.12 shows that SQNRs at DSP output for $[\text{DNR}_T(1)]_C$, $[\text{DNR}_T(2)]_B$ and $[\text{DNR}_T(3)]_B$ are approximately 6 dB higher than SQNR computed using DSPLIB SQRT. In other words, bit precision at DSP output improves by 1 bit if $\text{DNR}_T(n)$ algorithms are used for SRAM.

4.7 Conclusion

In this chapter, we presented a brief literature survey of the square rooting methods. The square rooting methods that are suitable for fixed-point processors with a multiplier unit were investigated in detail. A $\text{DNR}_T(n)$ algorithm for square rooting fixed-point numbers was proposed. The square rooting algorithms were simulated, and their bit accuracy and computational workload performances were compared for 8-bit, 16-bit and 32-bit fixed-point processors. The finite wordlength effect on each method was highlighted. $\text{DNR}_T(n)$ algorithms outperformed all square rooting methods in terms of bit accuracy and computational workload performance for 8-bit, 16-bit and 32-bit fixed-point processors. In particular, $\text{DNR}_T(1)$, $\text{DNR}_T(2)$ and $\text{DNR}_T(3)$ algorithms were implemented on the TI C55x DSP processor to validate their performances. Approximately 6 dB increment in SQNR (equivalent to 1 bit improvement in output accuracy) was achieved by using $\text{DNR}_T(n)$ algorithms compared to the C55x fixed-point square rooting DSP library routine.

Chapter 5

Single-sideband amplitude modulation

In Chapter 4, we discussed a number of fixed-point square rooting methods, and showed how a suitable method can reduce computational workload and increase SQNR significantly. With same objectives of minimizing computational workload and maximizing SQNR, we will now investigate different methods of generating SSBAM signal. Multi-rate processing is a standard way of reducing computational workload. However, interpolation filters used in multi-rate processing system increases the computational workload. Multi-rate processing is justified only when the reduction in computational workload under reduced rate processing outweighs the computational workload introduced by interpolation filters. Therefore, a comparative analysis of single-rate and multi-rate implementation of SSBAM signal generation methods is carried out in this chapter. Based on the analysis results, appropriate design decisions are made. In this chapter, we focus on different implementation issues of fixed-point SSBAM, which include saturation mode, clipping effect, and scaling for fixed-point implementation. In order to avoid saturation or clipping in fixed-point SSBAM, different scaling factors are introduced for different SSBAM generation methods.

This chapter is organized as follows. Section 5.1 gives a brief introduction of three methods of generating SSBAM signals, and these methods are investigated in detail in Section 5.2, 5.3, and 5.4. For each method, computational workload performance is compared for single-

rate and multi-rate implementations. The implementation with the lower computational workload is selected for each method. Section 5.5 compares these three methods and selects the best approach. An SQNR analysis is also performed for the selected approach. A real-time implementation of the chosen method is carried out in Section 5.6. Finally, Section 5.7 summarizes and concludes this chapter.

5.1 Single-sideband amplitude modulation for the ABS

For SSBAM signal spectrum, $y(t)$ consists of signal spectrum that is either the upper or lower-sideband of the carrier frequency. For the ABS, the carrier frequency f_c is set as the resonating frequency of the ultrasonic transducer. A number of ultrasonic transducers [71]-[73] with different resonating frequencies are available in the market. In our study, we select the resonating frequency of the ultrasonic transducer to be around 40 kHz. Therefore, $f_c=40$ kHz is used in our SSBAM based ABS. We investigate three methods [28], [29] of SSBAM signal generation, and they are listed as

- A. Truncated DSBAM
- B. Hilbert transform based SSBAM
- C. Weaver's method

The filtering operation constitutes the major signal processing block for each of these methods. As APOS and MPOS for filtering block are the same, we only show the MPOS values for different SSBAM generation methods. The SSBAM generation methods can be implemented as single-rate and multi-rate implementations. We investigate which mode (single-rate or multi-rate) is computationally more efficient for each of these methods. To quantify computational

workload, MPOS is computed for each method under both single-rate and multi-rate implementations. Based on the MPOS results, the method and the mode with the least MPOS is selected, and its SQNR analysis is performed. In the following sections, we investigate each of the SSBAM signal generation techniques for the ABS.

5.2 Truncated DSBAM

Conventional DSBAM has two sidebands: a LSB and a USB. SSBAM can be generated from DSBAM by removing either one of the two sidebands. This is achieved by using either a low-pass filter (LPF) or a high-pass filter (HPF). Consider a DSBAM signal with carrier frequency f_c . LSB and USB of DSBAM can be obtained by passing DSBAM signal through a LPF and a HPF, respectively. The output of LPF and HPF is referred to as LSBAM and USBAM, respectively. Figure 5.1(a) and (b) shows the block diagram of the LSBAM and USBAM, respectively, in floating-point implementation.

Figure 5.1 Single-rate (a) LSBAM and (b) USBAM for floating-point implementation.

For fixed-point implementation, a scaling factor is introduced to prevent overflow. Consider an N -bit fixed-point processor, and assume that the fixed-point algorithms use a $Q1.N-1$ format for data representation and arithmetic operations. For an N -bit fixed-point processor, $Q1.N-1$ arithmetic format provides maximum precision. This precision comes at the cost of dynamic range supported by $Q1.N-1$ format, which is restricted to $[-1, 1-2^{-(N-1)}]$. In order to prevent overflow or saturation at any stage of fixed-point truncated DSBAM (TDSBAM), the input is scaled by a factor of 0.5. The scaling factor of 0.5 can be merged with the modulation index m . The block diagram of single-rate LSBAM and USBAM for fixed-point implementation is shown in Fig. 5.2(a) and (b), respectively.

Figure 5.2 Single-rate (a) LSBAM and (b) USBAM for fixed-point implementation.

For both floating-point and fixed-point implementations, we need to design LPF and HPF to extract the desired frequency bands. To design LPF and HPF, we first define the following parameters

f_p : Pass-band edge of LPF and HPF in Hz,

f_s : Stop-band edge of LPF and HPF in Hz,

δ_p : Pass-band ripple for LPF and HPF in dB,

δ_s : Stop-band attenuation for LPF and HPF in dB.

In our design, we set $\delta_p = .01$ dB and $\delta_s = 40$ dB. In this section and subsequent sections, these values of δ_p and δ_s are used. To determine f_p and f_s for LPF and HPF, we make use of the fact that due to acoustic effects, frequencies below 1 kHz cannot be regenerated in air by ABS with sufficiently audible SPL [74]. Taking into account of this acoustic effect, minimum order LPF and HPF are designed using the specifications shown in Table 5.1. Equiripple FIR filter design method [75], [76] is used to design the minimum order LPF and HPF of length N . To quantify the computation workload of TDSBAM, we compute MPOS for LSBAM and USBAM. MPOS for the single-rate TDSBAM is given as

$$MPOS_{TDSBAM, SR} = N + 2. \quad (5.1)$$

Two multiplications are required for multiplying $x[n]$ with modulation index and ultrasonic carrier before passing through the filter. The MPOS results for single-rate TDSBAM are shown in Table 5.1.

Table 5.1 Filter specification for LSBAM and USBAM.

TDSBAM	f_p (kHz)	f_s (kHz)	N	$MPOS_{TDSBAM, SR}$
LSBAM	40	41	533	535
USBAM	40	39	521	523

Since TDSBAM is a simple process, its multi-rate implementation is not recommended. In the next section, we describe SSBAM generation using the Hilbert transform and investigate its computational workload performance.

5.3 SSBAM using Hilbert transformer

The second method of generating SSBAM signals is based on the Hilbert transformer (HT) [28]-[31]. We start our discussion with SSBAM generation using HT implemented in DP floating-point. Subsequently, we will introduce the scaling factor required for fixed-point implementation. The block diagram of SSBAM based on HT is shown in Fig. 5.3.

Figure 5.3 SSBAM using HT for floating-point implementation.

The HT based SSBAM has two paths, namely, inphase component generation path and quadrature component generation path. The quadrature component generation path uses the HT to generate the quadrature component $m\hat{x}[n]$ of the input signal $mx[n]$. $m\hat{x}[n]$ is 90° phase shifted with respect to $mx[n]$. The output of the block diagram in Fig. 5.3 can be mathematically expressed as

$$y[n] = (1 + mx[n])\cos[2\pi f_c n] + m\hat{x}[n]\sin[2\pi f_c n]. \quad (5.2)$$

For fixed-point implementation using Q1.N-1 arithmetic format, a scaling factor of $1/\sqrt{5}$ is used to avoid saturation or clipping during the processing. The block diagram of fixed-point SSBAM based on HT is shown in Fig. 5.4.

Figure 5.4 SSBAM using HT for fixed-point implementation.

Mathematically,

$$y[n] = \left(\frac{1}{\sqrt{5}} + \frac{mx[n]}{\sqrt{5}}\right) \cos[2\pi f_c n] + \frac{m\hat{x}[n]}{\sqrt{5}} \sin[2\pi f_c n]. \quad (5.3)$$

The HT block is the major processing element in SSBAM, and is the most computational intensive block. Computational workload performance of SSBAM, therefore, greatly depends on the computational workload of the HT block. The HT can be implemented as either finite-impulse response (FIR) filter [30] or IIR filter [31], [77]. Because of the linear-phase property of the FIR filters, FIR based HT is more commonly used than its IIR counterpart. In the next section, we shall discuss the FIR based HT for the SSBAM, and examine ways to reduce its computational workload.

5.3.1 Hilbert transformer with finite-impulse response

The frequency response of the HT with infinitely long impulse response is defined as [45]

$$H_\infty(e^{j\omega}) = \begin{cases} j, & -\pi < \omega < 0, \\ -j, & 0 < \omega < \pi. \end{cases} \quad (5.4)$$

The infinite length impulse response, $h_\infty[n]$ of HT is obtained by computing the inverse discrete-time Fourier transform of (5.4) and is given by [45]

$$h_\infty[n] = \frac{2 \sin^2[\pi n/2]}{\pi n}. \quad (5.5)$$

The infinitely long $h_\infty[n]$ should be truncated to generate a finite length sequence $h[n]$. $h[n]$ of length N_{HT} is given by

$$h[n] = \begin{cases} 0, & \text{for } n \text{ even,} \\ \frac{2}{\pi n}, & \text{for } n \text{ odd.} \end{cases} \quad (5.6)$$

FIR based HT of length N_{HT} causes a delay of $(N_{HT} - 1)/2$ samples in the quadrature path. To time align the inphase path with the quadrature path, a time delay of $(N_{HT} - 1)/2$ samples should be introduced in the inphase path, as shown in Fig. 5.5.

Figure 5.5 Time alignment for FIR-HT based fixed-point SSBAM.

If N_{HT} is even, fractional delay needs to be introduced in the system. To avoid fractional delay, we consider $h[n]$ of odd lengths only. The odd length impulse response $h[n]$ has anti-symmetric coefficients. An FIR filter of odd length with anti-symmetric coefficients is called the Type III FIR filter and is band-pass in nature [78]. From now onward, the term “HT” also implies that it is FIR based HT. For single-rate implementation, the HT runs at an input sampling frequency of f_{IN} . With band-pass performance, it is difficult to calculate Hilbert transform of signals at low frequencies and high frequencies (i.e. around $f_{IN}/2$). A practical work around is to take advantage of the acoustic properties of ABS described in Section 5.2. Due to acoustic effects, ABS cannot generate frequencies < 1 kHz with sufficient SPL [74]. We can therefore

safely use the frequency bands of 0 to 1 kHz and $(f_{IN}-2)/2$ to $f_{IN}/2$ as lower and upper transition bands, respectively for the band-pass HT. These lower and upper transition band values are used to determine the HT order in the subsequent sections. We start with the simplest case of single-rate FIR-HT based SSBAM.

5.3.2 FIR-HT based single-rate SSBAM

Single-rate SSBAM is the mode in which the HT and ultrasonic modulation run at the same sampling frequency. In our study, the sampling frequency, $f_{IN}=192$ kHz is used for the single-rate implementation. For designing a minimum order HT, we need to specify its lower transition band f_{lt} , upper transition band f_{ut} , pass-band ripple δ_p , and stop-band attenuation δ_s . As described in Section 5.3.1, at $f_{IN}=192$ kHz, f_{lt} extends from 0 to 1 kHz and f_{ut} extends from 95 kHz to 96 kHz. The normalized transition bandwidth for HT, ω_t is calculated as

$$\omega_t = \frac{2(f_{lt} + f_{ut})}{f_{IN}}. \quad (5.7)$$

The filter's length of HT, N_{HT} depends on the transition bandwidth, which in turn inversely depends on f_{IN} . With $\delta_p=.01$ dB and $\delta_s=-40$ dB, the number of taps for the minimum order FIR-HT, N_{HT} is 385. Considering, multiplication with modulation index $(m/\sqrt{5})$ and orthogonal carriers, MPOS for single-rate SSBAM using FIR-HT is stated as

$$MPOS_{FIRHT,SR} = N_{HT} + 3. \quad (5.8)$$

The $MPOS_{FIRHT,SR}$ value is 388, which is considerably large. Single-rate processing has simplicity of design but suffers from the disadvantage of high computational workload

requirement due mainly to the HT implementation. To overcome this disadvantage, multi-rate SSBAM design is evaluated in the next section.

5.3.3 FIR-HT based multi-rate SSBAM

Multi-rate processing is a standard way to reduce the computation workload of DSP algorithms. The theory of multi-rate processing is well developed and described in great details in [45]. In this section, we apply the concept of multi-rate processing to reduce the computational workload of HT based single-rate SSBAM. In HT based single-rate SSBAM, the same HT is used for both narrowband (example speech) and broadband (example audio) signals. As Hilbert transformation is a computationally expensive operation, it is advantageous to perform Hilbert transform at a lower sampling rate of f_{IN} to reduce computational workload. To generate SSBAM signal, the Hilbert transformed signal is up sampled by a factor of L , filtered by a low-pass interpolation filter (LPIF), and modulated with orthogonal carriers at higher sampling rate of f_{OUT} . The up sampling factor L is defined as

$$L = \frac{f_{OUT}}{f_{IN}}. \quad (5.9)$$

Here, HT operates at input sampling frequency f_{IN} , and ultrasonic modulation operates at output sampling frequency f_{OUT} . Figure 5.6 shows the block diagram of the multi-rate SSBAM.

Figure 5.6 FIR-HT based multi-rate SSBAM.

In total, $(1+N_{HT})$ multiplications are performed at sampling frequency f_{IN} , one multiplication is required for multiplying $x[n]$ with modulation index $(m/\sqrt{5})$; and N_{HT} multiplications are required by HT. These $(1+N_{HT})$ multiplications are carried out at a sampling frequency of f_{IN} . The inphase and quadrature components are up sampled and filtered by a LPIF, which requires $MPOS_{IF}$ MPOS. The outputs of LPIFs at inphase and quadrature paths are multiplied by inphase and quadrature carriers, respectively. Interpolation filtering and multiplication with inphase and the quadrature carriers is performed at sampling frequency f_{OUT} . MPOS for FIR-HT based multi-rate SSBAM is the summation of the MPOS of the individual operations and is calculated as

$$MPOS_{FIRHT,MR} = 2 + \frac{1+N_{HT}}{L} + 2MPOS_{IF}. \quad (5.10)$$

The bandwidths of audio and speech signals considered in our work are 20 kHz and 4 kHz, respectively. Parameters f_{IN} , f_{OUT} and L values used for audio and speech signal are shown in Table 5.2. These values will be used in the following descriptions.

Table 5.2 f_{IN} , f_{OUT} and L values for audio and speech signals.

Signal	Bandwidth (kHz)	f_{IN} (kHz)	f_{OUT} (kHz)	L
Audio	20	48	192	4
Speech	4	12	192	16

HT is designed using the procedure described in Section 5.3.2. Table 5.3 shows the values of f_{lt} , f_{ut} , ω_t , and N_{HT} for HT based multi-rate SSBAM.

Table 5.3 Specifications of FIR-HT based multi-rate SSBAM.

	f_{lt} (kHz)	f_{ut} (kHz)	ω_t	N_{HT}
Audio	0 to 1	23 to 24	.08	99
Speech	0 to 1	5 to 6	.33	25

When compared to speech, the HT designed for audio signal has a tighter specification and results in a higher order filter. Given the values stated in Table 5.3, we can now compute the MPOS of the multi-rate SSBAM based on (5.10) if $MPOS_{IF}$ is known. LPIF can be designed in a number of ways. Let's consider the case of audio signal. Audio signal requires interpolation by a factor of 4, which can be done in 2 ways. Figure 5.7(a) performs interpolation by a factor of 4 in a single-stage, whereas Fig. 5.7(b) performs the interpolation in 2 stages.

Figure 5.7 Multi-rate SSBAM with (a) single-stage and (b) multi-stage interpolation design for audio signal.

To select the optimal interpolation design from two interpolation designs shown in Fig. 5.7, multi-stage interpolation design is presented in the next section.

A Multi-stage interpolation design

In this section, multi-stage interpolation design is described for a general case. Using this method, MPOS values for different multi-stage interpolation designs are obtained, and the optimal design is identified. For multi-rate SSBAM, HT operates at a lower sampling frequency

f_{IN} and ultrasonic modulation is carried out at a higher sampling frequency f_{OUT} . To transit from f_{IN} to f_{OUT} , the output of HT is interpolated by a factor of L . Interpolation by factor of L using M stages, where

$$L = \prod_{i=1}^M L_i, \quad (5.11)$$

is shown in Fig. 5.8.

Figure 5.8 Interpolation by factor of L using M stages.

At each interpolation stage, an LPIF is required to remove unnecessary replicas generated by up sampling process. The optimal multi-stage interpolation design is discussed in detail in [79]-[81]. We briefly describe the method to obtain optimal design for multi-stage interpolation by a factor of L . For this, consider a band-limited signal extending from 0 to f_{sg} and sampled at sampling frequency of f_{IN} . The normalized bandwidth of the signal is

$$\omega_{sg} = \frac{2f_{sg}}{f_{IN}}. \quad (5.12)$$

Figure 5.9(a) shows the normalized spectrum of band-limited input signal.

(a)

(b)

(c)

(d)

(e)

Figure 5.9 Normalized signal spectrum (with reference to Nyquist frequency) for multi-stage interpolation design: (a) normalized spectrum of base-band signal, (b) normalized signal spectrum after interpolation by a factor of L_1 , (c) normalized signal spectrum at output of first interpolation stage, (d) normalized signal spectrum after interpolation by a factor of L_2 , and (e) normalized signal spectrum at output of second interpolation stage.

The increment in sampling rate from f_{IN} to Lf_{IN} can be carried out in single-stage or multiple stages. Single-stage interpolation is a special case of $M=1$. For simplicity of illustration, we first describe the interpolation by a factor of L using 2 stages, and then generalize the results for arbitrary number of interpolation stages. Suppose first stage interpolates by a factor of L_1 , and second stage interpolates by a factor of L_2 , such that $L=L_1L_2$. After first stage of interpolation, the spectrum is filtered by IF1, which has a normalized pass-band edge at

$$\omega_{p,IF1} = \frac{\omega_{sg}}{L_1}, \quad (5.13)$$

and has normalized transition bandwidth of

$$\omega_{t,IF1} = \frac{2(1-\omega_{sg})}{L_1}. \quad (5.14)$$

The normalized stop-band edge of IF1 $\omega_{s,IF1}$ can be calculated from $\omega_{p,IF1}$ and $\omega_{t,IF1}$ as

$$\omega_{s,IF1} = \omega_{p,IF1} + \omega_{t,IF1}. \quad (5.15)$$

Figure 5.9(b) shows the signal spectrum after interpolation by a factor of L_1 . The interpolation filter IF1 used to remove the unnecessary replicas after interpolation is also shown in Fig. 5.9(b). Figure 5.9(c) shows the frequency spectrum of signal after filtering by IF1. It can be observed that an additional normalized transition bandwidth of $g\omega_{t,IF1}$ is gained at first interpolation stage. The normalized additional transition bandwidth gained at first interpolation stage is given by

$$g\omega_{t,IF1} = 1 - \omega_{p,IF1} - \omega_{t,IF1}. \quad (5.16)$$

This normalized additional transition bandwidth gained at first interpolation stage contributes towards the transition bandwidth of the LPIF used after second stage of interpolation. In second

interpolation stage, the output of IF1 is up sampled by a factor of L_2 and filtered by LPIF IF2. IF2 has a normalized pass-band edge at

$$\omega_{p,IF2} = \frac{\omega_{p,IF1}}{L_2}. \quad (5.17)$$

The normalized transition bandwidth of IF2, $\omega_{t,IF2}$ is calculated from $\omega_{t,IF1}$ and $g\omega_{t,IF1}$, and is given by

$$\omega_{t,IF2} = \frac{2(\omega_{t,IF1} + g\omega_{t,IF1})}{L_2}. \quad (5.18)$$

The normalized stop-band edge of IF2, $\omega_{s,IF2}$ is calculated from $\omega_{p,IF2}$ and $\omega_{t,IF2}$ as

$$\omega_{s,IF2} = \omega_{p,IF2} + \omega_{t,IF2}. \quad (5.19)$$

The frequency spectrum of signal after interpolation by a factor of L_2 , and the interpolation filter IF2 used for filtering are shown in Fig. 5.9(d). Similar to $g\omega_{t,IF1}$, $g\omega_{t,IF2}$ is the normalized addition transition bandwidth gained at the second interpolation stage and is given by

$$g\omega_{t,IF2} = 1 - \omega_{p,IF2} - \omega_{t,IF2}. \quad (5.20)$$

Figure 5.9(e) shows the IF2 filtered signal spectrum and additional normalized transition bandwidth gained at second interpolation stage. In general, for an M -stage interpolation design, pass-band edge and transition bandwidth of LPIF at each stage can be determined. So, we move on to a generalized case, where interpolation by a factor of L is performed using M stages. Suppose L_i is the interpolation factor at the i^{th} interpolation stage ($1 \leq i \leq M$). For the first interpolation stage, $\omega_{p,IF1}$, $\omega_{t,IF1}$, and $\omega_{s,IF1}$ are obtained from ω_{sg} using (5.13), (5.14) and

(5.15), respectively. $g\omega_{t,IF1}$ is obtained from $\omega_{p,IF1}$ and $\omega_{t,IF1}$ using (5.16). For subsequent interpolation stages, the normalized pass-band edge at the i^{th} interpolation stage is calculated as

$$\omega_{p,IFi} = \frac{\omega_{p,IF(i-1)}}{L_i}, \quad \text{for } i > 1. \quad (5.21)$$

Similarly, $\omega_{t,IFi}$ is expressed as

$$\omega_{t,IFi} = \frac{2(\omega_{t,IF(i-1)} + g\omega_{t,IF(i-1)})}{L_i}, \quad \text{for } i > 1, \quad (5.22)$$

where $\omega_{s,IFi}$ and $g\omega_{t,IFi}$ are determined as

$$\omega_{s,IFi} = \omega_{p,IFi} + \omega_{t,IFi}, \quad \text{for } i > 1. \quad (5.23)$$

$$g\omega_{t,IFi} = 1 - \omega_{p,IFi} - \omega_{t,IFi}, \quad \text{for } i > 1. \quad (5.24)$$

Using (5.13)-(5.16), (5.21)-(5.24), pass-band edge, stop-band edge and the transition bandwidth of LPIF at each interpolation stage can be determined. Next, we describe the conditions for obtaining pass-band ripple and stop-band attenuation values for LPIF at each interpolation stage.

Suppose δ_p and δ_s are the desired pass-band ripple (in dB) and stop-band attenuation (in dB), respectively for a single-stage design for interpolation by a factor of L . If the same factor of L interpolation is realized using M stages, then each LPIF should have a pass-band ripple of δ_p/M and a stop-band attenuation of δ_s . Given the pass-band edge, transition bandwidth, pass-band ripple and stop-band attenuation for each LPIF, minimum order LPIF can be designed for each interpolation stage using equiripple FIR filter design method. If L_i is the up sampling factor and N_i is the filter order of the LPIF at i^{th} interpolation stage, MPOS of the multi-stage interpolator [82] is calculated as

$$MPOS_{IF} = \sum_{i=1}^M \frac{N_i}{\prod_{j=i+1}^M L_j}. \quad (5.25)$$

Among different possible configurations, the multi-stage design with minimum $MPOS_{IF}$ is the optimal design. The computational workload can be further reduced by polyphase implementation of LPIFs. The polyphase decomposition is well-known and studied in [45], [78], [82], and is described in the next section.

B Design of multi-stage polyphase interpolator

Computationally efficient interpolation structures employing LPIFs can be derived by applying polyphase decomposition to their corresponding transfer functions [45]. Polyphase decomposition makes use of the fact that for a factor of L interpolation, $(L-1)$ out of L samples are ignored. In other words, for an LPIF of length $N=nL$, where n is an integer, $n(L-1)$ samples are always 0. Therefore, we need to perform only L multiplications per sample. The transfer function of an LPIF with N taps is given as

$$H(z) = h(0) + h(1)z^{-1} + \dots + h(N-1)z^{-(N-1)}, \quad (5.26)$$

where $h(0), h(1) \dots h(N-1)$ are the coefficients of the filter. Using polyphase decomposition, L sub-filter structures can be obtained from a single LPIF for a factor of L interpolation. These sub-filters have the following transfer functions:

$$\begin{aligned}
H_1(z) &= h(0) + h(L)z^{-1} + \dots + h(N-L)z^{-(L-1)}, \\
H_2(z) &= h(1) + h(L+1)z^{-1} + \dots + h(N-L+1)z^{-(L-1)}, \\
&\vdots \\
&\vdots \\
&\vdots \\
H_L(z) &= h(L-1) + h(2L-1)z^{-1} + \dots + h(N-1)z^{-(L-1)}.
\end{aligned} \tag{5.27}$$

Figure 5.10 shows the polyphase implementation of LPIF for a factor of L interpolation using sub-filters.

Figure 5.10 Polyphase implementation of factor of L LPIF using sub-filters.

For polyphase implementation of the multi-stage interpolator, the filter order of LPIF at each stage is determined as discussed in Section 5.3.3.A. MPOS for the multi-stage interpolator with polyphase implementation is calculated as

$$MPOS_{IF, PP} = \sum_{i=1}^M \frac{N_i}{\prod_{j=i}^M L_j}. \quad (5.28)$$

Using multi-stage interpolation with polyphase implementation reduces computational workload significantly. The computational workload performance of various configurations of multi-stage interpolator is compared in the next section.

C Computational workload analysis of interpolation filters

In this section, we investigate the computational workload performance of various configurations of multi-stage interpolator that are possible for multi-rate SSBAM. The comparison is used to select the optimal interpolation design. Table 5.4 shows different possible configurations for speech and audio signals. It shows all the possible combinations when interpolation factor L is spread across M interpolation stages for all possible permutations.

Table 5.4 Different configurations of multi-stage interpolator for audio and speech signal.

Multi-rate SSBAM	L	M	Splitting of interpolation factor L across M interpolation stages				Configuration
			L_1	L_2	L_3	L_4	
Audio	4	1	4	N.A.	N.A.	N.A.	A
		2	2	2	N.A.	N.A.	B
Speech	16	1	16	N.A.	N.A.	N.A.	C
		2	2	8	N.A.	N.A.	D
		2	4	4	N.A.	N.A.	E
		2	8	2	N.A.	N.A.	F
		3	2	2	4	N.A.	G
		3	2	4	2	N.A.	H
		3	4	2	2	N.A.	I
		4	2	2	2	2	J

L_1, L_2, L_3, L_4 – Interpolation factors at first, second, third and fourth interpolation stages, respectively.

For each multi-stage configuration, the minimum order LPIFs for each interpolation stage are obtained using equiripple filter design method. To calculate the minimum order of the LPIF at each interpolation stage (N_i), we need to specify δ_p and δ_s values. For our study, we set $\delta_p = 0.01$ dB and $\delta_s = -40$ dB. $MPOS_{IF}$ and $MPOS_{IF,PP}$ are calculated from L_i and N_i using (5.25) and (5.28), respectively. Table 5.5 compares the computational workload for different configurations with polyphase and non-polyphase implementation.

Table 5.5 Performance comparison for various configurations of multi-stage interpolator.

Multi-rate SSBAM	Configuration	N_i				MPOS	
		N_1	N_2	N_3	N_4	NPP	PP
Audio	A	72	N.A.	N.A.	N.A.	72.00	18.00
	B	38	10	N.A.	N.A.	29.00	14.50
Speech	C	144	N.A.	N.A.	N.A.	144.00	9.00
	F	80	6	N.A.	N.A.	46.00	8.00
	J	22	10	8	6	15.25	7.63
	I	40	8	6	N.A.	20.00	7.50
	H	22	20	6	N.A.	18.75	6.88
	G	22	10	16	N.A.	21.25	6.63
	E	40	16	N.A.	N.A.	26.00	6.50
	D	20	40	N.A.	N.A.	42.50	6.25

NPP – Non-polyphase implementation

PP – Polyphase implementation

For audio signals, Configuration B is computationally more efficient than Configuration A for both polyphase and non-polyphase implementations. For speech signal, Configuration J is computationally most efficient for non-polyphase implementation, whereas Configuration D is computationally most efficient for polyphase implementation. In general, computationally least expensive interpolators are obtained by polyphase implementation, for both speech and audio signals.

D Computational workload analysis of multi-rate SSBAM

For the multi-rate SSBAM, the total MPOS is the combination of MPOS for HT and LPIFs.

Therefore, the MPOS for multi-rate SSBAM, which uses multi-stage interpolator with non-polyphase implementation, is calculated as

$$MPOS_{FIRHT,MR} = 2 + \frac{1 + N_{HT}}{L} + 2 \sum_{i=1}^M \frac{N_i}{\underbrace{\prod_{j=i+1}^M L_j}_{MPOS_{IF}}} . \quad (5.29)$$

Similarly, MPOS for multi-rate SSBAM, which uses multi-stage interpolator with polyphase implementation, is calculated as

$$MPOS_{FIRHT,MR,PP} = 2 + \frac{1 + N_{HT}}{L} + 2 \sum_{i=1}^M \frac{N_i}{\underbrace{\prod_{j=i}^M L_j}_{MPOS_{IF,PP}}} . \quad (5.30)$$

Table 5.6 shows the comparison of MPOS for FIR-HT based multi-rate SSBAM, which uses multi-stage interpolator with non-polyphase and polyphase implementations.

Table 5.6 Performance comparison for FIR-HT based multi-rate SSBAM with multi-stage interpolator.

Multi-rate SSBAM	N_{HT}, L	Configuration	NPP		PP	
			$MPOS_{IF}$	$MPOS_{FIRHT, MR}$	$MPOS_{IF, PP}$	$MPOS_{FIRHT, MR, PP}$
Audio	99, 4	A	72.00	171.00	18.00	63.00
		B	29.00	85.00	14.50	56.00
Speech	25, 16	C	144.00	291.63	9.00	21.63
		F	46.00	95.63	8.00	19.63
		J	15.25	34.13	7.63	18.89
		I	20.00	43.63	7.50	18.63
		H	18.75	41.13	6.88	17.39
		G	21.25	46.13	6.63	16.89
		E	26.00	55.63	6.50	16.63
		D	42.50	88.63	6.25	16.13

MPOS values for polyphase implementation is lower than non-polyphase implementation for all configurations. For audio signal, Configuration B with polyphase implementation has the least MPOS. Similarly, Configuration D with polyphase implementation has the least MPOS for speech signals. As polyphase implementation is computationally less expensive than non-polyphase implementation, only polyphase implementation is used for comparison purpose in the next section.

5.3.4 Performance comparison of FIR-HT based single-rate and multi-rate SSBAM

In this section, we compare the computational workload requirements of single-rate and multi-rate SSBAM. For single-rate SSBAM, MPOS depends only on the HT order. But for multi-rate SSBAM, MPOS depends on the HT order, as well as the order of LPIFs of the multi-stage interpolator. Table 5.7 compares MPOS for single-rate SSBAM, $MPOS_{FIRHT, SR}$, and multi-rate SSBAM (with polyphase implementation of Configuration D), $MPOS_{FIRHT, MR}$.

Table 5.7 Computational workload comparison for FIR-HT based single-rate and multi-rate SSBAM.

FIR-HT based SSBAM	MPOS	
	Audio	Speech
Single-rate	388.00	388.00
Multi-rate (PP)	56.00	16.13

Significant reduction of 85.57% and 95.84% in MPOS are achieved for audio and speech, respectively by using multi-rate SSBAM (with multi-stage interpolation and polyphase implementation) as compared to single-rate SSBAM. With these results, we can conclude that the multi-rate SSBAM, which uses multi-stage interpolator with polyphase implementation, reduces computational workload significantly as compared to single-rate implementation. The present section uses a FIR filter based HTs to generate SSBAM signal. In order to reduce computational workload, an IIR filter based SSBAM is carried out in the next section.

5.3.5 SSBAM using IIR filters

IIR filters are computationally efficient structures with a feedback loop. The IIR-based all-pass filters can be used to alter the phase of the signal, keeping its magnitude unchanged. There are many approaches [83]-[85] in implementing an IIR-based all-pass filter. These approaches describe steps to design two all-pass IIR filters, whose outputs are 90° phase apart, as shown in Fig. 5.11.

Figure 5.11 SSBAM using all-pass IIR filters.

The first all-pass filter introduces phase distortion to the input $x[n]$ and generates $y_i[n]$. The second all-pass filter generates $y_q[n]$, which is 90° out of phase with respect to $y_i[n]$. Unlike HT, the all-pass filter approach generates a pair of output which is 90° phase apart from each other, but none of the output is 90° phase apart from the input. For audio applications, phase distortion is not desirable; hence this approach is not preferred for ABS. In the next section, the third method of generating SSBAM signal is described and analyzed.

5.4 SSBAM using Weaver's method

The Weaver's method is the third method of generating SSBAM signals described in this chapter. It was first proposed by Donal K. Weaver in 1956 [29], and it is used to generate and detect SSBAM signals. In contrast to the SSBAM generation methods introduced earlier, Weaver's method does not require sharp cutoff filters or wide-band 90° phase-difference networks. The block diagram of the Weaver's method is shown in Fig. 5.12.

Figure 5.12 Weaver's method for generation of SSBAM signals.

The Weaver's method assumes that the input signal is a band-pass signal and bandwidth does not extend to 0 Hz. We therefore, introduce a HPF with cut-off frequency of $f_{p,HPF}$ before the first DSBAM stage. The block diagram of the Weaver's method with the HPF is shown in Fig. 5.13.

Figure 5.13 Weaver's method for ABS.

No scaling is required to prevent saturation and overflow for fixed-point implementation using Q1.N-1 arithmetic format and the block diagram of Fig. 5.13 can be used to represent fixed-point SSBAM using Weaver's method.

The two DSBAM stages of Weaver's method require the values of f_{DSBAM1} and f_{DSBAM2} . The values of f_{DSBAM1} and f_{DSBAM2} depend on the bandwidth of input signal, $x[n]$, pass-band edge of HPF, $f_{p, \text{HPF}}$ and ultrasonic carrier frequency, f_c . Let's assume that the bandwidth of input signal $x[n]$, extends from 0 to $(f_{p, \text{HPF}} + B)$. The input signal spectrum is shown in Fig. 5.14(a). $x[n]$ is passed through a HPF with a stop-band edge at $f_{p, \text{HPF}}$ and length N_{HPF} . $x_{11}[n]$ is the output signal of HPF and can be expressed as a summation of sinusoidal terms as

$$x_{11}[n] = m \sum_{i=1}^N a_i \cos[2\pi f_i n + \phi_i]. \quad (5.31)$$

The bandwidth of $x_{11}[n]$ extends from $f_{p, \text{HPF}}$ to $(f_{p, \text{HPF}} + B)$ as shown in Fig. 5.14(b).

Modulating frequency of the first DSBAM modulator, f_{DSBAM1} is related to $f_{p, \text{HPF}}$ and B as

$$f_{\text{DSBAM1}} = f_{p, \text{HPF}} + \frac{B}{2}. \quad (5.32)$$

$x_{12}[n]$ and $x_{22}[n]$ are outputs of the first DSBAM modulator at first and second paths, respectively. $x_{12}[n]$ and $x_{22}[n]$ can be expressed as

$$x_{12}[n] = 2 \left\{ \frac{m}{2} \sum_{i=1}^N a_i \cos[2\pi(f_i - f_{DSBAM1})n + \phi_i] + \frac{m}{2} \sum_{i=1}^N a_i \cos[2\pi(f_i + f_{DSBAM1})n + \phi_i] \right\}, \quad (5.33)$$

$$x_{22}[n] = 2 \left\{ -\frac{m}{2} \sum_{i=1}^N a_i \sin[2\pi(f_i - f_{DSBAM1})n + \phi_i] + \frac{m}{2} \sum_{i=1}^N a_i \sin[2\pi(f_i + f_{DSBAM1})n + \phi_i] \right\}. \quad (5.34)$$

The coefficient 2 is used for amplification and can be considered a property of the first DSBAM modulator. Signal spectrum at the output of the first DSBAM modulator (for both first and second paths) is shown in Fig. 5.14(c). The lower frequency components of modulated signal, ranging from 0 to $B/2$ are extracted using an LPF. The pass-band edge of this LPF is at $B/2$ and transition region extends from $B/2$ to $(2f_{DSBAM1} - B/2)$. LPF with same specifications is used for both paths. The length of LPFs (for both first and second paths) is denoted by N_{LPF} . $x_{13}[n]$ and $x_{23}[n]$ are output of LPFs at the first and second path, respectively, and can be expressed as

$$x_{13}[n] = m \sum_{i=1}^N a_i \cos[2\pi(f_i - f_{DSBAM1})n + \phi_i], \quad (5.35)$$

$$x_{23}[n] = -m \sum_{i=1}^N a_i \sin[2\pi(f_i - f_{DSBAM1})n + \phi_i]. \quad (5.36)$$

The signal spectrum at the LPF output is shown in Fig. 5.14(d). The output of LPF is modulated using second DSBAM modulator. The modulating frequency of the second DSBAM modulator is f_{DSBAM2} , which depends on the modulating frequency of the first DSBAM modulator f_{DSBAM1} and the ultrasonic carrier frequency f_c as

$$f_{DSBAM2} = f_c + f_{DSBAM1}. \quad (5.37)$$

Figure 5.14 Signal spectra for Weaver's method: (a) spectrum of input signal, (b) signal spectrum at output of HPF, (c) signal spectrum after first DSBAM, (d) signal spectrum at output of LPF and (e) spectrum of output signal.

$x_{14}[n]$ and $x_{24}[n]$ are outputs of second DSBAM modulator at the first and second path, respectively. They can be expressed as

$$x_{14}[n] = \frac{m}{2} \sum_{i=1}^N a_i \cos[2\pi(f_{DSBAM2} + f_i - f_{DSBAM1})n + \phi_i] + \frac{m}{2} \sum_{i=1}^N a_i \cos[2\pi(f_{DSBAM2} - f_i + f_{DSBAM1})n + \phi_i], \quad (5.38)$$

$$x_{24}[n] = \frac{m}{2} \sum_{i=1}^N a_i \cos[2\pi(f_{DSBAM2} + f_i - f_{DSBAM1})n + \phi_i] - \frac{m}{2} \sum_{i=1}^N a_i \cos[2\pi(f_{DSBAM2} - f_i + f_{DSBAM1})n + \phi_i]. \quad (5.39)$$

The outputs of the second DSBAM modulator from first and second paths are added to generate the output of the Weaver's method $y[n]$ as

$$y[n] = m \sum_{i=1}^N a_i \cos[2\pi(f_{DSBAM2} + f_i - f_{DSBAM1})n + \phi_i], \quad (5.40)$$

and is shown in Fig. 5.14(e).

We have described the Weaver's method and relationship between its various parameters. The Weaver's method can now be used to generate SSBAM signals for ABS. For single-rate Weaver's method, we use $f_{IN} = f_{OUT} = 192$ kHz. Taking into account of the acoustic properties of the ABS, the filter design parameters and other parameters for single-rate Weaver's method are stated in Table 5.8.

Table 5.8 Parameters for single-rate Weaver's method.

Parameters	Single-rate	
	Speech	Audio
B (kHz)	3	19
f_{DSBAM1} (kHz)	2.5	10.5
f_{DSBAM2} (kHz)	42.5	50.5
$f_{\text{p, HPF}}$ (kHz)	1	1
$f_{\text{s, HPF}}$ (kHz)	.5	.5
$f_{\text{p, LPF}}$ (kHz)	1.5	9.5
$f_{\text{s, LPF}}$ (kHz)	3.5	11.5
N_{HPF}	1,033	1,033
N_{LPF}	267	267
$\text{MPOS}_{\text{Weaver, SR}}$	1,572	1,572

The MPOS for the single-rate Weaver's method $\text{MPOS}_{\text{Weaver, SR}}$ is expressed as

$$\text{MPOS}_{\text{Weaver, SR}} = 1 + N_{\text{HPF}} + 2(N_{\text{LPF}} + 2). \quad (5.41)$$

Based on the parameters of Table 5.8, $\text{MPOS}_{\text{Weaver, SR}}$ is 1,572 which is quite high. To reduce the MPOS for the Weaver's method, its multi-rate implementation is described in the next section.

5.4.1 Multi-rate SSBAM using Weaver's method

The computational workload of the single-rate SSBAM using Weaver's method can be reduced by multi-rate implementation. For this, single-rate Weaver's method is split into multiple stages, with each stage operating at different sampling rates. In our study, we split single-rate Weaver's method into 3 stages. The first stage is a HPF, the second stage performs first DSBAM followed by the LPF, and the third stage performs second DSBAM. The block diagram of the 3-stage multi-rate Weaver's method is shown in Fig. 5.15(a). A polyphase implementation of the multi-rate Weaver's method can be implemented to further reduce the computational workload. The block diagram of the polyphase implementation of multi-rate Weaver's method is shown in Fig.

5.15(b). The sampling frequencies at the first stage, f_{IN} are 12 kHz and 48 kHz for speech and audio signals, respectively. The sampling frequency at the third stage, f_{OUT} is 192 kHz for both speech and audio signals. To determine sampling frequency at the second stage, f_{INT} , we need to determine the interpolation factor, L_1 . For this, consider the signal spectrum of Fig. 5.14(b) and (c). The maximum frequency components at the first and second stages are $f_{p,HPF} + B$ and $2f_{DSBAM1} + B/2$, respectively. Now, L_1 can be determined as

$$\begin{aligned}
 L_1 &= \left\lceil \frac{2f_{DSBAM1} + \frac{B}{2}}{f_{p,HPF} + B} \right\rceil \\
 &= \left\lceil \frac{2f_{p,HPF} + \frac{3B}{2}}{f_{p,HPF} + B} \right\rceil \\
 &= \left\lceil 2 - \frac{B}{2(f_{p,HPF} + B)} \right\rceil \\
 &= \left\lceil 2 - \frac{1}{2\left(1 + \frac{f_{p,HPF}}{B}\right)} \right\rceil.
 \end{aligned} \tag{5.42}$$

The sampling frequency at the second stage, f_{INT} is determined as

$$f_{INT} = L_1 \times f_{IN}. \tag{5.43}$$

L_2 is the up sampling factor for the second stage, and is determined as

$$L_2 = \left\lceil \frac{f_{OUT}}{f_{INT}} \right\rceil. \tag{5.44}$$

Once L_1 , L_2 and f_{INT} are determined, interpolation filters IFX and IFY can be designed. The parameters B , f_{DSBAM1} and f_{DSBAM2} for multi-rate implementation are the same as single-rate implementation and can be referred from Table 5.8. Table 5.9 shows values of f_{IN} , f_{INT} , f_{OUT} , L_1 and L_2 for non-polyphase and polyphase implementation of the multi-rate Weaver's method.

Table 5.9 Parameters for multi-rate Weaver's method.

Parameters	Multi-rate Weaver's method	
	Speech	Audio
f_{IN} (kHz)	12	48
f_{INT} (kHz)	24	96
f_{OUT} (kHz)	192	192
L_1	2	2
L_2	8	2

The parameters chosen in Table 5.9 are used to design HPF, LPF, IFX and IFY for the multi-rate Weaver's method. It should be noted that $L_1=L_2=2$ are used for audio signal. This implies that multi-stage interpolation design is not possible for audio signals. However, for speech signals, $L_1=2$ and $L_2=8$, and multi-stage interpolation is possible while transiting from 2nd to 3rd stage. Our simulations show that the single-stage interpolator has the minimum MPOS. Table 5.10 shows the specifications, operating frequencies and filter lengths for HPF, LPF, IFX and IFY.

Table 5.10 Filter specification and MPOS values for multi-rate Weaver's method.

	Filters	f_p (kHz)	f_s (kHz)	Operating frequency (kHz)		N	MPOS	
				NPP	PP		NPP	PP
Speech	HPF	1	.50	12	12	69	N.A.	N.A.
	IFX	4	8	24	12	18	2.25	1.13
	LPF	1.50	3.5	24	24	34	N.A.	N.A.
	IFY	1.50	22.5	192	24	32	32.00	4.00
Total MPOS for speech							83.63	25.38
Audio	HPF	1	.50	48	48	273	N.A.	N.A.
	IFX	20	28	96	48	36	18.00	9.00
	LPF	9.50	11.50	96	96	134	N.A.	N.A.
	IFY	9.50	86.50	192	96	8	8.00	4.00
Total MPOS for audio							257.50	231.50

The MPOS for IFX and IFY are calculated from N_{IFX} and N_{IFY} for non-polyphase implementation, as shown

$$MPOS_{IFX} = \frac{N_{IFX}}{L_2}, \quad (5.45)$$

$$MPOS_{IFY} = N_{IFY}. \quad (5.46)$$

For polyphase implementation, MPOS for IFX and IFY are calculated as

$$MPOS_{IFX, PP} = \frac{N_{IFX}}{L_1 L_2}, \quad (5.47)$$

$$MPOS_{IFY, PP} = \frac{N_{IFY}}{L_2}. \quad (5.48)$$

As such, the MPOS for the multi-rate Weaver's method is calculated as follows

$$MPOS_{Weaver, MR} = \frac{1 + N_{HPF}}{L_1 L_2} + 2 \left(MPOS_{IFX} + MPOS_{IFY} + \frac{1 + N_{LPF}}{L_2} + 1 \right). \quad (5.49)$$

For polyphase implementation, MPOS is expressed as

$$MPOS_{Weaver,MR,PP} = \frac{1+N_{HPF}}{L_1L_2} + 2 \left(MPOS_{IFX,PP} + MPOS_{IFY,PP} + \frac{1+N_{LPF}}{L_2} + 1 \right). \quad (5.50)$$

The MPOS for the multi-rate Weaver's method for non-polyphase and polyphase implementation is calculated using (5.49) and (5.50), respectively and is shown in Table 5.10. In the next section, MPOS for the single-rate and multi-rate Weaver's method (with polyphase implementation) are examined.

5.4.2 Performance comparison of single-rate and multi-rate Weaver's method

The MPOS result for the single-rate Weaver's method, $MPOS_{Weaver,SR}$ and the multi-rate Weaver's method, $MPOS_{Weaver,MR}$ are compared for both speech and audio signals in Table 5.11.

Table 5.11 MPOS comparison for single-rate and multi-rate Weaver's method.

Weaver's method	MPOS	
	Speech	Audio
Single-rate	1,572.00	1,572.00
Multi-rate (PP)	25.38	231.50

It can be observed from Table 5.11 that MPOS is reduced significantly when the multi-rate Weaver's method with polyphase implementation is used. Therefore, multi-rate implementation of the Weaver's method is used for SSBAM signal generation. So far, we have discussed three methods of generating SSBAM signals. We will next select the computationally most efficient method among these three methods.

5.5 Performance comparison of SSBAM generation methods

In this section, we compare the computational workload of the three SSBAM generation methods described previously. SSBAM using FIR-HT and the Weaver's method can be implemented either in single-rate or multi-rate. As shown in Sections 5.3.4 and 5.4.2, their multi-rate implementations are computationally more efficient than single-rate implementations. For multi-rate implementation, several interpolator configurations are realizable, and each configuration can be implemented as polyphase or non-polyphase. The polyphase implementation of interpolator configuration with the minimum MPOS is selected for both methods. However for TDSBAM, multi-rate implementation increases computational workload and design complexity. Due to its simplicity in design and reduced computational workload, single-rate TDSBAM is preferred. The MPOS scores for SSBAM based on TDSBAM, HT, and the Weaver's method are shown in Table 5.12.

Table 5.12 Computational workload comparison of SSBAM generation methods.

Method	Single-rate		Multi-rate	
	Speech	Audio	Speech	Audio
TDSBAM (USBAM)	523	523	N.A.	N.A.
HT	388	388	16.13	56.00
Weaver's method	1,572	1,572	25.38	231.50

It can be observed that the HT based SSBAM has the least computational workload among the three methods for single-rate, as well as multi-rate implementation. We, therefore, investigate SQNR performance of the HT based SSBAM in detail. The SQNR analysis is performed as described in Section 2.3.1. For the FIR-HT based multi-rate SSBAM, which uses multi-stage interpolator with polyphase implementation, floating-point and fixed-point HT and LPIFs are used to generate floating-point and fixed-point outputs, respectively. Table 5.13 shows the

SQNR values for 8-bit, 16-bit and 32-bit fixed-point processors for both audio and speech signals. It is desirable to compare the computational workload and SQNR performance of different configurations (defined in Table 5.4) of multi-stage interpolator (with polyphase implementation) against the single-rate implementation. Table 5.13 shows the computational workload and SQNR performance comparison of HT based on single-rate and multi-rate SSBAM.

Table 5.13 MPOS and SQNR values for 8-bit, 16-bit and 32-bit fixed-point processors for audio and speech signals.

Configuration	MPOS	SQNR		
		8-bit	16-bit	32-bit
Single-rate SSBAM	388.00	36.00	84.34	180.14
A	63.00	30.52	76.60	175.30
B	56.00	34.18	81.60	178.08
C	21.63	21.05	67.92	164.57
F	19.63	28.15	74.50	171.23
J	18.89	31.85	82.02	176.53
I	18.63	30.18	82.97	177.85
H	17.39	30.22	78.21	174.49
G	16.89	30.85	78.79	176.00
E	16.63	30.01	79.57	176.46
D	16.13	26.70	75.69	171.32

Comparing computational workload versus SQNR performance for audio signals, Configuration B provides the best computational workload performance but the best SQNR performance is provided by the single-rate SSBAM (SR-SSBAM). This observation suggests that design with the best computational performance may not necessarily provide the best SQNR performance, and vice-versa. The SR-SSBAM has the highest SQNR and MPOS for all fixed-point processors. However, the reduction in MPOS for multi-rate implementation is significant enough to outweigh the marginal increment of SQNRs achieved by single-rate implementation.

Comparing different configurations of multi-rate SSBAM for speech signal, Configuration D provides the best computational workload performance; but the best SQNR performance is provided by Configuration J for an 8-bit fixed-point processor, and Configuration I for 16-bit and 32-bit fixed-point processors. Generally, no straightforward relationship is observed between computational workload and SQNR performance. Making a design decision depends on processor specifications and SQNR requirement at output. For example, Configuration E is the best choice for a slower processor, and Configuration I or J is more suitable for a faster processor. Configuration E has low computational requirements and provides relatively good SQNR performance. Configurations I and J have moderate computational requirements and provide the best SQNR performance for speech applications. If MIPS is not an issue, then the simple design and highest SQNR performance makes SR-SSBAM the most preferred choice. In the next section, we present processing performance for a real-time implementation of the HT based on single-rate and multi-rate SSBAM.

5.6 Real-time implementation

In this section, we describe a real-time implementation of the most computationally efficient SSBAM signal generation method. Based on the above analysis, single-rate and multi-rate (having multi-stage interpolator with polyphase implementation) SSBAM is implemented on the C5505 eZDSP USB Stick, which is a 16-bit fixed-point DSP development board. As shown in Section 5.5, Configurations B and D are computationally most efficient for speech and audio signals, respectively and are implemented in the DSP processor. Table 5.14 shows the MIPS results and compares computational workload of single-rate and multi-rate implementation of SSBAM for audio and speech signals.

Table 5.14 MIPS results for 16-bit fixed-point implementation of FIR-HT based SSBAM.

Implementation	SSBAM using FIR-HT	MIPS	Onchip memory (kBytes)
Single-rate	SR-SSBAM	82.46	42
Multi-rate (PP)	B	29.34	46
	D	18.70	36

Table 5.13 shows that Configuration B and D achieves MPOS reduction of 85.57 % and 95.84%, respectively compared to single-rate implementation. But for real-time implementation, the MIPS reductions are 64.42% for Configuration B, and 77.32% for configuration D. This mismatch is due to the complexities introduced by the code structure of the multi-rate SSBAM. Even though the reduction in MIPS is not exactly the same as the reduction in MPOS, but MPOS correlates well with MIPS.

5.7 Conclusion

This chapter analyzed three SSBAM signal generation methods and compared them in terms of computational workload and SQNR performance. Single-rate and multi-rate implementation of the three methods were also investigated. Multi-stage interpolation design and polyphase interpolator structures are used to reduce the MPOS of multi-rate implementation. Among the three methods, FIR-HT based SSBAM has the least MPOS value for both single-rate and multi-rate implementation. In addition, FIR-HT based multi-rate SSBAM (with multi-stage interpolation design and polyphase interpolators) outperforms other methods in terms of MPOS by a significant margin. It is therefore concluded that this approach is computationally most efficient and should be used for real-time implementation. A real-time implementation of FIR-HT based multi-rate SSBAM was also carried out on fixed-point DSP processor.

Chapter 6

Conclusions and future works

6.1 Conclusions

In this thesis, we investigated several proprietary modulation schemes used in the ABS. Different algorithms to implement each modulation scheme are evaluated for an N -bit fixed-point DSP processor. The algorithm with the maximum SQNR and the minimum computational workload is selected for each modulation scheme, and its real-time implementation is performed on a 16-bit fixed-point DSP processor (photographs of the ABS with modulation schemes implemented on a 16-bit fixed-point DSP processor are shown in Appendix D). We performed the distortion analysis for fixed-point modulation schemes by simulating the selected algorithm for each modulation scheme on a 16-bit fixed-point processor using MATLAB. The model given in [33] and described in Section 2.1.5 is used to calculate the THD and IMD for different modulation schemes. As the THD and IMD values of modulation schemes vary with transducer bandwidth, simulations are performed for transducers with bandwidth of 6 kHz and 12 kHz. Modulation index of 0.70 is used for THD and IMD calculations. Table 6.1 shows the THD (%) for modulation schemes simulated on a 16-bit fixed-point processor. The THD (%) is computed using a single tone input of 1 kHz. The computational workload and memory requirement for different modulation schemes are also shown in Table 6.1.

Table 6.1 Computational workload and THD comparison for modulation schemes simulated and implemented on 16-bit fixed-point processor.

Modulation scheme (Algorithm)	Simulation		Implementation		THD (%)	
	APOS, MPOS	Memory (Bytes)	MIPS	Onchip memory (kBytes)	$f_{\text{trans}}=6$ kHz	$f_{\text{trans}}=12$ kHz
DSBAM (Mixed Q format implementation)	1, 2	2	6.40	15	33.04	33.04
MAM1 (Mixed Q format implementation)	3, 4	4	8.90	34	3.35	3.03
MAM2 (Adaptive MAM with mixed Q format implementation)	$\leq 4, \leq 5$	6	9.79	35	2.59	.25
MAM3 (Adaptive MAM with mixed Q format implementation)	$\leq 5, \leq 6$	8	9.79	35	2.58	.03
SRAM ($\text{DNR}_T(n)$: $[\text{DNR}_T(1)]_C$)	3, 3	288	11.21	38	6.91	.57
SSBAM (FIR-HT with polyphase implementation: Configuration B)	55, 56	296	29.34	46	0	0

The THD results show that the DSBAM has maximum THD among all modulation schemes, and it does not reduce with increase in transducer bandwidth. THD of MAM2, MAM3 and SRAM reduce significantly with increase in transducer bandwidth whereas the THD of MAM1 reduces marginally. It is also noted that SSBAM incurs zero THD for both cases of transducer's bandwidth. The IMD (%) computation is derived from multi-tone input. In this case, we use a two-tone input. The first tone f_1 is fixed at 1 kHz, and the second tone f_2 varies from 1.5

kHz to 2.7 kHz. Figure 6.1(a) and (b) plots the IMD values for different modulation schemes for transducers with bandwidth of 6 kHz and 12 kHz, respectively.

(a)

(b)

Figure 6.1 IMD plot for modulation schemes for (a) $f_{\text{trans}}=6$ kHz and (b) $f_{\text{trans}}=12$ kHz.

The IMD for all modulation schemes decreases as the transducer bandwidth increases. For the two transducers' bandwidth, DSBAM has the highest IMD and MAM3 has the lowest IMD values for all combinations of f_1 and f_2 . SRAM, SSBAM and MAM have comparable IMD for small transducer bandwidth but as the transducer bandwidth increases, MAM enjoys more IMD reduction. Within the MAM family, MAM1 has the highest IMD followed by MAM2 and MAM3. Based on these observations, we compare the IMD and computational workload performances of different modulation schemes for small ($f_{trans} \leq 6$ kHz) and large ($f_{trans} > 6$ kHz) bandwidth transducers. For simplicity of discussion, we classify IMD to be low, medium or high depending on its value. The IMD is considered high if its value is greater than 15%. It is considered medium if IMD lies in between 5% and 15%, and the IMD is considered low if its value is less than 5%. The computational workload and memory requirement of different modulation schemes are also classified as low ($MIPS < 10$), medium ($10 \leq MIPS < 20$) or high ($MIPS \geq 20$) depending on their values shown in Table 6.1. A comparison of IMD and computational workload for different modulation schemes is shown in Table 6.2.

Table 6.2 Computational workload and IMD comparison for modulations schemes.

Modulation scheme	Computational workload, Memory	IMD	
		Small transducer bandwidth	Large transducer bandwidth
DSBAM	Low, Low	High	High
SRAM	Medium, High	Medium	Low
MAM1	Low, Low	Medium	Low
MAM2	Low, Low	Medium	Low
MAM3	Low, Low	Medium	Low
SSBAM	High, High	Medium	Medium

With the current transducer technologies, large bandwidth transducers are not available. For small transducer bandwidth, medium level of distortion is generated by all modulation schemes except for DSBAM, which generates high distortion. From computational workload point of view, SSBAM and SRAM have high and medium computation workload requirements, respectively. But both of them have high memory requirements. When larger bandwidth ultrasonic transducers become available in the future, MAM is an efficient algorithm with the lowest computational workload, memory requirement, and the least IMD. Overall, MAM possesses the flexibility in meeting the computational workload and distortion performance for different types of processors and transducer bandwidths.

6.2 Future works

In [33], Bertkay's solution is used to model the demodulation process in air. A more accurate modeling of the demodulation process can be done in the KZK numerical solution [35]. However, the ultrasonic transducer has nonlinear-phase response, which introduces phase distortion and affects the THD and IMD values. The phase distortion can be compensated by performing phase equalization on the modulated signal before it is fed to the ultrasonic transducer. Inclusion of phase equalization in the KZK numerical solution [35] results in a more accurate THD and IMD computations over different modulation schemes. The block diagram of this setup is shown in Fig. 6.2.

Figure 6.2 ABS with phase equalization.

With the advancement in DSP processor technologies, newer processors come with a number of additional features which help in performing signal processing tasks more efficiently. One such feature is the inclusion of tightly-coupled FFT hardware accelerator (HWAFFT). HWAFFT runs faster than the conventional FFT software routines. For example, the HWAFFT in the C5505 is around 3 times faster and 5 times more energy efficient compared to the FFT routine in the DSP library [86]. HWAFFT can perform FFT and IFFT conversions, thus enabling fast filtering of signals in frequency domain. In ABS implementation, the HWAFFT can be utilized to reduce the MIPS requirement of SSBAM by implementing Hilbert and interpolation filters in frequency domain. Alternatively, Hilbert transform can also be generated by processing in frequency domain [87]-[89]. In present work, we used FIR based HT for SSBAM to ensure linear phase response. But human hearing is generally very tolerant to time-invariant phase distortion. IIR filters are computationally efficient structures with nonlinear phase response. They incur significantly lesser computational workload as compared to FIR filters. However, fixed-point implementation of IIR filters necessitates consideration of issues like filter stability, and limit cycle oscillation. Therefore, the applicability of IIR filters to SSBAM for ABS needs further exploration, and is an interesting topic for future study.

Apart from the amplitude-based modulation schemes discussed in this thesis, phase modulation (PM) is recently studied and applied for directional sound generation using parametric array [90]. For PM,

$$y(t) = A_c \cos[2\pi f_c t + \phi_c + x(t)]. \quad (6.1)$$

The block diagram of ABS with PM is shown in Fig. 6.3.

Figure 6.3 ABS with PM.

Cosine (or sine) operation is the main DSP operation in PM. Similar to SQRT calculation, cosine (or sine) calculation is also performed with the help of library routines. The bit accuracy and computational workload of the cosine (or sine) function is the major consideration for fixed-point implementation. Therefore, more work can be carried out to evaluate the fixed-point performance of PM applied in ABS.

Another challenge in the ABS implementation is to improve the efficiency of ABS. The amplifier design is the major bottleneck. Class A, Class B, Class AB and Class D are the commonly used amplifier topologies. Beyond these, the other amplifier topologies are either very

esoteric or not applicable to ABS. The maximum theoretical efficiency that can be achieved by these amplifier topologies [91] is shown in Table 6.3.

Table 6.3 Efficiency of amplifier topologies.

Amplifier type	Theoretical efficiency
Class A	50%
Class AB	< 78.5%
Class B	78.50%
Class D	100%

Among various amplifier topologies, Class D amplifier exhibits the highest efficiency, and is the most suitable choice for ABS implementation. The use of Class D amplifier in parametric loudspeakers is not new and it is described in [92], [93]. Figure 6.4 shows ABS with Class D amplifier.

Figure 6.4 ABS with Class D amplifier.

For ABS with Class D amplifier, the output of modulation scheme is pulse width modulated by a square wave before sending to MOSFETs for amplification. The amplified signal is then fed to ultrasonic transducer. Pulse width modulation (PWM) is an integral part of Class D amplifier and

it can be performed using either analog [94] or digital circuits [95]-[98]. Several DSP boards [98] have counter and compare circuits, which can be configured to generate PWM output. In such case, both modulation scheme and PWM can be implemented digitally [99] on a single DSP board. Recently, [100] describes that PM spectrum is a subset of PWM spectrum. This implies that a suitable PWM method alone is sufficient to generate PM signals. The inclusion of PM and PWM in ABS shows the potential to improve its design and efficiency, and needs further research and exploration.

Appendix A

Two stage amplitude-based modulation

The output of the ABS is a modulated ultrasonic signal. This means that the digital-to-analog converter (DAC) of the digital implementation platform should support sampling rates lying in the ultrasonic range. Consider the scenario where implementation platform supports only sampling rates of $f_s \leq 48$ kHz. In such a case, the envelope, $E[n]$ of modulation schemes is generated on a digital processor, and the ultrasonic modulation is performed in analog domain. This digital-analog (or two stage modulation) implementation scheme provides a mean to reduce the implementation cost. But in practice, the DAC of digital implementation platforms, such as microcontrollers, DSP processor, and FPGA generally blocks the D.C. component of the signal fed to it. This causes problem for algorithms that generate a time varying D.C. component. The envelopes of DSBAM and SSBAM has constant D.C. component that can be added in the analog domain. But this is not possible for SRAM and MAM. To illustrate this, consider a multi-tone input

$$x[n] = \sum_{i=1}^L A_i \cos[2\pi f_i n]. \quad (\text{A.1})$$

Envelopes of SRAM and MAM consist terms of the form

$$(x[n])^m = \left(\sum_{i=1}^L A_i \cos[2\pi f_i n] \right)^m. \quad (\text{A.2})$$

For even m , (A.2) generates D.C. component, and as $x[n]$ varies with time, the D.C. component also varies. There is no way to determine this time varying D.C. component in advance and add it back. Therefore, D.C. blocking should be avoided for SRAM and MAM. D.C

blocking can be avoided by performing ultrasonic modulation in two stages. The block diagram of two stage modulation is shown in Fig. A.1.

Figure A.1 Block diagram of two stage modulation.

The first stage modulation is performed in digital domain using a carrier frequency of f_{c1} , and the second stage modulation is performed in analog domain with a carrier frequency of f_{c2} . The first stage modulation avoids D.C. blocking, and the second stage shifts the spectrum to the resonating frequency of ultrasonic transducer. An analog HPF is required after second stage modulation to remove the unwanted frequencies. Let's consider that the first modulation stage uses MAM1 as the modulation scheme. The spectrum of the input signal, the spectra after first modulation stage (using MAM1) and second modulation stage are shown in Fig. A.2.

Figure A.2 Block diagram of two stage modulation with MAM1 used in the first modulation stage.

Now, consider that B denotes the bandwidth of input signal then from Fig. A.2(b), we have

$$2B \leq f_{c1}. \quad (\text{A.3})$$

If the ultrasonic transducer has resonating frequency at f_c then from Fig. A.2(d), we have

$$f_{c1} + f_{c2} = f_c. \quad (\text{A.4})$$

Suppose that the digital processor running modulation schemes supports maximum sampling frequency of f_s then

$$f_{c1} < \frac{f_s}{2}, \quad (\text{A.5})$$

and

$$f_{c1} + 2B < \frac{f_s}{2}, \quad (\text{A.6})$$

The values of f_c and B that satisfy (A.5) and (A.6) lie in the shaded region of Fig. A.3.

Figure A.3 f_{c1} and B values for two stage modulation using MAM1 in the first modulation stage.

The maximum allowed bandwidth of the input signal, $B_{\max}$, is attained at $f_{c1} = f_s/8$. For given values of f_{c1} and B , f_{c2} can be determined from (A.4).

Consider the example of a desktop computer, which supports $f_s = 48$ kHz. For two stage modulation using MAM1 in first modulation stage, $B_{\max} = 6$ kHz is obtained for $f_{c1} = 12$ kHz. If the

ultrasonic transducer has a resonating frequency of 40 kHz then $f_{c2}=28$ kHz. Theoretically, B can be very close to 6 kHz but this will require stringent specifications for analog HPF which can be impractical. The analog HPF specifications are relaxed if B is decreased. In other words, the relaxation in the specifications of analog HPF comes at the cost of reduced B . Therefore, for this particular example we would chose $f_{c1}=10$ kHz, $f_{c2}=30$ kHz and $B=5$ kHz.

Appendix B

Automatic gain controller

Dynamic behavior of audio signal plays an important role in the perception of demodulated sound from ABS. Loudness of different parts of an audio track may vary. Some parts may be louder whereas some parts may be softer. When the audio track is reproduced by ABS, louder parts generate higher SPL and are easily audible. Softer parts may not generate enough SPL and may remain largely inaudible. This does not translate into good listening experience. To enhance the listening experience, an automatic gain controller (AGC) can be used before modulation schemes to equalize the amplitude of different portions of audio tracks, as shown in Fig. B.1.

Figure B.1 ABS with AGC.

The AGC consists of peak detector, static curve mapping and gain smoothening blocks as shown in Fig. B.2.

Figure B.2 Block diagram of AGC.

The peak detector determines the peak levels of the input signal. These peak levels are used to generate gain values based on a static curve. The generated gain values are smoothened by a gain smoothening block. The smoothened gain is applied to the input signal $x[n]$ to generate output signal $y[n]$.

A Peak Detector Block

The peak detector block is used to determine the peak value of the signal. The block diagram of peak detector is shown in Fig. B.3.

Figure B.3 Block diagram of peak detector.

The difference equation for peak detector is given by

$$x_{peak}[n] = (1 - AT - RT)x_{peak}[n-1] + AT|x[n]|, \quad (B.1)$$

with the transfer function

$$H_{peak}(z) = \frac{AT}{1 - (1 - AT - RT)z^{-1}}. \quad (B.2)$$

AT and RT are the parameters controlling the dynamic behavior of peak detector. They are defined as

$$AT = 1 - e^{\frac{-2.2T_s}{T_a}}, \quad (B.3)$$

and

$$RT = 1 - e^{\frac{-2.2T_s}{T_r}}, \quad (B.4)$$

where T_s is the sampling period, T_a and T_r are attack and release time, respectively, specified in seconds. T_a should be as small as possible whereas T_r should be as large as possible, i.e. $AT \approx 1$ and $RT \approx 0$.

B Static curve mapping and gain smoothening

The output of peak detector block is converted to logarithmic scale and a static curve is used to map it to the desired level. A gain curve can be obtained for the input-output relationship defined by the static curve. The logarithmic gain $G[n]$ can be obtained as

$$G[n] = Y[n] - X_{peak}[n], \quad (B.5)$$

where $X_{peak}[n]$ is logarithmic value for the output of peak detector block and $Y[n]$ is the desired output level. Linear gain $g[n]$ can be derived from $G[n]$ using the relation

$$g[n] = 10^{(G[n]/20)}. \quad (B.6)$$

The linear gain is smoothened and applied to $x[n]$ to generate the output as shown

$$y[n] = g_s[n]x[n]. \quad (\text{B.7})$$

Static curve mapping and gain smoothening blocks are shown in Fig. B.4.

Figure B.4 Static curve mapping and gain smoothening.

The static curve can amplify or attenuate an audio signal selectively depending on its dB value. We define three regions for the static curve: amplification region, attenuation region and linear region. Amplification region is the region bounded by two dB values on static curve for which $G[n] > 0$ dB or $g[n] > 1$. Similarly, for attenuation region $G[n] < 0$ dB or $g[n] < 1$, and for linear region $G[n] = 0$ dB or $g[n] = 1$. A static curve with the linear region lying in the interval $[-96 \text{ dB}, -60 \text{ dB})$, amplification region lying in the interval $[-60 \text{ dB}, -2 \text{ dB})$ and attenuation region lying in the interval $[-2 \text{ dB}, 0 \text{ dB}]$ is shown in Fig. B.5(a). The static curve has ‘hard knees’ at the boundaries of two different regions, which creates audible distortion. To reduce this distortions, hard knees are smoothened to soft knees by smoothening. The smoothened static curve is shown in Fig. B.5(b). The log and linear gain curves for smoothened static curve (Fig. B.5(b)) are shown in Fig. B.5(c) and (d), respectively.

(a)

(b)

(c)

(d)

Figure B.5 (a) Hard knee static curve, (b) Soft knee static curve, (c) Log scale gain curve for smoothed static curve, and (d) linear scale gain curve for smoothed static curve.

For real-time implementation, linear gain values for the range [0 dB, -96 dB], at the interval of 1 dB, are stored in a LUT. Therefore, the AGC LUT has 97 entries and its size is 194 bytes. These gain values are marked on the gain curve shown in Fig. B.6.

Figure B.6 Gain curve with gain values marked at an interval of 1 dB.

To generate the gain value from LUT, the dB value of the peak detector block output is calculated as

$$X_{peak}[n] = 20 \times \log_{10}(x_{peak}[n]). \quad (B.8)$$

$X_{peak}[n]$ is either an integer or a rational number. If $X_{peak}[n]$ is a rational number, it is converted to the integer nearest but greater than $X_{peak}[n]$ denoted by $\lceil X_{peak}[n] \rceil$. $\lceil X_{peak}[n] \rceil$ for each input sample is calculated and mapped to the gain table. The resultant gain $g[n]$ is smoothened by a first order IIR filter to produce $g_s[n]$, which is applied to the input signal as shown in (B.7).

C AGC simulation and implementation

AGC is first simulated on MATLAB and then implemented on C5505 eZDSP in real-time. To demonstrate the performance of AGC, a segment of an audio track is taken as input signal and the linear gain generated by AGC is plotted. Figure B.7(a) and (b) shows the linear gain generated from MATLAB and C5505 eZDSP, respectively. It can be noted that similar gain values are generated by MATLAB and C5505 eZDSP, which validates our implementation.

(a)

(b)

Figure B.7 (a) AGC simulated on MATLAB (64-bit precision) and (b) implemented on C5505 eZDSP (16-bit precision).

D Auto Mute

Auto mute is used to mute the output signal when the input signal level is below a certain threshold. Without auto mute for no input, a 40 kHz carrier is outputted, which is redundant. The auto mute function samples the ADC at fixed time intervals (sampling interval). If the sample values are less than a fixed threshold (limiting threshold) for a fixed amount of time (silence period), the DAC is muted. The block diagram of ABS with auto mute functionality is shown in Fig. B.8.

Figure B.8 ABS with AGC and auto mute functionality.

The DAC is unmuted as soon as a sample with level greater than the limiting threshold is detected by auto mute function. In our implementation, we set the sampling interval to 2 ms, limiting threshold to -60 dB and silence period to 3 seconds.

Appendix C

DNR_T(n): Derivation and examples

A Derivation of DNR_T(2)

The derivation of the DNR_T(2) expression of (4.14) proceeds as follows. To keep the notation simple we refer to the i^{th} entry of the ISQRT(x)/2 LUT, $LUT_{\text{ISQRT}/2}(i)$, as I , and refer to the i^{th} entry of the SQRT(x) LUT, $LUT_{\text{SQRT}}(i)$, as S . Referring to Fig. 4.8, denoting the difference between x_N and x_L by e , we write

$$e = x_N - x_L, \quad (\text{C.1})$$

where the output of first iteration is given by

$$y_{1,N} = S + eI. \quad (\text{C.2})$$

Subsequently, the $y_{2,N}$ output of the second iteration can be written as

$$\begin{aligned} y_{2,N} &= \left[x_N - (S + eI)^2 \right] I + S + eI \\ &= S + (x_N - S^2 - e^2 I^2) I + eI - 2eSI^2. \end{aligned} \quad (\text{C.3})$$

Because $S \cdot I \approx 1/2$, the last two terms in (C.3) cancel each other, and the $x_N - S^2$ terms can be replaced by e , giving us

$$y_{2,N} \approx S + I(e - e^2 I^2), \quad (\text{C.4})$$

or

$$y_{2,N} \approx LUT_{\text{SQRT}}(i) + LUT_{\text{ISQRT}/2}(i) \left[e - e^2 (LUT_{\text{ISQRT}/2}(i))^2 \right]. \quad (\text{C.5})$$

Our empirical studies have shown that $(LUT_{ISQRT/2}(i))^2$ can be replaced by a factor 0.5 without inducing significant error in the desired results. As such, this gives a single-iteration approximation of

$$y_{1,N} \approx y_{2,N} \approx LUT_{SQRT}(i) + LUT_{ISQRT/2}(i) \times \left(e - \frac{e^2}{2} \right)$$

as presented in (4.14).

B Examples for generating L -bit rounding errors from L -bit truncation errors

We present two examples to show that (4.12) and (4.13) are equivalent, and L -bit rounding can be replaced by L -bit truncation and arithmetic shifts. The binary numbers, in the examples, use two's complement number representation and Q1.15 format (first bit is sign bit). ' $\leftarrow$ ' and ' $\rightarrow$ ' denote arithmetic left and right shift operation, respectively. We use $N=16$ and $L=6$ for both the examples. Table C.1 shows error values generated by L -bit rounding operation of (4.12), and Table C.2 shows error values generated by L -bit truncation followed by arithmetic shifts as used by (4.13).

Table C.1 Generating error values using L -bit rounding.

	Example 1	Example 2
x_{16}	0101111011101101 ₂ = 0.741607666015625 ₁₀	0011000001101111 ₂ = 0.378387451171875 ₁₀
x_{16} rounded to 6 bits	0110000000000000 ₂ = .75 ₁₀	0011000000000000 ₂ = .375 ₁₀
e	-0.008392333984375 ₁₀	0.003387451171875 ₁₀

Table C.2 Generating error values using L -bit truncation and arithmetic shifts.

	Example 1	Example 2
x_{16}	0101111011101101_2 $= 0.741607666015625_{10}$	0011000001101111_2 $= 0.378387451171875_{10}$
x_{16} truncated to 6 bits	0101110000000000_2 $= 0.71875_{10}$	0011000000000000_2 $= .375_{10}$
e_T	0000001011101101_2 $= 0.022857666015625_{10}$	0000000001101111_2 $= 0.003387451171875_{10}$
$e_T \leftarrow 6$	1011101101000000_2	0001101111000000_2
$[e_T \leftarrow 6] \rightarrow 6$	1111111011101101_2 $= -0.008392333984375_{10}$	0000000001101111_2 $= 0.003387451171875_{10}$

It can be observed that ‘ e ’ values of Table C.1, and ‘ $[e_T \leftarrow L] \rightarrow L$ ’ values of Table C.2 are exactly the same for both examples.

Appendix D

Photographs of software and hardware used

A Breakpoints to calculate CPU cycles

In Section 2.4, we mentioned that CPU clock cycles required for the execution of a code segment can be known with the help of CCS and breakpoints. Figure D.1 shows the screenshot of CCSv4 with breakpoints.

Figure D.1 Screenshot of CCSv4 [49] with breakpoints.

‘Breakpoint-1’ and ‘Breakpoint-2’ are set at appropriate positions before the code is executed. When the code is executed, the CPU clock cycles taken to reach from ‘Breakpoint-1’ to ‘Breakpoint-2’ are shown at the bottom of the screenshot. This CPU cycle count is substituted into (2.23) or (2.25) to compute the MIPS for the code segment.

B ABS with C5505 eZDSP USB stick development board

C5505 eZDSP USB stick development board has 16 test points for the monitoring of signals [101]. The location of these test points is shown in Fig. D.2.

Figure D.2 Test points on C5505 eZDSP USB stick development board [101].

We attach switches to TP12, TP13, TP14 and TP15 to on/off the AGC and select different modulation schemes. TP12 is used to switch on/off the AGC, and TP13, TP14 and TP15 are used to select different modulation schemes. Table D.1 shows switch combinations to turn on/off AGC and select different modulation schemes.

Table D.1 Switch combinations for AGC and modulation schemes.

TP12	TP13	TP14	TP15	
0	X	X	X	AGC OFF
1	X	X	X	AGC ON
X	0	0	0	DSBAM
X	0	0	1	SRAM
X	0	1	0	MAM1
X	0	1	1	MAM2
X	1	0	0	MAM3
X	1	0	1	SSBAM
X	1	1	X	DSBAM

Figure D.3 shows the picture of C5505 eZDSP USB stick development board with switches and battery, which is used to power the C5505 eZDSP USB stick development board.

Figure D.3 C5505 eZDSP USB stick development board with switches and batteries.

A picture of ABS with C5505 eZDSP USB stick development board, Class AB amplifiers and ultrasonic transducer are shown in Fig. D.4.

Figure D.4 ABS with C5505 USB stick development board, Class AB amplifier and ultrasonic transducer.

Author's publications

Book chapters

- [1] **A. Seth** and W.S. Gan, "Fixed-point square roots using L -bit truncation," to appear in *Streamlining Digital Signal Processing: A tricks of the trade guidebook*, 2nd edition, IEEE Press/Wiley & Sons.

Journal publications

- [1] **A. Seth** and W. S. Gan, "Fixed-point square roots using L -bit truncation," *IEEE Signal Processing Magazine*, Vol. 28, No. 6, Nov. 2011, pp. 149-153.

Conference publications

- [1] **A. Seth** and W. S. Gan, "Fixed-point square roots," accepted in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2012)*, 25-30 March 2012, Kyoto, Japan.
- [2] W. S. Gan, **A. Seth**, and S. M. Kuo, "Versatile and portable DSP platform for learning embedded signal processing," *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2011)*, 22-27 May 2011, Prague, Czech Republic.

References

- [1] *The American Heritage Dictionary of the English Language*, 4th Edition, Houghton Mifflin Company,
- [2] E. W. Siemens, "Magneto-electric Apparatus," *US Patent 149797*, 1874.
- [3] E. W. Kellogg, "Production of Sound," *US Patent 1983377*, 1934.
- [4] C. Shi and W. S. Gan, "Development of parametric loudspeaker: a novel directional sound generation technology," *IEEE Potentials*, vol. 29, pp. 20-24, 2010.
- [5] K. Brown, "Acoustic Imaging Sound Dome," World Intellectual Property Organization, WO 95/12960, May 1995.
- [6] I Ltd, *Digital Sound Projector*. [Online]. Available: <http://www.illimited.com/>
- [7] P. J. Westervelt, "Parametric acoustic array," *Journal of the Acoustical Society of America*, vol. 35, pp. 535-537, 1963.
- [8] W. S. Gan, E. L. Tan, and S. M. Kuo, "Audio Projection-Directional sound and its application in immersive communication," *IEEE Signal Processing Magazine*, vol. 28, pp. 43-57, Jan. 2011.
- [9] M. B. Bennett and D. T. Blackstock, "Parametric Array in Air," *Journal of Acoustic Society of America*, vol. 57, pp 562-568, March 1975.
- [10] M. Yoneyama and J. Ichiroh Fujimoto, "The audio spotlight: An application of nonlinear interaction of sound waves to a new type of loudspeaker design", *Journal of Acoustical Society of America*, vol. 73, pp. 1532-1536, 1983.
- [11] T. Kamakura, M. Yoneyama, and K. Ikegaya, "Developments of parametric loudspeaker for practical use," in *International Symposium on Nonlinear Acoustics*, pp. 147-150, 1984.
- [12] F. J. Pompei, "The use of airborne ultrasonics for generating audible sound beams," *Journal of Audio Engineering Society*, vol. 47, no. 9, pp. 726-731, 1999.
- [13] T. D. Kite, J. T. Post, and M. F. Hamilton, "Parametric array in air: distortion reduction by preprocessing," *Journal of Acoustic Society of America*, vol. 103, pp. 2871, 1998.

- [14] J. J. Croft and J. O. Norris, "White paper on hypersonic sound," *American Technology Corporation*, 2001/2002, pp. 1-28.
- [15] J. Yang, K. Sha, W. S. Gan, and J. Tian, "Modeling of finite-amplitude sound beams: Second order fields generated by a parameter loudspeaker," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 52, no. 4, pp. 610-618, 2005.
- [16] F. A. Karnapi, W. S. Gan, Y. K. Chang, "FPGA implementation of parametric loudspeaker system", *Microprocessors & Microsystems*, pp. 261-272, 2004.
- [17] B. Widrow, I. Kollar, and Ming-Chang Liu, "Statistical Theory of Quantization", *IEEE Transactions on Instrumentation and Measurement*, vol. 45, pp. 353-361, 1996.
- [18] A. Sripad and D. L. Snyder, "A necessary and sufficient condition for quantization error to be uniform and white," *IEEE Transaction on Acoustics, Speech and Signal Processing*, vol. 25, pp. 442-448, 1977.
- [19] B. Widrow, "Statistical analysis of amplitude quantized sampled-data systems," *AIEE Transactions on Applications and Industry*, pp. 1-14, 1961.
- [20] C. Barnes, B. N. Tran, and S. Leung, "On the statistics of fixed-point roundoff error," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 33, pp. 595-606, 1985.
- [21] F. J. Pompei, *Sound from ultrasound: The parametric array as a audible sound source*, Ph. D. thesis, Massachusetts Institute of Technology, U. S., 2002.
- [22] M. A. Al-Alaoui, "Novel digital integrator and differentiator," *Electronic Letters*, vol. 29, pp. 376-378, 1993.
- [23] N. Papamarkos and C. Chamzas, "A new approach for the design of digital integrators," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 43, pp. 785-791, 1995.
- [24] N. Q. Ngo, "A new approach for the design of wideband digital integrator and differentiator," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, pp. 936-940, 2006.
- [25] Y. Wang, X. Li, L. Xu, and Leon Xu, "SSBAM modulation of the ultrasonic carrier for a parametric loudspeaker," in *IEEE International Conference on Electronic Computer Technology*, 2009.

- [26] E. L. Tan, D. Harliono, W. S. Gan, and J. Yang, "Preprocessing techniques for parametric loudspeakers", in *IEEE International Conference on Audio, Language and Image Processing*, 2008.
- [27] J. Munir and U. Imtiaz, "Comparative analysis of different distortion reduction techniques for parametric loudspeaker based on self-demodulation of amplitude modulated ultrasound waves," in *IEEE International Conference on Electrical Engineering*, 2008.
- [28] B. Gold, A. V. Oppenheim and, C. M. Rader, "Theory and implementation of the discrete Hilbert transform," in *Symposium on Computer Processing in Communications*, pp. 235-250, 1969.
- [29] D. K. Weaver, Jr., "A third method of generation and detection of single-sideband signals," *Proceeding of the IRE*, vol. 44, pp. 1703-1705, 1956.
- [30] L. R. Rabiner and R. W. Schafer, "On the behavior of minimax FIR digital Hilbert transformers," *The Bell system technical journal*, pp. 363-390, 1974.
- [31] R. Ansari, "IIR discrete time Hilbert transformers," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35, pp. 1116-1119, 1987.
- [32] Y.H. Liew, *Signal processing techniques for sound reproduction in parametric array*, M. Eng. thesis, Nanyang Technological University, Singapore, 2002.
- [33] E. L. Tan, W. S. Gan, and J. Yang, "On preprocessing techniques for bandlimited parametric loudspeakers", *Applied Acoustics*, vol. 71, pp. 486-492, 2010.
- [34] H. O. Bertkay, "Possible exploitation of nonlinear acoustic in underwater transmitting applications", *Journal of Sound and Vibration*, vol. 24, pp. 435-461, 1965.
- [35] N. S. Bakhvalov, Y. M. Zhileikin and E. A. Zabolotskaya, "*Nonlinear theory of sound beams*," American Institute of Physics, 1987.
- [36] S. M. Kuo and W. S. Gan, *Digital Signal Processors-Architectures, Implementations, and Applications*, Prentice Hall, 2004.
- [37] W. S. Gan and S. M. Kuo, *Embedded Signal Processing with the Micro Signal Architectures*, Wiley, 2007.
- [38] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

- [39] L. R. Rabiner and B. Gold, *Theory and Applications of Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1975.
- [40] A. V. Oppenheim and C. J. Weinstein, "Effects of finite register length in digital filtering and fast Fourier transforms," *Proceedings of the IEEE*, vol. 60, pp. 956- 976, 1972.
- [41] S. Kim, K. Kum, and S. Wonyong, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *IEEE Transaction on Circuits and Systems II*, vol. 45, pp. 1455-1464, 1998.
- [42] H. Keding, F. Hurtgen, M. Willems, and M. Coors, "Transformation of floating-point into fixed- point algorithms by interpolation applying a statistical approach," in *9th International Conference on Signal Processing Application Technology (ICSPAT 98)*, 1998.
- [43] K. Kum and W. Sung, "Wordlength optimization for high level synthesis of digital signal processing systems," in *IEEE Workshop on Signal Processing Systems (SIPS 98)*, Boston, US, pp. 142–151, 1998.
- [44] MATLAB. *Fixed-Point Toolbox*. [Online]. Available: <http://www.mathworks.com/help/toolbox/fixedpoint>.
- [45] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, Mc-Graw Hill, 2006.
- [46] Texas Instruments. *TMS320C55x DSP CPU Reference Guide*. SPRU371F, 2004.
- [47] Texas Instruments. *TMS320VC5505 DSP System User's Guide*. SPRUFP0A, 2009.
- [48] TMS320VC5505 Fixed-Point Digital Signal Processor, SPRS503B, 2010.
- [49] Texas Instruments. *Code Composer Studio Integrated Development Environment v4.x*. [Online]. Available: <http://focus.ti.com/docs/toolsw/folders/print/ccstudio.html>.
- [50] P. Montuschi, and M. Mezzalama, "Survey of square rooting algorithms," *IEE Proceedings*, vol. 137, pp. 31-40, 1990.
- [51] K. Hwang, *Computer Arithmetic: Principles, architecture and design*, John Wiley & Sons, 1978.
- [52] N. R. Scott, *Computer number systems & arithmetic*, Princeton Hall, 1985.
- [53] M. D. Ercegovic, "An on-line square rooting algorithm," in *IEEE Symposium on Computer Arithmetic*, pp. 183-189, Oct. 1978.

- [54] V. G. Oklobdzija and M. D. Ercegovac, "An on-line square root algorithms," *IEEE Transactions on Computers*, vol. 31, pp.70-75, 1982.
- [55] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Transactions on Electronic Computers*, vol. 8, pp. 330-334, 1959.
- [56] J. S. Walther, "A unified algorithm for elementary functions," in *AFIPS Spring Joint Computer Conference*, pp. 379-385, 1971.
- [57] B. G. Delugish, *A class of algorithms for automatic evaluation of certain elementary functions in a binary computer*, Ph. D. Dissertation, Department of Computer Science, University of Illinois, Urbana, 1970.
- [58] T. C. Chen, "The automatic computation of exponentials, logarithms, ratios and square roots," *IBM Journal of Research and Development*, pp. 380-388, 1972.
- [59] M. Abramowitz, and I. Stegun, *Handbook of mathematical functions*, Dover Publicationss, New York, 1968.
- [60] C. Clenshaw, "Polynomial approximations to elementary functions," *Mathematical Tables Aids Computation*, 1954.
- [61] C. Ramamoorthy, J. Goodman, and K. Kim, "Some properties of iterative square rooting methods using high-speed multiplication," *IEEE Transactions on Computers*, vol. 21, pp. 837-847, 1972.
- [62] K. S. Kunz, *Numerical analysis*, McGraw-Hill, 1957.
- [63] R. W. Southworth and S. L. Deleeuw, *Digital computation on numerical methods*, McGraw-Hill, 1965.
- [64] M. Ercegovac, and T. Lang, "On-the-fly conversion of redundant into conventional representations," *IEEE Transactions on Computers*, vol. 36, pp 895-897, 1987.
- [65] N. Mikami, M. Kobayashi, and Y. Yokoyama, "A new DSP-oriented algorithm for calculation of square root using a nonlinear digital filter," *IEEE Transactions on Signal Processing*, vol. 40, pp. 1663-1669, 1992.
- [66] M. Allie and R. Lyons, "A root of less evil," *IEEE Signal Processing Magazine*, vol. 22, pp.93-96, 2005.
- [67] David G. Moursund, "Optimal starting values for Newton-Raphson calculation of $\sqrt{x}$," *Communications of the ACM*, pp. 430-432, 1967.

- [68] C. T. Fike, "Starting approximations for square root calculation on IBM system /360," *Communications of the ACM*, pp. 297-299, 1966.
- [69] M. Wayne Wilson, "Optimal starting approximations for generating square root for slow or no divided," *Communications of the ACM*, pp. 559-560, 1970.
- [70] Texas Instruments. *TMS320C55x DSP Library Programmer's Reference*. SPRU422J, 2000.
- [71] NICERA. *Ultrasonic transducers*. [Online].Available: <http://www.nicera.ph>
- [72] MURATA. *Ultrasonic transducers*. [Online].Available: <http://www.murata.com>
- [73] PARSONICS. *Ultrasonic transducers*. [Online].Available: <http://parsonicscorp.com>
- [74] R. Johannes, J. W. Beh, E. L. Tan, W. S. Gan and E. L. Tan, "Investigation of 3D audio rendering with parametric array loudspeakers," in *128th Audio Engineering Convention (AES)*, 22-25 May 2010, London, UK.
- [75] T. W. Parks and J. H. McClellan, "Chebyshev approximation for non recursive digital filters with linear phase," *IEEE Transactions on Circuit Theory*, vol. 19, pp. 189-194, 1972.
- [76] D. J. Shpak and A. Antoniou, "A generalized Remez method for the design of FIR digital filters," *IEEE Transactions on Circuits & Systems*, vol. 37, pp. 161-174, 1990.
- [77] D. W. Harris, E. Berdahl and J. S. Abel, "An IIR Hilbert transformer filter design technique for audio," in *AES Convention*, pp. 1-8, Nov. 2010.
- [78] R. G. Lyons, *Understanding Digital Signal Processing*, Princeton Hall, 2004.
- [79] R. Schafer and L. R. Rabiner, "A digital signal processing approach to Interpolation," *Proceedings of the IEEE*, vol. 61, pp. 692-702, June 1973.
- [80] R. E. Crochiere and L. R. Rabiner, "Optimal FIR digital filter implementations for decimation, interpolation, and narrow band filtering," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 23, pp. 444-456, 1975.
- [81] R. E. Crochiere and L. R. Rabiner, "Further considerations in the design of decimators and interpolators," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 24, pp. 444-456, 1976.
- [82] D. B. Turek, *Design of efficient digital interpolation filters for integer upsampling*, M. Eng. Thesis, Massachusetts Institute of Technology, June 2004.

- [83] S. Shimizu and T. Yahagi, "Design of allpass networks for Hilbert transformers," *Electronics Letters*, vol. 29, pp. 2087-2088, 1993.
- [84] H. Johansson and L. Wanhammar, "Digital Hilbert transformers composed of identical allpass subfilters," in *International Symposium on Circuits and Systems*, pp. 437-440, 1998.
- [85] P. A. Regalia, S. K. Mitra and P. P. Vaidyanathan, "The digital all-pass filter: A versatile signal processing building block," *Proceedings of the IEEE*, vol. 76, pp. 19-37, Jan. 1998.
- [86] Texas Instruments. *FFT Implementation on the TMS320VC5505, TMS320C5505, and TMS320C5515 DSPs*, SPRABB6A, 2010
- [87] Claerbout, J.F., *Fundamentals of Geophysical Data Processing*, McGraw-Hill, 1976, pp.59-62.
- [88] S. L. Marple, "Computing the discrete-time analytic signal via FFT," *IEEE Transactions on Signal Processing*, vol. 47, pp. 2600-2603, 1999.
- [89] A. V. Oppenheim and R.W. Schaffer, *Discrete-Time Signal Processing*, 2nd edition, Prentice-Hall, 1998.
- [90] M. Watanabe and M. Mori, "Method and device for driving a directional loudspeaker," *US Patent 7,801,315 B2*, Sep. 2010.
- [91] Maxim. *High-efficiency Class D audio amplifiers extend battery life in portable applications*. AN 1760, 2010.
- [92] G. J. Bank and J. J. Croft, "Power amplification for parametric loudspeakers," *US Patent 7,319,763 B2*, 2008.
- [93] G. J. Bank and J. J. Croft, "Power amplification for parametric loudspeakers," *Journal of Acoustical Society of America*, vol. 123, pp. 4032-4032, 2008.
- [94] National Semiconductor, *LM555 Timer*. [Online]. Available: <http://www.national.com/ds/LM/LM555.pdf>
- [95] Microchip, *PIC microcontroller unit*. [Online]. Available: <http://www.microchip.com>
- [96] Texas Instruments. *MSP430 Mixed signal microcontroller*, SLAS694E, 2011.
- [97] CYGNAL. *16-bit PWM using an onchip timer*, AN010, 2001.
- [98] Analog Devices, *Blackfin processors*. [Online]. Available: <http://www.analog.com>

- [99] Y. H. Liu, J. H. Tang and C. H. Hsieh, "Design and implementation of a fully-digital, high-efficiency Class-D amplifier system," in *IEEE TENCON*, pp. 1-4, 2007.
- [100] C. Pascual, Z. Song, P. T. Krein, D. V. Sarwate, P. Midya and W. J. Roeckner, "High-Fidelity PWM inverter for digital audio amplification: spectrum analysis, real-time DSP implementation, and results," *IEEE Transactions on Power Electronics*, vol. 18, pp. 473-485, 2003.
- [101] Spectrum Digital. *TMS320VC5505 eZDSP USB Stick Technical Reference*, 512325-0001 Rev. A, 2009.