University of Calgary

PRISM Repository **https://prism.ucalgary.ca**

The Vault Open Theses and Dissertations

2013-10-10

Design and Comparative Analysis of FRS Architectures for the AI DCT

Onen, Denis

Onen, D. (2013). Design and Comparative Analysis of FRS Architectures for the AI DCT (Doctoral thesis, University of Calgary, Calgary, Canada). Retrieved from https://prism.ucalgary.ca. doi:10.11575/PRISM/25564

http://hdl.handle.net/11023/1143

Downloaded from PRISM Repository, University of Calgary

UNIVERSITY OF CALGARY

Design and Comparative Analysis of FRS Architectures for the AI DCT

by

Denis Onen

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

SEPTEMBER, 2013

© Denis Onen 2013

ABSTRACT

Computationally intensive sinusoidal transforms such as the Fast Fourier Transform (FFT) and Discrete Cosine Transform (DCT) play a large role in science and engineering. The DCT has found valuable application in image and video processing. Its broad use has prompted many researchers to seek to improve its computational properties.

In the DCT, multiplication by irrational constants has been identified as a source of error, due to the introduction of quantization error. Efforts to overcome this quantization error have led to the relatively recent application of Algebraic Integer (AI) representations, by Dimitrov. The use of AI allows the irrational constants to be encoded, without quantization error. Numbers encoded in AI can be easily operated on by the arithmetic operations of addition, subtraction, and multiplication, all without the introduction of quantization error. A large computational penalty is paid when the AI encoded number, is reconstructed into a single number, using a process known as the Final Reconstruction Step (FRS).

This research investigates and develops a large number of novel designs for the FRS, in the application of 2D AI to the 2D DCT. These designs include the application of Multiple Constant Multiplication (MCM) and a substitution method, to reduce the computational complexity of the FRS. These new designs offer a significant improvement in Mean-Squared Error (MSE) and computational complexity, in comparison with standard FRS designs.

This research also provides a comprehensive comparative analysis of AI methods versus non-AI methods for the 2D DCT, and finds that non-AI methods are superior to AI in many cases, for the metrics of MSE and computational complexity, despite the significant

FRS improvements reported in this thesis. For the case of area-limited designs, this thesis demonstrates the superiority of the AI method over the non-AI method, by the use of a single, multiplexed FRS stage.

ACKNOWLEDGEMENTS

I would like to express my sincerest gratitude and appreciation to my supervisor, Dr. Vassil Dimitrov, for his guidance and support during the course of my PhD. His brilliant mind and positive, cheerful attitude made my research experience a pleasure. I am indebted to the legacy of my predecessor, Dr. Khan Wahid, upon whose excellent work I built my research. I was very fortunate to have had my MSc supervisor, Dr. Martin Mintchev, who first taught me academic rigor, which shaped my outlook towards research and writing. I was also very fortunate to have outstanding collaboration with Drs. Arjuna Madanayake, Renato Cintra, and Arnaud Tisserand, which greatly expanded the scope of my results. Special thanks go to my former mentor in industry, Peter Graumann, who sparked my initial interest in constant integer multiplication.

Many thanks are due to Dr. Bill Rosehart, who introduced me to teaching, which I loved. I had the pleasure to learn from my students, as well as from the professors with whom I taught. It is an experience that I will never forget.

Most of all, my most heartfelt thanks go to my family, without whose support, this research would never have been completed. My thanks go to my mother, Harika, for always providing me books on science, while I was a child. These books always delighted me, demanding to be read from cover to cover, repeatedly. My deepest thanks go to my wife Zeynep, for her unwavering support and encouragement. My very special thanks go to my children, Arda and Elif, for their patience while I completed my PhD.

For my wife and children, with all my love.

TABLE OF CONTENTS

Abstract ii

Acknowledgments iv

Dedication v

Table of Contents vi

List of Tables ix

List of Figures xi

List of Abbreviations xiii

1 INTRODUCTION 1

1.1 Introduction 1

1.2 Literature Review 3

1.3 Motivation for this Research 11

1.4 Contributions of the Thesis 12

1.5 Structure of the Thesis 16

2 BACKGROUND 17

2.1 Introduction 17

2.2 Discrete Cosine Transform 17

2.3 Fast Algorithm for the Discrete Cosine Transform 19

2.4 Algebraic Integers 20

2.4.1 One-dimensional Algebraic Integers 21

2.4.1.1 Encoding 21

2.4.1.2 Operations 23

2.4.2 Multidimensional Algebraic Integers 26

2.4.3 Approximation of 1D Algebraic Integers 33

2.5 Single-Constant Integer Multiplication in VLSI 34
2.5.1 Conversion of a Real Number to an Integer Approximation 35
2.5.2 Constant Integer Multiplication Using the Binary Implementation . 37
2.5.3 Constant Integer Multiplication Using the Canonic Signed Digits Implementation 38
2.5.4 Constant Integer Multiplication Using the Dempster-Macleod Implementation 39
2.5.5 Multiple Word-Length Paradigm 39
2.6 The Dempster-Macleod Algorithm 40
2.6.1 DM Algorithm Rules to Minimize Search Space 42
2.6.1.1 Rule for Generating Only Odd Vertex Graphs 42
2.6.1.2 Rule for Generating Even Numbers 43
2.6.1.3 Rule for Fractional Magnitude Edges 44
2.6.1.4 Rule for the Maximum Edge Value to Evaluate 44
2.7 Chapter Summary 44

3 SOFTWARE TOOLS AND MODELS 46
3.1 Single Constant Multiplication Tool 46
3.2 Multiple Constant Multiplication Tool (PARI/GP) 51
3.3 Mean-Squared Error Analysis 58
3.4 AI Model and Adder-bits Computation Tool 59
3.4.1 Adder-bits Computation Tool 59
3.4.2 AI Model 64
3.4.2.1 Full AI Model 64
3.4.2.2 Substitution Method for AI 71
3.5 Sample Implementations 78
3.5.1 Non-AI Designs 79
3.5.1.1 8-bit Designs 80
3.5.1.2 16-bit Designs 81
3.5.2 AI Designs 83
3.5.2.1 Common Subexpressions 84
3.5.2.2 SCM and MCM Designs for Full AI 85
3.5.2.2.1 SCM Designs - 8-bit 85
3.5.2.2.2 SCM Designs - 16-bit 87
3.5.2.2.3 MCM Designs - 8-bit 90
3.5.2.2.4 MCM Designs - 16-bit 95

3.5.2.3 MCM Designs for Approach 0 97
3.5.2.3.1 Approach 0: 8-bit Coefficient Quantization . . 97
3.5.2.3.2 16-bit Coefficient Quantization 99
3.5.2.3.3 Comparison of Lowest MSE Performance for 8- to 16-bit Coefficient Quantization in Approach 0 100
3.5.2.4 MCM Designs for Approach 2 101
3.5.2.4.1 8-bit Coefficient Quantization 102
3.5.2.4.2 16-bit Coefficient Quantization 103
3.5.2.5 Comparison of Lowest MSE Performance for 8- to 16-bit Coefficient Quantization for Approach 2 106
3.5.2.6 MCM Designs for Approach 4 106
3.5.2.6.1 8-bit Coefficient Quantization 107
3.5.2.6.2 16-bit Coefficient Quantization 108
3.5.2.6.3 Comparison of Lowest MSE Performance for 8- to 16-bit Coefficient Quantization for Approach 4 109
3.6 Chapter Summary 111

4 RESULTS AND COMPARATIVE ANALYSIS OF AI DESIGNS 112
4.1 Introduction 112
4.2 Comparison of Design Approaches 112
4.2.1 Fully Parallel Design Approach 113
4.2.2 Time-Multiplexed Design Approach 123
4.2.3 Fully Parallel Design Approach, with Single FRS 127
4.2.4 Time-Multiplexed Design Approach, with Single FRS 130
4.3 Chapter Summary 137

5 CONCLUSIONS 138

BIBLIOGRAPHY 140

LIST OF TABLES

1.1 Constant approximations for [1] 6
1.2 Constant approximations for [1] 6
1.3 Constant approximations for [1] 7
1.4 Constant approximations for [2,3] 7
1.5 Constant approximations for [2–5] 8
1.6 Constant approximations for [2–5] 8
1.7 Constant approximations for [4,5] 9
1.8 Constant approximations for [4,5] 9
1.9 Constant approximations for [4,5] 10
1.10 Constant approximations for [4,5] 10
1.11 Constant approximations for [6] 11
1.12 Constant approximations for [7] 12
1.13 Constant approximations for [7] 12

2.1 Arai 1D DCT multipliers 20
2.2 Scaling Factors for Arai 1D DCT 20
2.3 Arai 1D DCT multipliers with 1D AI encodings 23
2.4 Arai 1D DCT multipliers with 2D AI encodings 28
2.5 Approximation of $2\cos(2\pi k/32)$ 34
2.6 Approximation of $\sqrt{2}$ 37

3.1 Example SCM algorithms used for 12-bit full AI 52
3.2 Example combined SCM algorithms 54
3.3 Example MCM algorithm 55
3.4 Comparison of SCM vs MCM number of adders for a sum of 7 integer coefficient multiplications 55
3.5 Comparison of SCM vs MCM number of adders for the FRS 56
3.6 Comparison of SCM vs MCM number of adderbits for the FRS 57
3.7 Arai 1D DCT in tool input format 61

3.8 Comparison of multipliers for simplifying FRS equations 78
3.9 SCM algorithms for 8-bit Arai m-values 80
3.10 SCM algorithms for 16-bit Arai m-values 82
3.11 SCM algorithms for 8-bit full AI, part 1 86
3.12 SCM algorithms for 8-bit full AI, part 2 86
3.13 SCM-based FRS algorithm for S(i), 8-bit, full AI 87
3.14 SCM-based FRS algorithm for S(j), 8-bit, full AI 88
3.15 SCM-based FRS algorithm for S(k), 8-bit, full AI 88
3.16 SCM algorithms for 16-bit full AI, part 1 91
3.17 SCM algorithms for 16-bit full AI, part 2 92
3.18 SCM-based FRS algorithm for S(i), 16-bit, full AI 92
3.19 SCM-based FRS algorithm for S(j), 16-bit, full AI 93
3.20 SCM-based FRS algorithm for S(k), 16-bit, full AI 94
3.21 MCM-based FRS algorithm for S(i), 8-bit, full AI 95
3.22 MCM-based FRS algorithm for S(j), 8-bit 95
3.23 MCM-based FRS algorithm for S(i), 16-bit, full AI 96
3.24 MCM-based FRS algorithm for S(j), 16-bit, full AI 96
3.25 MCM-based FRS algorithm for S, 8-bit, approach 0 98

4.1 Ranges of MSEs for 2D DCT Designs vs Quantization Bits 114
4.2 MSE vs Adders for Fully Parallel Implementation - Part 1 115
4.3 MSE vs Adders for Fully Parallel Implementation- Part 2 117
4.4 MSE vs Adder-bits for Fully Parallel Implementation- Part 1 119
4.5 MSE vs Adder-bits for Fully Parallel Implementation- Part 2 119
4.6 Minimum number of adders for MSE thresholds in the fully parallel implementation 121
4.7 Minimum number of adder-bits for MSE thresholds in the fully parallel implementation 122
4.8 MSE vs Adders for Time-Multiplexed Implementation- Part 1 123
4.9 MSE vs Adders for Time-Multiplexed Implementation- Part 2 123
4.10 MSE vs Adder-bits for Time-Multiplexed Implementation- Part 1 125
4.11 MSE vs Adder-bits for Time-Multiplexed Implementation- Part 2 127
4.12 MSE vs Adders for Fully Parallel Implementation with Single FRS- Part 1 . 127
4.13 MSE vs Adders for Fully Parallel Implementation with Single FRS- Part 2 . 130
4.14 MSE vs Adder-bits for Fully Parallel Implementation with Single FRS-Part 1 134

4.15 MSE vs Adder-bits for Fully Parallel Implementation with Single FRS-Part 2 134
4.16 MSE vs Adders for Time-Multiplexed Implementation with Single FRS-Part 1 136
4.17 MSE vs Adders for Time-Multiplexed Implementation with Single FRS-Part 2 136
4.18 MSE vs Adder-bits for Time-Multiplexed Implementation with Single FRS-Part 1 136
4.19 MSE vs Adder-bits for Time-Multiplexed Implementation with Single FRS-Part 2 137

LIST OF FIGURES

2.1 Signal-flow graph of the modified Arai 1D DCT 19
2.2 Hardware interpretation of an AI representation vector 22
2.3 Constant Coefficient Multiplication 34
2.4 Using shifting to multiply by 2 35
2.5 Example of multiple solutions for a given SCM 38
2.6 Example representing SCM circuits as graphs 40
2.7 All possible graph topologies for circuits with 1 to 4 adders (from [8]) . . . 41
2.8 Vertex generation equations 42
2.9 Modification of vertex values without changing the circuit behavior 42
2.10 Creating even numbers from odd vertices 43
2.11 Fractional Magnitude Edges 44

3.1 Example of adder-bits growth 47
3.2 SCM design tool 49
3.3 SCM design tool verification 50
3.4 Frequency distribution of SCMs 50
3.5 Adder types and hardware costs (adapted from [9]) 60
3.6 AI model and simulation design tool 63
3.7 Overview of time-multiplexed AI-based 2D DCT 65
3.8 Overview of fully parallel AI-based 2D DCT 66
3.9 Pipelined SCM architecture for S0 and S4 72
3.10 Pipelined SCM architecture for S1, S3, S5, S7 73
3.11 Pipelined SCM architecture for S2 and S6 74
3.12 MSE for 8-bit, Non-AI 81
3.13 MSE for 16-bit, Non-AI 83
3.14 MSE for 8-bit, Full AI Implementation 89
3.15 MSE for 16-bit, Full AI Implementation 90
3.16 Lowest MSE for 8-bit, Approach 0 AI Implementation 99
3.17 Lowest MSE for 16-bit, Approach 0 AI Implementation 101

3.18 Summary of Lowest MSEs for Coefficient Quantizations of 8- to 16-bits . . 101
3.19 Lowest MSE for 8-bit, Approach 2 AI Implementation 104
3.20 Lowest MSE for 16-bit, Approach 2 AI Implementation 105
3.21 Summary of Lowest MSEs for Coefficient Quantizations of 8- to 16-bits . . 106
3.22 Lowest MSE for 8-bit, Approach 4 AI Implementation 108
3.23 Lowest MSE for 8-bit, Approach 4 AI Implementation 110
3.24 Summary of Lowest MSEs for Coefficient Quantizations of 8- to 16-bits . . 110

4.1 Ranges of MSEs for 2D DCT Designs vs Quantization Bits 115
4.2 MSE vs Adders for Fully Parallel Implementation 116
4.3 MSE vs Adder-bits for Fully Parallel Implementation 118
4.4 MSE vs Adders for Time-Multiplexed Implementation 124
4.5 MSE vs Adder-bits for Time-Multiplexed Implementation 126
4.6 MSE vs Adders for Fully Parallel Implementation with Single FRS 128
4.7 MSE vs Adders for Fully Parallel Implementation with Single FRS (expanded) 129
4.8 MSE vs Adder-bits for Fully Parallel Implementation with Single FRS . . . 131
4.9 MSE vs Adder-bits for Fully Parallel Implementation with Single FRS (zoomed into AI designs) 132
4.10 MSE vs Adders for Time-Multiplexed Implementation with Single FRS . . 133
4.11 MSE vs Adder-bits for Time-Multiplexed Implementation with Single FRS 135

LIST OF ABBREVIATIONS

1D	-	One Dimensional
2D	-	Two Dimensional
AI	-	Algebraic Integer
ASIC	-	Application Specific Integrated Circuit
CODEC	-	COder-DECoder
CORDIC	-	COordinate Rotation DIgital Computer
CNRS	-	Centre National de la Recherche Scientifique
CSD	-	Canonic Signed Digits
CSE	-	Common Subexpression Elimination
DCT	-	Discrete Cosine Transform
DM	-	Dempster-Macleod
DSP	-	Digital Signal Processing
FFT	-	Fast Fourier Transform
FIR	-	Finite-duration Impulse Response
FPGA	-	Field Programmable Gate Array
FRS	-	Final Reconstruction Step
KCM	-	Constant Coefficient Multiplication
MCM	-	Multiple Constant Multiplication
MSE	-	Mean-Squared Error
MWLP	-	Multiple Word-Length Paradigm
SCM	-	Single Constant Multiplication
VHDL	-	VHSIC Hardware Description Language
VHSIC	-	Very High-Speed Integrated Circuits
VLSI	-	Very Large Scale Integration

CHAPTER 1

INTRODUCTION

1.1 Introduction

Sinusoidal transforms, such as the Fast Fourier Transform (FFT) and the Discrete Cosine Transform (DCT), are widely used in science and engineering. The DCT has seen extensive application in image and video compression, for such standards as JPEG, MPEG-1, MPEG-2, H.261, H.263, and H.264 [10, 11]. In these applications, the DCT component of a CODEC is used to process 8x8 sub-images, with very high throughput requirements. It is desirable that the computations of the DCT be completed with high speed, high accuracy, low circuit area, and low energy [12]. Accordingly, VLSI implementations for the DCT, and other sinusoidal transforms, has been an active research area [13–28].

A problem with computing the DCT, as with other sinusoidal transforms, is multiplication by irrational numbers, which are difficult to encode in binary representations. Floating-point units can be used to compute these multiplications with high accuracy, but they are known to be costly in terms of circuit area and energy consumption, compared to fixed-point units [29]. A subset of fixed-point multipliers are multiplierless Constant (K) Coefficient Multipliers (KCMs), as used in this research. In this thesis, KCMs are divided into two types, Single Constant Multipliers (SCMs) [30] and Multiple Constant Multipliers (MCMs) [31]. In these "multiplierless" multipliers, "shift-and-add" circuits are used to efficiently implement the multiply operation. According to Dempster, traditionally, power consumption has been considered proportional to the number of adders in shift-and-add based SCM circuits, but when comparing circuits with similar adder cost, significant

variations in power consumption can be expected, due to switching activity within each specific circuit [30]. Regardless of whether fixed-, or floating-point units are used, the irrational numbers must be approximated, whether through truncation or rounding [32, 33], which introduces errors that must be managed by the designer. A further problem with approximated constants is the propagation of errors throughout the computation of the DCT [34, 35], which has led to efforts to address it [32–35].

In this thesis, Algebraic Integer (AI) based encoding is used to exactly represent the irrational numbers, resulting in approximation-free computations [36, 37]. In AI representation, irrational numbers are mapped to a vector or array of integers, which enable exact arithmetic operations. However, decoding from the AI representation results in the introduction of approximation errors, at the end of the transform [36]. This decoding step is referred to as the Final Reconstruction Step (FRS), and it is very computationally expensive.

This thesis addresses the computational complexity problem of the FRS, through the development of new algorithms to reduce the number and size of the computational circuits required to implement the AI DCT. In this research, custom software tools were developed to design optimal KCMs and to calculate the:

1. Mean-Squared Error (MSE), and

2. the number and size of the adder and subtractor circuits.

At the time this research was initiated, software tools for designing optimal KCMs did not exist, nor did they exist for modeling and analyzing AI-based hardware.

Two basic classes of architectures are developed and analyzed in this thesis, and they are compared to the traditional fixed-point implementation of the DCT, as a benchmark.

1.2 Literature Review

A small, but emerging body of work has focused on applications of AI to sinusoidal transforms such as the discrete Fourier transform (DFT) [38], discrete cosine transform (DCT) [2, 3, 6, 36, 39–41], discrete Hartley transform (DHT) [41], and the Haar transform [41]. Other applications have focused on digital filtering with the frequency-sampling filter (FSF) [42] and the finite-impulse response (FIR) filter [43].

Historically, the use of number theoretic techniques in signal processing began with the works of Agarwal and Burrus [44], Reed and Truong [45], and Pollard [46] in the mid-1970's and continued to the beginning of the 1980's with Dubois and Venetsanopoulis [47, 48], followed by Cozzens and Finkelstein [38] in the mid-1980's. An application in 1990 by Games, Moulin, and Rushanan [43] was followed by renewed interest in the late 1990's to the present by Dimitrov and Jullien [2, 6, 7, 36, 39–41] and Meyer-Base and Taylor [42]. Dimitrov introduced the concept of algebraic integer representation, where multiplication by sinusoidal functions, leading to irrational constants, could alternatively be encoded as operations on integer vectors, first reported in [36]. Particularly noteworthy to this area was the development of multidimensional AI by Dimitrov [40], which allowed more efficient AI encodings, which led to simpler AI operations. The works of Dimitrov have utilized mathematical operations on AI representations of data, which allowed for the claim of "error-free" operations, meaning that quantization error has been eliminated, until the FRS is computed. The FRS introduces quantization error to the final result.

The research presented in this thesis focuses deeply on the most studied and most recent AI approach, applied to the 2D Arai DCT, which has been previously explored by Wahid [2, 3, 6, 37, 49–52], and recently, Pradini [4, 5]. Although these valuable works have explored several interesting approaches towards the application of AI to the 2D DCT, some limitations identified below, prevent these works from clearly establishing the merit of AI versus non-AI approaches. These limitations include:

- suboptimal selection of constant multipliers, leading to increased quantization error for the algorithm;
- suboptimal encoding of constant multipliers, leading to excessive adder count; and
- counting of the number of physical adders in a design, as opposed to the number of addition operations, leading to an exaggeration of the purported benefit of AI for reduced computational complexity.

Wahid has focused on investigating and developing multidimensional (1D, 2D, and 3D) AI algorithms that minimize the physical adder count of the required circuits. He has also provided innovative approaches to the reuse of adders within his circuits, through recursion or multiplexing [1–3, 6, 7, 51]. Pradini has developed an innovative 2D AI algorithm for the 2D DCT that requires no transposition memory, relying on the 2D AI 2D DCT of Wahid. However, in terms of addressing the question of whether AI is superior to non-AI for this application, some limitations of the implementations of the algorithms of Wahid and Pradini preclude a direct answer, which this thesis seeks to address.

To demonstrate the areas for improvement in the state of the art, Tables 1.1 to 1.13 detail the constants used in the literature for the 2D AI 2D DCT, with comparison to the known optimal Dempster-Macleod (DM) algorithm [8,30,53] to design single constant multipliers (SCMs). As will be detailed in Section 3.1, the DM algorithm consists of an exhaustive search to find all possible SCMs for a given integer constant, and returns the SCM with the least number of adders, which is the optimality criterion for SCMs by Dempster and Macleod. A short comment also is included within the tables, to summarize the item(s) of note. For each citation within the table title, the Tables 1.1 to 1.13 show:

- the cited authors' label for the SCM;
- the cited authors' stated desired accuracy (if stated), in bit length;
- the exact value that the cited author attempts to approximate within his AI design;

- the cited authors' formula (i.e. algorithm) for the approximated value, given as a summation of binary terms;
- the numerical result of the cited authors' approximation, which also shows the equivalent integer value of the approximated value, scaled by a binary factor;
- the numerical result of the corrected (by the author of this thesis) integer value, scaled by a binary factor, to more accurately compute the approximation within the cited authors' specified accuracy;
- the percentage difference between the cited authors' approximated value and exact value;
- the percentage difference between the thesis authors' approximated value and exact value;
- the number of adders (i.e. the cost) required by the cited authors' SCM, given by the number of additions and subtractions in the approximation;
- the number of adders (i.e. the cost) required to implement the thesis author's approximation, using the DM algorithm (described in Section 3.1); and
- a short comment summarizing the important features for each comparison between the cited authors' and thesis author's SCM designs. In some cases, the cited authors' SCMs have been well chosen, meaning that the integer approximation could not be improved for the accuracy in the given bit length, and that the implementation could not be improved by using the DM algorithm.

Pradini [4, 5] follows Wahid's [2, 3] design approach for the DCT and AI components of his design, at 10-bits of approximation accuracy for z-values, as seen in Tables 1.5 and 1.6. Pradini created a 2D DCT image compression system, which he compared to a conventional, separable implementation. Comparison between Pradini's two implementations

Table 1.1: Constant approximations for [1]

z-Label	z_1
Accuracy	16-bit
Exact Value	$1+\sqrt{2}$
Wahid Approximation	$2+2^{-1}-2^{-3}+2^{-5}+2^{-7}+2^{-13}$
Wahid Value	$19777/2^{13} = 2.41418457$
SCM by DM Value	$19777/2^{13} = 2.41418457$
Wahid % Difference	-0.00120
SCM % Difference	-0.00120
Wahid Adder Cost	5
DM SCM Adder Cost	4
Comment	Wahid's integer constant, 19777, could have been encoded with one fewer adder using DM. Otherwise, it was well-chosen.

Table 1.2: Constant approximations for [1]

z-Label	z_2
Accuracy	16-bit
Exact Value	$\sqrt{2+\sqrt{2}}$
Wahid Approximation	$2-2^{-3}-2^{-5}+2^{-8}+2^{-13}-2^{-15}$
Wahid Value	$60547/2^{15} = 1.847747803$
SCM by DM Value	$60547/2^{15} = 1.847747803$
Wahid % Difference	-0.000610
SCM % Difference	-0.000610
Wahid Adder Cost	5
DM SCM Adder Cost	4
Comment	Wahid's integer constant, 60547, could have been encoded with one fewer adder using DM. Otherwise, it was well-chosen.

(AI and non-AI) is not meaningful because they have radically different algorithms, of which the AI component is only one part. Thus, the contribution of the AI component is not possible to separately assess for its merit over non-AI approaches.

Pradini [4, 5] implemented the scaling corrections factors for the outputs of the Arai

Table 1.3: Constant approximations for [1]

z-Label	z_3
Accuracy	16-bit
Exact Value	$\sqrt{2}$
Wahid Approximation	$2-2^{-1}-2^{-3}+2^{-5}+2^{-7}+2^{-13}$
Wahid Value	$23170/2^{14} = 1.41418457$
SCM by DM Value	$23170/2^{14} = 1.41418457$
Wahid % Difference	-0.00210
SCM % Difference	-0.00210
Wahid Adder Cost	5
DM SCM Adder Cost	4
Comment	Wahid's integer constant, 23170, could have been encoded with one fewer adder using DM. Otherwise, it was well-chosen.

Table 1.4: Constant approximations for [2, 3]

z-Label	z	z
Accuracy	10-bit	12-bit
Exact Value	$\sqrt{2+\sqrt{2}}$	$\sqrt{2+\sqrt{2}}$
Wahid Approximation	$2-2^{-3}-2^{-5}+2^{-8}$	$2-2^{-3}-2^{-5}+2^{-8}$
Wahid Value	$473/2^8 = 1.84765625$	$1892/2^{10} = 1.84765625$
SCM by DM Value	$473/2^8 = 1.84765625$	$1892/2^{10} = 1.84765625$
Wahid % Difference	-0.00556	-0.00556
SCM % Difference	-0.00556	-0.00556
Wahid Adder Cost	3	3
DM SCM Adder Cost	3	3
Comment	Wahid's approximation and encoding were well done.	Wahid's approximation and encoding were well done.

DCT algorithm [54] using the s-values in Tables 1.7 to 1.10.

Although Wahid's and Pradini's work contained areas for improvement, they are nonetheless very valuable for the exploration of using AI for the 2D DCT. Their use of suboptimal selection of SCMs and suboptimal encoding of SCMs leads to unrepresentative quantiza-

Table 1.5: Constant approximations for [2–5]

z-Label	z_1	z_1
Accuracy	10-bit	12-bit
Exact Value	$\sqrt{2+\sqrt{2}}+\sqrt{2-\sqrt{2}}$	$\sqrt{2+\sqrt{2}}+\sqrt{2-\sqrt{2}}$
Wahid Approximation	$2+2^{-1}+2^{-3}-2^{-6}$	$2+2^{-1}+2^{-3}-2^{-6}+2^{-8}-2^{-10}$
Wahid Value	$334/2^7 = 2.609375$	$2675/2^{10} = 2.612304688$
SCM by DM Value	$334/2^7 = 2.609375$	$2676/2^{10} = 2.61328125$
Wahid % Difference	-0.144	-0.0314
SCM % Difference	-0.144	0.00594
Wahid Adder Cost	3	5
DM SCM Adder Cost	3	3
Comment	Wahid's approximation and encoding were well done.	Wahid's approximation and encoding could have been improved by an order of magnitude in accuracy, with 2 fewer adders, if integer constant 2676 was selected (instead of 2675).

Table 1.6: Constant approximations for [2–5]

z-Label	z_2	z_2
Accuracy	10-bit	12-bit
Exact Value	$\sqrt{2+\sqrt{2}}-\sqrt{2-\sqrt{2}}$	$\sqrt{2+\sqrt{2}}-\sqrt{2-\sqrt{2}}$
Wahid Approximation	$1+2^{-4}+2^{-6}-2^{-8}$	$1+2^{-4}+2^{-6}-2^{-8}$
Wahid Value	$277/2^7 = 1.08203125$	$1108/2^{10} = 1.08203125$
SCM by DM Value	$277/2^7 = 1.08203125$	$1108/2^{10} = 1.08203125$
Wahid % Difference	-0.0333	-0.0333
SCM % Difference	-0.0333	-0.0333
Wahid Adder Cost	3	3
DM SCM Adder Cost	3	3
Comment	Wahid's approximation and encoding were well done.	Wahid's approximation and encoding were well done.

Table 1.7: Constant approximations for [4, 5]

z-Label	s_1	s_2
Exact Value	$1/(2\sqrt{2})$	$1/(4cos(\pi/16))$
Pradini Approximation	$2^{-2}+2^{-3}-2^{-5}$	2^{-2}
Pradini Value	$44/2^7 = 0.34375$	$32/2^7 = 0.25$
SCM by DM Value	$45/2^7 = 0.3515625$	$33/2^7 = 0.2578125$
Pradini % Difference	-2.77	-1.92
SCM % Difference	-0.563	1.14
Pradini Adder Cost	2	0
DM SCM Adder Cost	2	1
Comment	The accuracy of Pradini's integer approximation could have been improved at the same adder cost, by selecting 45 (instead of 44).	The accuracy of Pradini's integer approximation could have been improved at the cost of an additional adder, by using the DM algorithm.

Table 1.8: Constant approximations for [4, 5]

z-Label	s_3	s_4
Exact Value	$1/(4cos(2\pi/16))$	$1/(4cos(3\pi/16))$
Pradini Approximation	2^{-2}	$2^{-2}+2^{-4}$
Pradini Value	$32/2^7 = 0.25$	$40/2^7 = 0.3125$
SCM by DM Value	$35/2^7 = 0.2734375$	$38/2^7 = 0.296875$
Pradini % Difference	-7.61	3.93
SCM % Difference	1.05	-1.26
Pradini Adder Cost	0	1
DM SCM Adder Cost	2	2
Comment	The accuracy of Pradini's integer approximation could have been improved at the cost of two adders.	The accuracy of Pradini's integer approximation could have been improved at the cost of one adder.

tion errors and unrepresentative adder counts, for the algorithms being analyzed. Furthermore, Wahid's counting of the number physical adders, instead of the number of times they are used, leads to a greatly reduced apparent computational complexity, than what is actually present in the algorithm. Wahid's and Pradini's use of the FRS after the first 1D DCT introduces quantization error within the AI algorithm, thus eliminating the "error-

Table 1.9: Constant approximations for [4, 5]

z-Label	s_5	s_6
Exact Value	$1/(5cos(4\pi/16))$	$1/(5cos(3\pi/16))$
Pradini Approximation	$2^{-2}+2^{-3}-2^{-5}$	$2^{-1}-2^{-4}$
Pradini Value	$44/2^7 = 0.34375$	$56/2^7 = 0.4375$
SCM by DM Value	$45/2^7 = 0.3515625$	$58/2^7 = 0.453125$
Pradini % Difference	-2.77	-2.77
SCM % Difference	-0.563	0.697
Pradini Adder Cost	2	1
DM SCM Adder Cost	2	2
Comment	The accuracy of Pradini's integer approximation could have been improved at the same adder cost, by selecting 45 (instead of 44).	The accuracy of Pradini's integer approximation could have been improved at the cost of one adder by selecting 58, instead of 56, for the integer approximation.

Table 1.10: Constant approximations for [4, 5]

z-Label	s_7	s_8
Exact Value	$1/(5cos(6\pi/16))$	$1/(6cos(7\pi/16))$
Pradini Approximation	$2^{-1}+2^{-3}-2^{-5}$	$1+2^{-2}-2^{-5}$
Pradini Value	$76/2^7 = 0.59375$	$164/2^7 = 1.28125$
SCM by DM Value	$84/2^7 = 0.65625$	$164/2^7 = 1.28125$
Pradini % Difference	-9.11	-0.0162
SCM % Difference	0.454	-0.0162
Pradini Adder Cost	2	2
DM SCM Adder Cost	2	2
Comment	The accuracy of Pradini's integer approximation could have been improved the same adder cost by selecting 84, instead of 76, for the integer approximation.	Pradini's approximation and encoding were well done.

free" claim, because a second 1D DCT is required to compute the complete 2D DCT, as a separable transform. The use of only SCMs penalizes the AI methods applied by Wahid and Pradini as well, where the use of Multiple Constant Multiplication (MCM) would lead

Table 1.11: Constant approximations for [6]

z-Label	z_3
Accuracy	8-bit
Exact Value	$\sqrt{2}/2$
Wahid Approximation	$1-2^{-2}-2^{-4}+2^{-6}$
Wahid Value	$90/2^7 = 0.703125$
SCM by DM Value	$91/2^7 = 0.7109375$
Wahid % Difference	-0.563
SCM % Difference	0.542
Wahid Adder Cost	3
DM SCM Adder Cost	3
Comment	The accuracy of Wahid's integer approximation could have been increased by using 91 (instead of 90), at the same adder cost. Or 90 could have been encoded with only 2 adders, using DM.

to a significant reduction of the FRS computational complexity, as will be shown in this thesis.

1.3 Motivation for this Research

The study of the relative advantages and disadvantages of the use of AI versus non-AI approaches have been lacking in the literature. Particularly, the highly punitive cost of the FRS step has not been fully investigated, although it has been suggested in [55]. Furthermore, a full, comparative error analysis between AI and non-AI approaches has been lacking.

The research presented in this thesis provides insight into the merits of AI versus non-AI approaches by employing consistent design approaches, including the use of optimal KCMs (SCM or MCM) and the use of the same basic DCT algorithm (Arai DCT), for both AI and non-AI designs for the 2D Arai DCT. This enables strong comparisons to be made for each design's MSE, adder cost, and adder-bits cost.

Table 1.12: Constant approximations for [7]

z-Label	z
Accuracy	8-bit
Exact Value	$2+\sqrt{2+\sqrt{2}}$
Wahid Approximation	$4-2^{-3}-2^{-5}+2^{-7}$
Wahid Value	$1972/2^9 = 3.8515625$
SCM by DM Value	$1970/2^9 = 3.84765625$
Wahid % Difference	0.0988
SCM % Difference	-0.00267
Wahid Adder Cost	3
DM SCM Adder Cost	3
Comment	Wahid's integer approximation accuracy could have been greatly improved the same adder cost, by using 1970 instead of 1972, for the integer constant.

Table 1.13: Constant approximations for [7]

z-Label	z_1	z_2
Accuracy	8-bit	8-bit
Exact Value	$\sqrt{2+\sqrt{2}}$	$\sqrt{2-\sqrt{2}}$
Wahid Approximation	$2-2^{-3}-2^{-5}+2^{-8}$	$1-2^{-2}+2^{-6}-2^{-11}$
Wahid Value	$1892/2^{10} =$ 1.84765625	$1567/2^{11} =$ 0.765136719
SCM by DM Value	$1892/2^{10} =$ 1.84765625	$1567/2^{11} =$ 0.765136719
Wahid % Difference	-0.00556	0.0301
SCM % Difference	-0.00556	0.0301
Wahid Adder Cost	3	3
DM SCM Adder Cost	3	3
Comment	Wahid's approximation and encoding were well done.	Wahid's approximation and encoding were well done.

1.4 Contributions of the Thesis

1. *In this work, a class of architectures is presented that utilizes the FRS only at the end of the 2D DCT, as reported in [55–57]. Furthermore, this architecture is imple-*

mented using optimal single constant coefficient multipliers, of which a 12-bit implementation was reported in [55–57]. This architecture is also implemented using multiple constant coefficient multipliers, which are shown to provide reductions in computational complexity, compared to single constant multipliers. These AI architectures for the 2D DCT are developed for constant coefficient quantizations covering the range of 8- to 16-bits. These architectures are further developed to include fully parallel and time-multiplexed implementations. These architectures are also developed with a version that uses a single FRS, to further reduce the number of physical adders. Previously reported architectures for AI applied to the 2D Arai DCT employed suboptimal selection and implementation of constant coefficient multipliers, and a suboptimal architecture for employing the FRS after the first 1D DCT and again at the end of the second 1D DCT, instead of just at the end of the complete 2D DCT. This contribution eliminates the intermediate FRS at the end of the first 1D DCT and incorporates superior methods for designing constant coefficient multipliers, than what was used in the state of the art AI 2D DCT. In total, the 72 AI designs were evaluated in this thesis for Contribution 1, including:

(a) 9 for single-constant coefficient multipliers in the range of 8- to 16 bits, for a fully parallel AI implementation, with a full FRS;

(b) 9 for single-constant coefficient multipliers in the range of 8- to 16 bits, for a time-multiplexed AI implementation, with a full FRS;

(c) 9 for single-constant coefficient multipliers in the range of 8- to 16 bits, for a fully parallel AI implementation, with a single FRS;

(d) 9 for single-constant coefficient multipliers in the range of 8- to 16 bits, for a time-multiplexed AI implementation, with a single FRS;

(e) 9 for multiple-constant coefficient multipliers in the range of 8- to 16 bits, for a fully parallel AI implementation, with a full FRS;

(f) 9 for multiple-constant coefficient multipliers in the range of 8- to 16 bits, for a time-multiplexed AI implementation, with a full FRS;

(g) 9 for multiple-constant coefficient multipliers in the range of 8- to 16 bits, for a fully parallel AI implementation, with a single FRS; and

(h) 9 for multiple-constant coefficient multipliers in the range of 8- to 16 bits, for a time-multiplexed AI implementation, with a single FRS.

To facilitate the evaluation of these architectures in Contribution 1, a series of non-AI based designs were developed, using the optimal single constant coefficient multipliers, with constant coefficient quantizations covering the range of 8- to 16-bits, in order to provide a consistent basis for comparison of results in terms of MSE, adder cost, and adder-bits cost. For a fair comparison, these non-AI designs were created using the same SCM tools (MCM multipliers are not possible for these designs) as for the AI designs. This allowed the non-AI and AI design comparisons to isolate the only relevant design difference, namely the application of AI, rather than using the improved SCM/MCM algorithms for AI only. In total, 18 non-AI designs were evaluated, including:

(a) 9 for single-constant coefficient multipliers in the range of 8- to 16 bits, for a non-AI fully parallel AI implementation, as a reference; and

(b) 9 for single-constant coefficient multipliers in the range of 8- to 16 bits, for a non-AI time-multiplexed AI implementation, as a reference.

In summary, Contribution 1 advances the state of the art in the 2D AI 2D DCT by the creation of a new architecture, characterized by the use of

- optimal single constant coefficient multiplication,
- by multiple constant coefficient multiplication, and
- by processing the FRS only at the end of the DCT.

2. *An alternative class of architectures for the 2D Arai DCT is developed using simplifying decompositions, to simplify the design of the FRS. Three variations of the new, simplified architecture are designed for the range of 8- to 16-bits of coefficient quantization, using the optimal single constant coefficient multipliers and multiple constant multipliers. Following Contribution 1, the architectures in Contribution 2 are implemented for the range of 8- to 16-bits of constant coefficient quantization, and include fully parallel, time-multiplexed, and single FRS variations.* The high cost of the FRS is a serious design challenge for the use of AI in the 2D DCT and the architectures presented for Contribution 2 provide significant improvements in terms of MSE, adder cost, and adder-bits cost, compared to Contribution 1. In total, the designs evaluated in this thesis for Contribution 2 follow the same pattern as for Contribution 1, meaning that they have been designed in the fully parallel and time-multiplexed implementations, with either a full or single FRS, covering a range of constant coefficient quantizations of 8- to 16-bits, using the superior MCM multipliers (and not SCM, if possible). In total, the number of AI designs evaluated for Contribution 2 include:

   (a) 648 for Approach 0 (MCM only);

   (b) 1944 for Approach 2 (SCM/MCM hybrid); and

   (c) 1944 for Approach 4 (MCM only).

   As for Contribution 1, the Contribution 2 designs are compared against the 18 non-AI designs.

In summary, Contribution 2 further advances the state of the art in the 2D AI 2D DCT by:

- the use of simplifying decompositions to simplify the FRS; and
- reformulating the FRS equations to produce more design alternatives; while using the SCM and MCM techniques of Contribution 1.

Contribution 2 shows significant improvement over Contribution 1, and shows significant reduction in hardware cost, compared to the non-AI architecture.

1.5 Structure of the Thesis

This thesis is organized in five chapters. Chapter 2 provides background into the DCT, algebraic integers, and constant integer multiplication. It also provides a tutorial in the application of algebraic integers in the signal processing relevant for this thesis.

Chapter 3 discusses the software tools that were used or developed to facilitate this research. This chapter also illustrates the advantage of MCM over SCM. It also discusses the AI and non-AI architectures, including the development of the simplified AI architectures (Contribution 2). This chapter introduces the 12-bit SCM-based FRS architectures of Contribution 1, along with a detailed example of its hardware implementation.

Chapter 4 presents the results of this research effort by providing a comparative analysis of the best designs developed in this thesis, including their MSE performance and hardware cost for both Contributions 1 and 2.

Chapter 5 is the conclusion and it provides a summary of the thesis.

CHAPTER 2

BACKGROUND

2.1 Introduction

This chapter is a tutorial on the background material relevant to understanding this thesis. This background includes the Arai DCT, the 2D Arai DCT, adder circuits, SCMs, MCMs, and AI methods.

2.2 Discrete Cosine Transform

The most studied application of AI in the literature is the 1-dimensional (1D) forward and inverse DCT. The MATLAB version of the forward 1D DCT [58] was used as a reference and it is defined as:

$$y(k) = w(k) \sum_{n=0}^{N-1} x(n) \cos\left[\frac{\pi(2n+1)k}{2N}\right]; \quad 0 \leq k \leq N-1 \tag{2.1}$$

$$w(k) \quad = \quad \begin{cases} 1/\sqrt{N} & k = 0 \\ \sqrt{2/N} & 1 \leq k \leq N-1 \end{cases}$$

where $x(n)$ is an input sequence of length N.

The 2D version of the DCT is used extensively in image and video compression. The

MATLAB version of the forward 2D DCT [59] was used as a reference, and it is defined as:

$$B(p,q) = \alpha(p)\alpha(q)\sum_{n=0}^{M-1}\sum_{m=0}^{N-1} X(m,n)\cos\left[\frac{\pi(2m+1)p}{2M}\right]\cos\left[\frac{\pi(2n+1)q}{2N}\right]; \quad 0 \le p \le M-1; 0 \le q \le N-1 \tag{2.2}$$

$$\alpha(p) = \begin{cases} 1/\sqrt{M} & p = 0 \\ \sqrt{2/M} & 0 \le p \le M-1 \end{cases}$$

$$\alpha(q) = \begin{cases} 1/\sqrt{N} & q = 0 \\ \sqrt{2/N} & 0 \le k \le N-1 \end{cases}$$

where $X(n,m)$ is an $N \times M$ array of input data.

The 2D DCT is a separable transform and is often computed as a 1D DCT on the rows and another 1D DCT on the columns. This approach requires transposition memory to store the intermediate results between transforms. Computation of equation (2.1) can be seen by inspection to require 64 additions and 56 multiplications. Fast algorithms for the DCT exist [60], as they do for the Fourier Transform (e.g. the FFT), to reduce its computational complexity. Another approach to compute the 2D DCT is to directly develop a fast algorithm for it [60], thus eliminating the transposition memory. Another approach to eliminating transposition memory is to use two 1D DCT's, where the second 1D DCT can accept the data order from the first 1D DCT (the first and second 1D DCTs each have a different structure), without intermediate storage [61].

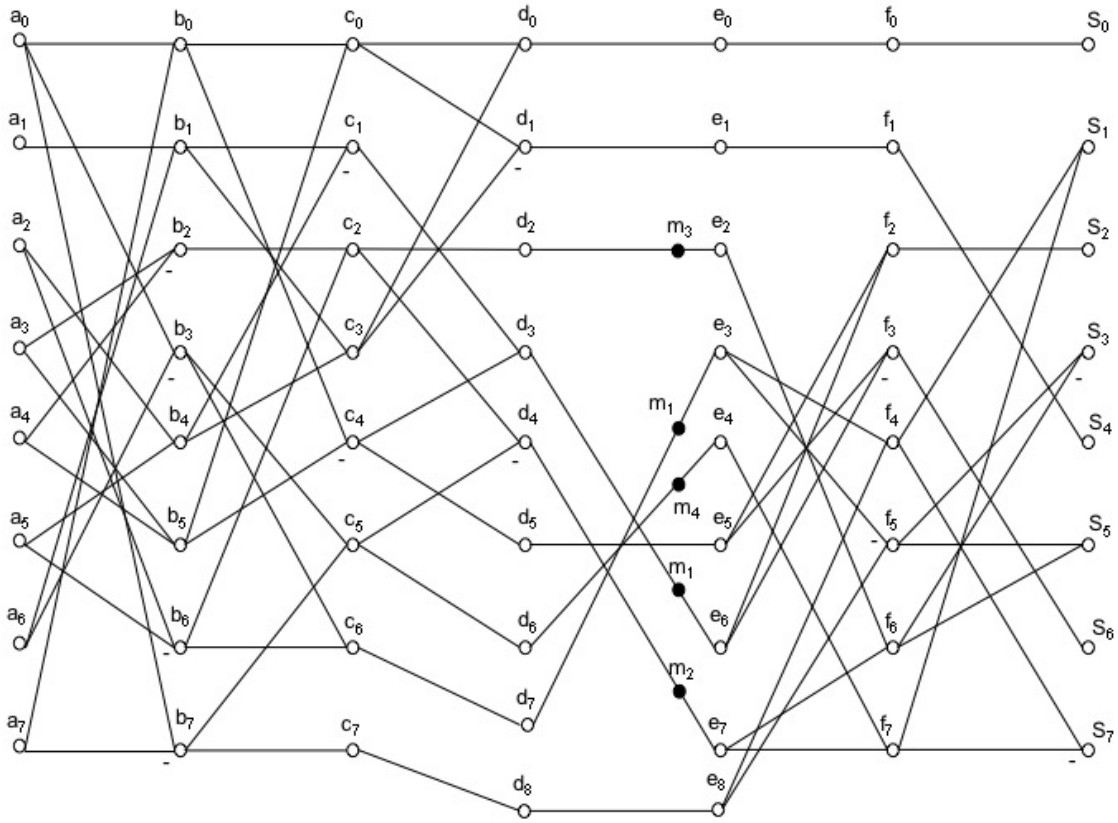

Figure 2.1: Signal-flow graph of the modified Arai 1D DCT

2.3 Fast Algorithm for the Discrete Cosine Transform

Arai has created the 1D DCT algorithm with the least known number of multiplications [54]. Figure 2.1 shows the signal flow graph of Arai's fast 1D DCT, which has been modified by Kovac and Ranganathan [62] to reduce the number of two's complement additions. The signals flow left-to-right, thus the inputs are the nodes denoted by a_0 to a_7 and the outputs are nodes S_0 to S_7. Note that the outputs of the DCT are also referred to as output "coefficients." The intermediate nodes represent addition if two lines enter the node from the left of the node. A minus sign indicates negation of the signal entering the node. Column 2 of Table 2.1 shows the values of the multipliers, m_1 to m_4. In total, as can be seen

Table 2.1: Arai 1D DCT multipliers

Multiplier	Value
m_1	$\cos(4\pi/16) = \sqrt{2}/2$
m_2	$\cos(6\pi/16) = \left(\sqrt{2-\sqrt{2}}\right)/2$
m_3	$\cos(2\pi/16) - \cos(6\pi/16) = \left(\sqrt{2+\sqrt{2}} - \sqrt{2-\sqrt{2}}\right)/2$
m_4	$\cos(2\pi/16) + \cos(6\pi/16) = \left(\sqrt{2+\sqrt{2}} + \sqrt{2-\sqrt{2}}\right)/2$

Table 2.2: Scaling Factors for Arai 1D DCT

Output	Constant
S_0	$1/2\sqrt{2}$
S_1	$1/4cos(\pi/16)$
S_2	$1/4cos(2\pi/16)$
S_3	$1/4cos(3\pi/16)$
S_4	$1/4cos(4\pi/16)$
S_5	$1/4cos(5\pi/16)$
S_6	$1/4cos(6\pi/16)$
S_7	$1/4cos(7\pi/16)$

by inspection of Figure 2.1, there are 29 additions[1] and 5 multiplications (m_1 is used twice) in the Arai algorithm. These improvements for computational complexity are obtained because the Arai algorithm is a scaled algorithm, meaning that its outputs must be scaled by the factors shown in Table 2.2, to obtain the correct DCT values.

An alternative fast 1D DCT has been proposed by Linzer and Feig [63], which was reported to have 26 additions and 10 multiplications.

2.4 Algebraic Integers

Algebraic numbers are the roots of integer coefficient polynomials. Algebraic integers are the roots of monic polynomials, and they are a subset of the algebraic numbers [64]. An example of an algebraic number is $\cos(\pi/4) = \sqrt{2}/2$, which is the root of $4 \cdot x^2 + 0 \cdot x - 2 = 0$. An example of an algebraic integer is $2 \cdot \cos(\pi/4) = \sqrt{2}$, which is the root of

[1]Note the since adders and subtractors have approximately the same hardware cost, the "number of adders" usually refers to the "number of adders and subtractors" in the literature. This will be the convention used in this thesis.

$x^2 + 0 \cdot x - 2 = 0$.

The method for the development of specific algebraic integer representations has not been disclosed in the literature. Rather, these algebraic integer representations have been developed on an empirical basis, using mathematical insights coupled with exhaustive searches to find suitable representations, per personal communications with Dimitrov. In this thesis, algebraic integer representations, as reported in [3], have been used as the starting point for the development of the new architectures presented in this thesis.

2.4.1 One-dimensional Algebraic Integers

2.4.1.1 Encoding

To represent the multipliers, m_1 to m_4, of Table 2.1, in algebraic integer representation, a z-value must be chosen that will allow the m_1 to m_4 values to be encoded by an integer polynomial. The constraint within this method is that the z-value be an algebraic integer. The initial research by Dimitrov in the selection of z-values used one of the multipliers present in the target algorithm, such as for the DCT [41]. Later research used alternative z-values to construct AI representations for the multipliers with the algorithm [3]. Following [3], let $z = 2 \cdot \cos(\pi/8) = \sqrt{2+\sqrt{2}}$, then using the polynomial expansion:

$$f(z) = a_0 + a_1 z^1 + a_2 z^2 + a_3 z^3 \tag{2.3}$$

we can write m_1 to m_4 in terms of the 1D AI encodings (for a_0 to a_3) given in column 3 of Table 2.3. These AI encodings were found by Dimitrov using an exhaustive search. This is the most attractive 1D AI encoding in the literature, because the representation is sparse (i.e. there are only two non-zero elements in each encoding vector), and because the non-zero elements are powers of two, this leads to efficient hardware implementation. Note that there is a scaling factor of 2 on the encoding, in order to create an algebraic integer value for z (not shown, as is customary in the AI literature), so $f(z)$ must be divided by 2

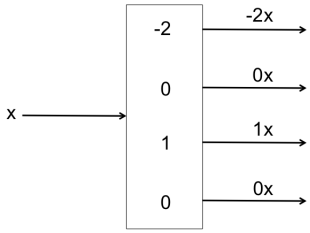

Figure 2.2: Hardware interpretation of an AI representation vector

to obtain the final result.

For example, in Table 2.3, the AI encoding for m_1 is represented as [-2 0 1 0]. The hardware interpretation of the AI representation vector is shown in Figure 2.2, where an input channel x (e.g. an input bus), is transformed into 4 output channels with the indicated, scaled values (-2x, 0x, 1x, 0x). The example in the figure shows an AI multiplication operation on x by m_1. Because of the sparseness of the output (-2x, 0x, 1x, 0x), only the two non-zero values actually need to be transmitted to the next stage, i.e. (-2x, 1x). Other 1D AI encodings for the DCT have had up to 4 non-zero elements for the AI representation vector [41], e.g. (-2x, 0x, 9x, 0x, -6x, 0x, 1x, 0x, 0x) or (0x, -7x, 0x, 14x, 0x, -7x, 0x, 1x, 0x), which lead to more complex architectures (i.e. more outputs to process). Since all the non-zero AI encodings of Table 2.3 are powers of two, very efficient hardware implementations are possible, since a binary shift can be implemented with only with wires, and is considered "cost-free." In contrast, it is noted that in [41], the AI representation vectors contain non-zero elements that are not powers of two, e.g. 9x, -6x, -7x, 14x, etc., necessitating further hardware to compute the value, beyond a wired shift.

To verify that m_1 is correctly encoded, we must apply the AI encoding vector for m_1 to Equation 2.3. Using Table 2.3, the AI encoding for m_1 is [-2 0 1 0] (i.e. $a_0 = -2$, $a_1 = 0$, $a_2 = 1$, and $a_3 = 0$), and recalling that $z = \sqrt{2+\sqrt{2}}$, and recalling that the result must be scaled by the factor of 2, we can see from Equation 2.4 that AI encoding for m_1 does indeed represent $\sqrt{2}/2$.

Table 2.3: Arai 1D DCT multipliers with 1D AI encodings

Multiplier	Value	1D AI Encoding $[a_0\ a_1\ a_2\ a_3]$
m_1	$\cos(4\pi/16) = \sqrt{2}/2$	[-2 0 1 0]
m_2	$\cos(6\pi/16) = \left(\sqrt{2-\sqrt{2}}\right)/2$	[0 -3 0 1]
m_3	$\cos(2\pi/16) - \cos(6\pi/16) = \left(\sqrt{2+\sqrt{2}} - \sqrt{2-\sqrt{2}}\right)/2$	[0 4 0 -1]
m_4	$\cos(2\pi/16) + \cos(6\pi/16) = \left(\sqrt{2+\sqrt{2}} + \sqrt{2-\sqrt{2}}\right)/2$	[0 -2 0 1]

$$\frac{f(z)}{2} = \frac{-2+0z^1+1z^2+0z^3}{2} = \frac{-2+\left(\sqrt{2+\sqrt{2}}\right)^2}{2} = \frac{\sqrt{2}}{2} = m_1 \tag{2.4}$$

The numbers produced by $f(z)$ form a ring denoted as $Z\left[\sqrt{2+\sqrt{2}}\right]$. In Section 2.4.1.2, it will be important to note that $\sqrt{2+\sqrt{2}}$ is the root of the integer coefficient polynomial

$$z^4 - 4z^2 + 2 = 0. \tag{2.5}$$

This integer coefficient polynomial will be used in modulo reduction for polynomial multiplication.

2.4.1.2 Operations

Addition/Subtraction between an AI and an Integer: Addition or subtraction between the two number representations is between the integer and the a_0 element (i.e. the constant) of the AI, except the integer must be shifted to accommodate for the scaling factor (i.e. 2). For example, adding 7 to m_1 of Table 2.1 is computed by:

7 + [-2 0 1 0] = [(-2+7<<1) 0 1 0] = [12 0 1 0], where <<1 represents a binary shift of 1, or a multiplication by 2. This binary shift was necessary to compensate for the initial scaling described in paragraph two of Section 2.4.1.1.

As another example, again using m_1 of Table 2.1, $1+\sqrt{2}/2$ can be transformed as an AI operation by:

1+[-2 0 1 0] = [(-2+1<<1) 0 1 0] = [0 0 1 0].

Addition/Subtraction between two AIs: Addition or subtraction between two AIs is element-by-element, for example:

[1 0 4 2] + [2 0 1 0] = [3 1 4 3].

Multiplication between an AI and an Integer: Multiplication is performed between the integer and all the elements of the AI as a scalar operation, for example:

7 x [-2 0 1 0] = [-14 0 7 0].

Multiplication between two AIs: Multiplication between two AIs is performed as between polynomials, for example:

Consider $m_1 = -2 + z^2$ and $m_4 = -2z + z^3$, then

$$m_1 \times m_4 = (-2+z^2)(-2z+z^3) = 4z - 4z^3 + z^5. \tag{2.6}$$

Since there is now a z^5 term in Equation 2.6, the original polynomial expansion equation, Equation 2.3, must be extended by a z^5 term, to $f(z) = a_0 + a_1 z^1 + a_2 z^2 + a_3 z^3 + a_4 z^4 + a_5 z^5$.

This resulting representation of Equation 2.6 requires an expansion of the AI encoding vector, i.e. [0 4 0 -4 -0 1], to accommodate the z^5 term.

However, since $f(z)$ forms a ring of numbers, $m_1 \times m_4$ can be simplified through polynomial modulo reduction, using the integer coefficient polynomial Equation 2.5. For example, to reduce Equation 2.6, we divide it by Equation 2.5, and take the remainder, as follows:

$4z - 4z^3 + z^5 \equiv 2z \pmod{z^4 - 4z^2 + 2}$.

Thus $4z - 4z^3 + z^5$ is congruent to $2z$, so the reduced AI representation, [0 2 0 0], could be used instead of [0 4 0 -4 0 1]. The benefit of using the reduced AI representation in this example is that only one non-zero output is generated for processing (for the "2" element),

rather than the three non-zero outputs to be processed in the non-reduced AI representation (for the "4", "-4", and "1" elements).

It should be noted that polynomial reduction does not always lead to a simpler AI representation. Following [41], let $z = \sqrt{2+\sqrt{2+\sqrt{2}}}$, which is a root of the integer coefficient polynomial $2 - 16z^2 + 20z^4 - 8z^6 + z^8 = 0$. Then, let $m = 2\cos(\pi/16) = \sqrt{2+\sqrt{2+\sqrt{2}}}$, which would be represented as [0 1 0 0 0 0 0 0], using polynomial expansion $f(z) = \sum_{i=0}^{7} a_i z^i$. If the algorithm required m^9 to be represented as an AI encoded vector, then one option would be to allow the AI representation to grow to [0 0 0 0 0 0 0 0 0 1], using polynomial expansion $f(z) = \sum_{i=0}^{9} a_i z^i$. But if we use modulo polynomial reduction, we obtain:

$z^9 \equiv -2z + 16z^3 - 20z^5 + 8z^7 \pmod{2 - 16z^2 + 20z^4 - 8z^6 + z^8}$,

which generates the AI vector [0 -2 0 16 0 -20 0 8]. Computationally, it is far easier to work with [0 0 0 0 0 0 0 0 0 1] than [0 -2 0 16 0 -20 0 8], because there is only one output to process. This was a contrived example, but its purpose is to demonstrate that the use of modulo polynomial reduction must be compared to allowing the original AI encoding to remain in expanded form, in order to select a satisfactory AI representation vector format to use for a design.

Another potential problem with AI is that the integer coefficient polynomial may have large coefficients, such as the following example:

$z = \sqrt[4]{2} + \sqrt[4]{3}$, then the integer coefficient polynomial is:

$1 - 3860z^4 - 666z^8 - 20z^{12} + z^{16} = 0$.

Clearly in this case, attempts at modulo polynomial reduction will result in AI representations with large coefficients and make AI operations more difficult.

Conversion of an AI to a rational number approximation: Conversion, or decoding, of an AI vector is referred to as the Final Reconstruction Step (FRS) [3] and is performed using the polynomial expansion used to generate the encoding vector. Recall the first example

in Section 2.4.1.1, where the encoding vector was $f(z) = a_0 + a_1 z^1 + a_2 z^2 + a_3 z^3$, with $z = \sqrt{2+\sqrt{2}}$. Then, substitution of the AI elements into the polynomial expansion yields the single-number representation of the AI representation. For example, consider the AI vector [4 0 3 0], then the FRS would be:

$$FRS([4\,0\,3\,0]) = \frac{4+0z+3z^2+0z^3}{2} = \frac{4+3\left(\sqrt{2+\sqrt{2}}\right)^2}{2} = 5+3\sqrt{2}/2. \tag{2.7}$$

However, in terms of computer arithmetic, the irrational number z cannot be represented precisely, so a fixed-precision representation for z leads to rounding or truncation errors, so the FRS is where error is introduced in the AI method. The FRS is a costly step, taking 26% of the area in the Meyer-Base and Taylor design [42], which uses the CORDIC algorithm for FRS. In the author's design, the FRS took 66% of the Look-Up-Tables (LUTs) and 35% of FPGA registers, for a 2D AI implementation of the 2D DCT [55].

A benefit of using AI is that fixed-point approximation errors occur at the end of the computations, at the FRS. This allows the intermediate computations to be performed error-free and also minimizes word growth during computations. In two's complement systems, multiplication by fixed-point approximations of irrational numbers require larger word widths to carry out computations, in order to minimize error.

2.4.2 Multidimensional Algebraic Integers

Multidimensional AI was introduced by Dimitrov in [40], whereby multiple variables are used to create the polynomial expansion. This approach has been reported to minimize the AI representation over the 1D approach. This approach has been referred to as *bivariate* in [55, 56]. As with the one dimensional AI case, the z-values and AI encodings for multidimensional AI have been found found empirically, by Dimitrov.

Encoding

For a 2D AI example [3], consider the following two variables, $z_1 = \sqrt{2+\sqrt{2}} + \sqrt{2-\sqrt{2}}$ and $z_2 = \sqrt{2+\sqrt{2}} - \sqrt{2-\sqrt{2}}$, and the polynomial expansion:

$$f(z_1, z_2) = a_{00} + a_{01}z_2 + a_{10}z_1 + a_{11}z_1z_2 \tag{2.8}$$

.

The use of two z-values leads to the description of "2D" AI (where the use of one z-value is referred to as "1D" AI). We can now write m_1 to m_4 in terms of the 2D AI encodings (for a_{00} to a_{11}) as given in column 3 of Table 2.4. Note that there is a scaling factor of 4 on the encoding (not shown), so $f(z_1, z_2)$ must be divided by 4 to obtain the final result, similar to what was done for the one dimensional AI, to make the z-values algebraic integers.

For the AI encodings m_1 to m_4 in Table 2.4, it is observed that there are 8 non-zero elements for 1D AI and 5 non-zero elements for 2D AI. The 5 non-zero elements (i.e. 1, 1, -1, 2, and 2) infer a multiplication operation, as was the case for one dimensional AI, in Section 2.4.1. It is also observed that multiplication by an input for 2D AI would, at worst, only consist of a shift by one bit-position (i.e. 2x, which is only a binary shift, considered cost-free in hardware). However, in 1D AI, one element is a -3 (in m_2), so an additional adder would be required to complete the multiplication (e.g. 3x = x + 2x).

Multidimensional AI has multiple roots, which means that there are more degrees of freedom to obtain efficient AI encodings. To date, the highest reported AI is 3D [6], and it has been reported to be more efficient in implementation than 1D or 2D AI.

Operations

For all operations, except modulo polynomial reduction, the multidimensional AI works in a similar manner as 1D AI. At the present time, it is unknown whether modulo polynomial

Table 2.4: Arai 1D DCT multipliers with 2D AI encodings

Multiplier	Value	2D AI Encoding $\begin{bmatrix} a_{00} & a_{10} \\ a_{01} & a_{11} \end{bmatrix}$
m_1	$\cos(4\pi/16) = \sqrt{2}/2$	$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$
m_2	$\cos(6\pi/16) = \left(\sqrt{2-\sqrt{2}}\right)/2$	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$
m_3	$\cos(2\pi/16) - \cos(6\pi/16) = \left(\sqrt{2+\sqrt{2}} - \sqrt{2-\sqrt{2}}\right)/2$	$\begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix}$
m_4	$\cos(2\pi/16) + \cos(6\pi/16) = \left(\sqrt{2+\sqrt{2}} + \sqrt{2-\sqrt{2}}\right)/2$	$\begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix}$

reduction is possible on multidimensional AI. However, a method of simplifying decomposition has been found by Dimitrov (in the course of our graduate student-supervisor relationship), which has similar results to modulo polynomial reduction, as will be reported and utilized later in this thesis. The 2D DCT is explored in this thesis, so the AI addition/subtraction, multiplication, and FRS operations will be discussed as below.

The scaling factor for the 2D AI DCT is 4, which can be applied to integer additions/subtractions, and at the FRS.

For integer-AI multiplication, considering the example of multiplying 2 by m_2 of Table 2.4, we have:

$$2 \times \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ -2 & 0 \end{bmatrix}.$$

To compute the FRS of $2 \times m_2$, where $\circ$ is the Hadamard or Schur product, and recalling the scaling factor of 4 (i.e. <<2) we have:

$$FRS\left(\begin{bmatrix} (0 << 2) & 2 \\ -2 & 0 \end{bmatrix}\right) = FRS\left(\begin{bmatrix} 0 & 2 \\ -2 & 0 \end{bmatrix}\right) =$$

$$SUM\left(\begin{bmatrix} 0 & 2 \\ -2 & 0 \end{bmatrix} \circ \begin{bmatrix} 1 & z_1 \\ z_2 & z_1 z_2 \end{bmatrix}\right) \cdot \frac{1}{4} = SUM\left(\begin{bmatrix} 0 & 2 \cdot z_1 \\ -2 \cdot z_2 & 0 \end{bmatrix}\right) \cdot \frac{1}{4}. \tag{2.9}$$

Where *SUM* denotes the summation of all elements of the matrix. The resulting operation is:

$$SUM\left(\begin{bmatrix} 0 & 2 \cdot z_1 \\ -2 \cdot z_2 & 0 \end{bmatrix}\right) \cdot \frac{1}{4} = (0 + 2 \cdot z_1 - 2 \cdot z_2 + 0) \cdot \frac{1}{4}. \tag{2.10}$$

Now recalling that $z_1 = \sqrt{2+\sqrt{2}} + \sqrt{2-\sqrt{2}}$ and $z_2 = \sqrt{2+\sqrt{2}} - \sqrt{2-\sqrt{2}}$, we can

solve for the numerical value. As an alternative and equivalent method of computation, we note that, using the 2D AI encoding matrix, $\begin{bmatrix} a_{00} & a_{10} \\ a_{01} & a_{11} \end{bmatrix}$, that 2 is in the a_{10} position and that -2 is in the a_{01} position, and that the a_{00} and a_{11} positions are zero. We resolve Equation 2.9 by substituting the z-values and the a-values into the polynomial expansion Equation 2.8, as follows, $\frac{0+2\cdot z_1+(-2)\cdot z_2+0\cdot z_1 z_2}{4} = \frac{2\cdot\left(\sqrt{2+\sqrt{2}}+\sqrt{2-\sqrt{2}}\right)+(-2)\cdot\left(\sqrt{2+\sqrt{2}}-\sqrt{2-\sqrt{2}}\right)}{4}$, to obtain the numerical solution.

For integer-AI addition, we add the integer to the a_{00} position, as follows for the example of $2+m_2$:

$$2+\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 0+2 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ -1 & 0 \end{bmatrix} \tag{2.11}$$

then to compute the FRS, we have,

$$FRS\left(\begin{bmatrix} (2<<2) & 1 \\ -1 & 0 \end{bmatrix}\right) = FRS\left(\begin{bmatrix} 8 & 1 \\ -1 & 0 \end{bmatrix}\right) = SUM\left(\begin{bmatrix} 8 & 1 \\ -1 & 0 \end{bmatrix} \circ \begin{bmatrix} 1 & z_1 \\ z_2 & z_1 z_2 \end{bmatrix}\right)\cdot\frac{1}{4}, \tag{2.12}$$

which can then be solved to obtain a numerical value.

For multiplication between two 2D AI representations, it is observed that the AI operation can be represented as a Kronecker product, $\otimes$. For example, to multiply the two AI representations $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$, we have:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1\cdot5 & 1\cdot6 & 2\cdot5 & 2\cdot6 \\ 1\cdot7 & 1\cdot8 & 2\cdot7 & 2\cdot8 \\ 3\cdot5 & 3\cdot6 & 4\cdot5 & 4\cdot6 \\ 3\cdot7 & 3\cdot8 & 4\cdot7 & 4\cdot8 \end{bmatrix} = \begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix} \tag{2.13}$$

To convert this expanded matrix of Equation 2.13, the FRS is expanded as a Kronecker product.

$$\begin{bmatrix} 1 & z_1 \\ z_2 & z_1z_2 \end{bmatrix} \otimes \begin{bmatrix} 1 & z_1 \\ z_2 & z_1z_2 \end{bmatrix} = \begin{bmatrix} 1\cdot1 & 1\cdot z_1 & z_1\cdot1 & z_1\cdot z_1 \\ 1\cdot z_2 & 1\cdot z_1z_2 & z_1\cdot z_2 & z_1\cdot z_1z_2 \\ z_2\cdot1 & z_2\cdot z_1 & z_1z_2\cdot1 & z_1z_2\cdot z_1 \\ z_2\cdot z_2 & z_2\cdot z_1z_2 & z_1z_2\cdot z_2 & z_1z_2\cdot z_1z_2 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & z_1 & z_1 & z_1^2 \\ z_2 & z_1z_2 & z_1z_2 & z_1^2z_2 \\ z_2 & z_1z_2 & z_1z_2 & z_1^2z_2 \\ z_2^2 & z_1z_2^2 & z_1z_2^2 & z_1^2z_2^2 \end{bmatrix} \tag{2.14}$$

Then the FRS is computed, with the scaling factor, which becomes squared (i.e. $(\frac{1}{4})^2 = \frac{1}{16}$), as follows, continuing the example with Equation 2.13:

$$FRS\left(\begin{bmatrix} (5<<2) & (6<<2) & 10 & 12 \\ (7<<2) & (8<<2) & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}\right) =$$

$$SUM\left(\begin{bmatrix} 20 & 24 & 10 & 12 \\ 28 & 32 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix} \circ \begin{bmatrix} 1 & z_1 & z_1 & z_1^2 \\ z_2 & z_1z_2 & z_1z_2 & z_1^2z_2 \\ z_2 & z_1z_2 & z_1z_2 & z_1^2z_2 \\ z_2^2 & z_1z_2^2 & z_1z_2^2 & z_1^2z_2^2 \end{bmatrix}\right) \cdot \frac{1}{16}.$$

The above operation can written abstractly, with s_i, where $i \in$ 1,2, 3, ...16, as a label for

the value in each position of the AI representation of the data matrix, as:

$$SUM\left(\begin{bmatrix} s_1 & s_2 & s_5 & s_6 \\ s_3 & s_4 & s_7 & s_8 \\ s_9 & s_{10} & s_{13} & s_{14} \\ s_{11} & s_{12} & s_{15} & s_{16} \end{bmatrix} \circ \begin{bmatrix} 1 & z_1 & z_1 & z_1^2 \\ z_2 & z_1z_2 & z_1z_2 & z_1^2z_2 \\ z_2 & z_1z_2 & z_1z_2 & z_1^2z_2 \\ z_2^2 & z_1z_2^2 & z_1z_2^2 & z_1^2z_2^2 \end{bmatrix}\right) \cdot \frac{1}{16} \tag{2.15}$$

However, it is noted that the expanded z-matrix of Equation 2.14 has a number of common terms shown below in Equation 2.16. The overall FRS computation can be implemented much more efficiently by collecting terms before computing the Hadamard product with the now simplified z-matrix, transforming the matrix from 4x4 to 3x3.

$$\begin{bmatrix} 1 & z_1 & z_1 & z_1^2 \\ z_2 & z_1z_2 & z_1z_2 & z_1^2z_2 \\ z_2 & z_1z_2 & z_1z_2 & z_1^2z_2 \\ z_2^2 & z_1z_2^2 & z_1z_2^2 & z_1^2z_2^2 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & z_1 & z_1^2 \\ z_2 & z_1z_2 & z_1^2z_2 \\ z_2^2 & z_1z_2^2 & z_1^2z_2^2 \end{bmatrix} \tag{2.16}$$

The data matrix can now be simplified by collecting terms as shown in Equation 2.17, transforming the matrix from 4x4 to 3x3, by collecting s_i terms, following the pattern of Equation 2.16.

$$\begin{bmatrix} s_1 & s_2 & s_5 & s_6 \\ s_3 & s_4 & s_7 & s_8 \\ s_9 & s_{10} & s_{13} & s_{14} \\ s_{11} & s_{12} & s_{15} & s_{16} \end{bmatrix} \Longrightarrow \begin{bmatrix} s_1 & (s_2+s_5) & s_6 \\ (s_3+s_9) & (s_4+s_7+s_{10}+s_{13}) & (s_8+s_{14}) \\ s_{11} & (s_{12}+s_{15}) & s_{16} \end{bmatrix} \tag{2.17}$$

The above simplifications of Equations 2.16 and 2.17 reduces the number of multiplication operations from 16 (from the 4x4 Kronecker product in Equation 2.15), to 9 (from the simplified 3x3 Kronecker product in Equation 2.18).

$$FRS\left(\begin{bmatrix} s_1 & (s_2+s_5) & s_6 \\ (s_3+s_9) & (s_4+s_7+s_{10}+s_{13}) & (s_8+s_{14}) \\ s_{11} & (s_{12}+s_{15}) & s_{16} \end{bmatrix}\right) =$$

$$SUM\left(\begin{bmatrix} s_1 & (s_2+s_5) & s_6 \\ (s_3+s_9) & (s_4+s_7+s_{10}+s_{13}) & (s_8+s_{14}) \\ s_{11} & (s_{12}+s_{15}) & s_{16} \end{bmatrix} \circ \begin{bmatrix} 1 & z_1 & z_1^2 \\ z_2 & z_1z_2 & z_1^2z_2 \\ z_2^2 & z_1z_2^2 & z_1^2z_2^2 \end{bmatrix}\right) \cdot \frac{1}{16} \tag{2.18}$$

2.4.3 Approximation of 1D Algebraic Integers

Dimitrov [41] has extended the method of 1D AI by approximating the AI representation. In this method, only some of the multipliers have exact AI representations and the remainder have approximate AI representations. Consider the example from [41], which uses the following polynomial expansion and z-value.

$f(z) = a_0 + a_1z^1 + a_2z^2 + a_3z^3$, where $z = \cos(4\pi/32) = \sqrt{2+\sqrt{2}}$

Table 2.5 shows the encoding of 2cos(2πk/32), where k values of 0, 2, 4, 6, and 8 are exact AI representations and k values of 1, 3, 5, and 7 are approximate AI representations. The approximate AI representations introduce error within the AI computations, which are then compounded at the FRS.

For 1D AI implementations of the DCT and DHT, it has been reported that exact AI implementations require about twice the number of multiplications, compared to approximate AI implementations [41]. The application of approximation methods has not been reported for multidimensional AI.

Table 2.5: Approximation of 2cos(2πk/32)

k	Value 2cos(2πk/32)	1D AI Encoding $[a_0\ a_1\ a_2\ a_3]$	Error FRS($[a_0\ a_1\ a_2\ a_3]$) - 2cos(2πk/32)
0	2cos(0πk/32)	[2 0 0 0]	0
1	2cos(2πk/32)	[9 8 1 -4]	0.0001392826
2	2cos(4πk/32)	[0 1 0 0]	0
3	2cos(6πk/32)	[-1 1 -9 5]	0.0001180780
4	2cos(8πk/32)	[-2 0 1 0]	0
5	2cos(10πk/32)	[7 -16 -6 7]	0.0000584621
6	2cos(12πk/32)	[0 -3 0 1]	0
7	2cos(14πk/32)	[4 6 -8 2]	0.0000466333
8	2cos(16πk/32)	[0 0 0 0]	0

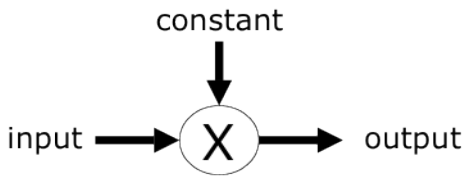

Figure 2.3: Constant Coefficient Multiplication

2.5 Single-Constant Integer Multiplication in VLSI

A Single Coefficient Multiplication (SCM) is a mathematical function that is frequently encountered in Digital Signal Processing (DSP), especially digital filtering such as Finite-duration Impulse Response (FIR) and Infinite-duration Impulse Response (IIR) filters. The difficulty with designing SCMs is that there are no simple algorithms that give an optimal result, in fact, this problem is known to be NP-complete [65]. An exhaustive search algorithm has been proposed by Dempster and Macleod, described later, to find optimal SCMs. A SCM is shown in Figure 2.3. All the inputs and the output are signed binary numbers and represent integers.

When implementing a SCM in hardware, it is desirable to take advantage of the property of binary numbers whereby a left-shift is a multiplication by 2. An example is shown in Figure 2.4, for multiplying the input of 5 by 2, to obtain the output of 10. The left-shift necessitates the addition of a zero at the LSB, which is obtained in a VLSI circuit by a

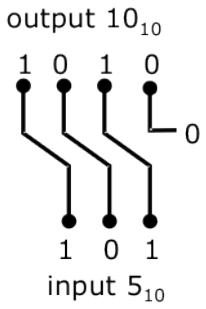

Figure 2.4: Using shifting to multiply by 2

grounded wire.

Multiplication by shifting is considered to be cost-free in hardware, because shifting only requires wires. Multiplication by numbers that are integer powers of 2, i.e. 2^1, 2^2, 2^3, etc. is therefore considered to be free, but multiplying by other integers requires the use of adders to form the final result as shown in the example of Figure 2.5 [66].

2.5.1 Conversion of a Real Number to an Integer Approximation

To multiply real numbers with a fractional component using integer multiplication, we must first convert the real number to a scaled integer approximation. The algorithm to find the integer and its scaling factor, to approximate the desired real number for a given number of bits of quantization accuracy, is as follows in MATLAB code:

```
% -- the inputs ------------------------------------------------
realnum = sqrt(2); % the desired real number, to be approximated
bits = 8; % the bit-length of the approximation (i.e.  quantization accuracy)
% -- algorithm to find the approximation -----------------------
exponent = floor(log2(2^bits/realnum); % this is the power of the scaling factor
integer = round(2^exponent*realnum); % the integer approximation
approximation = integer/2^exponent; % the closest, scaled approximation
percentageError = 100*(approximation-realnum)/realnum;
```

The output of this integer approximation finding algorithm is shown Table 2.6, for the real number $\sqrt{2}$ and for quantization accuracies ranging from 3 to 13 bits. The table shows, from the first column and moving right:

- the real number to be approximated;
- the number of bits of quantization accuracy desired (unsigned), for example if there are 3 bits, the largest unsigned number that could be represented is 7;
- the integer which is the most accurate approximation of the real number, for the given number of bits of quantization accuracy, note that this integer must be scaled to give the approximation;
- the exponent of the binary scaling factor required to scale the integer, to form the approximation of the real number;
- the approximation, which is the integer divided by the scaling factor $\left(i.e.\ \frac{Integer}{(2^{Exponent})}\right)$, note that the scaling factor is a power of 2, so it is a cost-free;
- the percentage error of the approximation, compared to the real number.

We can see that the percentage error (column 6) in Table 2.6 decreases monotonically, as the quantization accuracy (column 2) increases. However, we observe that for bits lengths of 8 to 12, the error remains the same, despite the increases in quantization accuracy. The reason that the percentage error does not decrease for every increase in quantization accuracy is because of the scaling used to generate integer, and is explained in Section 2.6.1.1.

Note that the simple algorithm presented in this section produces the most accurate integer approximation, for a given quantization accuracy. For example, considering row 4 of Table 2.6, we have the approximation of $\sqrt{2}$ at 5 unsigned bits of quantization accuracy as $23/2^4 = 1.4375$. The maximum number that can be represented by 5 unsigned bits is 31, and using the neighboring integers of 23, namely 22 and 24, yields approximations of $22/2^4 = 1.375$ (percentage error of -2.77%) and $24/2^4 = 1.5$ (percentage error of 6.07%), respectively. From this example, we see that $23/2^4$ is indeed the approximation that most accurately represents $\sqrt{2}$ at 5 bits of accuracy, because it has the lowest magnitude of percentage error, at 1.65%.

Table 2.6: Approximation of $\sqrt{2}$

Real Number	Number of Unsigned Bits	Integer	Exponent	Approximation $\left(\frac{Integer}{(2^{Exponent})}\right)$	Percentage Error
$\sqrt{2}$	3	6	2	1.5	6.06602
$\sqrt{2}$	4	11	3	1.375	-2.77282
$\sqrt{2}$	5	23	4	1.4375	1.64660
$\sqrt{2}$	6	45	5	1.40625	-0.56311
$\sqrt{2}$	7	91	6	1.421875	0.54175
$\sqrt{2}$	8	181	7	1.4140625	-0.01068
$\sqrt{2}$	9	362	8	1.4140625	-0.01068
$\sqrt{2}$	10	724	9	1.4140625	-0.01068
$\sqrt{2}$	11	1448	10	1.4140625	-0.01068
$\sqrt{2}$	12	2896	11	1.4140625	-0.01068
$\sqrt{2}$	13	5793	12	1.414306640625	0.00658

2.5.2 Constant Integer Multiplication Using the Binary Implementation

Creating a hardware realization of a constant integer multiplier is very simple, when viewing the integer as a binary number. Since binary numbers use a positional weighting system, the weights can be viewed as binary shifts of the input value, for the constant integer multiplier. For example, the binary representation of 231 is 11100111. Writing the binary representation in decimal, using the positional weighting system, we have $231 = 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$. Removing the terms multiplied by zero simplifies the representation to $231 = 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$. Now, to compute the constant integer multiplication of 231 by some input x, we can write $231 \cdot x = (1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) \cdot x = x \cdot 2^7 + x \cdot 2^6 + x \cdot 2^5 + x \cdot 2^2 + x \cdot 2^1 + x \cdot 2^0$. The final form makes it clear that x is the sum of the products in the binary representation. Fortunately, these products are multiplications by powers of two, and can be implemented cost-free using wired shifts. The top design of Figure 2.5 shows the binary implementation of the constant integer multiplier for 231.

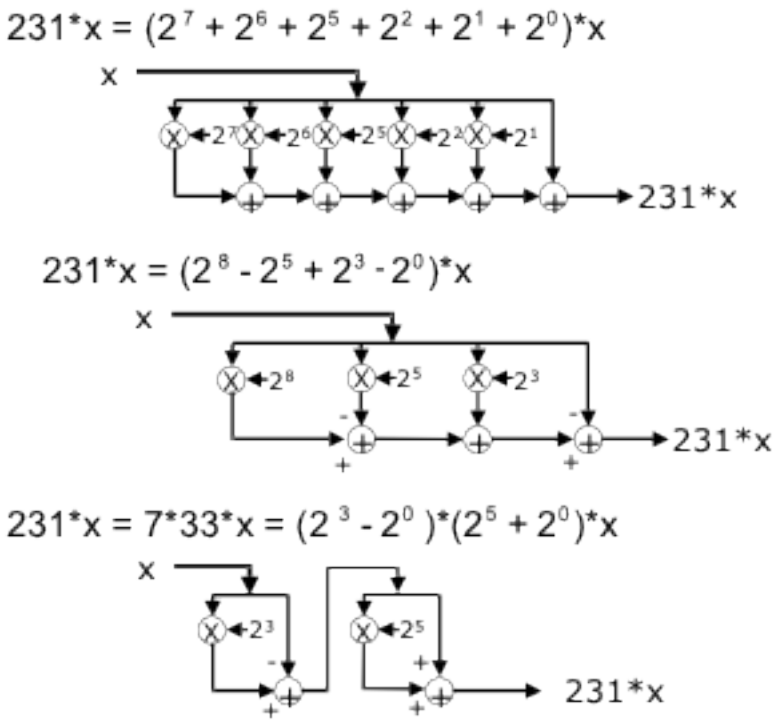

Figure 2.5: Example of multiple solutions for a given SCM

2.5.3 Constant Integer Multiplication Using the Canonic Signed Digits Implementation

Canonic Signed Digits (CSD) is a variation of the binary case, which on average results in a 33% reduction in the number of non-zero elements, compared to binary. Whereas binary has two elements, 1 and 0, CSD is ternary, with elements 1, 0, and -1. The number -1 is often denoted as $\bar{1}$. A further characteristic of CSD is that no strings of non-zero elements are permitted, rather, they are transformed as in the following example for the number 7. In binary, 7 is represented by 111, but the CSD representation is $100\bar{1}$. Any binary string can be transformed in a similar manner. The CSD representation for each integer is unique. For 231, the CSD representation is $100\bar{1}0100\bar{1}$, resulting the equation $231 \cdot x = x \cdot 2^8 - x \cdot 2^5 + x \cdot 2^3 - x \cdot 2^0$, and its hardware implementation is given in the middle design of Figure 2.5.

2.5.4 Constant Integer Multiplication Using the Dempster-Macleod Implementation

In Figure 2.5, we can see that there is more than one way to implement the multiplication of x by the constant integer 231. Since shifts by integer powers of 2 are free, the first representation (a binary multiplier) has a cost of 5 adders, whereas the second representation (a CSD multiplier) has a cost of 3 adders, and the third has a cost of 2 adders. Note that a subtractor is considered to have the same cost as the adder. The third representation is superior because it has the fewest adders. The problem with designing SCMs is that there are no simple algorithms that specify what representation is optimal for a given SCM. Dempster and Macleod [8] have developed an algorithm that finds optimal SCMs by an exhaustive search method, and has been used to design the third design example of Figure 2.5. The Dempster-Macleod (DM) algorithm exhaustively evaluates all possible SCM circuit topologies and stores the optimal SCM circuits for each constant multiplier value, as detailed in Section 2.6.

2.5.5 Multiple Word-Length Paradigm

The Multiple Word-Length Paradigm (MWLP), as described by Constantinides in [9, 67], is an efficient method to represent binary numbers. In traditional, fixed-point DSP systems, the word-length is kept constant, for example, 52 and 215 would be represented in binary as 0000000000110100 and 0000000011010111, respectively, for a 16-bit signed representation. The sum of 52 plus 215 in binary would be 0000000100001011, and we note that the summation operation would have required circuitry to compute the entire 16-bit word. In MWLP, 52 and 215 would be represented as 01101 and 011010111, and their sum as 0100001011. Note that in MWLP, the leading and trailing zeros are omitted, except for the sign bit. We note that this representation is smaller that the constant word-length example, and the circuitry to compute the sum would be correspondingly smaller, as the leading and trailing zeros do not have to be computed. The challenge with the MWLP is aligning the multiple word lengths correctly, to keep the binary points aligned. The description of the

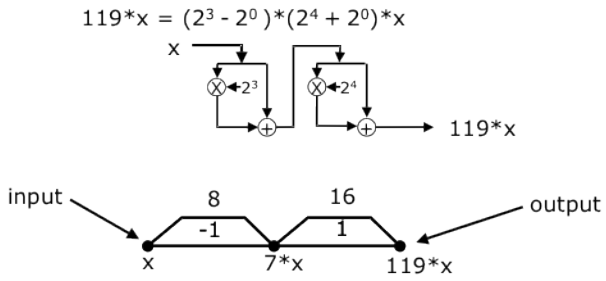

Figure 2.6: Example representing SCM circuits as graphs

MWLP is elaborated in Section 3.4.1, in its application to this thesis.

2.6 The Dempster-Macleod Algorithm

The DM algorithm is a method to obtain optimal constant coefficient multipliers, by using an exhaustive search to find the multipliers. The optimality criterion is that the constant coefficient multiplier has the minimum number of 2-input adders, for any particular integer constant, as illustrated in the example for the integer constant 231 in Figure 2.5. The DM algorithm represents SCM circuits by graphs, as shown in Figure 2.6. Each vertex of the graph (except the first) represents a 2-input adder. Each edge has a multiplier value associated with it that is an integer power of 2, i.e. 2^n, where n is an integer. Thus, each edge represents a wired shift operation (i.e. a binary shift) that is considered to be free in hardware. If the edge value is negative, then the adder associated with that edge is converted to a subtractor.

The DM algorithm generalizes all the possible SCM circuit topologies as graphs, as shown in Figure 2.7. It has been shown in [8] that for SCM constants up to 12 bits, i.e. –4095 to 4095, SCM circuits of 4 or fewer adders are required.

To implement the DM algorithm, all the possible edge values for each of the circuit topologies in Figure 2.7 are exhaustively tested. The set of graphs in the figure represents all the possible ways that circuits with up to 4 adders, can be connected. During the exhaustive search, if the output of the SCM circuit results an integer within the legal range, i.e. –4095 to 4095, then that SCM circuit is stored temporarily. Then, if a new SCM circuit is later

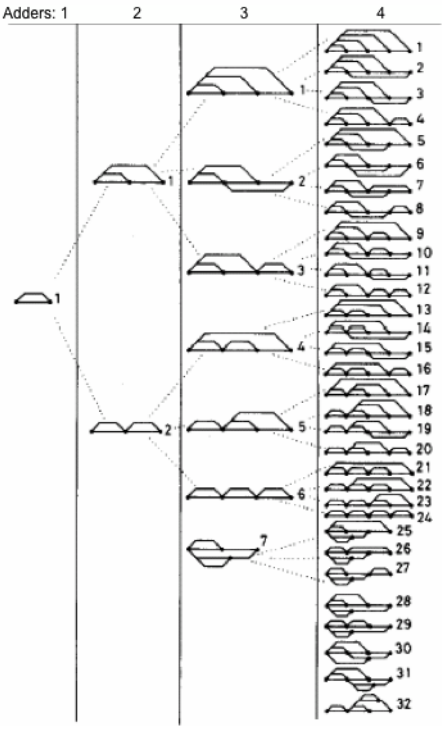

Figure 2.7: All possible graph topologies for circuits with 1 to 4 adders (from [8])

found to have a lesser number of 2-input adders, the previous result is replaced by the new result, for each integer value. This process is repeated until all the graphs of Figure 2.7 have been searched, for all possible edge values.

It is interesting to note that the binary and CSD types of SCM circuits are only represented by the top circuit topology of each column of Figure 2.7, which represent the sum (or negation) of binary shifts of the input value, as described in the examples of Sections 2.5.2 and 2.5.3. The DM method takes advantage of all the remaining circuit topologies to produce optimal SCM circuits, resulting in the expansion of the search space for possible solutions for the SCM, beyond the simple binary or CSD circuits.

The equations demonstrating how to compute vertex values are shown in Figure 2.8. The vertices are represented by b_n, the polarity [-1, 1] is represented by "s" and the magnitude is represented by "2^j"or "2^k" [k, j are integers].

Since the search space for this problem is quite large, the DM algorithm specifies constraints to limit the search space.

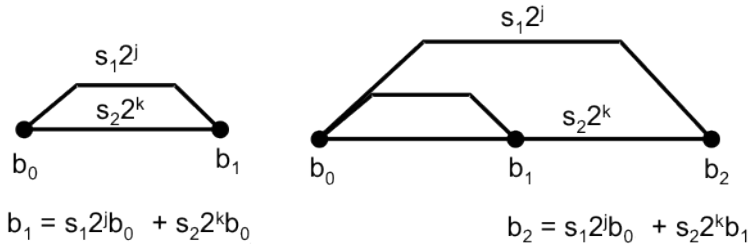

Figure 2.8: Vertex generation equations

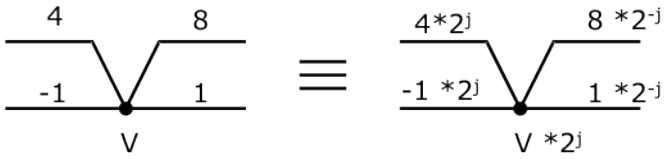

Figure 2.9: Modification of vertex values without changing the circuit behavior

2.6.1 DM Algorithm Rules to Minimize Search Space

The following rules for minimizing the search space come from [8].

2.6.1.1 Rule for Generating Only Odd Vertex Graphs

Figure 2.9 demonstrates a technique for modifying the vertex value, by a power of two, without changing the circuit behavior. The two circuits in Figure 2.9 are considered equivalent, it can be concluded that any even vertex can be converted to an odd vertex, with appropriate binary scaling. The DM algorithm specifies that only odd vertex SCM circuits need to be computed because they represent unique circuits, i.e. all the even vertex graphs can be computed from odd vertex graphs.

In Table 2.6 of Section 2.5.1, it was observed that increases in quantization accuracy do not always lead to decreases in percentage error, and this is related to the rule for generating odd vertices. To explain this observation, consider the example of taking $\sqrt{2}$ to an integer approximation of 8 bits of quantization accuracy (i.e. row 7 of Table 2.6), which is $181/2^7 = 1.414065$. Let us compare the previous approximation to $\sqrt{2}$ at a quantization accuracy of 9 bits ((i.e. row 7 of Table 2.6), which is $362/2^8 = 1.414065$, which has the

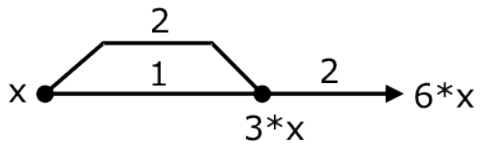

Figure 2.10: Creating even numbers from odd vertices

same numerical value as for the approximation at 8 bits of quantization accuracy. Now we note that the approximation at 9 bits of quantization accuracy has an even integer (i.e. 362), and can be rewritten as $(181 \cdot 2^1)/(2^7 \cdot 2^1) = 1.414065$, which by canceling the 2^1 in each of the numerator and denominator, can be reduced to $181/2^7 = 1.414065$, which is same numerical value as the approximation for 8 bits of quantization accuracy. Therefore, both representations must have the same quantization accuracy and the same percentage error.

From this brief analysis, we can infer that when constructing approximations for real numbers, the accuracy of the approximation only increases when the integer used for the approximation changes to a new odd number, as the quantization accuracy increases. When the integer used for the approximation is an even number, then the accuracy of the approximation remains the same as the previous odd integer approximation, as the quantization accuracy increases. This is because an even integer approximation can be reduced to an odd integer approximation, by canceling out the binary scaling, as we saw in the previous paragraph. This behavior can be seen in Table 2.6, and is noted in some occasions in this thesis, when results are reported for designs using successively higher quantization accuracies.

2.6.1.2 Rule for Generating Even Numbers

Creating even numbers from odd vertex graphs only requires multiplication by a power of 2 as demonstrated by the example of Figure 2.10. This operation only requires wires and is therefore considered to be free in hardware.

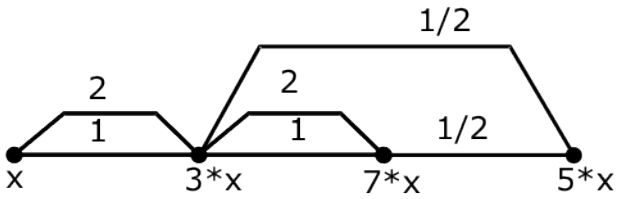

Figure 2.11: Fractional Magnitude Edges

2.6.1.3 Rule for Fractional Magnitude Edges

Following the rule that only odd vertex graphs need to be considered, Figure 2.11 demonstrates an example of a graph with fractional edge values (that are powers of 2). Since the rule is only odd vertex circuits need to be considered, only fractional edges that produce odd integer vertices need to be evaluated. Mathematically, this can only occur when the fractional edges are equal in magnitude to each other.

Figure 2.11 does not represent the optimal SCM for 5*x, it is only an illustration of fractional magnitude edges.

2.6.1.4 Rule for the Maximum Edge Value to Evaluate

The maximum edge value to be considered is 2^k where $k = max(log_2(b))$ and b is the SCM value to be represented. In our case, $b = 4095$. Then, $k = max(log_2(4095)) = 12$. Then $2^k = 2^{12} = 4096$ is the maximum edge value to be considered when generating the optimal SCM database.

2.7 Chapter Summary

This chapter provided a tutorial in background material covering the Discrete Cosine Transform and the Arai fast algorithm for its implementation. The Arai algorithm is used as a sample algorithm with which to compare the alternative design implementations explored in this thesis. This chapter also introduced Algebraic Integers and their application for the representation of irrational constants, using 1D and 2D formulations. The Algebraic Inte-

ger representations allowed for error-free representation of the irrational constants within the Arai algorithm. Binary and Canonic Signed Digit representation were discussed and the Multiple Word-Length Paradigm was described. This chapter concluded with a description of how to design optimal constant coefficient multipliers, using the method of Dempster-Macleod. These multipliers are used to implement the designs described in this thesis.

CHAPTER 3

SOFTWARE TOOLS AND MODELS

This chapter describes the tools and models that were used to conduct the research described in this thesis. Several software tools were developed by the author in the course of investigating this topic and two tools specifically relevant to the research presented in this thesis are briefly described in this chapter. One software tool (PARI), developed by an external research institution, was also used to help develop many of the designs in this thesis, as described in this chapter. The architecture for the 2D DCT is also presented in this chapter, as well as an example of a 12-bit SCM implementation of the full AI design, as part of Contribution 1. The development of the simplified method for AI is also presented, as part of Contribution 2.

This chapter concludes with design details of samples of the best designs developed for the AI approaches. These designs were evaluated for their performance on MSE and their hardware cost.

3.1 Single Constant Multiplication Tool

A software tool was designed and written by the author to implement and extend the DM method for designing SCMs [68]. At the time this research was started, no design tools were available to implement the DM method to design SCMs, although, Carnegie Mellon University presently has a similar tool available [69]. The author's SCM tool finds optimal SCMs for integers up to 16 bits in magnitude, using the minimum number of adder-bits for the SCM. For example, in Figure 3.1, x is a 16-bit number and the adder-bits are shown for

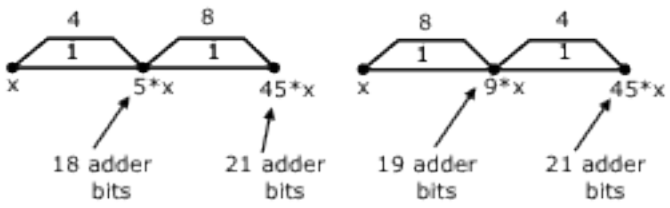

Figure 3.1: Example of adder-bits growth

two possible implementations for the SCM for 45, both of which have 2 adders. We can observe that the leftmost implementation has a cost of 18+21=39 adder-bits, whereas the rightmost implementation has a cost of 19+21=40 adder-bits, the difference arising from different word-growth within the algorithms. The use of the number of adder-bits as the parameter to minimize gives more hardware efficient designs than simply relying on the number of adders. As an aside, the integer 45 is the smallest positive integer that has a more efficient graph (i.e. lower number of adders) than that of the binary or CSD graphs. To implement the SCM for 45, the DM algorithm has 2 adders (Figure 3.1), binary has 3 adders ($2^5+2^3+2^2+2^1$) and CSD has 3 adders ($2^6-2^4-2^2+2^1$).

The SCM tool designed and implemented by the author is described in Figure 3.2, and follows the DM algorithm described in Section 2.6. Referring to Figure 3.2, the elements of the software tools developed by the author are as follows:

- A) The graph topologies netlist is an encoded version of the graphs of Figure 2.7. The topologies in the netlist represent all the possible ways that SCMs can be constructed, using from one to four, 2-input adders within the SCM.

- B) The program to search the netlist uses the graph topologies of A), and exhaustively applies different edge values, described in Section 2.6, to iteratively search each of the graph topologies in Figure 2.7. Each edge can have the value of $\pm 2^i$, where $i \in \{-12,$-11, -10,...,10, 11, 12$\}$. Each iteration of the program produces an SCM, whose integer value is checked whether it falls within the legal range of -4095 to 4095. The program of B) finds many solutions for each SCM, and keeps the solution

that has the least number of adder-bits, which represents the smallest hardware cost of the SCM. To reduce the search space, the rules described in Section 2.6.1 are used, namely: selecting only odd SCMs; generating even SCMs from odd SCMs; evaluating fractional edges only when fractional edges are equal to each other; and limiting the maximum edge value to 2^{12}. This reduction of the search space has been proven by Macleod and Dempster to not affect the optimality of the resulting solutions, defined as the minimum number of 2-input adders for SCMs of any integer in the legal range $[8, 30]$.

- C) The database generated by B), of optimal SCM circuits covering the range of constant integer coefficient multipliers, from -4095 to 4095. This database was used extensively in the author's tool described in Section 3.4.

- D) The program to search the database of C), in response to user inputs for the desired SCM and the number of bits in the input. This program searches the database and writes out synthesizable VHDL code for the SCM, and its associated VHDL testbench, to verify the design. The program of D) was used extensively to produce the designs in [55–57], which used several thousand lines of VHDL code.

- E) The VHDL program generated by program D). This program is synthesizable, meaning that it can be used to generate a circuit with FPGA or ASIC design tools. This VHDL code implements the SCM desired by the user.

- F) The VHDL testbench, which can be used to verify the VHDL program in E).

To verify that the SCM tool was correctly designed, two approaches were taken. The initial approach was to compare graphs of the cumulative average number of adders versus the wordlength of the SCM integer, between the author's SCM circuit database (i.e. element C in Figure 3.2) and the result reported by Dempster-Macleod in [8]. This comparison is extracted from the author's report in [68] and is shown in Figure 3.3. It can be seen

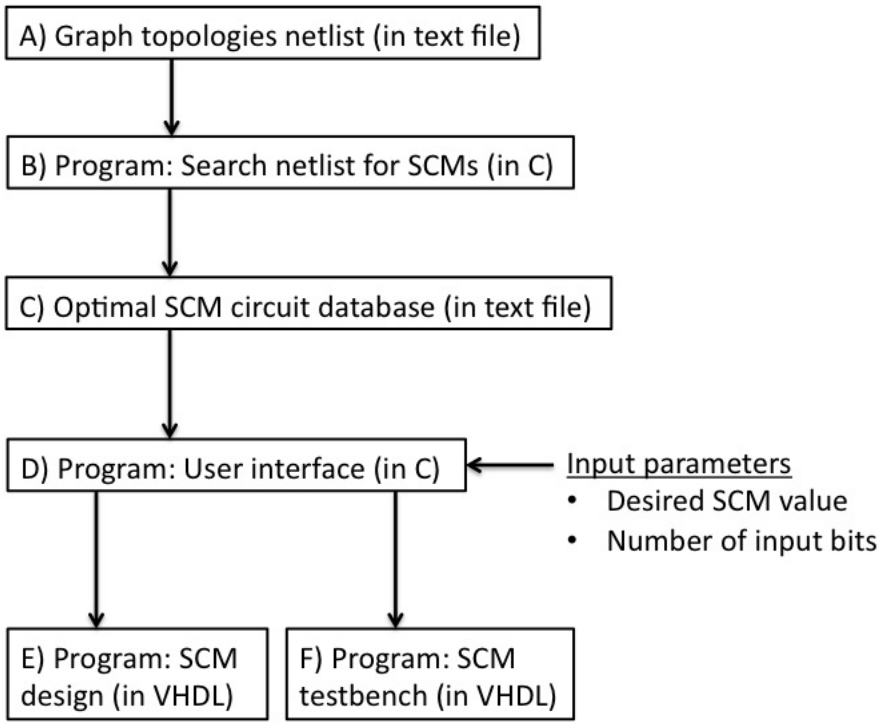

Figure 3.2: SCM design tool

that the sublinear behavior of the author's graph matches the behavior of the Dempster-Macleod graph (MAG algorithm). The second verification approach used a large number of comparisons between the author's SCM tool and that of Spiral [69] [70], an online software tool from Carnegie Mellon University used to generate SCMs, amongst many other DSP-related designs. Spiral implements the DM algorithm, and the author's tool produced the same results, over a large number of test cases.

The result of short study by the author on the distribution of optimal graph topologies in the SCM circuit database is shown in Figure 3.4, from the author's report in [68]. A plot of the frequency distribution of SCM graphs (or circuit topologies) in Figure 2.7 is shown in Figure 3.4, for 12-bit numbers (ranging from 0 to 4095). The graphs of Figure 2.7 are enumerated from 1 to 42, starting from the first row and column, then proceeding down each column from top to bottom, then repeating for each column from left to right. For example, the top graphs in each column of Figure 2.7 would be enumerated 1, 2, 4, 11.

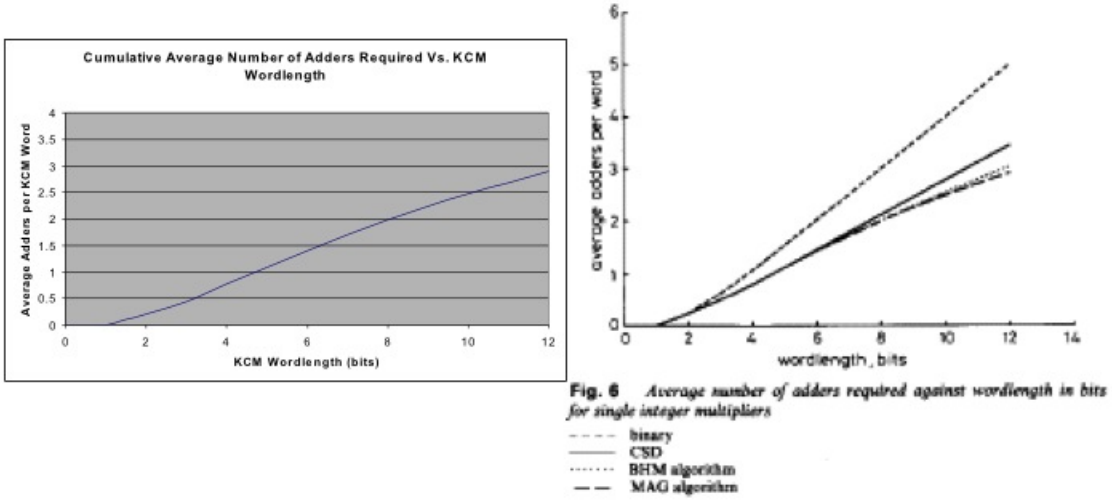

Figure 3.3: SCM design tool verification

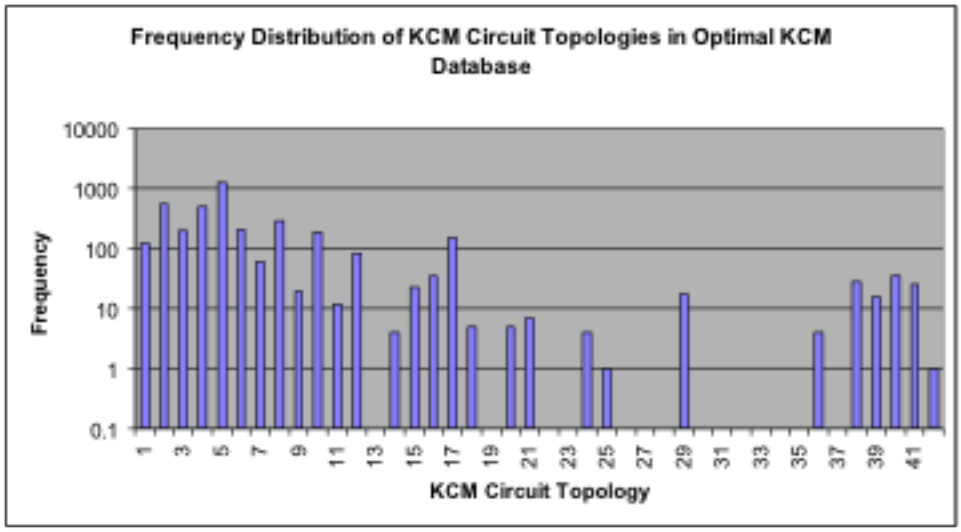

Figure 3.4: Frequency distribution of SCMs

It is noted that some SCM graphs have zero incidence in Figure 3.4, namely 13, 19, 22, 23, 26, 27, 28, 30, 31, 32, 33, 34, 35, and 37. All of these non-appearing graphs have 4 adders in them and are more costly in terms of hardware compared to smaller solutions. It is also noted that the graphs enumerated as 1, 2, 4, and 11 represent the SCM circuit topologies that correspond to binary and CSD SCM multipliers. Even though graphs 1, 2, 4, and 11 have a fairly high frequency of incidence, they represent only slightly more than ¼ of the circuits in the optimal SCM database. This means that the majority of optimal SCM

circuits cannot be found using the binary or CSD algorithm. According to Dempster [8,53], the average reduction of adders using the DM method, compared to CSD, is 16% for 12-bit words, 25% for 19-bit words, and 26.6% for 32-bit words.

3.2 Multiple Constant Multiplication Tool (PARI/GP)

The MCM algorithms reported in this thesis were provided by the author's collaborator, CNRS researcher Arnaud Tisserand[1]. Dr. Tisserand used the open source computer algebra system called PARI/GP [31], to obtain the MCM algorithms specified by the author. The MCM algorithms were verified by the author, and provided significant improvement over SCM algorithms, as demonstrated in the following example.

Consider the following equation as a candidate for a multiplierless algorithm for 12-bit integer coefficients, whose basic form appeared in [55, 56],

$$S = 2676 \cdot x_1 + 2217 \cdot x_2 + 2896 \cdot x_3 + 3135 \cdot x_4 + 3784 \cdot x_5 + 3496 \cdot x_6 + 2399 \cdot x_7 \quad (3.1)$$

Using the optimal DM methods for SCM design for Equation 3.1, the SCM algorithms in Table 3.1 are obtained. Recall that in DM, integers are factored to their odd vertex, to obtain the optimal algorithm, then scaled by powers of two, to generate even vertices, if required. In Table 3.1, each row shows an example of the SCM to compute the integer for the SCM: the first column shows the desired constant integer multiplier, C; the second column show the SCM algorithm. In the SCM algorithm, x is the input value, the v's are temporary variables or intermediate signals, and y is the output value.

The final SCM algorithm for equation (3.1), in the format for the author's software design tool, is presented in Table 3.2. The algorithm is read row-wise starting from the top of the first column, proceeding to the bottom of the first column, then repeating for the

[1] http://people.irisa.fr/Arnaud.Tisserand/#vitae. Accessed on May 1, 2012.

Table 3.1: Example SCM algorithms used for 12-bit full AI

Constant (C)	Algorithm to compute the constant, x is the input, y is the output, then $y = C \cdot x$
$2676 = 669 \cdot 2^2$	$v_1 = (1+2) \cdot x$ $v_2 = (1-2^3) \cdot v_1$ $y = -v_1 - 2^5 \cdot v_2$ $y = y \cdot 2^2$
2217	$v_1 = (1+2^4) \cdot x$ $v_2 = (1+2) \cdot x$ $v_3 = (v_1 + 2^3) \cdot v_2$ $y = 2^7 \cdot v_1 + v_3$
$2896 = 181 \cdot 2^4$	$v_1 = (1+2) \cdot x$ $v_2 = 2^3 \cdot x + v_1$ $y = 2^6 \cdot v_1 - v_2$ $y = y \cdot 2^4$
3135	$v_1 = (1+2) \cdot x$ $v_2 = (1-2^6) \cdot x$ $y = 2^{10} \cdot v_1 - v_2$
$3784 = 473 \cdot 2^3$	$v_1 = (1+2^2) \cdot x$ $v_2 = x - 2^3 \cdot v_1$ $y = 2^9 \cdot x + v_2$ $y = y \cdot 2^3$
$3496 = 437 \cdot 2^3$	$v_1 = (1+2^2) \cdot x$ $v_2 = 2^5 \cdot x - v_1$ $y = v_1 + 2^4 \cdot v_2$ $y = y \cdot 2^3$
2399	$v_1 = (1+2^2) \cdot x$ $v_2 = x + 2^5 \cdot v_1$ $y = 2^9 \cdot v_1 - v_2$

second column, and so on. The algorithm implements the following equations,

$$X_1 = 2676 \cdot x_1 \tag{3.2}$$

$$X_2 = 2217 \cdot x_2 \tag{3.3}$$

$$X_3 = 2896 \cdot x_3 \tag{3.4}$$

$$X_4 = 3135 \cdot x_4 \tag{3.5}$$

$$X_5 = 3784 \cdot x_5 \tag{3.6}$$

$$X_6 = 3496 \cdot x_6 \tag{3.7}$$

$$X_7 = 2399 \cdot x_7 \tag{3.8}$$

$$S = X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7. \tag{3.9}$$

In Table 3.2 "<<" represents a binary shift by a power of 2. Note that the labels for the intermediate values, V0 to V3, are reused for each SCM, because the mathematical form of the algorithm is shown. In a VLSI implementation, the V's would have a unique label for each SCM, or be hidden within a module for each SCM. By inspection, this SCM algorithm computes equation (3.1) in 28 additions and subtractions.

Using an MCM provided by PARI, equation (3.1) can be implemented by the algorithm of Table 3.3. This MCM algorithm computes equation (3.1) in 20 additions and subtractions, an approximately 29% reduction compared to SCM.

Table 3.5 shows the comparison between SCM-based and MCM-based algorithms for 9 designs for the sum of 7 integer coefficient products. These 9 designs are based on the Full AI algorithms described in this thesis, for coefficient quantization accuracies between 8 to 16 bits. There are significant savings in the number of adders required for MCM, across all accuracies, 29.6% on average.

As an alternative, Table 3.5 shows the comparison in the number of adders for SCM versus MCM, across all coefficient quantization accuracies for the FRS component for the Full AI algorithms. Again there are a significant savings in the number of adders, 20%

Table 3.2: Example combined SCM algorithms

V0 = x1	V0 = x3	V1 = V0 + V0<<2	V2 = V0 + V1<<5
V1 = V0<<0 + V0<<1	V1 = V0 + V0<<1	V2 = V0 - V1<<3	X7 = V1<<9 - V2
V2 = V1<<0 - V1<<3	V2 = V0<<3 + V1	X5 = V0<<12 + V2<<3	t = X1 + X2
X1 = -V1<<2 - V2<<7	X3 = V1<<10 - V2<<4	V0 = x6	t = t + X3
V0 = x2	V0 = x4	V1 = V0 + V0<<2	t = t + X4
V1 = V0 + V0<<4	V1 = V0 + V0<<1	V2 = V0<<5 - V1	t = t + X5
V2 = V0 + V0<<1	V2 = V0 - V0<<6	X6 = V1<<3 + V2<<7	t = t + X6
V3 = V1 + V2<<3	X4 = V1<<10 - V2	V0 = x7	S = t + X7
X2 = V1<<7 + V3	V0 = x5	V1 = V0 + V0<<2	

Table 3.3: Example MCM algorithm

t3 = x1<<4+x2	t10 = x4+x2<<5	t1 = t6<<2+t7	t16 = t12+t13<<2
t4 = x1-x4<<8	t11 = x3+x6	t12 = t8+t9	t17 = t14<<2+t15
t5 = x6+x3<<1	t6 = -x5<<1+t3	t13 = t4+t3<<1	t18 = t17<<4+t16
t8 = -x4+x2	t2 = -x7<<2+t5	t14 = x5<<6+t10	t19 = t1+t1<<2
t9 = x5<<3-x7	t7 = t2-t2<<4	t15 = -t4+t11<<8	S = t18+t19<<3

Table 3.4: Comparison of SCM vs MCM number of adders for a sum of 7 integer coefficient multiplications

Quantization Bits	8	9	10	11	12	13	14	15	16	Average
MCM adders for 7 elements	18	17	17	18	20	21	21	23	24	19.9
SCM adders for 7 elements	26	25	25	26	28	28	32	32	32	28.2
Percentage reduction for MCM algorithms	30.8	32.0	32.0	30.8	28.6	25.0	34.4	28.1	25.0	29.6

on average. These savings are not as large for in Table 3.4 because the FRS is not just comprised of 7-coefficient multiplications for each output. Some FRS outputs require only 3-coefficient multiplication, where the benefit of MCM over SCM is not as great as in the case of 7-coefficient multiplication. Besides the 7- and 3-coefficient multiplication cases, the remaining FRS outputs require no multiplications, so there no difference between SCM and MCM, in this final case.

Table 3.6 shows the comparison in the number of adders for SCM versus MCM, across all coefficient quantization accuracies for the FRS component for the Full AI algorithms. The number of adder-bits is a more refined measure of circuit complexity, for VLSI implementations, than the number of adders. Again, MCM is significantly better than SCM, having average adder-bit savings of 18.6%.

Unlike the author's SCM tool, the MCM tool using PARI does not take into account the adder-bits cost of the algorithm, but rather, returns the algorithm with the minimum

Table 3.5: Comparison of SCM vs MCM number of adders for the FRS

Quantization Bits	8	9	10	11	12	13	14	15	16	Average
Full AI MCM FRS adders	836	820	836	868	932	992	1008	1040	1104	973.3
Full AI SCM FRS adders	1060	1060	1060	1092	1156	1168	1312	1312	1312	1170.2
Percentage reduction for MCM algorithms	21.1	22.6	21.1	20.5	19.4	15.1	23.2	20.7	15.9	20.0

Table 3.6: Comparison of SCM vs MCM number of adderbits for the FRS

Quantization Bits	8	9	10	11	12	13	14	15	16	Average
Full AI MCM FRS adderbits	12182	11726	12339	12437	13608	14493	15444	15854	17264	13927.4
Full AI SCM FRS adderbits	14905	15074	15217	15607	16669	17074	19529	19562	20255	17099.1
Percentage reduction for MCM algorithms	18.3	22.2	18.9	20.3	18.4	15.1	20.9	19.0	14.8	18.6

number of adders. Some of the designs presented in this thesis can only be implemented using SCMs, or must be partially implemented by SCMs. Otherwise, the MCM algorithms provided by PARI, are used. For the full AI algorithms, both SCM and MCM versions of the algorithms are implemented, to demonstrate the value of MCM versus SCM algorithms.

3.3 Mean-Squared Error Analysis

The Mean-Squared Error (MSE) was the metric used to evaluate the quality of the results of the implementation. The 2D DCT is commonly used in image compression and a variety of metrics could have been used to evaluate the quality of results. However, the MSE is very commonly used in signal processing [71], and metrics that concentrated on signal fidelity based on human perception were deemed inappropriate since the intention of this research was not to compress data, but rather, to model a signal processing platform that extensively used constant multiplications by irrational numbers, that could be transformed to use algebraic integer representations.

The MSEs of the designs were obtained by averaging the squared error over 10000 8x8 input matrices with uniformly distributed random 8-bit signed integers for each element of the matrices, computed by equation (3.10). For all MSE trials, a constant seed value[2] was used to obtain the same random input numbers to obtain comparative MSE data for different designs. In the designs analyzed, the word-growth as allowed to happen without restriction, i.e. no truncation nor rounding was used.

$$MSE(x,y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2 \tag{3.10}$$

[2]The seed value was generated by the following Matlab command: rand('twister')

3.4 AI Model and Adder-bits Computation Tool

3.4.1 Adder-bits Computation Tool

A set of bit-accurate software tools were developed to evaluate the hardware costs of the designs evaluated in this research. These tools generate a hardware model of the mathematical equations supplied as inputs. The hardware model emulates the adder circuitry (hence, bit-accurate) implied by the mathematical equations within the designs, accommodating word growth as data flows through the hardware model, as a simple form of hardware synthesis. These tools were also used to calculate the MSE of the designs.

Since the implementation of the DCT algorithm relies only on additions/subtractions and multiplications, and the multiplications are transformed to "shift-and-add" operations, then the only essential hardware cost is the number and size of the adders/subtractors. Shift operations are not considered in VLSI design at the algorithmic level, as they are considered cost-free, since they simply consist of wires, which are insignificant compared to digital logic. The only other reasonable additional hardware would be registers, which would be used for pipelining, to improve the speed of operations. The number of registers will be proportional to the size of the adders/subtractors, and is not counted.

To develop the costs for the adders, the Multiple Word-Length Paradigm (MWLP) [9, 67] was used to account for the different adder types, as shown in Figure 3.5. In the MWLP, the size of the data words is not constant, as would be in the case of a fixed-point DSP processor. Rather, since custom hardware is considered in this thesis, for a VLSI implementation, the data words can be sized to precisely the width they need to be, for worst-case word-growth due to arithmetic operations. This will result in the most efficient use of hardware resources, but at the cost of increased design complexity. In Figure 3.5, three of the six basic types of addition are shown (the remaining three types can be obtained by the commutative property). Figure 3.5 shows a pictorial view of the addition operations. The top row shows three addends, one each for (a), (b), and (c). The sign bit is shown,

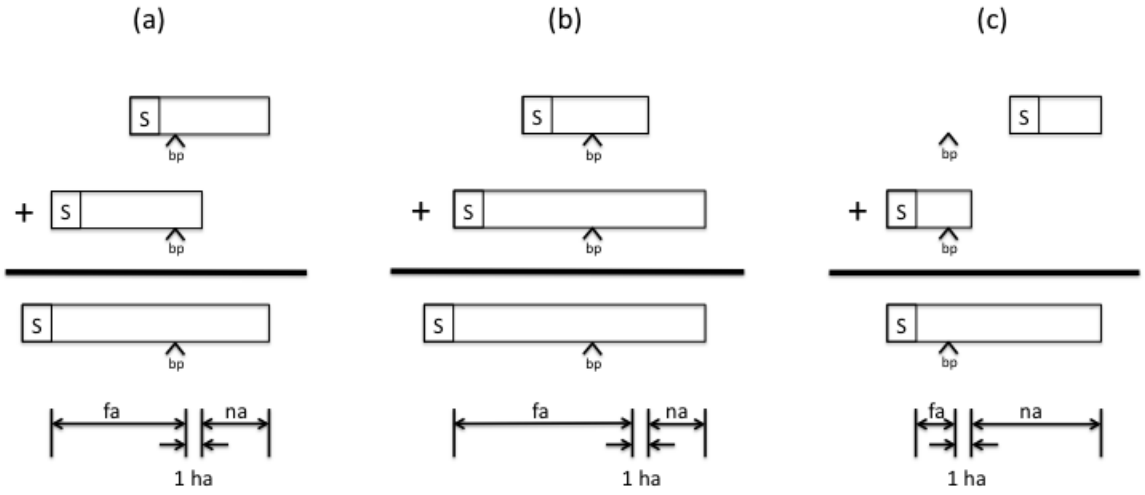

Figure 3.5: Adder types and hardware costs (adapted from [9])

represented by the box inclosing "S", and the unlabeled rectangle to the right of the sign bit represents the addend (excluding the sign bit). The length of the unlabeled rectangle represents the relative word-length of the addend. Thus, Figure 3.5 represents addition operations on words of multiple word-lengths.

For each of the examples, (a), (b), and (c), the binary point (denoted "bp" in the figure) is aligned for each addend, even if the bp is not within the word (as seen in (c)). Within the tool, the sign extension and word growth of the sum is accommodated. In the examples of Figure 3.5, word growth has been shown, but the tool only allows the summation word to grow if there is potential for overflow, resulting in efficient hardware design, by limiting unnecessary word growth.

If the least significant bits (LSB's, reading right to left) of the addends are not aligned, as shown in (a), (b), and (c), then there will be no adder cost, denoted as "na." Moving leftwards toward the most significant bits (MSBs), the first aligned bits will require a single half-adder, denoted "ha" as there is no carry-in. The remaining MSB bits will require full-adders, denoted "fa," as there can be a carry-in, and the sign and length of the sign extension is unknown, in a general case.

The tool accepts mathematical operations in the form of two operand operations in

Table 3.7: Arai 1D DCT in tool input format

b0 = x0+x7	c5 = b3+b7	e1 = d1	f5 = e8-e3
b1 = x1+x6	c6 = b3+b6	e2 = d2*m3	f6 = e2+e7
b2 = x3-x4	c7 = b7	e3 = d7*m1	f7 = e4+e7
b3 = x1-x6	d0 = c0+c3	e4 = d6*m4	S0 = f0
b4 = x2+x5	d1 = c0-c3	e5 = d5	S1 = f4+f7
b5 = x3+x4	d2 = c2	e6 = d3*m1	S2 = f2
b6 = x2-x5	d3 = c1+c4	e7 = d4*m2	S3 = f5-f6
b7 = x0-x7	d4 = c2-c5	e8 = d8	S4 = f1
c0 = b0+b5	d5 = c4	f0 = e0	S5 = f5+f6
c1 = b1-b4	d6 = c5	f1 = e1	S6 = f3
c2 = b2+b6	d7 = c6	f2 = e5+e6	S7 = f4-f7
c3 = b1+b4	d8 = c7	f3 = e5-e6	
c4 = b0-b5	e0 = d0	f4 = e3+e8	

equation form, including addition, subtraction, and multiplication, with multiplications converted to shift-and-add forms. An example of the tool input format for the Arai 1D DCT is shown in Table 3.7, with x's replacing the a's in Figure 2.1. Additions, subtractions, and multiplications can be handled as either AI, non-AI, or mixed AI and non-AI operations. The expansion of the AI representations (by AI multiplication) is handled automatically by the tool. The format of the equations in Table 3.7 is converted to an intermediate format, by the tool, as shown in Table 3.1. The tool also accepts the format of Table 3.1 directly.

The software tool written by the author to model and simulate the designs reported in this thesis is described in Figure 3.6. The elements of this software are described as follows:

- A) The signal-flow graph of the Arai algorithm in Figure 2.1, encoded in the form of equations, as shown in Table 3.7.

- B) The set of equations for the FRS, which are developed later in the thesis, in Section 3.4.2. Many variations of the FRS were were explored in this thesis, to find which ones gave the best performance.

- C) The set of SCM and MCM equations used within the FRS. The SCM equations were generated by the author's tool, as described in Section 3.1. The SCMs were used in the non-AI designs and some of the AI designs, only if MCM equations were

not possible. The MCM equations were generated by the author's collaborator, as described in Section 3.2, using specifications provided by the author. The MCMs were only applicable in the AI designs.

- D) The program which translates the design equations of A), B), and C), into a program which models the circuit behavior of the design. The user specifies input parameters regarding whether the design is to be AI or non-AI, and the number of bits of desired quantization accuracy.

- E) The program generated by D) is a bit-accurate model of the design. This means that the model represents the hardware implementation of the design, emulating the adder circuits required for implementation as a digital circuit. Accordingly, the data resulting from the simulation of this model is exactly what would be generated from a physical circuit, thus metrics for mean-squared error, the number of adders, and the number of adder-bits can be obtained for analysis.

- F) The simulation program exercises the model in E), through a user specified number of simulations, and records the MSE for each output coefficient, the number of adders used, and the number of adder-bits used. To determine the MSE, the program in F) uses the MATLAB DCT as the reference, to compare against the outputs of the model in E).

The software described by Figure 3.6 was verified using two techniques. The first technique was designer level testing with manual checking that the equations of A), B), and C), were correctly converted to E). This meant verifying that the correct adder circuits were inferred by the program, at stage F), based on the equations inputs of A), B), and C), for a large number of progressively more complex design examples. This testing technique was applied repeatedly by the extensive debugging of program F). Since the entire modeling and simulation process for Figure 3.6 is very complex, a large number (i.e. several hundred) of detailed investigations were carried out to verify the results, during development.

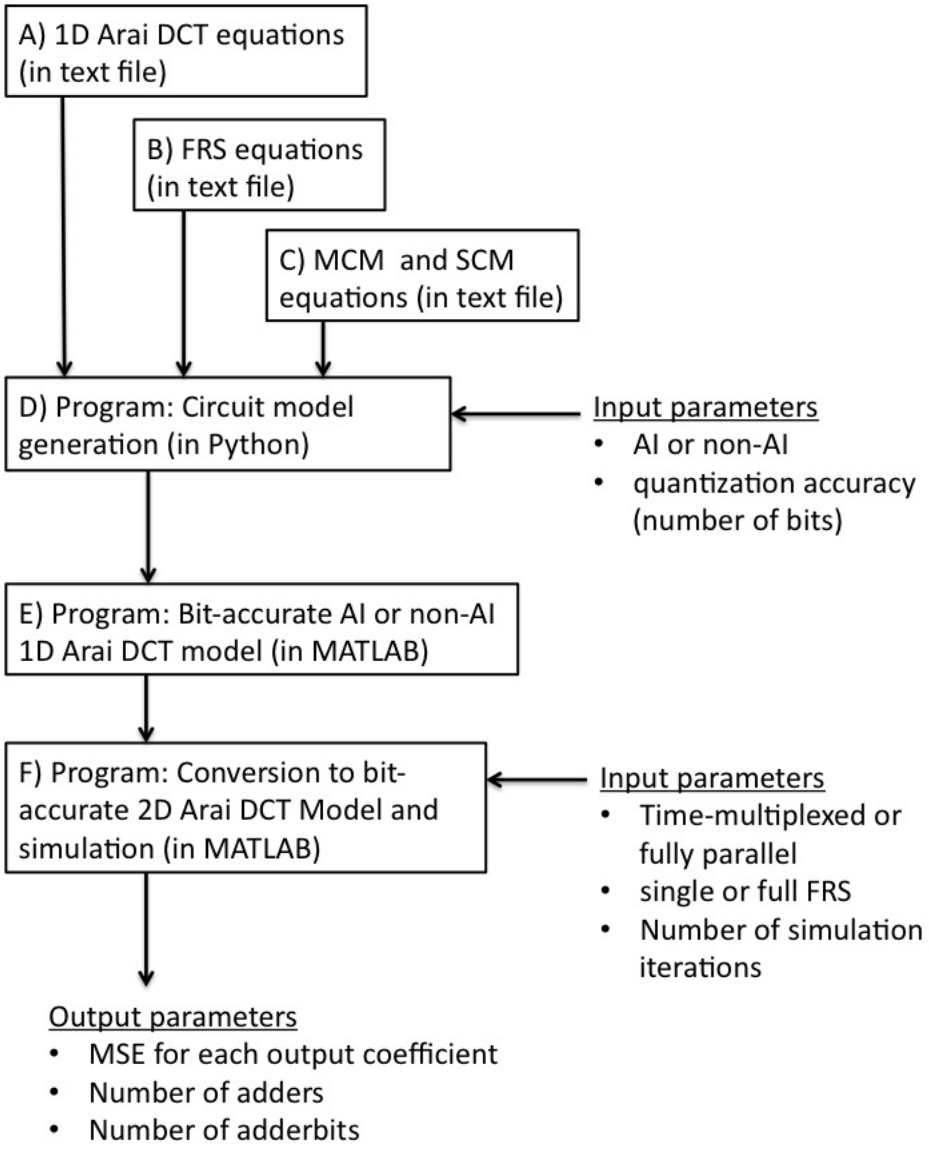

Figure 3.6: AI model and simulation design tool

The second technique used to verify the software was in-system testing, whereby the equation models of A), B), and C), were propagated to F), with their numerical approximations, and the simulation was carried out for both the numerical simulation and the circuit simulation, and the results compared with each other. For example, if an equation in A), B), or C) called for the multiplication of $\sqrt{2}$ at 5 bits of coefficient quantization accuracy, the

numerical approximation would be 1.4375[3], then this numerical approximation would be carried out through the simulation in F). Concurrently, the equation for the multiplication of $\sqrt{2}$ at 5 bits of coefficient quantization accuracy would be converted by D) to E), and simulated as a digital circuit by F). The results of both simulations (i.e. the numerical approximation simulation and the digital circuit simulation) must be identical, to be correct. The program in F) checks to verify that the result for both of the parallel simulation tracks (i.e. for the numerical approximation and the digital circuit) are consistent with each other, and reports an error message, if there is a violation. This check is automatically done for every output value.

3.4.2 AI Model

3.4.2.1 Full AI Model

This section details the design of the 2D AI implementation for the 2D Arai DCT. There are two main components to the design, the DCT transform and the FRS. The 2D DCT is based on the Arai DCT of Section 2.3 and Table 3.7. This Arai DCT is modified to be able to accommodate AI operations is described in Section 2.4, whereby multiplication operations are converted to AI operations, and AI addition and subtraction operations can accommodate vector arrays representing AI data. The 2D DCT is implemented on a separable basis, whereby the inputs are passed through an 8-point 1D DCT, then transposed by a transposition matrix, then passed through another 8-point 1D DCT. The majority of the outputs of the 2D DCT are in algebraic integer format and must be converted to a fixed-point binary format, by using the FRS. This separable implementation is shown in Figure 3.7. Note that the 1D DCT sections must be reused 8 times, for an 8x8 input matrix, to compute the 8x8 output coefficients. The transposition matrix is typically memory or register banks.

Figure 3.7 indicates a generalized transposition matrix, which is implemented as trans-

[3]Note that the numerical approximation is exactly what the circuit simulation would compute. Another way to say "numerical approximation," for this example, would be, "quantized value."

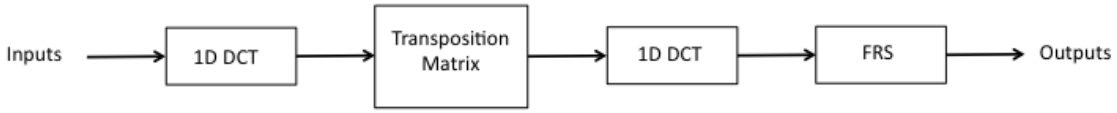

Figure 3.7: Overview of time-multiplexed AI-based 2D DCT

position memory. This transposition memory has 64 AI elements, as an 8x8 AI matrix, denoted as TM. Each AI element can have four integer elements, as described in Sections 2.4.1 and 2.4.2. The 8x8 input matrix is X. The TM matrix is populated by the first (i.e. leftmost) 1D DCT operating on input X, as follows:

for row = 1:8

$TM(row, 1:8) = DCT(X(row, 1:8))$

end

To transform TM with the DCT, in order to obtain the FRS inputs, the following operation is used, which accommodates the transposition:

for col = 1:8

$FRS(1:8, col) = DCT(TM(1:8, col))$

end

Figure 3.8 shows the fully parallel implementation of the 2D DCT, which can operate on an 8x8 input matrix (flattened to a 64x1 array), denoted Y. The fully parallel design can operate much faster than the time-multiplexed design, as each 1D DCT is used only once, but at the cost of much greater hardware. The transposition matrix, TM, for the fully parallel design is simply wires, with no need for physical storage. The TM matrix performs the following function.

for i = 1:8

$TM((8 \cdot i - 7):8 \cdot i)) = DCT_i(Y((8 \cdot i - 7):8 \cdot i))$

end

To transform the TM with the DCT, in order to obtain the FRS inputs, the following operation is used, which accommodates the transposition (note that the argument for TM

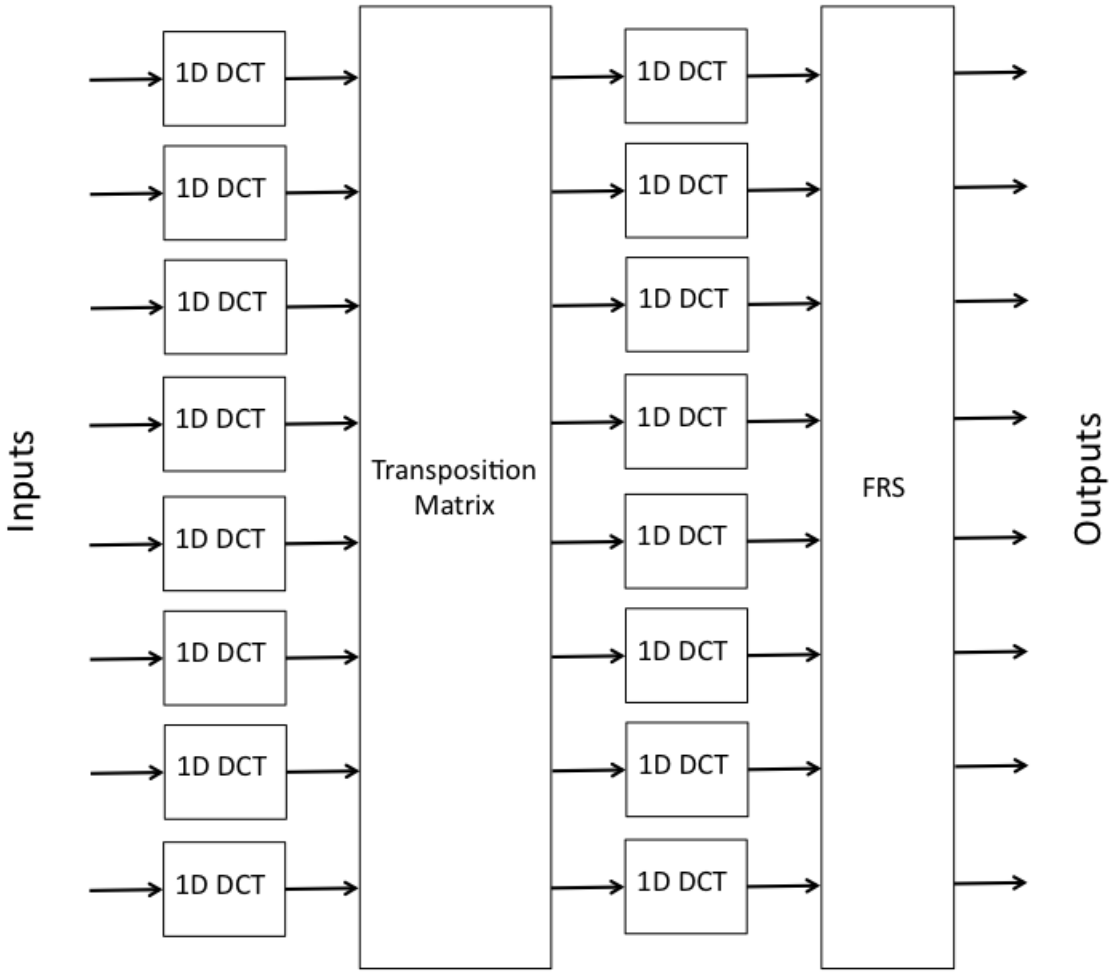

Figure 3.8: Overview of fully parallel AI-based 2D DCT

is an 8-element vector):

for j = 1:8

$FRS((8\cdot i-7):8\cdot i)) = DCT_j(TM([j \quad j+8 \quad j+16 \quad j+24 \quad j+32 \quad j+40 \quad j+48 \quad j+56]))$

end

The fully parallel and time-multiplexed 2D DCT designs are implemented in the design tools. The AI and non-AI designs have identical architectures to Figures 3.7 and 3.8, except there is no FRS block in the non-AI models. However, internally to the 1D DCT block, the

non-AI implementation uses SCMs to implement the m-values of Table 2.1 and Figure 2.1, whereas the AI implementation uses AI multiplication as described in Section 2.4. For the AI designs, the 2D AI encoding is as shown in Table 2.4 and the Arai algorithm is used for the 8-point 1D DCT, as shown in Figure 2.1. A variety of AI designs are presented in this thesis, and they are differentiated from each other strictly by their FRS sections.

A further variation of the FRS is explored in this thesis, which involves using a single FRS output to compute one 2D AI DCT output coefficient at a time. In this design, the single FRS output is multiplexed to the DCT. For the time-multiplexed design, there is a 8:1 multiplexer. For the fully parallel design there is a 60:1 multiplexer[4]. The single FRS approach results in reduced physical hardware for the FRS, because of reuse, at the cost of reduced data throughput rate.

The FRS is based on Equation 2.18, which describes a time-multiplexed design, which is used iteratively to generate eight 2D DCT coefficients per iteration, S0 to S7. The eight equations have the form of either of one of the three prototype forms given in the equations below. The Equations 3.11 to 3.13 are referred to in this thesis as full AI.

Type 1:

$$S0, S4 = x_1 + z_1 \cdot x_2 + z_2 \cdot x_3 + z_1 z_2 \cdot x_4 \tag{3.11}$$

Type 2:

$$S1, S3, S5, S7 = x_1 + z_1 \cdot x_2 + z_2 \cdot x_3 + z_1 z_2 \cdot x_4 + z_1^2 \cdot x_5 + z_1^2 z_2 \cdot x_6 + z_2^2 \cdot x_7 + z_1 z_2^2 \cdot x_8 + z_1^2 z_2^2 \cdot x_9 \tag{3.12}$$

[4] It is 60:1, instead of 64:1, because four coefficients do not require the FRS, as they are scalar values, not AI values.

Type 3:

$$S2,S6 = x_1 + z_1 \cdot x_2 + z_2 \cdot x_3 + z_1 z_2 \cdot x_4 + z_1^2 z_2 \cdot x_5 + z_1 z_2^2 \cdot x_6 + z_1^2 z_2^2 \cdot x_7 \tag{3.13}$$

In the above equations, x_1 to x_7 represents the AI-encoded output coefficients of the 2D DCT. The Type 1 Equation 3.11 is used to compute the FRS of 4-element AI vectors, as shown in Equation 2.12. The Type 2 Equation 3.12 is used to compute the FRS of 9-element AI vectors, which have arisen due to AI-by-AI multiplication, as shown in Equation 2.18. The Type 3 Equation 3.13 is a special case of Type 2, but where the z_1^2 and z_2^2 elements have zero as the AI element. The use of the Type 3 equation allows a smaller MCM (i.e. sum of 7 products) to be used for it, rather than using the full MCM for the Type 2 equation (i.e. sum of 9 products), since there are 2 fewer multiplies to be accommodated by the MCM. In this thesis, the z-values are rounded, using the method described in Section 2.5.1, to their closest signed approximations, ranging from 8- to 16-bits of quantization. Using an example from [55, 56] to illustrate, the z-values are rounded to their closest 12-bit signed approximations, as follows.

Exact Values:

$$z_1 = \sqrt{2+\sqrt{2}} + \sqrt{2-\sqrt{2}} = 2.61312592975275\ldots \tag{3.14}$$

$$z_2 = \sqrt{2+\sqrt{2}} - \sqrt{2-\sqrt{2}} = 1.08239220029239\ldots \tag{3.15}$$

$$z_1^2 = 4 + 2\sqrt{2} = 6.82842712474619\ldots \tag{3.16}$$

$$z_2^2 = 4 - 2\sqrt{2} = 1.171572875253810\ldots$$

$$z_1 z_2 = 2\sqrt{2} = 2.82842712474619\ldots \tag{3.17}$$

$$z_1 z_2^2 = 4\sqrt{2-\sqrt{2}} = 3.06146745892072\ldots \tag{3.18}$$

$$z_1^2 z_2 = 4\sqrt{2+\sqrt{2}} = 7.39103626009029\ldots \tag{3.19}$$

$$z_1^2 z_2^2 = 8 \tag{3.20}$$

Approximated Values:

$$\widetilde{z_1} = 2676/2^{10} = 669/2^8 = 2.61328125 \tag{3.21}$$

$$\widetilde{z_2} = 2217/2^{11} = 1.08251953125 \tag{3.22}$$

$$\widetilde{z_1^2} = 3496/2^{11} = 437/2^6 = 6.828125 \tag{3.23}$$

$$\widetilde{z_2^2} = 2399/2^6 = 1.17138671875 \tag{3.24}$$

$$\widetilde{z_1 z_2} = 2896/2^{10} = 181/2^6 = 2.828125 \tag{3.25}$$

$$\widetilde{z_1 z_2^2} = 3135/2^{10} = 3.0615234375 \tag{3.26}$$

$$\widetilde{z_1^2 z_2} = 3784/2^9 = 473/2^6 = 7.390625 \tag{3.27}$$

Using the DM methods, the above constants in Equations 3.14 to 3.20 can be optimally encoded as single-constant multipliers, by using the minimum possible number of 2-input adders per Section 3.1, as shown in Table 3.1. This table lists the SCMs for each integer associated with the approximated z-values in Equations 3.21 to 3.27. For example, the integer associated with the approximation $\widetilde{z_1}$ is 2676 (Equation 3.21). The row for 2676 in Table 3.1 shows the SCM algorithm for computing the integer multiplier. The result of the multiplication by 2676 must be scaled down by 2^{10}, to correctly implement the multiplication by $\widetilde{z_1}$.

From the table of constants, Equations of 3.11 to 3.13 can be implemented in hardware to construct the 12-bit SCM-based FRS for 2D AI, as shown in Figures 3.9, 3.10, and 3.11. These designs were used by the author in [55, 56] and obtained by the tool described in Section 3.1. In these figures, a pipelined architecture is shown whereby a sequence of registers gates the data flow through combinational logic, in order to increase the data throughput rate. In the figures: a flip-flop is shown to represent a register, wires represent data buses, and adders represent the combinational logic. Powers of two on the wires (e.g. 2^3 or 2^{-10}) represent wired shifts, which are considered free in hardware. Negative values on the wires (e.g -1, or -2^3) transform the associated adders into subtractors. Since adders and subtractors are very similar in hardware cost, these negative edge values are considered cost-free.

For example, consider the hardware implementation of Equation 3.11, as shown in Figure 3.9. Input x_2 is multiplied by z_1, which is approximated by $\widetilde{z_1} = 2676/2^{10}$. The circuit that implements this operation is the second circuit from the top of Figure 3.9, with input x_2. The data flow can be followed and compared to the SCM algorithm for 2676 in Table 3.1, culminating in a final scaling of 2^{-10}, before being added to the other terms of the S-equation. In detail, the explanation of how the circuit for input x_2, as shown in Figure 3.9, implements the algorithm for the constant integer multiplication for 2676 as shown in Table 3.1, we can trace the data flow as follows. The input x_2 passes through an input

register (symbolized by a Flip Flop) and splits into three branches, where the middle branch passes x_2 to one input of a 2-input adder. The upper branch scales x_2 by 2, and feeds it to input the other input of the first 2-input adder. This operation implements the operation $v_1 = (1+2) \cdot x_2$, where v_1 is a label for an intermediate variable in the algorithm ("v" actually represents "vertex", in the DM nomenclature). This equation for v_1 corresponds to the first line of the SCM algorithm for 2676, in row 2 and column 2 of Table 3.1. The remaining branches of the circuit can be similarly traced to obtain the remaining steps for the SCM algorithm for 2676. As well, a similar analysis can be performed to verify the remaining circuits of Figures 3.9, 3.10, and 3.11, based on Equations 3.21 to 3.27, and using the SCM algorithms of Table 3.1.

3.4.2.2 Substitution Method for AI

Examination of the z-values in Equations 3.14 to 3.20 led Dimitrov to notice that some z-values could be rewritten as functions of other z-values. This led to the development of the substitution method for AI. It is noted that this substitution method was not previously published, and arose during our work on AI. The summary of z-substitutions is as follows.

$$z_1 = (z_1^2 \cdot z_2 - 2 \cdot z_2)/2 \tag{3.28}$$

$$z_2 = (z_1^2 \cdot z_2 - 2 \cdot z_1)/2 \tag{3.29}$$

$$z_2 = (2 \cdot z_1 - z_1 \cdot z_2^2)/2 \tag{3.30}$$

$$z_1^2 = 4 + z_1 \cdot z_2 \tag{3.31}$$

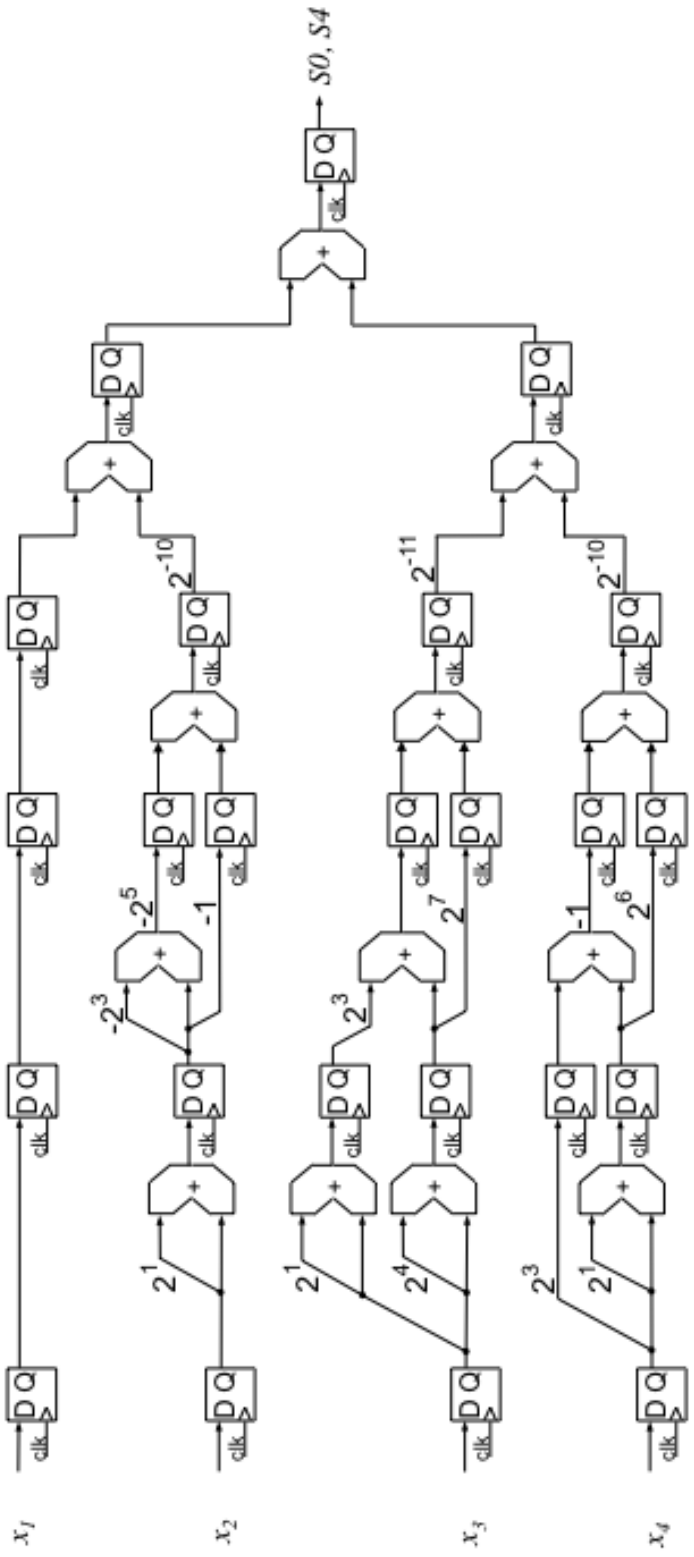

Figure 3.9: Pipelined SCM architecture for S0 and S4

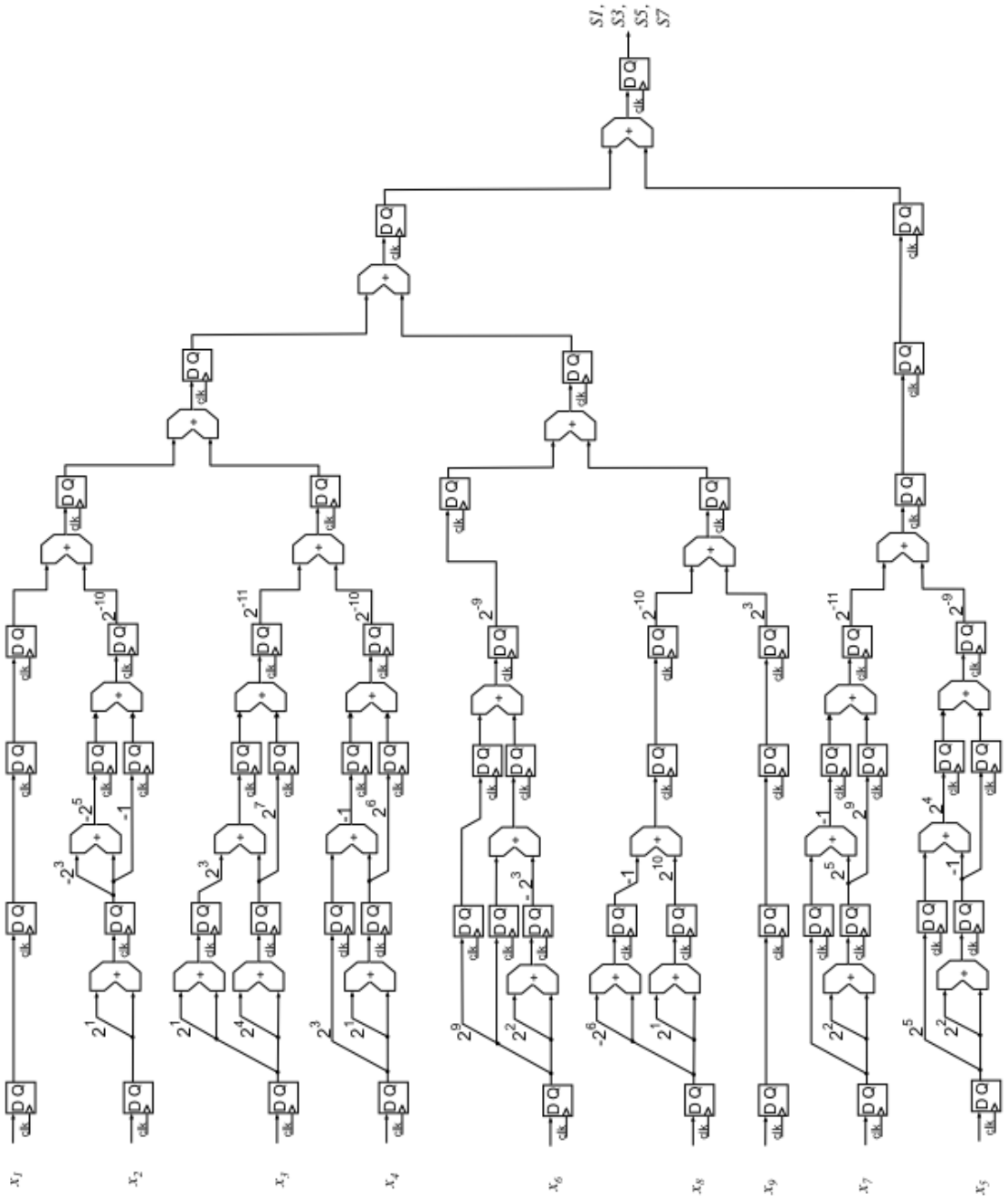

Figure 3.10: Pipelined SCM architecture for S1, S3, S5, S7

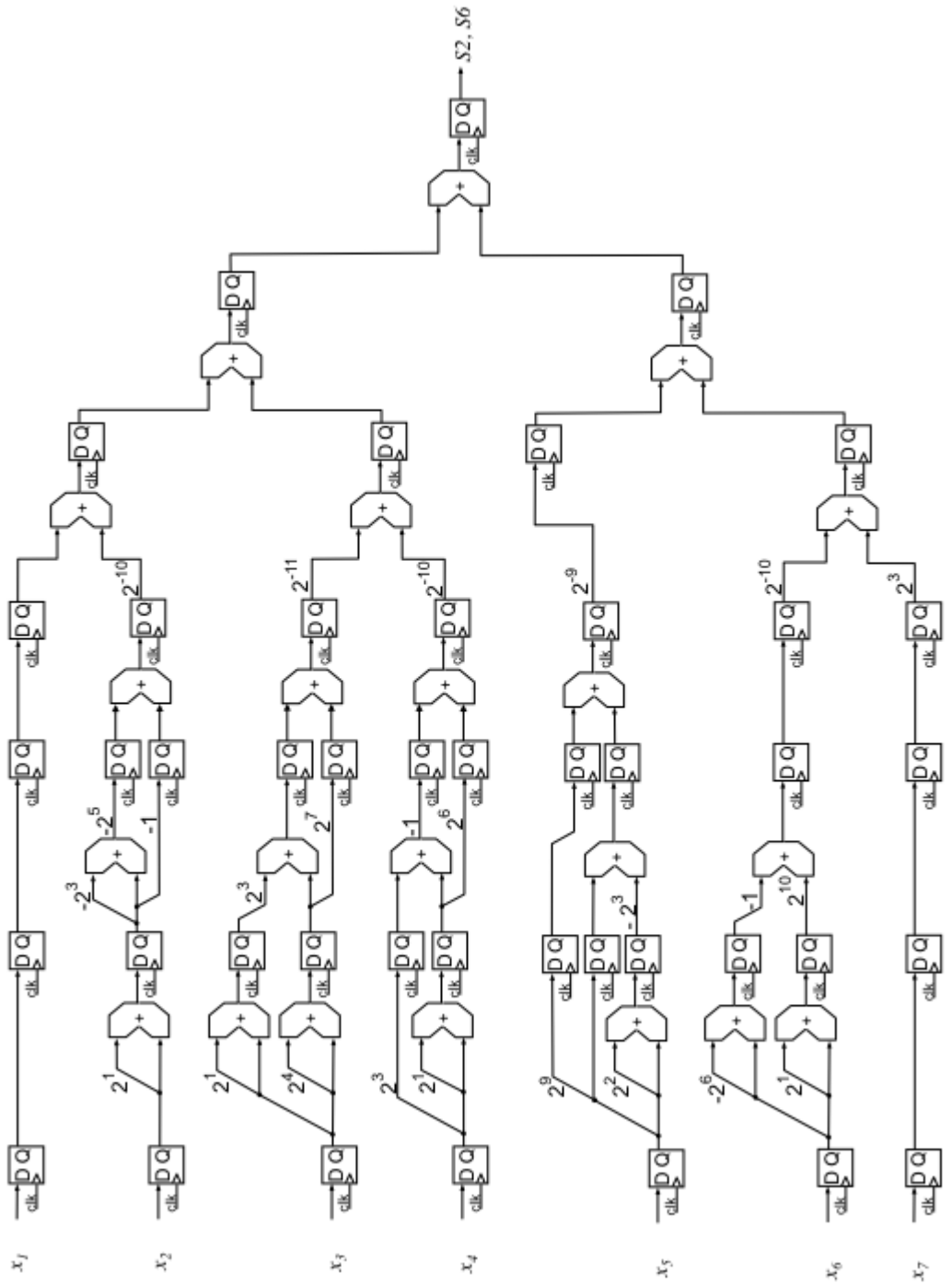

Figure 3.11: Pipelined SCM architecture for S2 and S6

$$z_1^2 = 8 - z_2^2 \tag{3.32}$$

$$z_2^2 = 4 - z_1 \cdot z_2 \tag{3.33}$$

$$z_2^2 = 8 - z_1^2 \tag{3.34}$$

$$z_1 z_2 = z_1^2 - 4 \tag{3.35}$$

$$z_1 z_2 = 4 - z_2^2 \tag{3.36}$$

$$z_1^2 z_2 = 2 \cdot z_1 + 2 \cdot z_2 \tag{3.37}$$

$$z_1 z_2^2 = 2 \cdot z_1 - 2 \cdot z_2 \tag{3.38}$$

Using these 11 substitutions, the FRS equations can be rewritten in 18 different variations, as found through exhaustive trials with Mathematica. The 18 variations have only 3 z-values, because they are simplified decompositions, using Equations 3.28 to 3.38. Since the Type 2 is a superset of Type 1 and 3, it will be used to illustrate the variations below.

$$S = x_1 + z_1 \cdot x_2 + z_2 \cdot x_3 + z_1 z_2 \cdot x_4 + z_1^2 \cdot x_5 + z_1^2 z_2 \cdot x_6 + z_2^2 \cdot x_7 + z_1 z_2^2 \cdot x_8 + z_1^2 z_2^2 \cdot x_9 \tag{3.39}$$

where x_1 to x_9 are generalized input data coming the FRS, following the format of Equation 2.18.

Five basic approaches were considered to apply the z-substitutions, as below:

Approach 0: Use the substitutions to rewrite Equation 3.39 as a function of three distinct z-values.

E.g. Select Equations 3.31, 3.37, 3.33, 3.38, and recognizing that $z_1^2 z_2^2 = 8$, rewrite Equation 3.39 as 3.40, which is now a function of only z_1, z_2, and $z_1 z_2$. Then manipulate the equation to obtain the form of Equation 3.41.

$$\begin{aligned} S &= x_1 + z_1 \cdot x_2 + z_2 \cdot x_3 + z_1 z_2 \cdot x_4 + (4 + z_1 \cdot z_2) \cdot x_5 + \\ &\quad (2 \cdot z_1 + 2 \cdot z_2) \cdot x_6 + (4 - z_1 \cdot z_2) \cdot x_7 + (2 \cdot z_1 - 2 \cdot z_2) \cdot x_8 + 8 \cdot x_9 \end{aligned} \tag{3.40}$$

$$S = d_1 + z_1 \cdot d_2 + z_2 \cdot d_3 + z_1 z_2 \cdot d_4 \tag{3.41}$$

Where the d-values are the simplified input data values of the x-values (e.g. $d_1 = x_1 + 4 \cdot x_5 + 4 \cdot x_7 + 8 \cdot x_9$, etc.). This type of procedure resulted in 18 unique solutions for approach 0. The most important result is that with approach 0, all Equations 3.11 to 3.13 are all functions of only 3 z-values. This is a dramatic decrease from Equations 3.12 and 3.13, leading to designs with lower computational complexity and improved MSE performance. The reason there is reduced computational complexity is because there are fewer constant multiplications to encode, from 7 down to 3. The reason there is improved MSE performance is because there are fewer constants to quantize, and also because there are more possible equations to create a design (18 versus 1). All of the subsequent approaches are variations of Approach 0.

Approach 1: Extract a common factor, $\frac{1}{m}$, from the z-values in Approach 0,

$$S = d_1 + \frac{1}{m} \cdot (z_1 \cdot m \cdot d_2 + z_2 \cdot m \cdot d_3 + z_1 z_2 \cdot m \cdot d_4) \tag{3.42}$$

then simplify the constants to,

$$S = d_1 + \frac{1}{m} \cdot (k_1 \cdot d_2 + k_2 \cdot d_3 + k_3 \cdot d_4) \tag{3.43}$$

Approach 2: Similar to Approach 1, extract a common z-value from the z-values in Approach 0,

$$S = d_1 + z_1 \cdot (d_2 + \frac{z_2}{z_1} \cdot d_3 + \frac{z_1 z_2}{z_1} \cdot d_4) \tag{3.44}$$

There are 54 unique solutions for Approach 2, as any one of the 3 z-values can be extracted from each equation of Approach 0.

Approach 3: Extract a common factor from the entire equation, $\frac{1}{m}$, from the z-values in Approach 0,

$$S = \frac{1}{m} \cdot (m \cdot d_1 + z_1 \cdot m \cdot d_2 + z_2 \cdot m \cdot d_3 + z_1 z_2 \cdot m \cdot d_4) \tag{3.45}$$

then simplify the constants to,

$$S = \frac{1}{m} \cdot (k_1 \cdot d_1 + k_2 \cdot d_2 + k_3 \cdot d_3 + k_3 \cdot d_4) \tag{3.46}$$

Approach 4: Similar to Approach 3, extract a common z-value from the entire equation,

$$S = z_1 \cdot (\frac{1}{z_1} \cdot d_1 + d_2 + \frac{z_2}{z_1} \cdot d_3 + \frac{z_1 z_2}{z_1} \cdot d_4) \tag{3.47}$$

Note that the extracted common factor will not be computed in the FRS, as it is consid-

Table 3.8: Comparison of multipliers for simplifying FRS equations

Approach	No. SCM	No. MCM	**Total No. KCM**
Full (i.e. Equation (3.39))	7	0	**7**
Approach 0	0	3	**3**
Approach 1	1	3	**4**
Approach 2	1	2	**3**
Approach 3	0	4	**4**
Approach 4	0	3	**3**

ered "embedded" within the Arai constant. There are 54 unique solutions for Approach 4, as any one of the 3 z-values can be extracted from each equation of Approach 0.

Examination of the above approaches yields the varying multiplication requirements for each approach, as shown in Table 3.8. Reduction of the number of KCMs in the FRS will result in smaller and more efficient designs, so Approaches 0, 2, and 4, were selected as the best candidates for further analysis.

3.5 Sample Implementations

This section presents samples of the best designs for each of the design approaches for the 2D Arai DCT, as developed in this thesis. These design approaches include:

- the full AI SCM approach (Contribution 1),
- the full AI MCM approach (Contribution 1),
- the simplified AI approaches 0, 2, 4, (Contribution 2), and
- the non-AI approach, as a benchmark.

The metrics for assessing the designs were MSE performance, adder cost, and adder-bits cost. MSE was used to determine how suitable the design was for its function, with the lowest MSE being most desirable. The adder cost is the number of adders or subtractors required to implement the design. The adder cost is useful for assessing the computational

complexity for software implementation, or for circuit complexity for hardware implementation. The adder-bits cost is a more accurate measure for the circuit complexity of hardware implementations, because it also measures the size of the adders and subtractors, rather than simply counting the number of adders and subtractors. The author's tool was used for designing the SCMs, per Section 3.1. PARI was used to design the MCMs, per Section 3.2.

Each of the 4608 AI designs was analyzed for its MSE performance, and the lowest MSE AI design was selected for each design approach, at each coefficient quantization level using the author's tool in Section 3.4. The AI approaches were evaluated for the coefficient quantization levels 8- to 16-bits. The non-AI approaches were evaluated for the coefficient quantization levels 8- to 16-bits. The rationale for evaluating such a large range of design approaches is to provide a basis for comparing AI versus non-AI approaches and form general conclusions of their respective merits in application to the 2D Arai DCT.

Each of the best MSE candidates was then evaluated for their adder cost and their adder-bits cost using the author's tool described in Section 3.4. The results are given in a comprehensive analysis and summary of results, in Chapter 4. In the immediately following sections, selected samples of the best AI designs' FRSs are described mathematically and their MSE performance is plotted for each 2D DCT output coefficient. For the samples of the best FRS's, selected samples of the algorithm to compute the FRSs are also provided. The next section provides samples of the non-AI approach.

3.5.1 Non-AI Designs

The designs of the non-AI approach directly follows the Arai 1D DCT algorithm [54], as shown in Figure 2.1 and Table 3.7. The 2D DCT algorithms of Section 3.4.2 are followed in these designs. The designs for non-AI range from 8- to 16-bits of coefficient quantization accuracy. This section presents the SCMs of the 8-bit and 16-bit quantized coefficients for the non-AI designs. The author's design tool, Section 3.1, was used to generate the SCMs

Table 3.9: SCM algorithms for 8-bit Arai m-values

$m_1 = 181/2^8$	$m_2 = 196/2^9$	$m_3 = 139/2^8$	$m_4 = 167/2^7$
V0 = in	V0 = in	V0 = in	V0 = in
V1 = V0 + V0<<1	V1 = V0 + V0<<1	V1 = V0 + V0<<1	V1 = V0 + V0<<2
V2 = V0<<3 + V1	V2 = V0<<2 + V1<<6	V2 = V0<<3 + V1	V2 = V0<<1 + V1
V3 = V1<<6 - V2	out = V2<<-9	V3 = V0<<7 + V2	V3 = V1<<5 + V2
out = V3<<-8		out = V3<<-8	out = V3<<-7

for the 8- to 16-bit quantized coefficients. A summary of the performance of all the 8- to 16-bit designs is shown in Chapter 4.

The purpose of the non-AI designs was to provide a reference to which the AI designs could be compared.

3.5.1.1 8-bit Designs

This section details the SCMs for the 8-bit quantized coefficients, m_1 to m_4. In Table 3.9, the 8-bit quantized coefficients, which approximate the m-values, are shown, followed beneath by their DM SCM algorithm. It can been seen that only m_3 follows the binary circuit (i.e. graph 1 of column 3 in Figure 2.7). It can also be seen that all the SCMs have 3 adders, except m_2, which has 2.

Figure 3.14 shows the MSEs for the 2D DCT coefficients of this design, completed over 10000 iterations of a randomized 8-bit unsigned 8x8 input matrix. The average MSE of all the coefficients is 0.666. It is noted that 2D DCT coefficient number 10 is particularly high, over 5. Coefficient number 28 is also very high, at 2.6. The reason why some coefficients have much higher error is because, by examining the 2D DCT, there can seen 64 computation channels (i.e. one for each coefficient), and the coefficients with the highest error (i.e. numbers 10 and 28) have SCMs (i.e. m_1 to m_4) with the highest quantization error. As noted previously, a feature of AI methods is to allow the accuracy of each output coefficient to be individually calibrated, which is not applicable to this non-AI example.

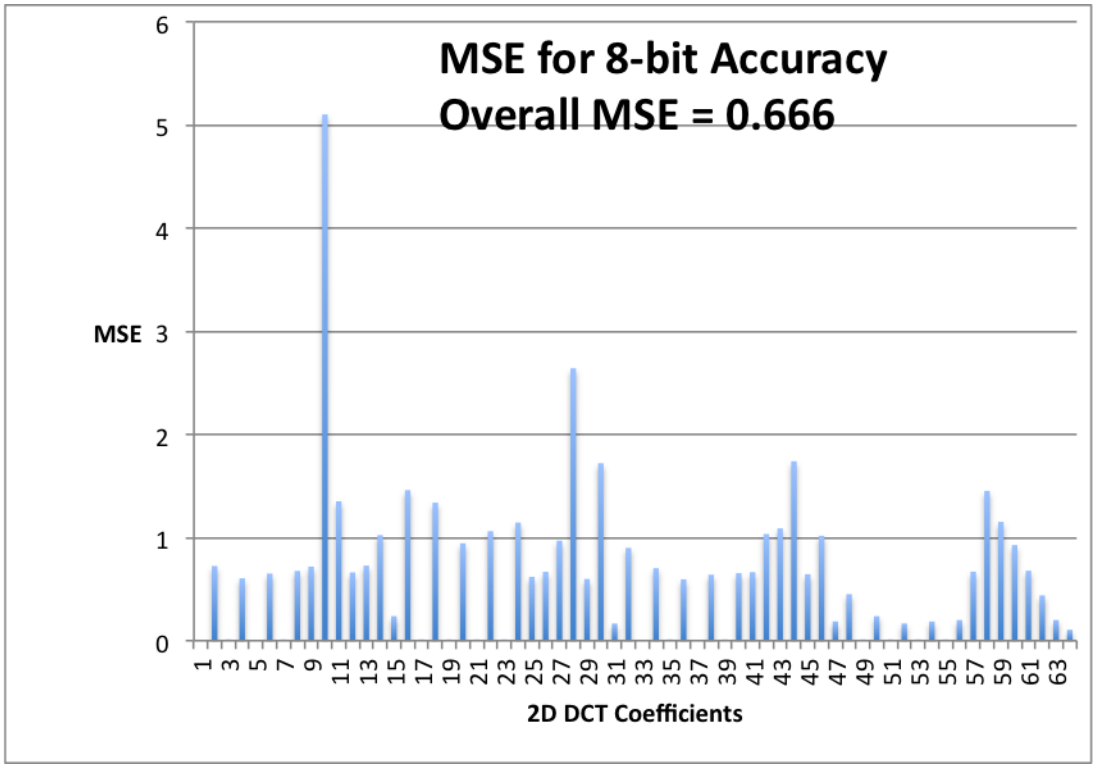

Figure 3.12: MSE for 8-bit, Non-AI

3.5.1.2 16-bit Designs

This section details the 16-bit SCMs for the non-AI design. It is observed in Table 3.10 that all the SCMs have 4 adders, in order to accurately represent m_1 to m_4, a higher number of adders, compared with the previous 8-bit design.

Figure 3.15 shows the MSE plot for the 16-bit design. It is observed that the average MSE is 1.67-e5, a large improvement over the 8-bit design of 0.666. Coefficient 10 is again very high, compared to over coefficient's MSEs (as in the 8-bit design), at about 0.00021. However, we can see that coefficient 28 is no longer prominent, as it was in the 8-bit design.

As is typical, and expected, increases in quantization accuracy results in decreased MSE and increased hardware costs. This is because when coefficients are quantized to higher accuracy, their quantization error decreases, hence their MSE decreases. There are increased hardware costs because SCMs tend to have more adders when representing larger integers, on average. This makes the 16-bit design more costly in terms of hardware, as can

Table 3.10: SCM algorithms for 16-bit Arai m-values

$m_1 = 46341/2^{16}$	$m_2 = 50159/2^{17}$	$m_3 = 35468/2^{16}$	$m_4 = 42813/2^{15}$
V0 = in	V0 = in	V0 = in	V0 = in
V1 = V0+V0<<2	V1 = V0+V0<<2	V1 = V0+V0<<1	V1 = V0<<0+V0<<1
V2 = V0<<4+V1	V2 = V0+V1<<6	V2 = V0<<5+V1	V2 = V1<<0-V1<<3
V3 = V1<<5+V2	V3 = V0<<5-V1	V3 = V0<<7-V2	V3 = V1<<0+V1<<6
V4 = V1+V3<<8	V4 = V2<<1+V3<<11	V4 = V2<<10-V3<<2	V4 = V2<<11-V3
out = V4<<-16	out = V4<<-17	out = V4<<-16	out = V4<<-15

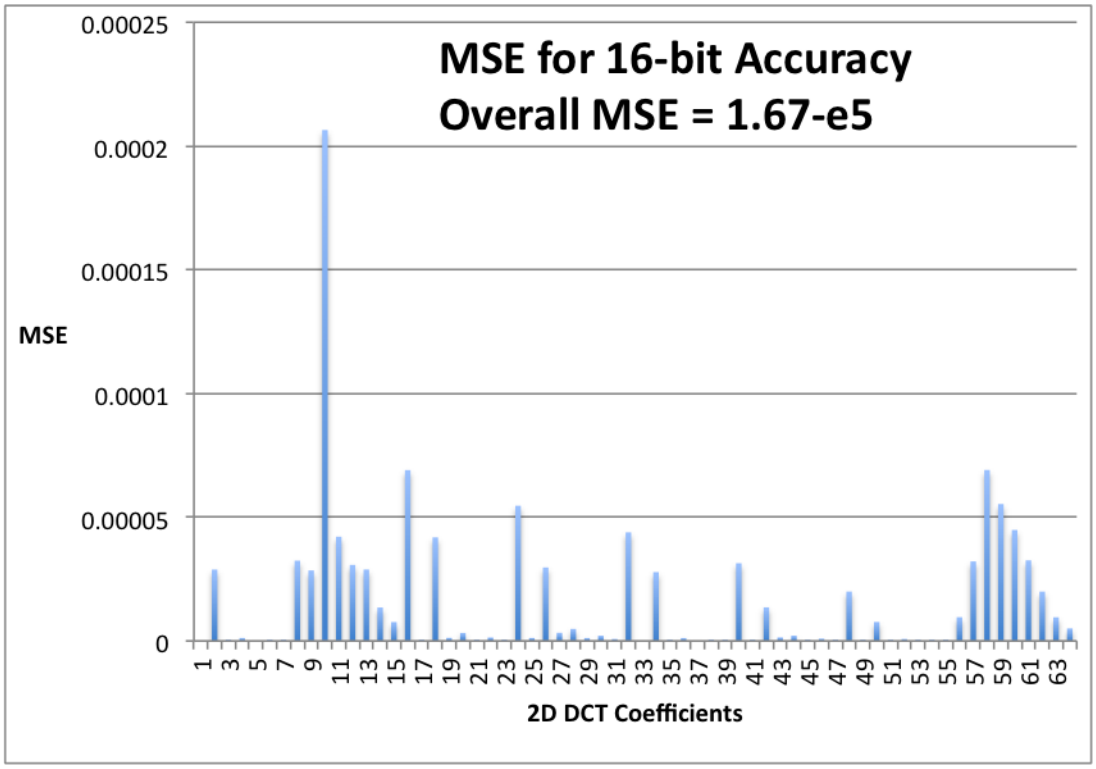

Figure 3.13: MSE for 16-bit, Non-AI

be expected.

3.5.2 AI Designs

The section details the AI designs investigated in this thesis. The full AI approach is discussed, in its SCM and MCM variants, primarily as a reference for the improved, simplified AI designs. The SCM and MCM variants are discussed to illustrate the benefits of MCM over SCM. All the AI designs are for 8- to 16-bits of coefficient quantization. Selected samples of each design are described in this section, at 8- and 16-bits of coefficient quantization. A summary for all 8- to 16-bit levels of coefficient quantization for AI is given in Chapter 4.

For AI approaches 0, 2, and 4, a larger set of designs was analyzed. For AI approach 0, there are 18 possible designs for each bit level of accuracy, leading to 162 possible designs. Similarly, AI approaches 2 and 4 each have 54 possible designs at each bit level

of accuracy, leading to 486 possible designs each. The best design (i.e. lowest MSE) was chosen at each level of coefficient quantization, for each AI approach. The 9 best designs, for each approach, are reported as a summary in Chapter 4. Selected samples of the best designs for each approach, at 8- and 16-bits, is given in Sections 3.5.2.3 to 3.5.2.6.

The sample designs are presented in each of their 3 prototype forms, Type 1, 2, and 3, as described in Section 3.4.2, as S(i), S(j), and S(k), respectively. The purpose of identifying and presenting the 3 types, is to recognize that there are different computational requirements between types, and to generalize the set of 8 FRS outputs (S0, S1, ..., S7) in a compact, non-redundant format (3 equations instead of 8).

All of the SCMs for the AI designs were provided by the author's software tools, per Section 3.1. The MCMs were provided by PARI, per Section 3.2. Similar to the non-AI designs, it is observed that, as the accuracy of coefficient quantization increases, hardware costs increase and MSE decreases.

3.5.2.1 Common Subexpressions

Common subexpression elimination (CSE) reduces computational effort by pre-computing common subexpressions that are used multiple times within the overall computation. The common subexpressions are obtained from the FRS equation 2.18, and are stated below:

$$t_1(j) = s_2(j) + s_5(j), \quad j \in \{1,3,5,7\} \tag{3.48}$$

$$t_2(j) = s_3(j) + s_9(j), \quad j \in \{1,3,5,7\} \tag{3.49}$$

$$t_3(j) = s_4(j) + s_7(j) + s_{10}(j) + s_{13}(j), \quad j \in \{1,3,5,7\} \tag{3.50}$$

$$t_4(j) = s_8(j) + s_{14}(j), \quad j \in \{1,3,5,7\} \tag{3.51}$$

$$t_5(j) = s_{12}(j) + s_{15}(j), \quad j \in \{1,3,5,7\} \tag{3.52}$$

$$t_6(k) = s_4(k) + s_{13}(k), \quad k \in \{2,6\} \tag{3.53}$$

CSE assists in reducing the computational complexity of the simplified AI designs, because the subexpressions are used more than once, so they can be precomputed for the multiple usage. For the full AI designs, it merely assists in presenting the S-equations in a more compact manner, with no computational advantage because any CSE appears at most once in the FRS.

3.5.2.2 SCM and MCM Designs for Full AI

This section details the sample designs for the full AI implementation. The full AI implementation has a direct implementation of the FRS Equation 2.8, represented in [2–6, 37, 49–52], and Equation 2.18, represented in [55, 56]. The formulation of these equations is described in Section 2.4.2. Note that Equation 2.18 has only been previously investigated for coefficient quantization accuracies of 12-bits [55, 56], as an SCM.

3.5.2.2.1 SCM Designs - 8-bit The SCM-based design for the full AI design for 8-bit coefficient quantization, is presented in this section. The FRS equations for S0 to S7 are given below.

$$S(i) = s_1(i) + z_1 \cdot s_5(i) + z_2 \cdot s_9(i) + z_1 \cdot z_2 \cdot s_{13}(i), \quad i \in \{0,4\} \tag{3.54}$$

Table 3.11: SCM algorithms for 8-bit full AI, part 1

$\widetilde{z_1} = 167/2^6$	$\widetilde{z_2} = 139/2^7$	$\widetilde{z_1^2} = 219/2^5$	$\widetilde{z_2^2} = 150/2^7$
V0 = x5	V0 = x7	V0 = x2	V0 = x6
V1 = V0 + V0<<2	V1 = V0 + V0<<1	V1 = V0 + V0<<2	V1 = V0 + V0<<2
V2 = V0<<1 + V1	V2 = V0<<3 + V1	V2 = V0<<1 + V1	V2 = V1<<1 + V1<<5
V3 = V1<<5 + V2	V3 = V0<<7 + V2	V3 = V1 + V2<<5	X6=V2<<-7
X5 = V3<<-6	X7 = V3<<-7	X2 = V3<<-5	

Table 3.12: SCM algorithms for 8-bit full AI, part 2

$\widetilde{z_1 z_2} = 181/2^6$	$\widetilde{z_1 z_2^2} = 196/2^6$	$\widetilde{z_1^2 z_2} = 237/2^5$	$z_1^2 z_2^2 = 8$
V0 = x4	V0 = x3	V0 = x1	X8 = x8<<3
V1 = V0 + V0<<1	V1 = V0 + V0<<1	V1 = V0 + V0<<1	
V2 = V0<<3 + V1	V2 = V0<<2 + V1<<6	V2 = V0<<4 + V1	
V3 = V1<<6 - V2	X3 = V2<<-6	V3 = V0<<8 - V2	
X4 = V3<<-6		X1 = V3<<-5	

$$\begin{aligned} S(j) &= s_1(j) + z_1 \cdot t_1(j) + z_2 \cdot t_2(j) + \\ &\quad z_1 z_2 \cdot t_3(j) + z_1^2 \cdot s_6(j) + \\ &\quad z_1^2 z_2 \cdot t_4(j) + z_2^2 \cdot s_{11}(j) + z_1 z_2^2 \cdot t_5(j) + \\ &\quad z_1^2 z_2^2 \cdot s_{16}(j), \quad j \in \{1,3,5,7\} \end{aligned} \tag{3.55}$$

$$\begin{aligned} S(k) &= s_1(k) + z_1 \cdot s_5(k) + z_2 \cdot s_9(k) + z_1 z_2 \cdot t_6(k) + \\ &\quad z_1^2 z_2 \cdot s_8(k) + z_1 z_2^2 \cdot s_{12}(k) + z_1^2 z_2^2 \cdot s_{16}(k), \quad k \in \{2,6\} \end{aligned} \tag{3.56}$$

The 8-bit coefficient quantization values, which approximate the z-values, are given in Tables 3.11 and 3.12. Below each z-value approximation, its associated DM-based SCM algorithm follows.

Note that $z_1^2 z_2^2$ does not require approximation, as 8 is an exact value, implemented by

Table 3.13: SCM-based FRS algorithm for S(i), 8-bit, full AI

$S(i) = x_0 + z_1 \cdot x_5 + z_2 \cdot x_7 + z_1 z_2 \cdot x_4$			
c=x0<<2	V0=x7	V1 = V0 + V0<<1	t = t + X4
V0=x5	V1 = V0 + V0<<1	V2 = V0<<3 + V1	S = t<<-2
V1 = V0 + V0	V2 = V0<<3 + V1	V3 = V1<<6 - V2	
V2 = V0<<1 + V1	V3 = V0<<7 + V2	X4=V3<<-6	
V3 = V1<<5 + V2	X7=V3<<-7	t = c + X5	
X5=V3<<-6	V0=x4	t = t + X7	

<<3.

Tables 3.13 to 3.15 are the generalized algorithms for the 3 FRS equation types for S0 to S7. The x-values in the tables can be easily obtained from Equations 3.54 to 3.56. For example, x_0 in the 3 equations below will refer to $s_1(\cdot)$ in Equations 3.54 to 3.56. As another example, x_5 in the 3 equations below will refer to $s_5(i)$ in Equation 3.54, $t_1(j)$ in Equation 3.55, and $s_5(k)$ in Equation 3.56.

Each S-equation in Tables 3.13 to 3.15 shares a common set of constants, which eases the effort of a design implementation. After each S-equation, the algorithm for generating the final S-value is shown, incorporating the 8-bit quantized coefficient's SCMs above. As can be observed, the algorithm for $S(j)$ is the most computationally expensive, because it has 7 SCMs, and $S(i)$ is the least expensive, because it has only 3 SCMs. The algorithms are read starting from the top of the first row and leftmost column, then progressing down the rows of the first column, then progressing to the next column to the right. Note that the labels for V0 to V3 are reused for each SCM, because the mathematical form of the algorithm is shown. In a VLSI implementation, the V's would have a unique label for each SCM, or be hidden within a module for each SCM.

Figure 3.14 shows the MSE plot for 8-bit full AI for the 2D DCT. Its average MSE of 0.525 is lower than the 0.666 MSE of the 8-bit non-AI design. As well, Figure 3.14 does not have a singleton outlying spike, as was the case for the non-AI designs. Rather, the MSEs for the DCT coefficients appear more uniformly distributed than the non-AI designs.

Table 3.14: SCM-based FRS algorithm for S(j), 8-bit, full AI

$S(j) = x_0 + z_1 \cdot x_5 + z_2 \cdot x_7 + z_1 z_2 \cdot x_4 + z_1^2 \cdot x_2 + z_1^2 z_2 \cdot x_1 + z_2^2 \cdot x_6 + z_1 z_2^2 \cdot x_3 + z_1^2 z_2^2 \cdot x_8$			
c=x0<<2+x8<<3	V0=x4	V2 = V0<<4 + V1	X6=V2<<-7
V0=x5	V1 = V0 + V0<<1	V3 = V0<<8 - V2	t = c + X1
V1 = V0 + V0	V2 = V0<<3 + V1	X1=V3<<-5	t = t + X2
V2 = V0<<1 + V1	V3 = V1<<6 - V2	V0=x2	t = t + X3
V3 = V1<<5 + V2	X4=V3<<-6	V1 = V0 + V0<<2	t = t + X4
X5=V3<<-6	V0=x3	V2 = V0<<1 + V1	t = t + X5
V0=x7	V1 = V0 + V0<<1	V3 = V1 + V2<<5	t = t + X6
V1 = V0 + V0<<1	V2 = V0<<2 + V1<<6	X2=V3<<-5	t = t + X7
V2 = V0<<3 + V1	X3=V2<<-6	V0=x6	S = t<<-4
V3 = V0<<7 + V2	V0=x1	V1 = V0 + V0<<2	
X7=V3<<-7	V1 = V0 + V0<<1	V2 = V1<<1 + V1<<5	

Table 3.15: SCM-based FRS algorithm for S(k), 8-bit, full AI

$S(k) = x_0 + z_1 \cdot x_5 + z_2 \cdot x_7 + z_1 z_2 \cdot x_4 + z_1^2 z_2 \cdot x_1 + z_1 z_2^2 \cdot x_3 + z_1^2 z_2^2 \cdot x_8$		
c=x0<<2+x8<<3	V0=x4	V2 = V0<<4 + V1
V0=x5	V1 = V0 + V0<<1	V3 = V0<<8 - V2
V1 = V0 + V0	V2 = V0<<3 + V1	X1=V3<<-5
V2 = V0<<1 + V1	V3 = V1<<6 - V2	t = c + X1
V3 = V1<<5 + V2	X4=V3<<-6	t = t + X3
X5=V3<<-6	V0=x3	t = t + X4
V0=x7	V1 = V0 + V0<<1	t = t + X5
V1 = V0 + V0<<1	V2 = V0<<2 + V1<<6	t = t + X7
V2 = V0<<3 + V1	X3=V2<<-6	S = t<<-4
V3 = V0<<7 + V2	V0=x1	
X7=V3<<-7	V1 = V0 + V0<<1	

3.5.2.2.2 SCM Designs - 16-bit The SCM algorithms for the 16-bit full AI design are presented in this section, following Equations 3.54 to 3.56, in the same format as the previous section. The equations are reproduced below for the convenience of the reader.

$$S(i) = s_1(i) + z_1 \cdot s_5(i) + z_2 \cdot s_9(i) + z_1 \cdot z_2 \cdot s_{13}(i), \quad i \in \{0,4\}$$

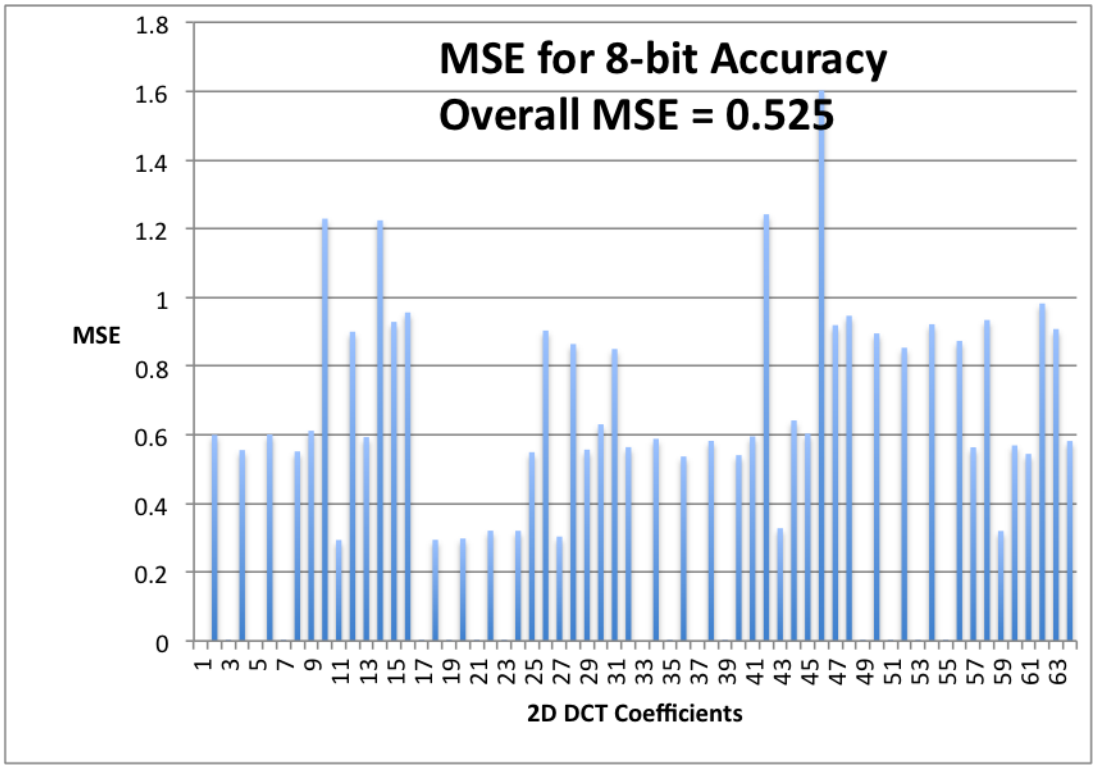

Figure 3.14: MSE for 8-bit, Full AI Implementation

$$\begin{aligned} S(j) &= s_1(j) + z_1 \cdot t_1(j) + z_2 \cdot t_2(j) + \\ &\quad z_1 z_2 \cdot t_3(j) + z_1^2 \cdot s_6(j) + \\ &\quad z_1^2 z_2 \cdot t_4(j) + z_2^2 \cdot s_{11}(j) + z_1 z_2^2 \cdot t_5(j) + \\ &\quad z_1^2 z_2^2 \cdot s_{16}(j), \quad j \in \{1,3,5,7\} \end{aligned}$$

$$\begin{aligned} S(k) &= s_1(k) + z_1 \cdot s_5(k) + z_2 \cdot s_9(k) + z_1 z_2 \cdot t_6(k) + \\ &\quad z_1^2 z_2 \cdot s_8(k) + z_1 z_2^2 \cdot s_{12}(k) + z_1^2 z_2^2 \cdot s_{16}(k), \quad k \in \{2,6\} \end{aligned}$$

The 16-bit quantized coefficients approximating the z-values are given in Tables 3.16 to 3.17, with their DM SCM algorithms. It can be seen that more adders are required for the

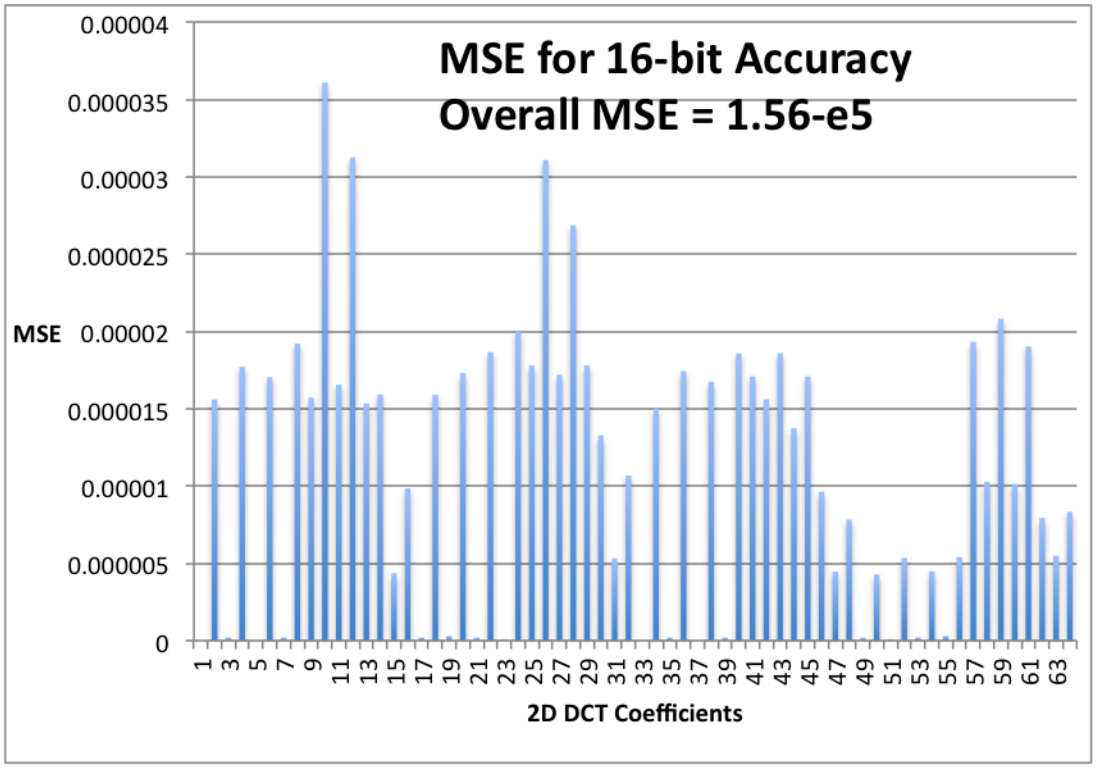

Figure 3.15: MSE for 16-bit, Full AI Implementation

16-bit design, compared to the 8-bit design, due to the increased quantization level, leading to more complex SCMs.

The algorithms incorporating the quantized coefficients in Tables 3.16 to 3.17, are presented in Tables 3.18 to 3.20 for $S(i)$, $S(j)$, and $S(k)$. Again, $S(j)$ has the highest complexity and $S(i)$ has the lowest complexity.

Figure 3.15 shows the MSE plot for the 16-bit accuracy design for full AI with SCM. It is observed that the overall MSE at 1.56e-5 is slightly lower than that of the 16-bit non-AI design of 1.67e-5.

3.5.2.2.3 MCM Designs - 8-bit The MCM designs for the 8-bit full AI design are presented in this section, following Equations 3.54 to 3.56, in the same format as the previous section. The equations are reproduced below for the convenience of the reader.

$$S(i) = s_1(i) + z_1 \cdot s_5(i) + z_2 \cdot s_9(i) + z_1 \cdot z_2 \cdot s_{13}(i), \quad i \in \{0,4\}$$

Table 3.16: SCM algorithms for 16-bit full AI, part 1

$\widetilde{z_1} = 42813/2^{14}$	$\widetilde{z_2} = 35468/2^{5}$	$\widetilde{z_1^2} = 50159/2^{14}$	$\widetilde{z_2^2} = 60547/2^{13}$
V0 = x5	V0 = x7	V0 = x2	V0 = x6
V1 = V0<<0+V0<<1	V1 = V0+V0<<1	V1 = V0+V0<<2	V1 = V0+V0<<2
V2 = V1<<0-V1<<3	V2 = V0<<5+V1	V2 = V0+V1<<6	V2 = V1-V1<<4
V3 = V1<<0+V1<<6	V3 = V0<<7-V2	V3 = V0<<5-V1	V3 = V1<<1-V2<<9
V4 = V2<<11-V3	V4 = V2<<10-V3<<2	V4 = V2<<1+V3<<11	X6 = V3<<-15
X5 = V4<<-14	X7 = V4<<-15	X2 = V4<<-13	

Table 3.17: SCM algorithms for 16-bit full AI, part 2

$\widetilde{z_1 z_2} = 46341/2^{14}$	$\widetilde{z_1 z_2^2} = 38390/2^{15}$	$\widetilde{z_1^2 z_2} = 55938/2^{13}$	$z_1^2 z_2^2 = 8$
V0 = x4	V0 = x3	V0 = x1	X8 = x8<<3
V1 = V0+V0<<2	V1 = V0+V0<<4	V1 = V0+V0<<1	
V2 = V0<<4+V1	V2 = V0<<5+V1	V2 = V0<<9-V1	
V3 = V1<<5+V2	V3 = V1+V2<<10	V3 = V0<<5-V2	
V4 = V1+V3<<8	X3 = V3<<-14	V4 = V2-V3<<7	
X4 = V4<<-14		X1 = V4<<-13	

Table 3.18: SCM-based FRS algorithm for S(i), 16-bit, full AI

$S(i) = x_0 + z_1 \cdot x_5 + z_2 \cdot x_7 + z_1 z_2 \cdot x_4$			
c = x0<<2+x8<<3	V0 = x7	V1 = V0+V0<<2	t = t+X7
V0 = x5	V1 = V0+V0<<1	V2 = V0<<4+V1	S = t<<-2
V1 = V0<<0+V0<<1	V2 = V0<<5+V1	V3 = V1<<5+V2	
V2 = V1<<0-V1<<3	V3 = V0<<7-V2	V4 = V1+V3<<8	
V3 = V1<<0+V1<<6	V4 = V2<<10-V3<<2	X4 = V4<<-14	
V4 = V2<<11-V3	X7 = V4<<-15	t = c+X4	
X5 = V4<<-14	V0 = x4	t = t+X5	

$$
\begin{aligned}
S(j) \;=\; & s_1(j) + z_1 \cdot t_1(j) + z_2 \cdot t_2(j) + \\
& z_1 z_2 \cdot t_3(j) + z_1^2 \cdot s_6(j) + \\
& z_1^2 z_2 \cdot t_4(j) + z_2^2 \cdot s_{11}(j) + z_1 z_2^2 \cdot t_5(j) + \\
& z_1^2 z_2^2 \cdot s_{16}(j), \quad j \in \{1,3,5,7\}
\end{aligned}
$$

$$
\begin{aligned}
S(k) \;=\; & s_1(k) + z_1 \cdot s_5(k) + z_2 \cdot s_9(k) + z_1 z_2 \cdot t_6(k) + \\
& z_1^2 z_2 \cdot s_8(k) + z_1 z_2^2 \cdot s_{12}(k) + z_1^2 z_2^2 \cdot s_{16}(k), \quad k \in \{2,6\}
\end{aligned}
$$

The quantized values for the 8-bit MCM design for full AI, are identical to that of the SCM design. However, the algorithms shown in Tables 3.21 to 3.22, used to compute the MCM-based S-equations, have significantly less complexity than their SCM counterparts.

Table 3.19: SCM-based FRS algorithm for S(j), 16-bit, full AI

$S(j) = x_0 + z_1 \cdot x_5 + z_2 \cdot x_7 + z_1 z_2 \cdot x_4 + z_1^2 \cdot x_2 + z_1^2 z_2 \cdot x_1 + z_2^2 \cdot x_6 + z_1 z_2^2 \cdot x_3 + z_1^2 z_2^2 \cdot x_8$			
c = x0<<2+x8<<3	V0 = x4	V2 = V0<<9-V1	V3 = V1<<1-V2<<9
V0 = x5	V1 = V0+V0<<2	V3 = V0<<5-V2	X6 = V3<<-15
V1 = V0<<0+V0<<1	V2 = V0<<4+V1	V4 = V2-V3<<7	t = c+X1
V2 = V1<<0-V1<<3	V3 = V1<<5+V2	X1 = V4<<-13	t = t+X2
V3 = V1<<0+V1<<6	V4 = V1+V3<<8	V0 = x2	t = t+X3
V4 = V2<<11-V3	X4 = V4<<-14	V1 = V0+V0<<2	t = t+X4
X5 = V4<<-14	V0 = x3	V2 = V0+V1<<6	t = t+X5
V0 = x7	V1 = V0+V0<<4	V3 = V0<<5-V1	t = t+X6
V1 = V0+V0<<1	V2 = V0<<5+V1	V4 = V2<<1+V3<<11	t = t+X7
V2 = V0<<5+V1	V3 = V1+V2<<10	X2 = V4<<-13	S = t<<-4
V3 = V0<<7-V2	X3 = V3<<-14	V0 = x6	
V4 = V2<<10-V3<<2	V0 = x1	V1 = V0+V0<<2	
X7 = V4<<-15	V1 = V0+V0<<1	V2 = V1-V1<<4	

Table 3.20: SCM-based FRS algorithm for S(k), 16-bit, full AI

$S(k) = x_0 + z_1 \cdot x_5 + z_2 \cdot x_7 + z_1 z_2 \cdot x_4 + z_1^2 z_2 \cdot x_1 + z_1 z_2^2 \cdot x_3 + z_1^2 z_2^2 \cdot x_8$		
c = x0<<2+x8<<3	V0 = x4	V2 = V0<<9-V1
V0 = x5	V1 = V0+V0<<2	V3 = V0<<5-V2
V1 = V0<<0+V0<<1	V2 = V0<<4+V1	V4 = V2-V3<<7
V2 = V1<<0-V1<<3	V3 = V1<<5+V2	X1 = V4<<-13
V3 = V1<<0+V1<<6	V4 = V1+V3<<8	t = c+X1
V4 = V2<<11-V3	X4 = V4<<-14	t = t+X3
X5 = V4<<-14	V0 = x3	t = t+X4
V0 = x7	V1 = V0+V0<<4	t = t+X5
V1 = V0+V0<<1	V2 = V0<<5+V1	t = t+X7
V2 = V0<<5+V1	V3 = V1+V2<<10	S = t<<-4
V3 = V0<<7-V2	X3 = V3<<-14	
V4 = V2<<10-V3<<2	V0 = x1	
X7 = V4<<-15	V1 = V0+V0<<1	

This is because MCM designs are usually have fewer adders than their equivalent SCM counterparts, because of the opportunity to share intermediate computations which are unavailable in SCM. The MSE plot is identical that of Figure 3.14, for 8-bit SCM for full AI. This is as would be expected, because SCM and MCM designs of full AI have identical quantized coefficients for each quantization accuracy level, and thus have identical quantization error. The only difference between SCM and MCM implementations is the hardware complexity in computing the result, where MCM will typically be at least as good, if not significantly better, than SCM. The quantized values for the z's are:

$$\widetilde{z_1} = 167/2^6$$

$$\widetilde{z_2} = 139/2^7$$

$$\widetilde{z_1^2} = 219/2^5$$

$$\widetilde{z_2^2} = 150/2^7$$

$$\widetilde{z_1 z_2} = 181/2^6$$

$$\widetilde{z_1 z_2^2} = 196/2^6$$

$$\widetilde{z_1^2 z_2} = 237/2^5$$

$$z_1^2 z_2^2 = 8.$$

$$S(k) = x_0 + z_1 \cdot x_5 + z_2 \cdot x_7 + z_1 z_2 \cdot x_4 + z_1^2 z_2 \cdot x_1 + z_1 z_2^2 \cdot x_3 + z_1^2 z_2^2 \cdot x_8$$

Table 3.21: MCM-based FRS algorithm for S(i), 8-bit, full AI

$S(i) = x_0 + z_1 \cdot x_1 + z_2 \cdot x_2 + z_1 z_2 \cdot x_3$			
c=x0<<2	t4 = x1<<3-x1	t6 = t1<<2+t1	out=y1+c
t2 = x2+x3<<4	t1 = -t2+t3	t7 = t5+t2<<4	S = out<<-4
t3 = x3+x1<<5	t5 = t4+x2<<7	y1 = t6+t7	

Table 3.22: MCM-based FRS algorithm for S(j), 8-bit

$S(j) = x_0 + z_1 \cdot x_1 + z_2 \cdot x_2 + z_1 z_2 \cdot x_3 + z_1^2 \cdot x_6 + z_1^2 z_2 \cdot x_5 + z_2^2 \cdot x_7 + z_1 z_2^2 \cdot x_4 + z_1^2 z_2^2 \cdot x_8$			
c = x0<<2+x8<<3	t6 = x7<<2+x7	t10 = x2<<5-x5	t15 = t13+t3<<1
t2 = x3+x4	t7 = t5-t2<<2	t11 = t4-t3	t16 = t14-t1
t3 = x1+x6<<5	t8 = x5<<3+t6	t12 = t4<<2+t9	t17 = t15+t1<<2
t4 = x1<<5-x6	t1 = t7+t8<<1	t13 = t10+t2<<6	y1 = t16+t17<<2
t5 = -x3+x2	t9 = x5-x2<<2	t14 = t12+t11	out = y1+c
			S = out<<-4

3.5.2.2.4 MCM Designs - 16-bit The designs for the16-bit MCM version of the full AI is presented in this section. Again, as for the 8-bit MCM version, the MSE performance is identical to the 16-bit SCM counterpart, but with significantly less circuit complexity. The following S-equations and z-values were implemented:

$$S(i) = s_1(i) + z_1 \cdot s_5(i) + z_2 \cdot s_9(i) + z_1 \cdot z_2 \cdot s_{13}(i), \quad i \in \{0,4\}$$

$$\begin{aligned} S(j) \;=\; & s_1(j) + z_1 \cdot t_1(j) + z_2 \cdot t_2(j) + \\ & z_1 z_2 \cdot t_3(j) + z_1^2 \cdot s_6(j) + \\ & z_1^2 z_2 \cdot t_4(j) + z_2^2 \cdot s_{11}(j) + z_1 z_2^2 \cdot t_5(j) + \\ & z_1^2 z_2^2 \cdot s_{16}(j), \quad j \in \{1,3,5,7\} \end{aligned}$$

Table 3.23: MCM-based FRS algorithm for S(i), 16-bit, full AI

$S(i) = x_0 + z_1 \cdot x_2 + z_2 \cdot x_3 + z_1 z_2 \cdot x_4$			
c = x0<<2	t7 = x2+x1<<4	t8 = t7+t3<<4	y1 = t10+t11<<2
t3 = x2+x3<<1	t5 = -x3<<5+t3	t9 = -t2<<2+t2	out = y1+c
t4 = x3+x1<<11	t1 = t4+t5<<7	t10 = t9+t1	S = out<<-4
t6 = -x2<<2+x1	t2 = x1<<6+t6	t11 = t1+t8<<9	

Table 3.24: MCM-based FRS algorithm for S(j), 16-bit, full AI

$S(j) = x_0 + z_1 \cdot x_5 + z_2 \cdot x_7 + z_1 z_2 \cdot x_4 + z_1^2 \cdot x_2 + z_1^2 z_2 \cdot x_1 + z_2^2 \cdot x_6 + z_1 z_2^2 \cdot x_3 + z_1^2 z_2^2 \cdot x_8$			
c=x0<<2+x8<<3	t14 = x4+x2<<5	t1 = t7+t8<<9	t21 = t19+t18<<10
t4 = x3-x7<<1	t15 = x6+x3	t2 = -x4<<14+t11	t22 = t20+t1
t5 = x1<<4+x2	t6 = -x5<<10+t4	t16 = t13<<4+t12	t23 = t1<<2+t21
t9 = -x5+x1	t3 = t4<<1+x6	t17 = x5<<9+t5	y1 = t22+t23
t10 = -x2+x1<<4	t7 = t6+t3<<7	t18 = t15<<6+t14	out = y1+c
t12 = x6<<1-x4	t11 = t10<<2+t9	t19 = t16+t17<<7	S = out<<-4
t13 = x5<<3-x4	t8 = -t3<<2+t5	t20 = -t2<<2+t2	

$$\begin{aligned} S(k) &= s_1(k) + z_1 \cdot s_5(k) + z_2 \cdot s_9(k) + z_1 z_2 \cdot t_6(k) + \\ &\quad z_1^2 z_2 \cdot s_8(k) + z_1 z_2^2 \cdot s_{12}(k) + z_1^2 z_2^2 \cdot s_{16}(k), \quad k \in \{2,6\} \end{aligned}$$

$\widetilde{z_1} = 42813/2^{14}$

$\widetilde{z_2} = 35468/2^{5}$

$\widetilde{z_1^2} = 50159/2^{14}$

$\widetilde{z_2^2} = 60547/2^{13}$

$\widetilde{z_1 z_2} = 46341/2^{14}$

$\widetilde{z_1 z_2^2} = 38390/2^{15}$

$\widetilde{z_1^2 z_2} = 55938/2^{13}$

$z_1^2 z_2^2 = 8.$

The algorithms to compute the S-equations are shown in Tables 3.23 to 3.24

3.5.2.3 MCM Designs for Approach 0

The initial formulation of algorithms for approach 0 began with rewriting the basic FRS Equations 3.11 to 3.13, by substituting the 11 simplification Equations 3.28 to 3.38, as described in Section 3.4.2. This resulted in 18 sets of equations for Equations 3.11 to 3.13, thus greatly expanding the search space to find good design solutions. The equations of approach 0 are now functions of only 3 z-values, a large decrease from Equations 3.12 to 3.13, leading to designs with lower complexity and MSE. Each of the 18 designs of approach 0 had differing MSE performance and circuit complexity. The design with the lowest MSE, at each of the 8- to 16-bit coefficient quantization levels was selected. In this section, the best designs at 8-bits and 16-bits are presented.

3.5.2.3.1 Approach 0: 8-bit Coefficient Quantization The 8-bit coefficient quantization design for approach 0, which has the lowest average MSE is presented in this section.

The substituted equations for best designs of approach 0 are:

$$S(i) = s_1(i) + z_1 \cdot s_5(i) + z_2 \cdot s_9(i) + z_1 z_2 \cdot s_{13}(i), \quad i \in \{0,4\}$$

$$\begin{aligned} S(j) \;=\; & s_1(j) + 4 \cdot s_6(j) + 4 \cdot s_{11}(j) + z_1 z_2 \cdot (t_3(j) + s_6(j) - \\ & s_{11}(j)) + z_2 \cdot (t_2(j) + 2 \cdot t_4(j) - 2 \cdot t_5(j)) \\ & + z_1 \cdot (t_1(j) + 2 \cdot t_4(j) + 2 \cdot t_1(j)) \\ & + 8 \cdot s_{16}(j), \quad j \in \{1,3,5,7\} \end{aligned}$$

Table 3.25: MCM-based FRS algorithm for S, 8-bit, approach 0

$S = x_0 + x_1 \cdot \widetilde{z_1} + x_2 \cdot \widetilde{z_2} + x_3 \cdot \widetilde{z_1 z_2}$			
c = x0<<2	t2 = x2+x3<<4	t5 = t4+x2<<7	S = y1+c
x1 = x1<<-6	t3 = x3+x1<<5	t6 = t1<<2+t1	S = S<<-2
x2 = x2<<-7	t4 = x1<<3-x1	t7 = t5+t2<<4	
x3 = x3<<-6	t1 = -t2+t3	y1 = t6+t7	

$$\begin{aligned} S(k) &= s_1(k) + z_2 \cdot (2 \cdot s_8(k) + s_9(k) - 2 \cdot s_{12}(k)) + z_1 \cdot (s_5(k) + 2 \cdot s_8(k) + \\ &\quad 2 \cdot s_{12}(k)) + z_1 z_2 \cdot t_6(k) + 8 \cdot s_{16}(k), \quad k \in \{2,6\} \end{aligned}$$

$$\widetilde{z_1} = 167/2^6 \tag{3.57}$$

$$\widetilde{z_2} = 139/2^7 \tag{3.58}$$

$$\widetilde{z_1 z_2} = 181/2^6 \tag{3.59}$$

The basic prototype equation is shown below and MCM algorithm for computing it is shown in Table 3.25. In comparison with Section 3.5.2.2.3, it an be seen that the designs for approach 0 has significantly reduced computational requirements, due to having only 3 constant multipliers.

$$S = x_0 + x_1 \cdot \widetilde{z_1} + x_2 \cdot \widetilde{z_2} + x_3 \cdot \widetilde{z_1 z_2} \tag{3.60}$$

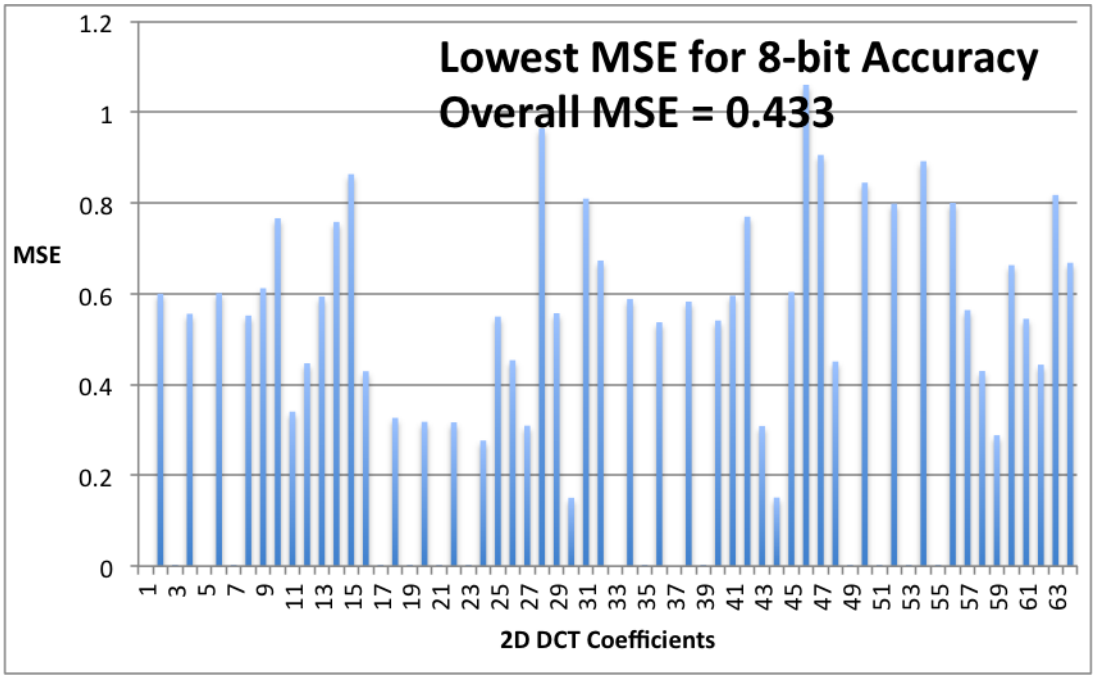

Figure 3.16: Lowest MSE for 8-bit, Approach 0 AI Implementation

In Figure 3.16, the plot of the MSE for approach 0 at 8-bits of accuracy is shown. At an overall average MSE of 0.433, it is superior to that of the full AI and non-AI approaches at 8-bits. This improvement is due to approach 0 having more possible equations to fulfill the requirements of low MSE, and because of fewer constant multiplications.

3.5.2.3.2 16-bit Coefficient Quantization The best design for approach 0 at 16-bits of coefficient quantization is presented in this section. The substituted equations are below.

$$S(i) = s_1(i) + 4 \cdot s_{13}(i) + s_5(i)/2 \cdot z_1 z_2^2 + (s_5(i) + s_9(i)) \cdot z_2 - s_{13}(i) \cdot z_2^2, \quad i \in \{0,4\}$$

$$\begin{aligned} S(j) \quad = \quad & s_1(j) + 8 \cdot s_6(j) + 8 \cdot s_{16}(j) + 4 \cdot t_3(j) + (t_1(j)/2 + t_4(j) + t_5(j)) \cdot z_1 z_2^2 + \\ & (t_1(j) + t_2(j) + 4 \cdot t_4(j)) \cdot z_2 + (-s_6(j) + s_{11}(j) - t_3(j)) \cdot z_2^2, \quad j \in \{1,3,5,7\} \end{aligned}$$

$$S(k) = s_1(k) + 8 \cdot s_{16}(k) + 4 \cdot t_6(k) + (s_5(k)/2 + s_8(k) + s_{12}(k)) \cdot z_1 z_2^2 + (s_5(k) + 4 \cdot s_8(k) + s_9(k)) \cdot z_2 - t_6(k) \cdot z_2^2, \quad k \in \{2,6\}$$

$$\widetilde{z_2} = 35468/2^{15} \tag{3.61}$$

$$\widetilde{z_2^2} = 38390/2^{15} \tag{3.62}$$

$$\widetilde{z_1 z_2^2} = 50159/2^{14} \tag{3.63}$$

The prototype equation is shown below.

$$S = x_0 + x_1 \cdot \widetilde{z_2} + x_2 \cdot \widetilde{z_2^2} + x_3 \cdot \widetilde{z_1 z_2^2} \tag{3.64}$$

Figure 3.17 shows the MSE performance of the 16-bit approach 0 design at 6.825e-7, an improvement by 2 orders of magnitude over the full AI and non-AI designs, at 16-bits of quantization accuracy. Again this is because of the same reasoning as for the previous, 8-bit case. Approach 0 has 18 possible equations with which to exceed MSE performance of the single equation for the full AI or the non-AI approaches, and it only has 3 constant multipliers.

3.5.2.3.3 Comparison of Lowest MSE Performance for 8- to 16-bit Coefficient Quantization in Approach 0 This section shows the MSE performance of the best approach 0 designs across 8- to 16-bit of accuracy. Figure 3.18 summarizes these results graphically. As quantization accuracy increases, the MSE decreases monotonically, due to more accurate quantization of the constant coefficients.

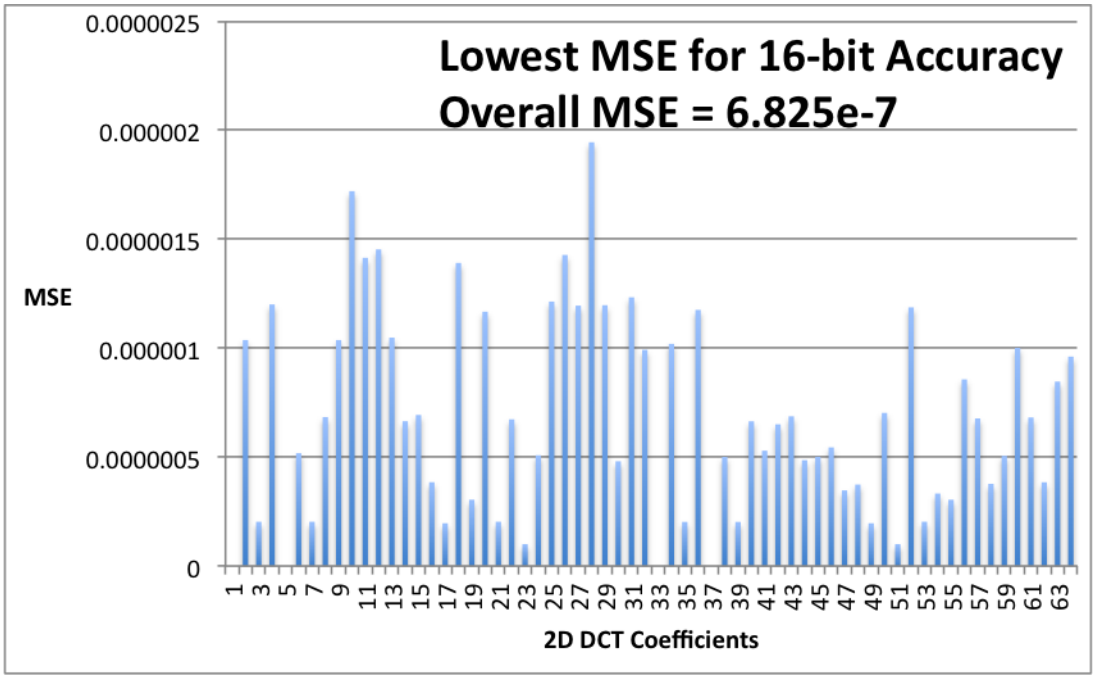

Figure 3.17: Lowest MSE for 16-bit, Approach 0 AI Implementation

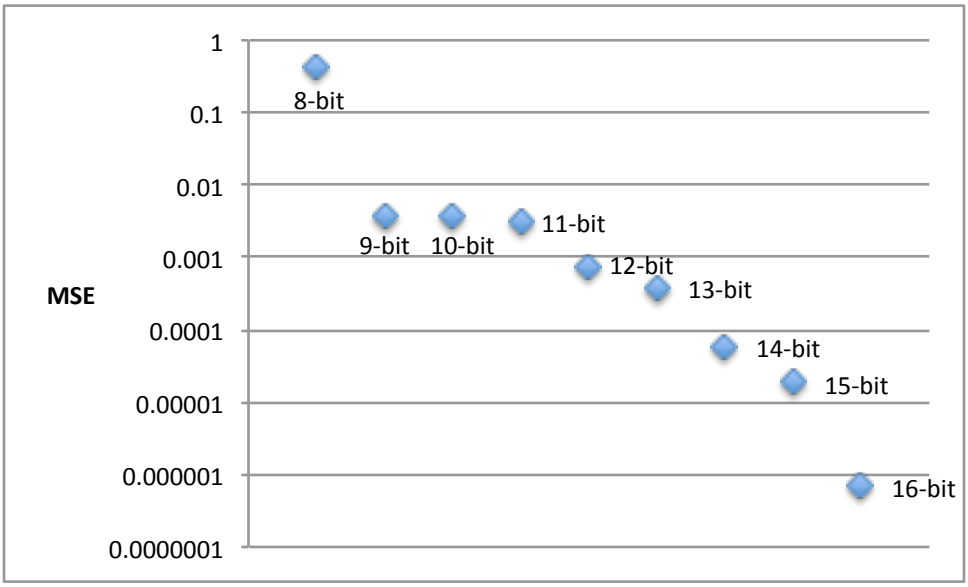

Figure 3.18: Summary of Lowest MSEs for Coefficient Quantizations of 8- to 16-bits

3.5.2.4 MCM Designs for Approach 2

The designs for approach 2 are an evolution of approach 0, as described in Section 3.4.2. The 18 approach 0 sets of equations are used a basis and one of the z-values is factored

out. Since there are 3 z-values in each of the sets of equations for approach 0, 54 approach 2 sets of equations are generated, leading to more degrees of design freedom to find good solutions within the design space.

In approach 2, two of the z-values can be converted to an MCM, and the remaining z-value must be an SCM by default. Approach 2 has relatively poor performance, compared to approaches 0 and 4. From examination of Equation 3.44, it can be deduced that the SCM penalizes this design approach by three factors. The first factor is that the SCM is unavailable to be converted to an MCM, thus losing the opportunity to be more efficiently encoded within an MCM, in terms of circuit complexity. The second factor is that the SCM operates on three computed terms, which will have experienced significant word growth, thus the SCM encoding must be implemented with relatively large adders. Lastly, the third factor is that the quantization error of the MCM terms is magnified by the quantization error of the SCM term.

3.5.2.4.1 8-bit Coefficient Quantization The lowest MSE design for approach 2, quantized with 8-bit coefficients is presented in this section. The S-equations show the z-values, and the SCM multiplication (e.g. $z_1 z_2^2$) described in the previous section.

$$S(i) = s_1(i) + 4 \cdot s_{13}(i) + z_1 z_2^2 \cdot (s_5(i)/2 + (s_5(i) + s_9(i)) \cdot 1/z_1 z_2 - s_{13}(i) \cdot 1/z_1), \quad i \in \{0,4\}$$

$$\begin{aligned} S(j) \;=\; & s_1(j) + 8 \cdot s_6(j) + 8 \cdot s_{16}(j) + 4 \cdot t_3(j) + z_1 z_2^2 \cdot ((t_1(j)/2 + t_4(j) + t_5(j)) + \\ & (t_1(j) + t_2(j) + 4 \cdot t_4(j)) \cdot 1/z_1 z_2 + (-s_6(j) + s_{11}(j) - t_3(j)) \cdot 1/z_1), \\ & j \in \{1,3,5,7\} \end{aligned}$$

$$\begin{aligned} S(k) &= s_1(k) + 8 \cdot s_{16}(k) + 4 \cdot t_6(k) + z_1 z_2^2 \cdot ((s_5(k)/2 + s_8(k) + s_{12}(k)) + \\ &\quad (s_5(k) + 4 \cdot s_8(k) + s_9(k)) \cdot 1/z_1 z_2 - t_6(k) \cdot 1/z_1), \quad k \in \{2,6\} \end{aligned}$$

The quantized z-values are presented below, with the SCM or MCM algorithm type used to implement them.

$$\widetilde{z_1 z_2^2} = 196/2^9 \ (SCM) \tag{3.65}$$

$$\widetilde{1/z_1 z_2} = 181/2^9 \ (MCM) \tag{3.66}$$

$$\widetilde{1/z_1} = 196/2^6 \ (MCM) \tag{3.67}$$

The prototype S-equation is given below.

$$S = x_0 + \widetilde{z_1 z_2^2} \cdot (x_1 + x_2 \cdot \widetilde{1/z_1 z_2} + x_3 \cdot \widetilde{1/z_1}) \tag{3.68}$$

The reported MSE for the best 8-bit approach 2 design is 0.0181, as shown in Figure 3.19. This MSE value is much better than that of approach 0, full AI, and non-AI. However, as will be shown later in the thesis, the hardware cost of this approach is high.

3.5.2.4.2 16-bit Coefficient Quantization The S-equations and constants for the best 16-bit design for approach 2 is shown below.

$$S(i) = s_1(i) + 4 \cdot s_{13}(i) + z_1 z_2^2 \cdot (s_5(i)/2 + (s_5(i) + s_9(i)) \cdot 1/z_1 z_2 - s_{13}(i) \cdot 1/z_1), \quad i \in \{0,4\}$$

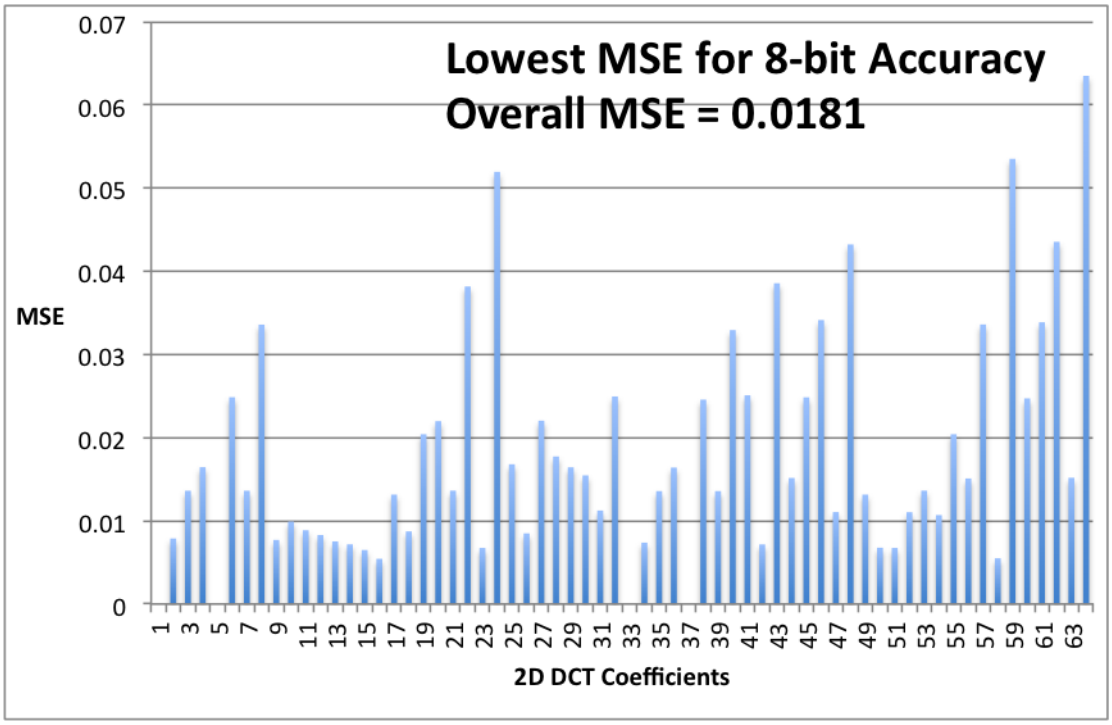

Figure 3.19: Lowest MSE for 8-bit, Approach 2 AI Implementation

$$\begin{aligned} S(j) &= s_1(j)+8\cdot s_6(j)+8\cdot s_{16}(j)+4\cdot t_3(j)+z_1z_2^2\cdot((t_1(j)/2+t_4(j)+t_5(j))+ \\ &\quad (t_1(j)+t_2(j)+4\cdot t_4(j))\cdot 1/z_1z_2+(-s_6(j)+s_{11}(j)-t_3(j))\cdot 1/z_1), \\ &\quad j\in\{1,3,5,7\} \end{aligned}$$

$$\begin{aligned} S(k) &= s_1(k)+8\cdot s_{16}(k)+4\cdot t_6(k)+z_1z_2^2\cdot((s_5(k)/2+s_8(k)+s_{12}(k))+ \\ &\quad (s_5(k)+4\cdot s_8(k)+s_9(k))\cdot 1/z_1z_2-t_6(k)\cdot 1/z_1), \quad k\in\{2,6\} \end{aligned}$$

$$\widetilde{z_1z_2^2}=50159/2^{14}\ (SCM) \tag{3.69}$$

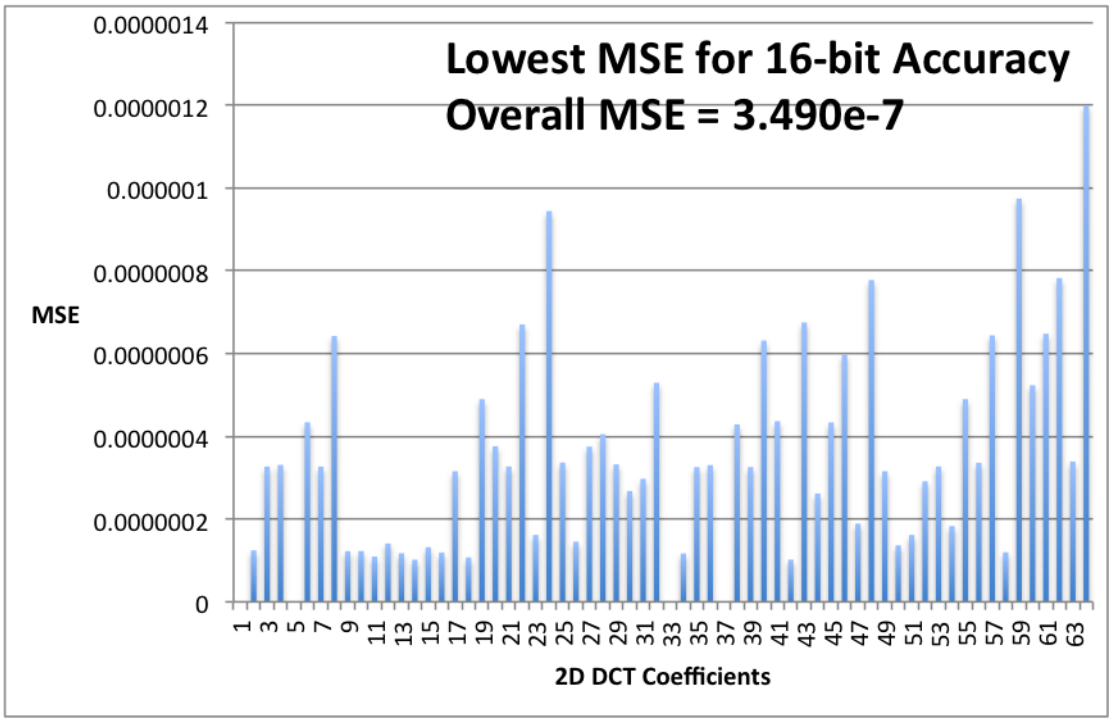

Figure 3.20: Lowest MSE for 16-bit, Approach 2 AI Implementation

$$\widetilde{1/z_1 z_2} = 46341/2^{17}\ (MCM) \tag{3.70}$$

$$\widetilde{1/z_1} = 50159/2^{17}\ (MCM) \tag{3.71}$$

The prototype S-equation is:

$$S = x_0 + \widetilde{z_1 z_2^2} \cdot (x_1 + x_2 \cdot \widetilde{1/z_1 z_2} + x_3 \cdot \widetilde{1/z_1}). \tag{3.72}$$

In Figure 3.20, the overall MSE value of 3.49e-7 is reported, which is the lowest MSE for any 16-bit coefficient quantization approach reported in this thesis.

3.5.2.5 Comparison of Lowest MSE Performance for 8- to 16-bit Coefficient Quantization for Approach 2

The summary of the MSE performance for approach 2 is provided in Figure 3.21 , for the lowest MSE at each quantization level. Again, it is seen that the MSE drops monotonically with increases in quantization accuracy.

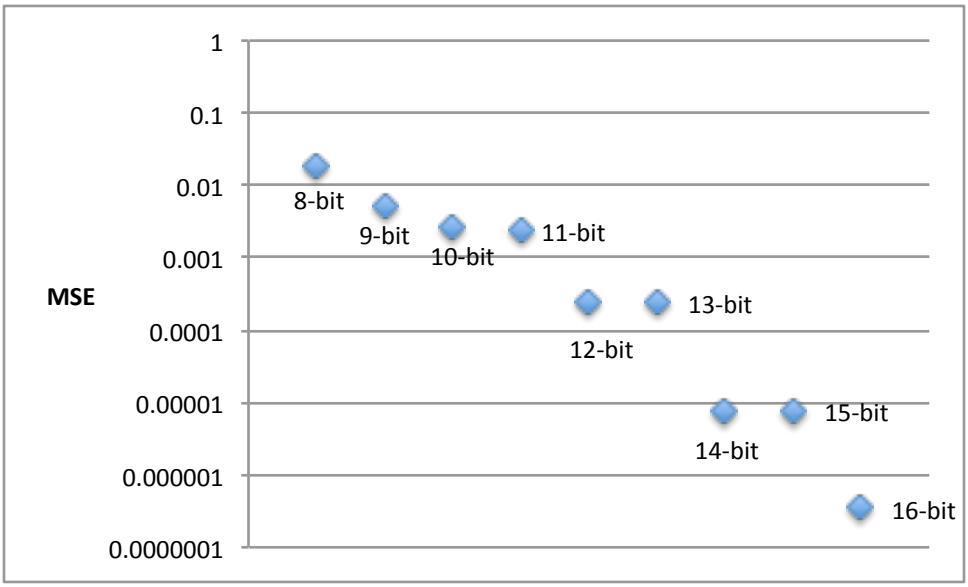

Figure 3.21: Summary of Lowest MSEs for Coefficient Quantizations of 8- to 16-bits

3.5.2.6 MCM Designs for Approach 4

The sample designs for approach 4 are reported in this section. Approach 4 is an extension of approach 2, with the difference being that the z-value being factored to the outside of the equation, as seen in Equation 3.47. This z-value, is not implemented at all, rather, it is considered to be "embedded" within the Arai constant, as discussed in Section 2.3, thus downstream processing unrelated to the AI component is considered to accommodate it. As with approach 2, there are 54 possible algorithms at each level of quantization accuracy. However, approach 4 has the benefit of 3 MCM terms, as does approach 0, with the drawback that some post-processing scaling is required to be available at the end of computation.

3.5.2.6.1 8-bit Coefficient Quantization The lowest MSE design for approach 4 at 8-bit quantization coefficient accuracy is presented, with S-equations and constants below.

$$S(i) = z_1 \cdot (s_1(i) \cdot 1/z_1 + (s_5(i) - s_9(i)) + z_1 z_2 \cdot s_9(i)/2 + z_2 \cdot s_{13}(i)), \quad i \in \{0,4\}$$

$$\begin{aligned} S(j) &= z_1 \cdot ((s_1(j) + 4 \cdot s_6(j) + 4 \cdot s_{11}(j) + 8 \cdot s_{16}(j)) \cdot 1/z_1 + (t_1(j) - t_2(j) + 4 \cdot t_5(j)) + \\ &\quad (t_2(j)/2 + t_4(j) - t_5(j)) \cdot z_1 z_2 + z_2 \cdot (t_3(j) + s_6(j) - s_{11}(j))), \quad j \in \{1,3,5,7\} \end{aligned}$$

$$\begin{aligned} S(k) &= z_1 \cdot ((s_1(k) + 8 \cdot s_{16}(k)) \cdot 1/z_1 + t_6(k) \cdot z_2 + z_1 z_2 \cdot (s_8(k) + \\ &\quad s_9(k)/2 - s_{12}(k)) + (s_5(k) - s_9(k) + 4 \cdot s_{12}(k))), \quad k \in \{2,6\} \end{aligned}$$

$$\widetilde{z_1 z_2} = 181/2^6 \tag{3.73}$$

$$\widetilde{z_2} = 139/2^7 \tag{3.74}$$

$$\widetilde{1/z_1} = 196/2^9 \tag{3.75}$$

The prototype S-equation is:

$$S = x_0 + x_1 \cdot \widetilde{z_1 z_2} + x_2 \cdot \widetilde{z_2} + x_3 \cdot \widetilde{1/z_1}. \tag{3.76}$$

Recall that the z_1 multiplier is not implemented, as it is considered to be embedded in

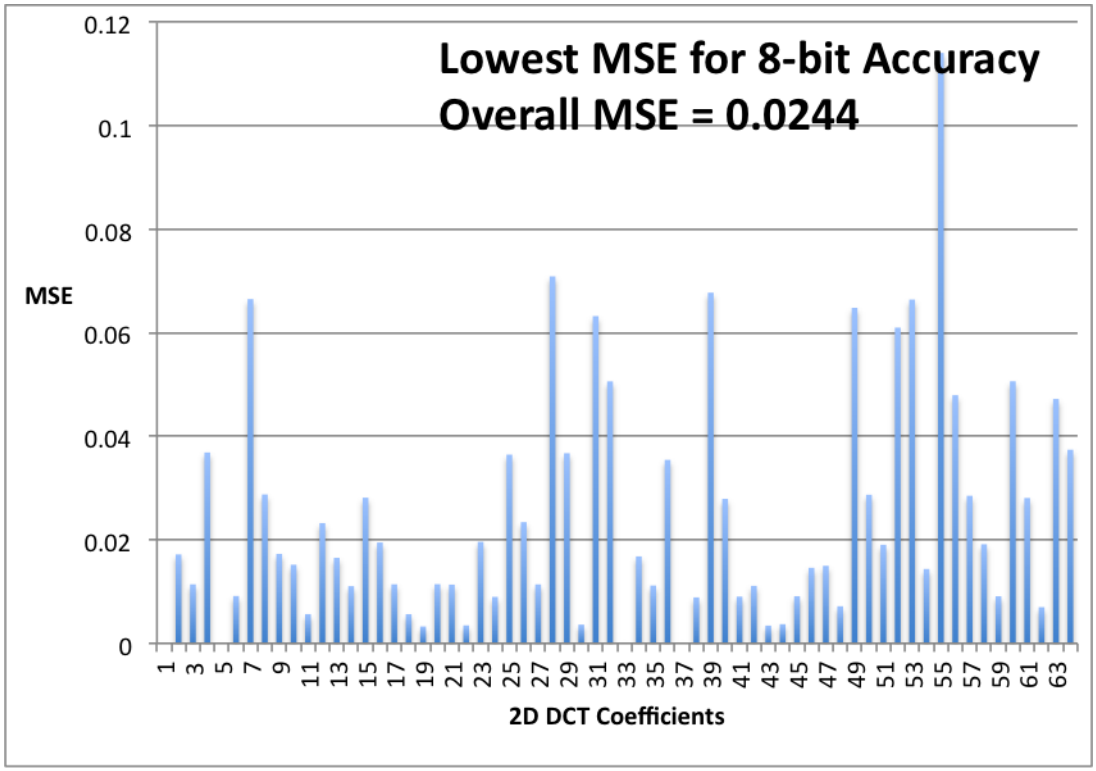

Figure 3.22: Lowest MSE for 8-bit, Approach 4 AI Implementation

the Arai constant.

Figure 3.22 shows that the reported overall average MSE is 0.0244.

3.5.2.6.2 16-bit Coefficient Quantization The S-equations and constants for the lowest MSE approach 4 design at 16-bits quantization accuracy is presented below.

$$S(i) = z_1^2 z_2 \cdot ((s_1(i) + 4 \cdot s_{13}(i)) \cdot 1/z_1^2 z_2 + (s_5(i)/4 + s_9(i)/4) + (s_5(i)/4 - s_9(i)/4) \cdot z_2/z_1 - s_{13}(i) \cdot z_2/z_1^2), \quad i \in \{0,4\}$$

$$S(j) = z_1^2 z_2 \cdot ((s_1(j) + 8 \cdot s_6(j) + 8 \cdot s_{16}(j) + 4 \cdot t_3(j)) \cdot 1/z_1^2 z_2 + (t_1(j)/4 + t_2(j)/4 + t_4(j)) + (t_1(j)/4 - t_2(j)/4 + t_5(j)) \cdot z_2/z_1 + (-s_6(j) + s_{11}(j) - t_3(j)) \cdot z_2/z_1^2), \quad j \in \{1,3,5,7\}$$

$$S(k) = z_1^2 z_2 \cdot ((s_1(k) + 8 \cdot s_{16}(k) + 4 \cdot t_6(k)) \cdot 1/z_1^2 z_2 + (s_5(k)/4 + s_8(k) + s_9(k)/4) + (s_5(k)/4 - s_9(k)/4 + s_{12}(k)) \cdot z_2/z_1 - t_6(k) \cdot z_2/z_1^2), \quad k \in \{2,6\}$$

$$\widetilde{1/z_1^2 z_2} = 35468/2^{18} \tag{3.77}$$

$$\widetilde{z_2/z_1^2} = 41553/2^{18} \tag{3.78}$$

$$\widetilde{z_2/z_1} = 54292/2^{17} \tag{3.79}$$

The prototype S-equation is:

$$S = x_0 + x_1 \cdot \widetilde{1/z_1^2 z_2} + x_2 \cdot \widetilde{z_2/z_1^2} + x_3 \cdot \widetilde{z_2/z_1}. \tag{3.80}$$

As shown in Figure 3.23, the reported MSE of approach 4 with 16-bit accuracy is 6.828e-7, a close second place to approach 2.

3.5.2.6.3 Comparison of Lowest MSE Performance for 8- to 16-bit Coefficient Quantization for Approach 4 Figure 3.24 shows the MSEs of the best approach 4 designs at each quantization accuracy, from 8- to 16-bits.

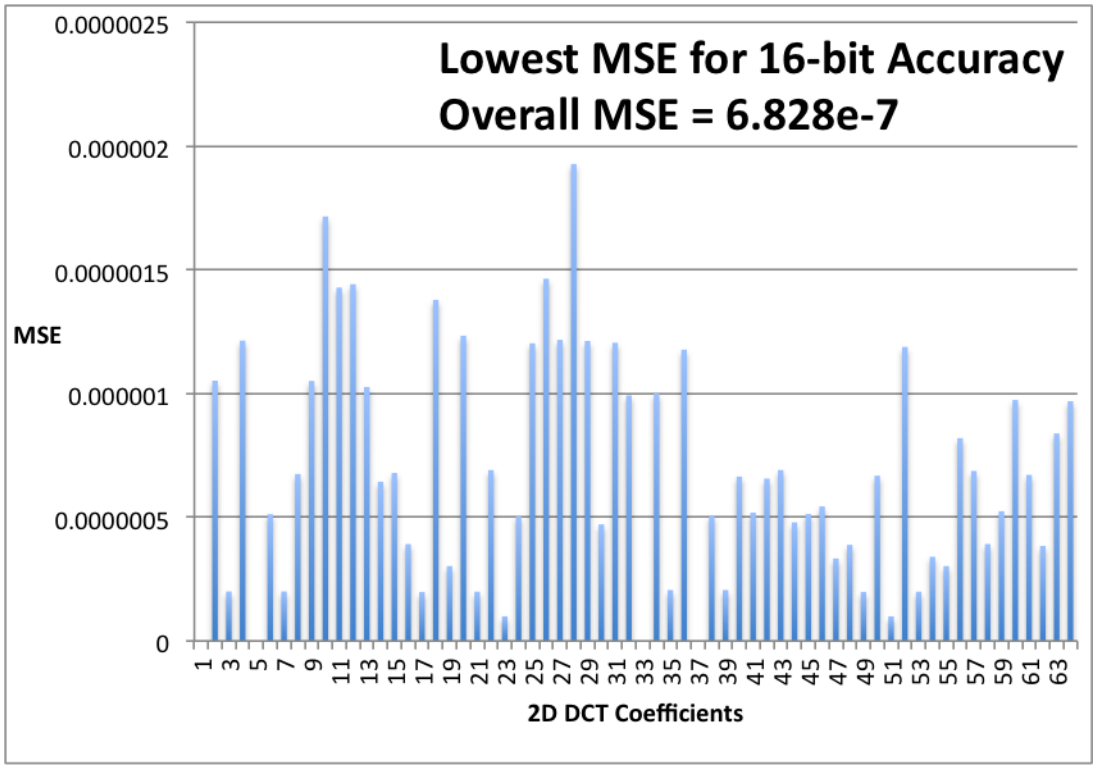

Figure 3.23: Lowest MSE for 8-bit, Approach 4 AI Implementation

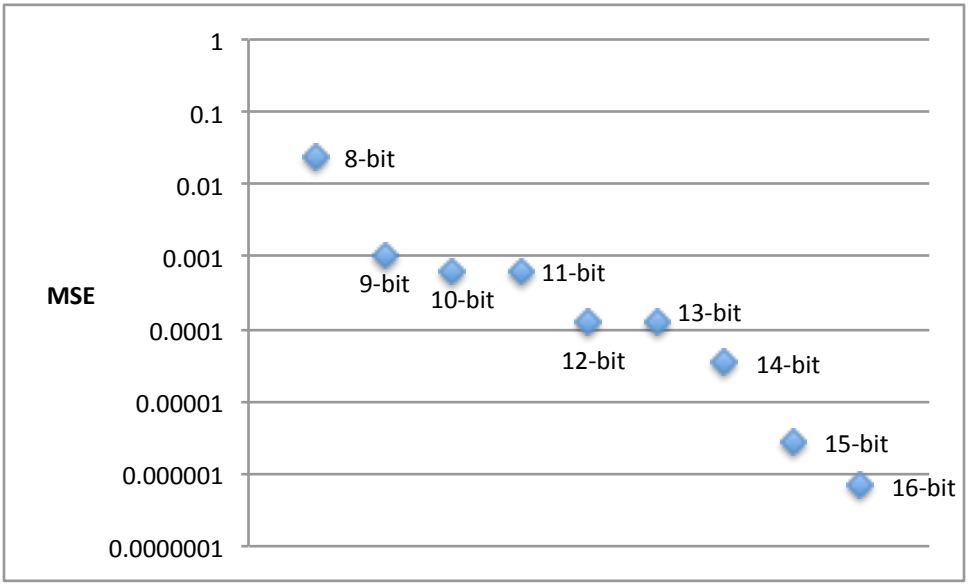

Figure 3.24: Summary of Lowest MSEs for Coefficient Quantizations of 8- to 16-bits

3.6 Chapter Summary

This chapter described the tools and models that were used to conduct this research. The development of five different algorithmic approaches to the full AI FRS was discussed and the justification for the selection of the three most promising approaches was provided. The algorithm and architecture for the 2D DCT was also presented in this chapter. This included an example of both an algorithmic, and hardware, 12-bit SCM implementation of the full AI algorithm, as part of Contribution 1. Selected samples of the best 8-, and 16-bit designs of the AI and non-AI design approaches were presented.

CHAPTER 4

RESULTS AND COMPARATIVE ANALYSIS OF AI DESIGNS

4.1 Introduction

This chapter details the summary and comparative analysis of the best AI and non-AI designs explored in this thesis. The best designs are defined as the designs with the lowest MSE, at each quantization bit level, for each of the design types in this thesis. The non-AI designs only have one design at each quantization bit level, from 8- to 16-bits. Similarly, the full AI designs for both the SCM and MCM designs each have one design at each quantization bit level, from 8- to 16-bits. The AI design approaches 0, 2, and 4, have 18, 54, and 54 designs at each quantization bit level, from 8- to 16-bits. There are 1152 basic AI designs, and 13 non-AI designs, evaluated in this thesis. These designs are further differentiated by being split into fully parallel and time-multiplexed versions, thus doubling the number of designs evaluated. The AI designs are further differentiated by having a full, or single, FRS unit, again doubling the number of designs. In total 4634 AI designs and 26 non-AI, are evaluated. The large number, and variety, of designs analyzed will help facilitate the drawing of general conclusions for merits of the application of AI to the 2D DCT.

4.2 Comparison of Design Approaches

Figure 4.1 shows a comparison of the 2D DCT designs, for AI and non-AI, using MSE versus the quantization bits accuracy for the best designs. Note that this comparison does

not include the circuit complexity directly because different designs approaches can have radically different hardware implications for circuit complexity, independent from quantization bits accuracy, as discussed in Section 3.5.2.4. Table 4.1 shows the specific values of Figure 4.1.

Since the AI designs for approaches 0, 2, and 4, have multiple options for design at each quantization level, error bars are added to their plots, to show the upper range of MSE, at each coefficient quantization accuracy level. These error bars show the large range of variation of the MSEs for each these designs. For example, approach 0 has 18 different designs at each quantization level, with different MSEs for each one. The lowest MSE at each quantization level is plotted in a line in Figure 4.1 and the error bar represents the highest MSE for that design at that quantization level. Since both approaches 2 and 4 have 54 designs each for each quantization level (3 times more than approach 0), their error bars' range significantly exceeds that of approach 0, because there are many more possibilities for a good (or bad) solution. Similarly, approaches 2 and 4 generally have the lowest MSE, compared to approach 0, because they have more possible design options for a good design.

Figure 4.1 shows that as the quantization accuracy increases, the algorithms' MSEs decrease monotonically. Since a lower MSE is desirable, it appears from the figure that either approach 2 or 4 is the best, and that non-AI is the worst. However, the circuit complexity costs required to achieve the respective MSEs are not represented in this figure. This problem is addressed in the figures in the following sections, to more accurately assess the suitability of the designs for their purpose.

4.2.1 Fully Parallel Design Approach

Figure 4.2 is the plot of the MSE versus the number of adders required, for the fully parallel implementations. It is desirable to have the least number of adders for given MSE, therefore, the best designs are towards the left of the figure. Since the AI designs are differentiated from each other by the FRS, a new line is introduced to represent the pre-FRS

Table 4.1: Ranges of MSEs for 2D DCT Designs vs Quantization Bits

Quantization Bits	MSE								
	Fixed-Point	Full AI SCM	Full AI MCM	AI Approach 0		AI Approach 2		AI Approach 4	
				Min Value	Max Value	Min Value	Max Value	Min Value	Max Value
8	6.66E-01	5.25E-01	5.25E-01	4.33E-01	5.17E+00	1.81E-02	1.34E+01	2.44E-02	5.16E+00
9	3.98E-01	1.56E-01	1.56E-01	3.61E-03	5.24E-01	5.04E-03	4.95E+00	1.08E-03	1.52E+00
10	5.80E-03	3.61E-03	3.61E-03	3.61E-03	3.61E-03	2.60E-03	1.47E-01	6.46E-04	2.36E-01
11	5.80E-03	3.46E-03	3.46E-03	3.17E-03	1.41E-02	2.36E-03	8.55E-02	6.46E-04	6.43E-02
12	2.91E-03	1.64E-03	1.64E-03	7.70E-04	3.61E-03	2.46E-04	1.07E-02	1.28E-04	1.27E-02
13	1.68E-03	9.54E-04	9.54E-04	3.90E-04	3.61E-03	2.46E-04	1.10E-02	1.28E-04	4.03E-03
14	3.78E-04	1.43E-04	1.43E-04	5.92E-05	8.12E-04	7.68E-06	2.95E-03	3.51E-05	1.26E-03
15	1.01E-04	5.81E-05	5.81E-05	1.85E-05	5.92E-05	7.68E-06	5.23E-04	2.82E-06	1.61E-04
16	1.67E-05	1.13E-05	1.13E-05	6.83E-07	6.37E-05	3.49E-07	2.70E-04	6.83E-07	2.11E-04

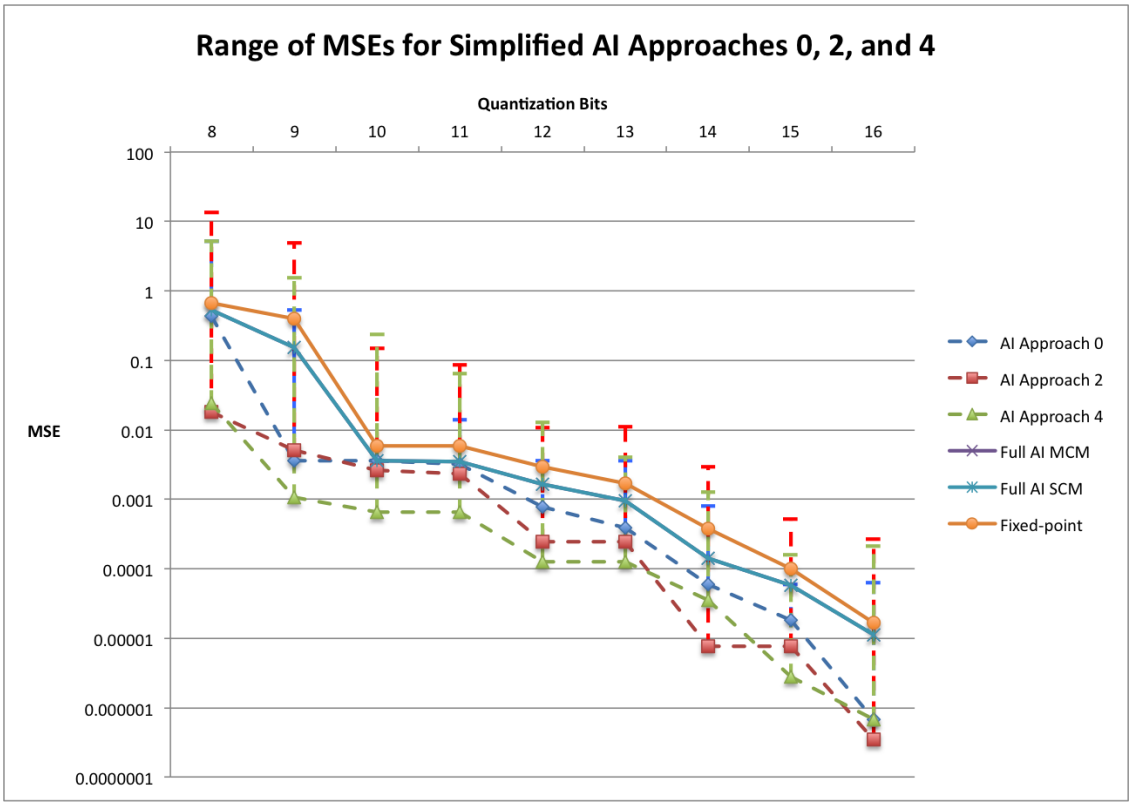

Figure 4.1: Ranges of MSEs for 2D DCT Designs vs Quantization Bits

Table 4.2: MSE vs Adders for Fully Parallel Implementation - Part 1

Quantization Bits	Fixed-Point		Full AI SCM		Full AI MCM	
	MSE	No. of Adders	MSE	No. of Adders	MSE	No. of Adders
8	6.66E-01	688	5.25E-01	1570	5.25E-01	1346
9	3.98E-01	688	1.56E-01	1570	1.56E-01	1330
10	5.80E-03	688	3.61E-03	1570	3.61E-03	1346
11	5.80E-03	688	3.46E-03	1602	3.46E-03	1378
12	2.91E-03	720	1.64E-03	1666	1.64E-03	1442
13	1.68E-03	736	9.54E-04	1678	9.54E-04	1502
14	3.78E-04	768	1.43E-04	1822	1.43E-04	1518
15	1.01E-04	768	5.81E-05	1822	5.81E-05	1550
16	1.67E-05	768	1.13E-05	1822	1.13E-05	1614

component of the AI designs, in blue and labeled “Pre-FRS AI”, with a value of 510 adders. This line represents the non-FRS part of the AI designs, which will be exactly the same for all the AI designs of the fully parallel implementations, regardless of coefficient quantization accuracy. This means that the designs are differentiated from each other by their FRS

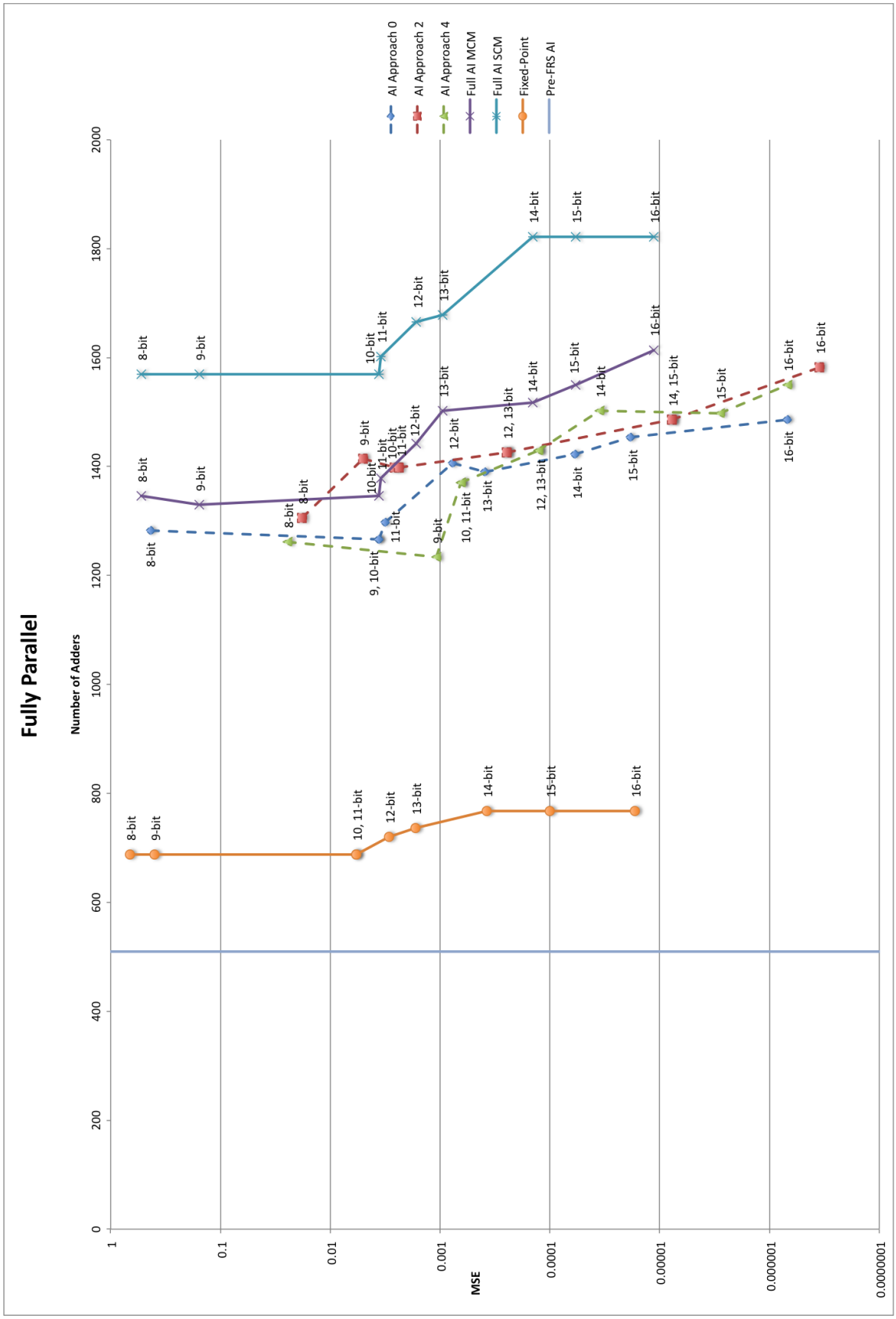

Figure 4.2: MSE vs Adders for Fully Parallel Implementation

Table 4.3: MSE vs Adders for Fully Parallel Implementation- Part 2

Quantization Bits	AI Approach 0		AI Approach 2		AI Approach 4	
	MSE	No. of Adders	MSE	No. of Adders	MSE	No. of Adders
8	4.33E-01	1282	1.81E-02	1306	2.44E-02	1262
9	3.61E-03	1266	5.04E-03	1414	1.08E-03	1234
10	3.61E-03	1266	2.60E-03	1398	6.46E-04	1370
11	3.17E-03	1298	2.36E-03	1398	6.46E-04	1370
12	7.70E-04	1406	2.46E-04	1426	1.28E-04	1430
13	3.90E-04	1390	2.46E-04	1426	1.28E-04	1430
14	5.92E-05	1422	7.68E-06	1486	3.51E-05	1502
15	1.85E-05	1454	7.68E-06	1486	2.82E-06	1498
16	6.83E-07	1486	3.49E-07	1582	6.83E-07	1550

only. The remaining lines in the figure show the hardware cost, in terms of the number of adders, for the fully parallel designs. The FRS-only hardware costs can be obtained by the distance between the Pre-FRS AI line to whichever AI algorithm line is of interest. Tables 4.2 and 4.3 show the specific values of Figure 4.2.

However, in contrast to Figure 4.1, Figure 4.2 shows that the non-AI designs are superior to the AI designs, when circuit cost is taken into account. In Figure 4.2, the best AI designs are approaches 0 and 4. The worst AI design is the SCM version of the full AI design, which is seen to be worse that its MCM version, due to less optimal encoding of its constant multipliers.

It was noted that in Figure 4.2, the distance from the pre-FRS line, to any of the AI lines, represents the circuit complexity cost of the FRS component of that particular AI design. The circuit complexity indicated by the pre-FRS line shows the limit of what is possible for the 2D AI approach investigated in this thesis. Since the non-AI line is relatively close to the pre-FRS line, a radically different AI approach must be taken in the future (if it is possible), to improve the AI approach so that it has a lower adder cost, compared to the non-AI approach, for the fully parallel design approach.

Consideration of the number of adders as the measure of circuit complexity has the drawback of ignoring the size of the adders, which is addressed in the next figure.

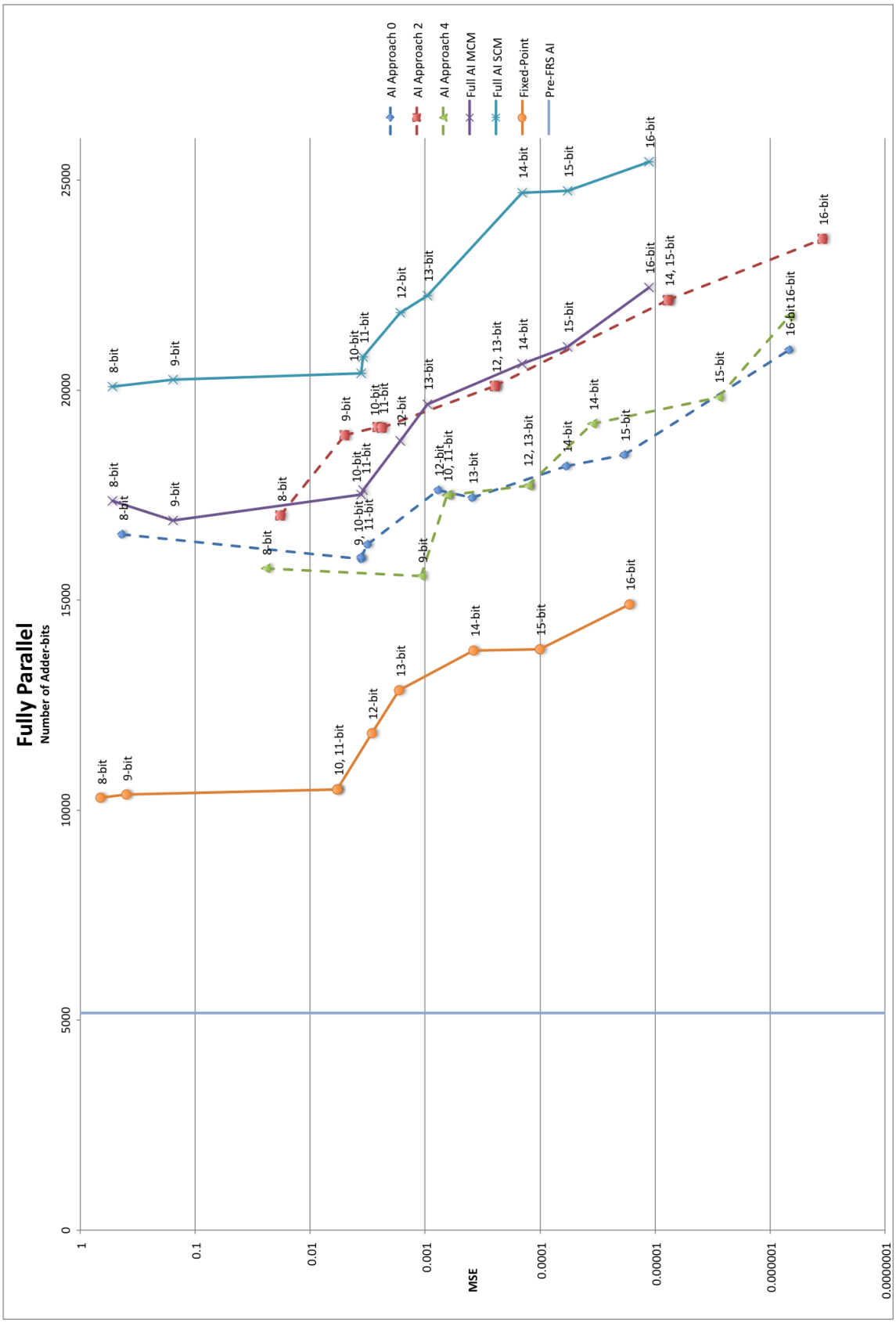

Figure 4.3: MSE vs Adder-bits for Fully Parallel Implementation

Table 4.4: MSE vs Adder-bits for Fully Parallel Implementation- Part 1

Quantization Bits	Fixed-Point		Full AI SCM		Full AI MCM	
	MSE	No. of Adder-bits	MSE	No. of Adder-bits	MSE	No. of Adder-bits
8	6.66E-01	10288.5	5.25E-01	20081.5	5.25E-01	17358
9	3.98E-01	10366.5	1.56E-01	20250	1.56E-01	16902.5
10	5.80E-03	10495.5	3.61E-03	20393	3.61E-03	17515
11	5.80E-03	10495.5	3.46E-03	20783	3.46E-03	17613.5
12	2.91E-03	11822.5	1.64E-03	21845	1.64E-03	18784.5
13	1.68E-03	12846.5	9.54E-04	22250.5	9.54E-04	19669
14	3.78E-04	13800.5	1.43E-04	24705.5	1.43E-04	20620.5
15	1.01E-04	13824.5	5.81E-05	24738.5	5.81E-05	21030
16	1.67E-05	14898.5	1.13E-05	25431	1.13E-05	22440

Table 4.5: MSE vs Adder-bits for Fully Parallel Implementation- Part 2

Quantization Bits	AI Approach 0		AI Approach 2		AI Approach 4	
	MSE	No. of Adder-bits	MSE	No. of Adder-bits	MSE	No. of Adder-bits
8	4.33E-01	16568.5	1.81E-02	17020.5	2.44E-02	15746
9	3.61E-03	15972	5.04E-03	18929	1.08E-03	15565.5
10	3.61E-03	16028	2.60E-03	19128.5	6.46E-04	17498
11	3.17E-03	16329	2.36E-03	19098.5	6.46E-04	17498
12	7.70E-04	17620	2.46E-04	20095	1.28E-04	17715.5
13	3.90E-04	17436.5	2.46E-04	20095	1.28E-04	17715.5
14	5.92E-05	18186	7.68E-06	22148	3.51E-05	19190.5
15	1.85E-05	18456.5	7.68E-06	22148	2.82E-06	19834
16	6.83E-07	20960.5	3.49E-07	23595.5	6.83E-07	21753.5

Figure 4.3 shows the MSE versus the number of adder-bits, for a fully parallel implementation. Tables 4.4 and 4.5 show the specific values of Figure 4.2. Immediately, we can see that the AI designs are much closer to the non-AI designs, as compared to the previous figure. This is due to the minimal word growth throughout much of the AI design, thus, the adders used in computation for AI are relatively much smaller than their non-AI counterparts. We can also observe that the line for AI approach 2 has shifted relatively rightwards (i.e. worsened), so that it is now comparable with the MCM version of full AI. This rightwards shift occurs due to the three factors discussed earlier in Section 3.5.2.4.

Namely: loss of an MCM term, which becomes inefficiently encoded as an SCM term; the SCM operating on a large input from the MCM operation, causing large word growth; and cascading of quantization errors through multiple quantized multipliers.

Table 4.6 shows an example application of using the results of Tables 4.2 and 4.3 and Figure 4.2, to obtain design guidance. In practice, a designer will seek a design that meets the minimum acceptable performance at the lowest cost. In this case, the performance will be determined by the MSE, and the cost by the number of adders in the design. If we make the assumption, for example, that the designer desires an MSE of lower than 0.001, then by using Table 4.6, the design with the lowest cost is the fixed-point design, with 768 adders, and therefore it should be chosen. By referring to Table 4.2, we see that the fixed-point designs that have MSE < 0.001, are the ones with 14-, 15-, or 16-bits of coefficient quantization accuracy, each with adder cost 768. Although any of the three designs can selected, the designer would most likely select the 16-bit fixed point design, because it offers the lowest MSE, at the same cost as the 14- or 15-bit fixed point designs. Again referring to Table 4.6, the lowest cost AI design that meets the MSE < 0.001 specification is AI Approach 4, at 10-bits of coefficient quantization accuracy (from Table 4.3), which has 1370 adders. However in this case, it is clear from Table 4.6 that the fixed-point design is superior to the AI design, with a percentage difference of 78.4% in the number of adders. The bolded entries of Tables 4.2 and 4.3 are the lowest cost AI designs, at each MSE level.

If we consider the same example of a designer requiring a design that has an MSE < 0.001, but using adder-bits as the cost measure, we can use Table 4.7 for design guidance. We see that again, the fixed-point design has the fewest adder-bits, at 13800.5, for 14-bits of coefficient quantization accuracy (using Table 4.4). This design has a percentage difference of only 26.3% compared to the best AI design, which is Approach 0 at 13-bits of coefficient quantization accuracy, with 17436.5 adder-bits. Since the AI designs have smaller word-growth than the fixed-point designs, the adder-bits cost measure is more favorable to the AI designs, as well as more representative for VLSI designs, rather than simply counting the

Table 4.6: Minimum number of adders for MSE thresholds in the fully parallel implementation

MSE Threshold	Minimum number of adders that meet the MSE threshold						% Difference of Best AI with Non AI
	Fixed-Point	Full AI SCM	Full AI MCM	AI Approach 0	AI Approach 2	AI Approach 4	
< 1	688	1570	1330	1266	1306	**1234**	79.4
< 0.1	688	1570	1346	1266	1306	**1234**	79.4
< 0.01	688	1570	1346	1266	1398	**1234**	79.4
< 0.001	768	1678	1502	1390	1426	**1370**	78.4
< 0.0001	768	1822	1550	**1422**	1486	1498	85.2
< 0.00001	n/a	n/a	n/a	**1486**	**1486**	1498	n/a
< 0.000001	n/a	n/a	n/a	**1486**	1582	1550	n/a

Table 4.7: Minimum number of adder-bits for MSE thresholds in the fully parallel implementation

MSE Threshold	Minimum number of adder-bits that meet the MSE threshold						% Difference of Best AI with Non AI
	Fixed-Point	Full AI SCM	Full AI MCM	AI Approach 0	AI Approach 2	AI Approach 4	
< 1	10288.5	20081.5	16902.5	15972	17020.5	**15565.5**	51.3
< 0.1	10495.5	20393	17515	15972	17020.5	**15565.5**	48.3
< 0.01	10495.5	20393	17515	15972	18929	**15565.5**	48.3
< 0.001	13800.5	22250.5	19669	**17436.5**	20095	17498	26.3
< 0.0001	14898.5	24738.5	21030	**18186**	22148	19190.5	22.1
< 0.00001	n/a	n/a	n/a	20960.5	22148	**19834**	n/a
< 0.000001	n/a	n/a	n/a	**20960.5**	23595.5	21753.5	n/a

Table 4.8: MSE vs Adders for Time-Multiplexed Implementation- Part 1

Quantization Bits	Fixed-Point		Full AI SCM		Full AI MCM	
	MSE	No. of Adders	MSE	No. of Adders	MSE	No. of Adders
8	6.66E-01	86	5.25E-01	329	5.25E-01	273
9	3.98E-01	86	1.56E-01	329	1.56E-01	265
10	5.80E-03	86	3.61E-03	329	3.61E-03	265
11	5.80E-03	86	3.46E-03	337	3.46E-03	273
12	2.91E-03	90	1.64E-03	353	1.64E-03	289
13	1.68E-03	92	9.54E-04	353	9.54E-04	297
14	3.78E-04	96	1.43E-04	385	1.43E-04	297
15	1.01E-04	96	5.81E-05	385	5.81E-05	313
16	1.67E-05	96	1.13E-05	385	1.13E-05	321

Table 4.9: MSE vs Adders for Time-Multiplexed Implementation- Part 2

Quantization Bits	AI Approach 0		AI Approach 2		AI Approach 4	
	MSE	No. of Adders	MSE	No. of Adders	MSE	No. of Adders
8	4.33E-01	257	1.81E-02	249	2.44E-02	249
9	3.61E-03	249	5.04E-03	265	1.08E-03	249
10	3.61E-03	249	2.60E-03	265	6.46E-04	265
11	3.17E-03	249	2.36E-03	265	6.46E-04	265
12	7.70E-04	265	2.46E-04	265	1.28E-04	273
13	3.90E-04	265	2.46E-04	265	1.28E-04	273
14	5.92E-05	273	7.68E-06	289	3.51E-05	289
15	1.85E-05	273	7.68E-06	289	2.82E-06	281
16	6.83E-07	281	3.49E-07	289	6.83E-07	281

number of adders.

Although Tables 4.6 and 4.7 were used as design aids, Figures 4.2 and 4.3, and Tables 4.2, 4.3, 4.4, and 4.5, can be used directly by a designer to find the best design tradeoffs, between MSE and hardware cost.

4.2.2 Time-Multiplexed Design Approach

Figure 4.4 shows the performance of the time-multiplexed versions of the 2D DCT designs, in terms of MSE and number of adders. Tables 4.8 and 4.9 show the specific values of Figure 4.4. The time-multiplexed designs trade off physical size for data throughput, so they

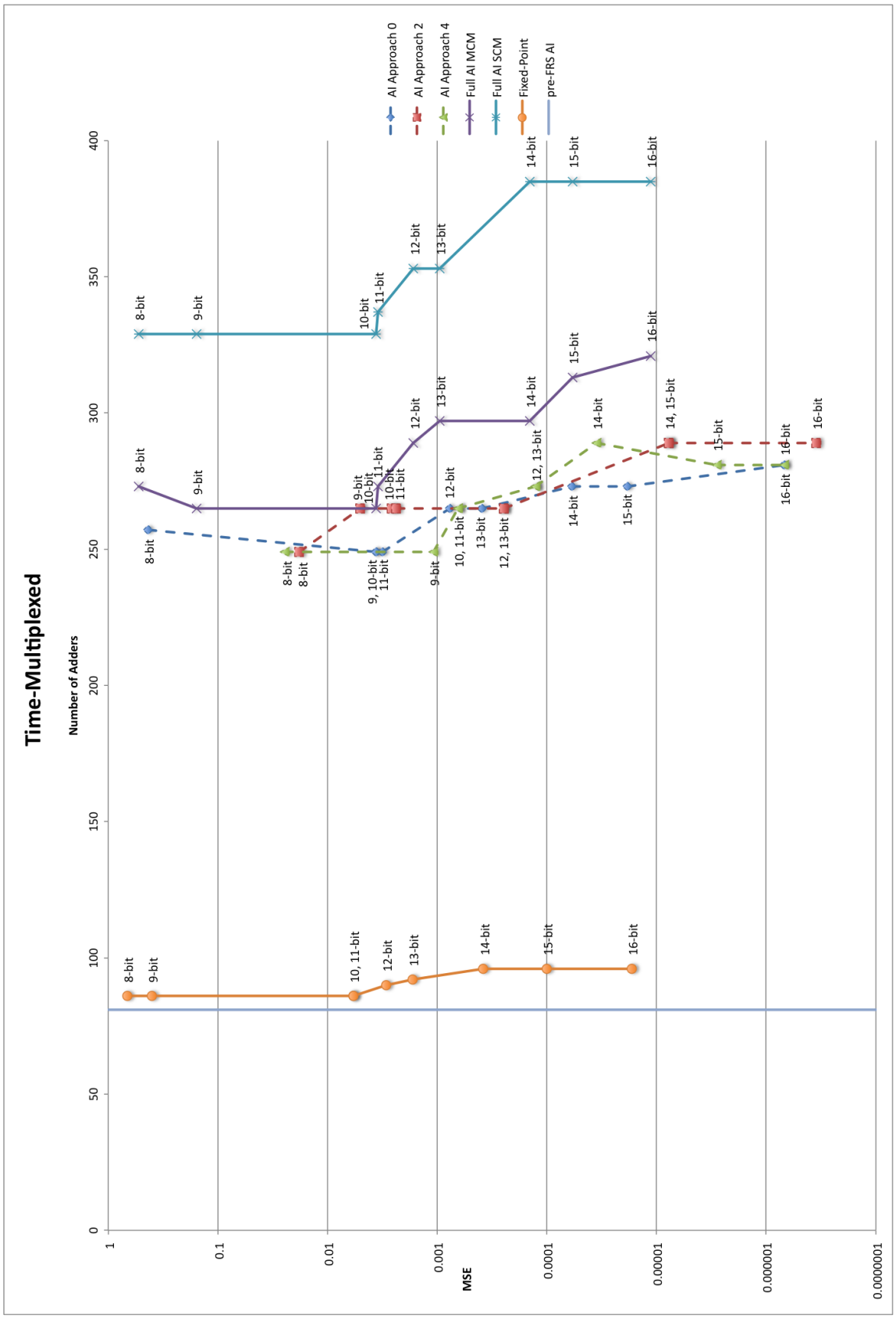

Figure 4.4: MSE vs Adders for Time-Multiplexed Implementation

Table 4.10: MSE vs Adder-bits for Time-Multiplexed Implementation- Part 1

Quantization Bits	Fixed-Point		Full AI SCM		Full AI MCM	
	MSE	No. of Adder-bits	MSE	No. of Adder-bits	MSE	No. of Adder-bits
8	6.66E-01	1387	5.25E-01	4651.5	5.25E-01	3847.5
9	3.98E-01	1419	1.56E-01	4695.5	1.56E-01	3651.5
10	5.80E-03	1435.5	3.61E-03	4715.5	3.61E-03	3751.5
11	5.80E-03	1435.5	3.46E-03	4823.5	3.46E-03	3767.5
12	2.91E-03	1696	1.64E-03	5111.5	1.64E-03	4115.5
13	1.68E-03	1772	9.54E-04	5179.5	9.54E-04	4243.5
14	3.78E-04	1911	1.43E-04	5759.5	1.43E-04	4391.5
15	1.01E-04	1914	5.81E-05	5771.5	5.81E-05	4607.5
16	1.67E-05	2090.5	1.13E-05	5935.5	1.13E-05	4791.5

have approximately 1/8 the size and 1/8 the throughput. The AI designs fare more poorly than they did for the fully parallel version, due to the need for the FRS to be specified for the worst-case size of input that could be multiplexed to them. In the fully parallel case, each individual FRS (i.e. one of the 64 individual FRSs for the 2D DCT) was exactly "tuned" to the specific input size that it was receiving for the pre-FRS component. This "tuning" refers to selecting one of Equation (3.11) to (3.13), which all have differing circuit complexities. In the time-multiplexed version, this opportunity for fine-tuning is not available so the worst-case FRS must be chosen. However, the design of Pradini [4,5] may be a good candidate to use to address this problem, because it is a time-multiplexed design that has been remapped to eliminate transposition memory. Pradini's design has constant computation channels and thus an AI implementation will have a predictable FRS output at each S-output of the 1D DCT, to fine-tune the hardware complexity to exactly what is needed.

Figure 4.5 shows the time-multiplexed designs in terms of MSE versus the number of adder-bits. Tables 4.10 and 4.11 show the specific values of Figure 4.5. As with the fully parallel version, the analysis using adder-bits benefits AI designs, as it more accurately reflects the actual cost of the circuit complexity.

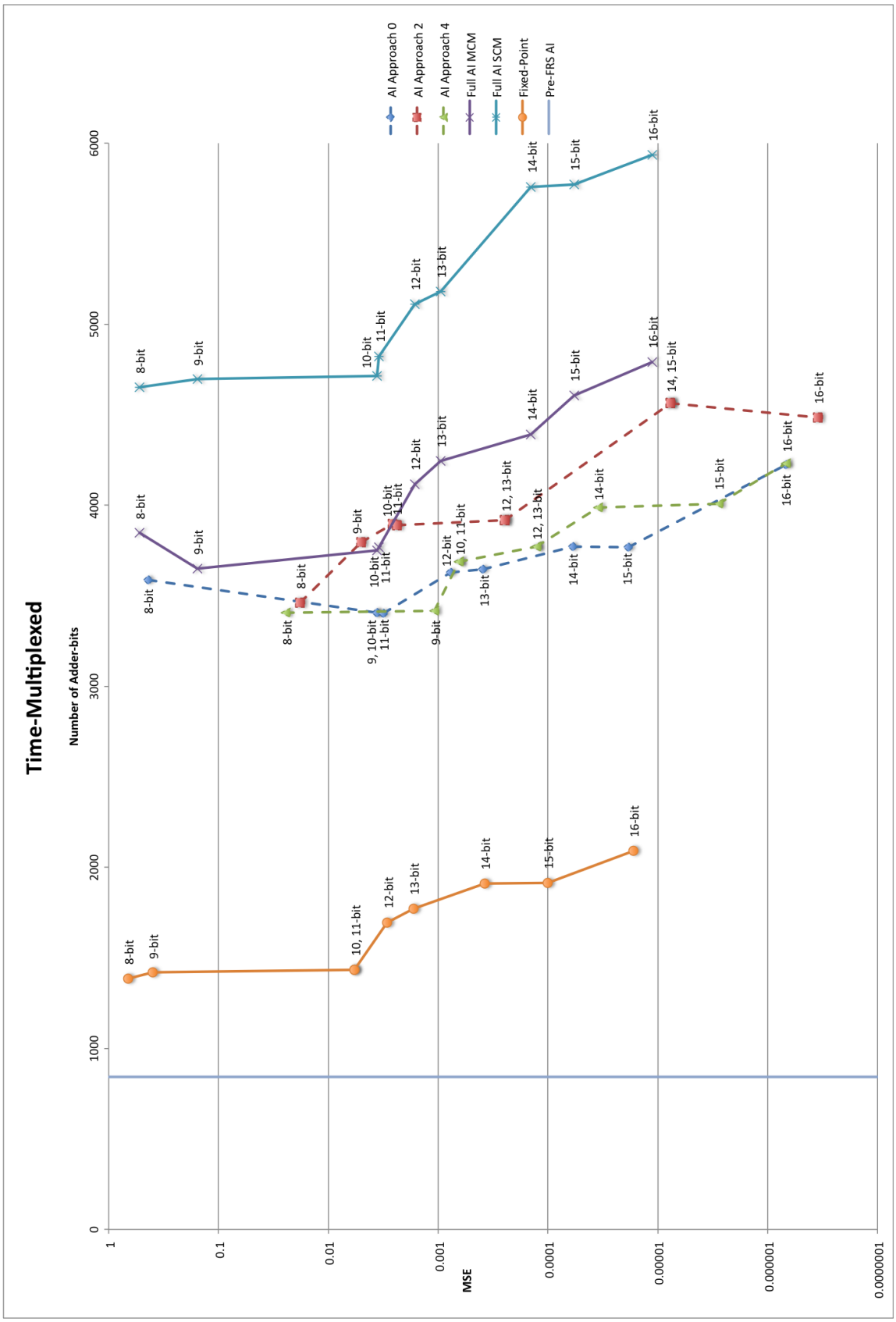

Figure 4.5: MSE vs Adder-bits for Time-Multiplexed Implementation

Table 4.11: MSE vs Adder-bits for Time-Multiplexed Implementation- Part 2

Quantization Bits	AI Approach 0		AI Approach 2		AI Approach 4	
	MSE	No. of Adder-bits	MSE	No. of Adder-bits	MSE	No. of Adder-bits
8	4.33E-01	3587.5	1.81E-02	3463.5	2.44E-02	3407.5
9	3.61E-03	3407.5	5.04E-03	3795.5	1.08E-03	3415.5
10	3.61E-03	3407.5	2.60E-03	3895.5	6.46E-04	3687.5
11	3.17E-03	3403.5	2.36E-03	3891.5	6.46E-04	3687.5
12	7.70E-04	3627.5	2.46E-04	3919.5	1.28E-04	3771.5
13	3.90E-04	3647.5	2.46E-04	3919.5	1.28E-04	3771.5
14	5.92E-05	3771.5	7.68E-06	4563.5	3.51E-05	3987.5
15	1.85E-05	3767.5	7.68E-06	4563.5	2.82E-06	4007.5
16	6.83E-07	4223.5	3.49E-07	4483.5	6.83E-07	4231.5

Table 4.12: MSE vs Adders for Fully Parallel Implementation with Single FRS- Part 1

Quantization Bits	Fixed-Point		Full AI SCM		Full AI MCM	
	MSE	No. of Adders	MSE	No. of Adders	MSE	No. of Adders
8	6.66E-01	688	5.25E-01	541	5.25E-01	534
9	3.98E-01	688	1.56E-01	541	1.56E-01	533
10	5.80E-03	688	3.61E-03	541	3.61E-03	533
11	5.80E-03	688	3.46E-03	542	3.46E-03	534
12	2.91E-03	720	1.64E-03	544	1.64E-03	536
13	1.68E-03	736	9.54E-04	544	9.54E-04	537
14	3.78E-04	768	1.43E-04	548	1.43E-04	537
15	1.01E-04	768	5.81E-05	548	5.81E-05	539
16	1.67E-05	768	1.13E-05	548	1.13E-05	540

4.2.3 Fully Parallel Design Approach, with Single FRS

Figure 4.6 shows the performance of a design approach to AI where a single FRS output is multiplexed to all the 2D AI DCT outputs. Tables 4.12 and 4.13 show the specific values of Figure 4.6. The 2D DCT has 64 output coefficients (i.e. 8x8), of which 60 are the result of a multiplication operation at some point in their computation (through an m-value). In an AI implementation, these coefficients require reconstruction with the FRS. Four of the output coefficients (1, 5, 33, and 37) are computed without any multiplications, and therefore do not need an FRS. In summary, an AI design with a single multiplexed FRS output will have

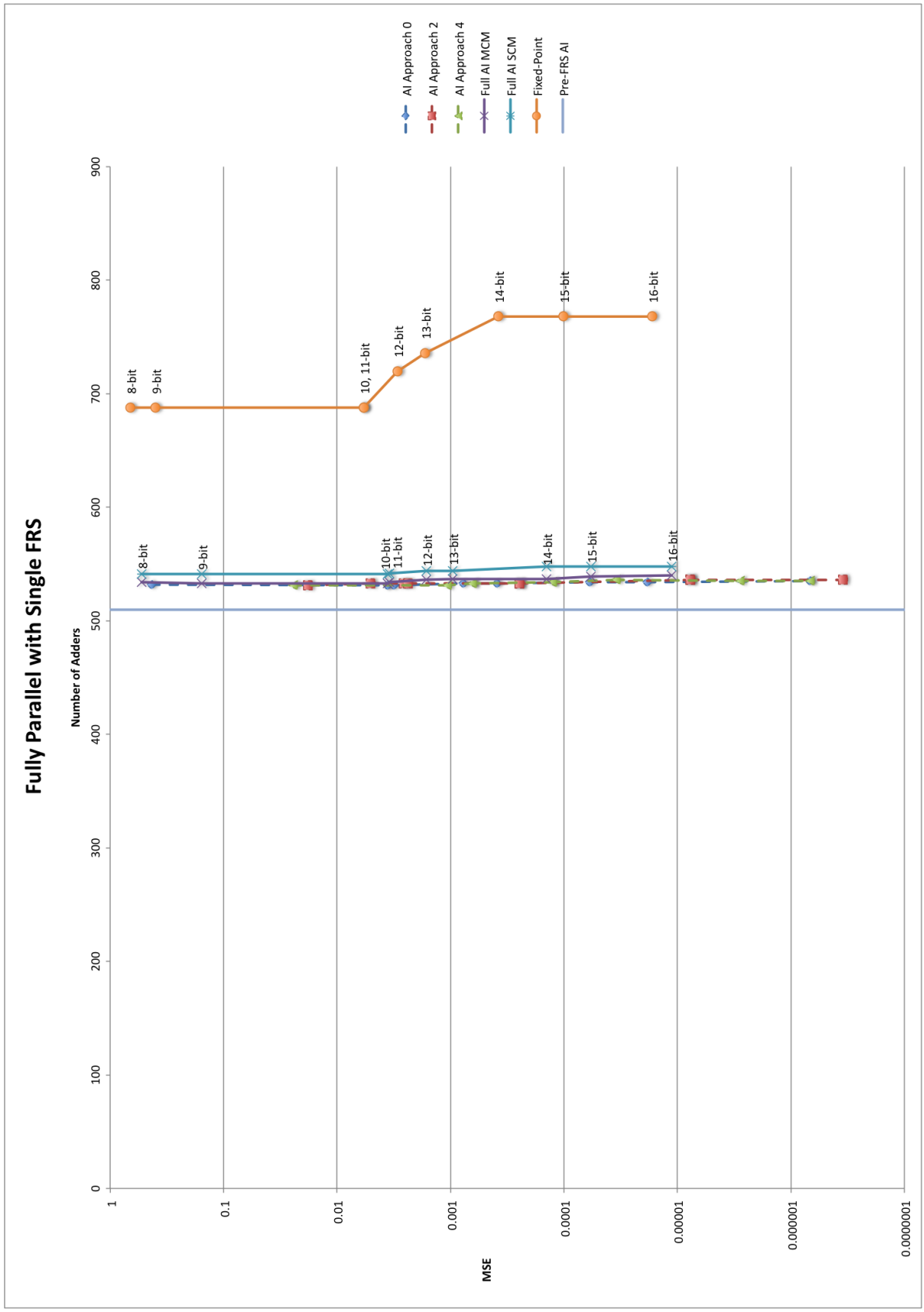

Figure 4.6: MSE vs Adders for Fully Parallel Implementation with Single FRS

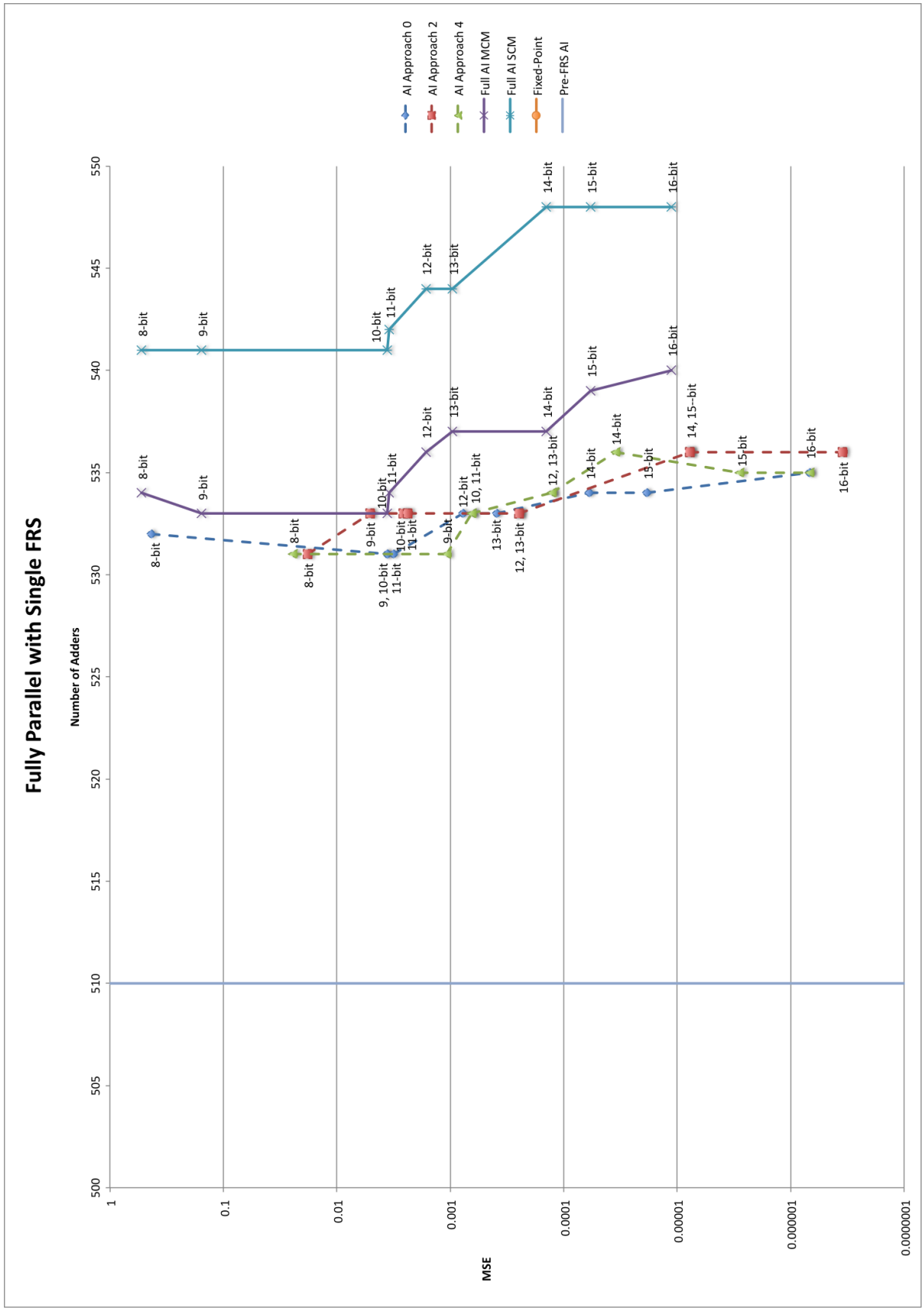

Figure 4.7: MSE vs Adders for Fully Parallel Implementation with Single FRS (expanded)

Table 4.13: MSE vs Adders for Fully Parallel Implementation with Single FRS- Part 2

Quantization Bits	AI Approach 0		AI Approach 2		AI Approach 4	
	MSE	No. of Adders	MSE	No. of Adders	MSE	No. of Adders
8	4.33E-01	532	1.81E-02	531	2.44E-02	531
9	3.61E-03	531	5.04E-03	533	1.08E-03	531
10	3.61E-03	531	2.60E-03	533	6.46E-04	533
11	3.17E-03	531	2.36E-03	533	6.46E-04	533
12	7.70E-04	533	2.46E-04	533	1.28E-04	534
13	3.90E-04	533	2.46E-04	533	1.28E-04	534
14	5.92E-05	534	7.68E-06	536	3.51E-05	536
15	1.85E-05	534	7.68E-06	536	2.82E-06	535
16	6.83E-07	535	3.49E-07	536	6.83E-07	535

approximately 1/60 the throughput of the same AI design with a complete FRS component for every output. In terms of the physical circuit size, excluding the multiplexor, Figure 4.6 shows that the AI design approach can result in a significantly smaller physical design that non-AI, making it a suitable candidate for area-limited designs. Since the size of the FRS component (in terms of adders) is the distance from the pre-FRS line to the AI line, there is significant opportunity to employ multiple, individual FRS outputs, to proportionally increase the throughput and still have less physical circuitry than non-AI.

It is observed in Figure 4.6 that the differences between the different AI designs is relatively insignificant. This means that the design procedure could be considerably simplified by using the easiest AI FRS to design, namely using the full AI SCM designs.

Figure 4.8 shows MSE versus the number of adder-bits, for a fully parallel implementation with a single, multiplexed FRS. Tables 4.14 and 4.15 show the specific values of Figure 4.8. As was previously observed, analysis with adder-bits improves the relative performance of the AI algorithms, compared to non-AI.

4.2.4 Time-Multiplexed Design Approach, with Single FRS

Figures 4.10 and 4.11 repeat the analysis for the single, multiplexed FRS, but for the time-multiplexed design approach. Tables 4.16 and 4.17 show the specific values of Figure 4.10.

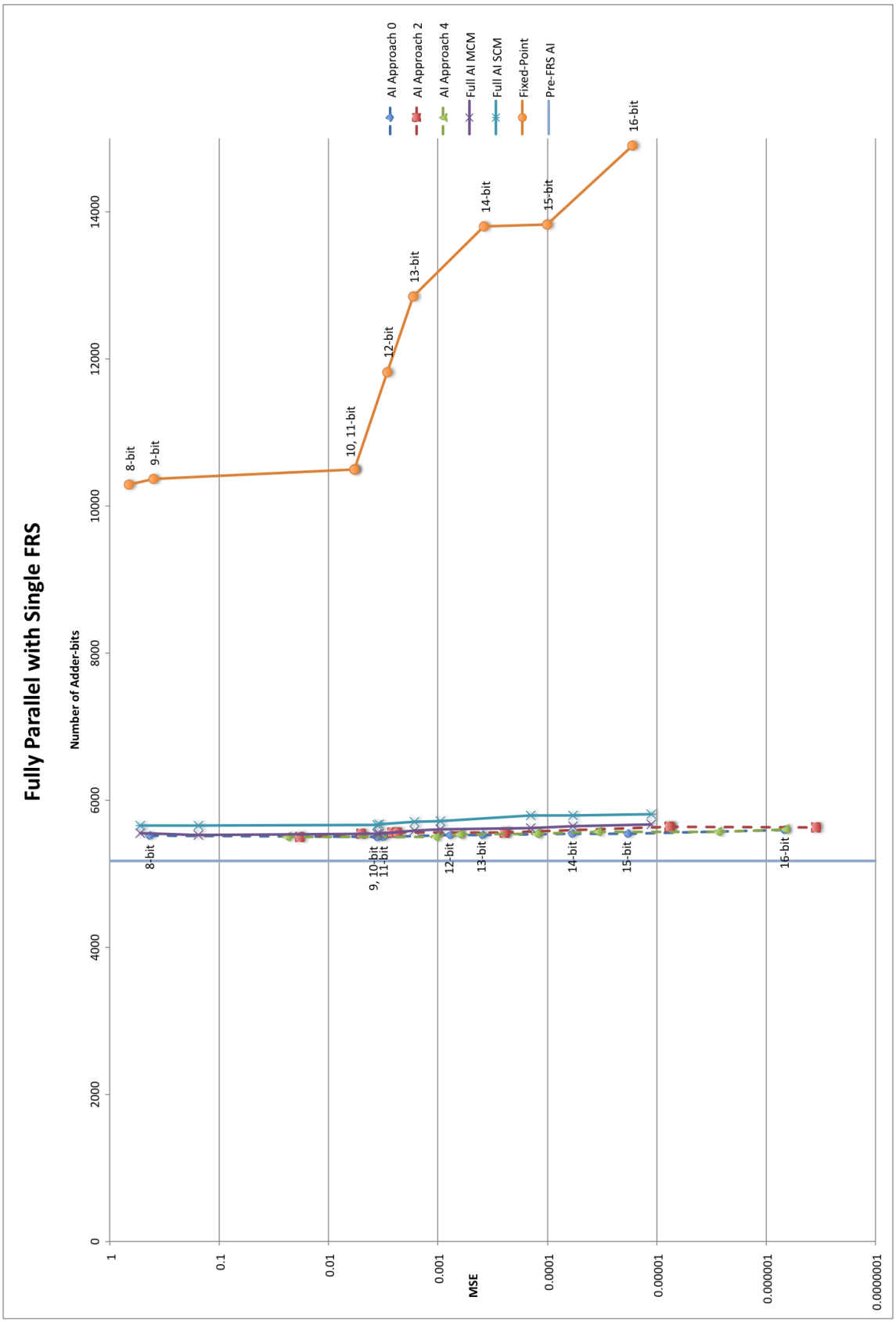

Figure 4.8: MSE vs Adder-bits for Fully Parallel Implementation with Single FRS

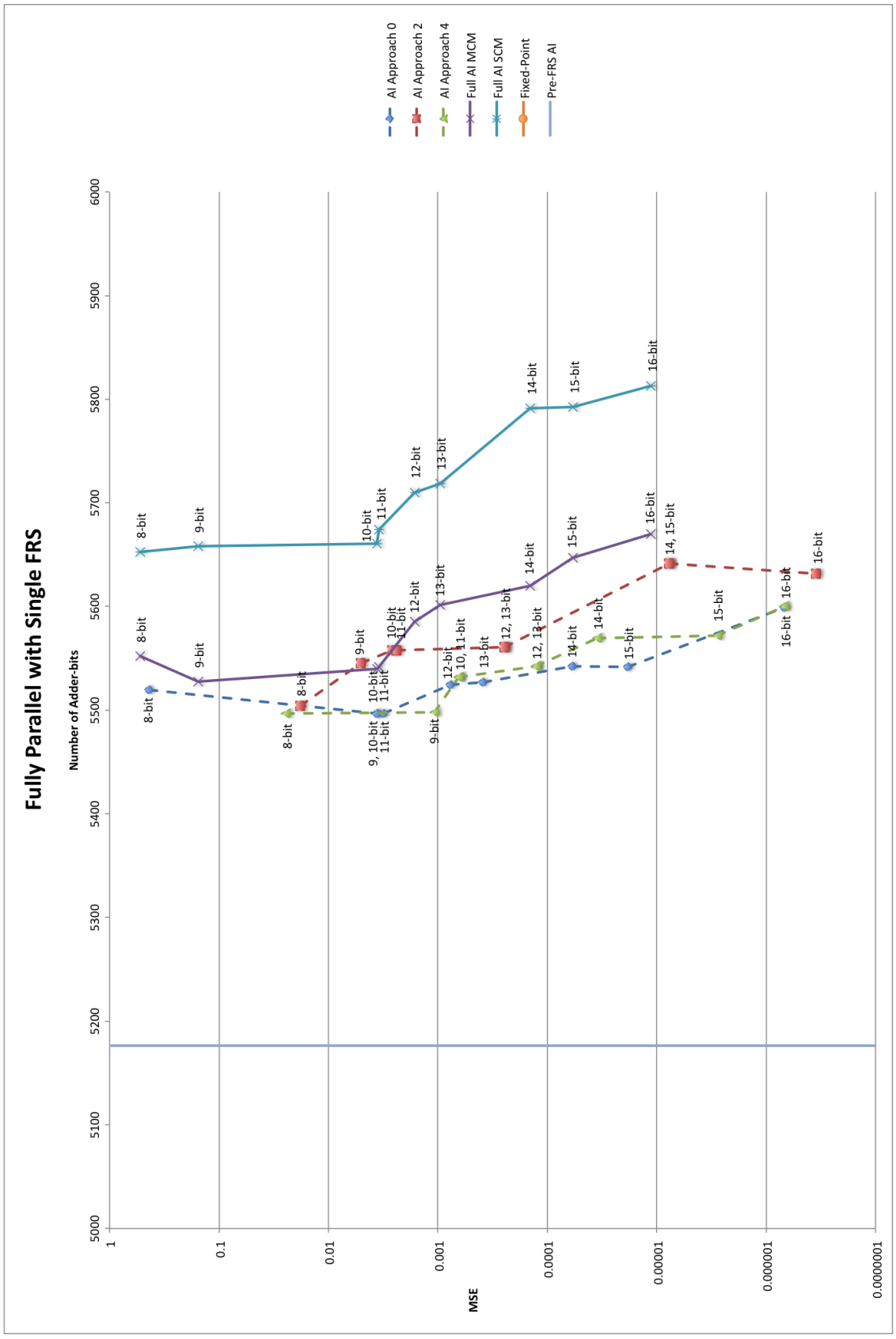

Figure 4.9: MSE vs Adder-bits for Fully Parallel Implementation with Single FRS (zoomed into AI designs)

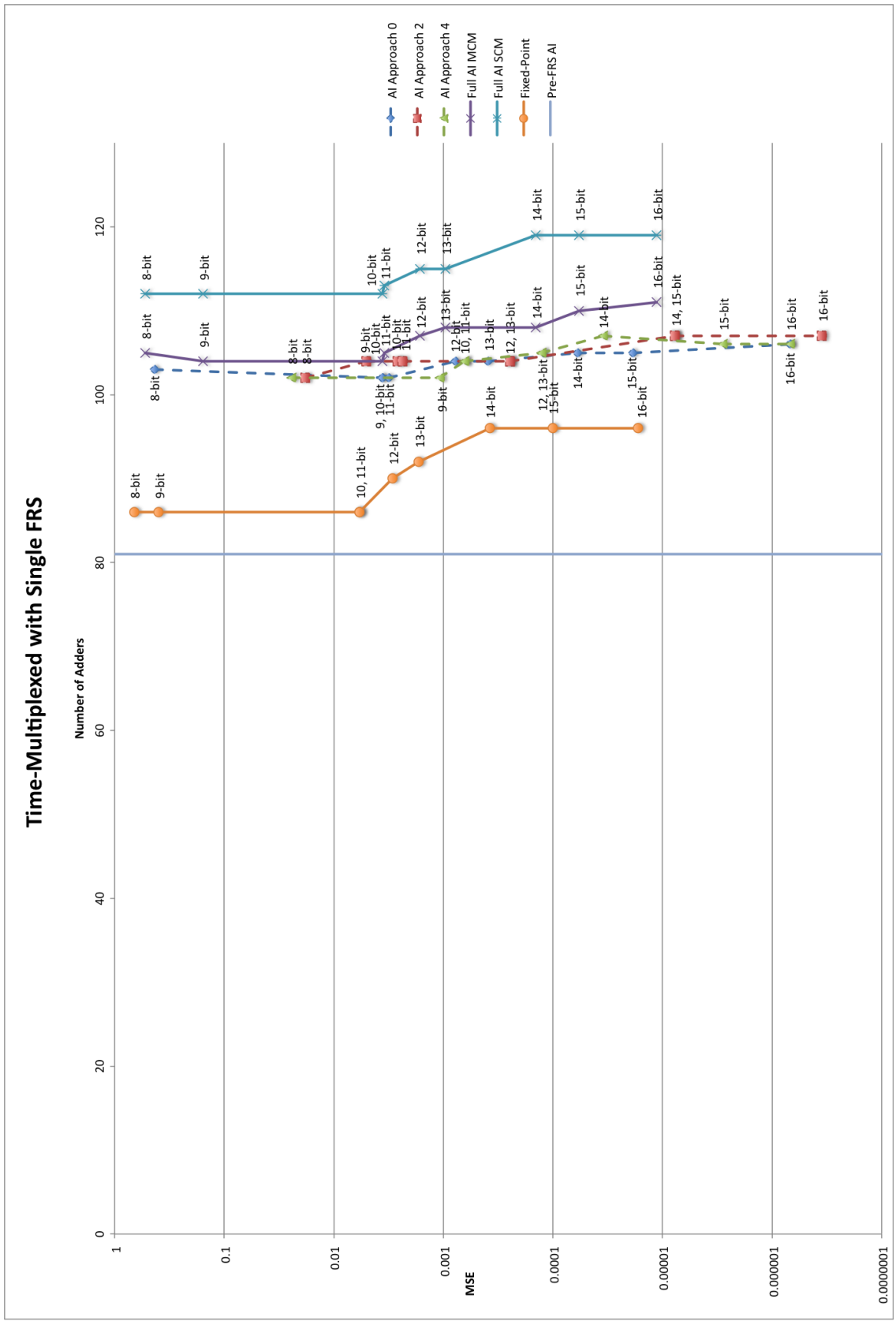

Figure 4.10: MSE vs Adders for Time-Multiplexed Implementation with Single FRS

Table 4.14: MSE vs Adder-bits for Fully Parallel Implementation with Single FRS- Part 1

Quantization Bits	Fixed-Point		Full AI SCM		Full AI MCM	
	MSE	No. of Adder-bits	MSE	No. of Adder-bits	MSE	No. of Adder-bits
8	6.66E-01	10288.5	5.25E-01	5652.5	5.25E-01	5552
9	3.98E-01	10366.5	1.56E-01	5658	1.56E-01	5527.5
10	5.80E-03	10495.5	3.61E-03	5660.5	3.61E-03	5540
11	5.80E-03	10495.5	3.46E-03	5674	3.46E-03	5542
12	2.91E-03	11822.5	1.64E-03	5710	1.64E-03	5585.5
13	1.68E-03	12846.5	9.54E-04	5718.5	9.54E-04	5601.5
14	3.78E-04	13800.5	1.43E-04	5791	1.43E-04	5620
15	1.01E-04	13824.5	5.81E-05	5792.5	5.81E-05	5647
16	1.67E-05	14898.5	1.13E-05	5813	1.13E-05	5670

Table 4.15: MSE vs Adder-bits for Fully Parallel Implementation with Single FRS- Part 2

Quantization Bits	AI Approach 0		AI Approach 2		AI Approach 4	
	MSE	No. of Adder-bits	MSE	No. of Adder-bits	MSE	No. of Adder-bits
8	4.33E-01	5519.5	1.81E-02	5504	2.44E-02	5497
9	3.61E-03	5497	5.04E-03	5545.5	1.08E-03	5498
10	3.61E-03	5497	2.60E-03	5558	6.46E-04	5532
11	3.17E-03	5496.5	2.36E-03	5557.5	6.46E-04	5532
12	7.70E-04	5524.5	2.46E-04	5561	1.28E-04	5542.5
13	3.90E-04	5527	2.46E-04	5561	1.28E-04	5542.5
14	5.92E-05	5542.5	7.68E-06	5641.5	3.51E-05	5569.5
15	1.85E-05	5542	7.68E-06	5641.5	2.82E-06	5572
16	6.83E-07	5599	3.49E-07	5631.5	6.83E-07	5600

As opposed to a throughput of approximately 1/60 for the fully parallel case, the throughput of the time-multiplexed case will have approximately 1/8 the throughput, as compared to similar non-AI designs. Reviewing the number of adders in Figure 4.10, it appears that non-AI has lower physical circuit complexity, despite ignoring the cost of multiplexing the single FRS. But review of the adder-bits in Figure 4.11 shows that the AI approaches have significantly lower circuit complexity, compared to non-AI (still ignoring the costs of multiplexing). Again the drawbacks of the AI approach are reduced throughput and much higher design effort.

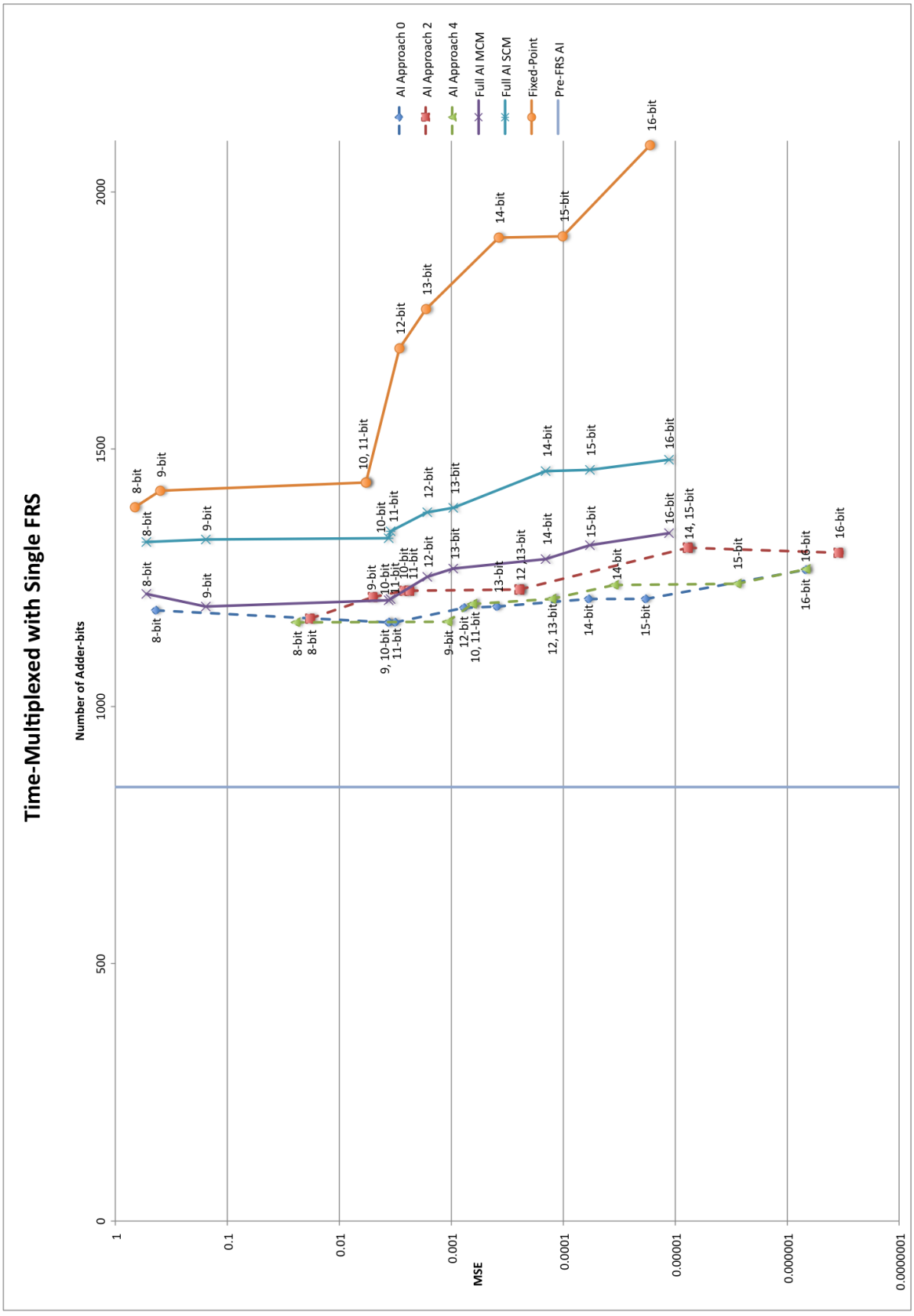

Figure 4.11: MSE vs Adder-bits for Time-Multiplexed Implementation with Single FRS

Table 4.16: MSE vs Adders for Time-Multiplexed Implementation with Single FRS- Part 1

Quantization Bits	Fixed-Point		Full AI SCM		Full AI MCM	
	MSE	No. of Adders	MSE	No. of Adders	MSE	No. of Adders
8	6.66E-01	86	5.25E-01	112	5.25E-01	105
9	3.98E-01	86	1.56E-01	112	1.56E-01	104
10	5.80E-03	86	3.61E-03	112	3.61E-03	104
11	5.80E-03	86	3.46E-03	113	3.46E-03	105
12	2.91E-03	90	1.64E-03	115	1.64E-03	107
13	1.68E-03	92	9.54E-04	115	9.54E-04	108
14	3.78E-04	96	1.43E-04	119	1.43E-04	108
15	1.01E-04	96	5.81E-05	119	5.81E-05	110
16	1.67E-05	96	1.13E-05	119	1.13E-05	111

Table 4.17: MSE vs Adders for Time-Multiplexed Implementation with Single FRS- Part 2

Quantization Bits	AI Approach 0		AI Approach 2		AI Approach 4	
	MSE	No. of Adders	MSE	No. of Adders	MSE	No. of Adders
8	4.33E-01	103	1.81E-02	102	2.44E-02	102
9	3.61E-03	102	5.04E-03	104	1.08E-03	102
10	3.61E-03	102	2.60E-03	104	6.46E-04	104
11	3.17E-03	102	2.36E-03	104	6.46E-04	104
12	7.70E-04	104	2.46E-04	104	1.28E-04	105
13	3.90E-04	104	2.46E-04	104	1.28E-04	105
14	5.92E-05	105	7.68E-06	107	3.51E-05	107
15	1.85E-05	105	7.68E-06	107	2.82E-06	106
16	6.83E-07	106	3.49E-07	107	6.83E-07	106

Table 4.18: MSE vs Adder-bits for Time-Multiplexed Implementation with Single FRS- Part 1

Quantization Bits	Fixed-Point		Full AI SCM		Full AI MCM	
	MSE	No. of Adder-bits	MSE	No. of Adder-bits	MSE	No. of Adder-bits
8	6.66E-01	1387	5.25E-01	1319.5	5.25E-01	1219
9	3.98E-01	1419	1.56E-01	1325	1.56E-01	1194.5
10	5.80E-03	1435.5	3.61E-03	1327.5	3.61E-03	1207
11	5.80E-03	1435.5	3.46E-03	1341	3.46E-03	1209
12	2.91E-03	1696	1.64E-03	1377	1.64E-03	1252.5
13	1.68E-03	1772	9.54E-04	1385.5	9.54E-04	1268.5
14	3.78E-04	1911	1.43E-04	1458	1.43E-04	1287
15	1.01E-04	1914	5.81E-05	1459.5	5.81E-05	1314
16	1.67E-05	2090.5	1.13E-05	1480	1.13E-05	1337

Table 4.19: MSE vs Adder-bits for Time-Multiplexed Implementation with Single FRS-Part 2

Quantization Bits	AI Approach 0		AI Approach 2		AI Approach 4	
	MSE	No. of Adder-bits	MSE	No. of Adder-bits	MSE	No. of Adder-bits
8	4.33E-01	1186.5	1.81E-02	1171	2.44E-02	1164
9	3.61E-03	1164	5.04E-03	1212.5	1.08E-03	1165
10	3.61E-03	1164	2.60E-03	1225	6.46E-04	1199
11	3.17E-03	1163.5	2.36E-03	1224.5	6.46E-04	1199
12	7.70E-04	1191.5	2.46E-04	1228	1.28E-04	1209.5
13	3.90E-04	1194	2.46E-04	1228	1.28E-04	1209.5
14	5.92E-05	1209.5	7.68E-06	1308.5	3.51E-05	1236.5
15	1.85E-05	1209	7.68E-06	1308.5	2.82E-06	1239
16	6.83E-07	1266	3.49E-07	1298.5	6.83E-07	1267

4.3 Chapter Summary

This chapter presented the results of the best designs found in this thesis for the full AI (Contributions 1), the simplified AI (Contribution 2) designs, and the non-AI designs. This chapter presented a detailed comparative analysis of the designs by using the metrics of MSE, the number of adders, and the number of adder-bits.

It was found that the simplified AI methods for approach 0 and 4 were the best of the AI methods. It was found that non-AI was superior to AI for both the fully parallel and time-multiplexed approaches for computing the 2D DCT. However, it was found that if a single FRS output was used, then AI would be superior to non-AI, for the number of adder-bits used in the design.

CHAPTER 5

CONCLUSIONS

This thesis has systematically explored the application of 2D AI to the 2D Arai DCT, building on such works as [2–6, 37, 49–52, 55, 56]. The SCM-based, 2D AI approach was developed and taken as reference point and many design variations were built upon it. The main focus of this thesis was to design and evaluate alternative FRS structures for the 2D AI 2D DCT, and compare the AI designs to the non-AI designs.

This thesis introduced the MCM paradigm [31] and the method of Dempster-Macleod [8, 30, 53] to design optimal constant coefficient multipliers, for the design of the full AI FRS. This thesis also introduced the total number of adder-bits as a measure of circuit complexity for AI hardware requirements, consistent with the literature in custom hardware design of DSP circuits [8,30,53,69,72]. This thesis also introduced the substitution method to reduce the order of the FRS equations, referred to in this thesis as approach 0. This thesis also introduced further enhancements to the substitution method, by developing approaches 2 and 4. These substitution methods generated many alternative design implementations to explore for each bit of coefficient quantization. Non-AI designs were also developed, to serve as a basis of comparison with the AI algorithms. These non-AI designs were kept as consistent with AI designs as possible, including following same algorithm (Arai [54]) and using the same design tools to develop constant coefficient multipliers. In total, 4608 AI designs and 26 non-AI designs were designed and evaluated, with coefficient quantization accuracies of 8- to 16-bits, to form a strong basis for comparison of the basic designs.

Custom software tools were designed to facilitate the development, simulation, and

analysis of the designs, including generating single-constant multipliers and modeling AI and non-AI circuits. Modeling of the circuits was bit-accurate so that MSE and hardware costs for the number of adders and number of adder-bits could be generated.

The findings of this thesis include:

- The demonstration of the superiority of the MCM paradigm over the SCM paradigm for full AI.
- That the substitution methods of approach 0, and 4, were significantly superior to that of the full AI designs, in both terms of mean-squared error and circuit complexity. Approach 2 was also compared but had inferior performance to approaches 0 and 4.
- It was found for both fully parallel and time-multiplexed versions of the 2D DCT, the non-AI designs were superior to the AI designs, both in terms of mean-squared error and circuit complexity.
- However it was demonstrated that an AI system could be designed with a single FRS output that was multiplexed to the AI outputs, leading to a decrease in throughput, but with reduced size of physical adders, compared to non-AI.

In general, this thesis contributes to the AI literature by providing lower circuit complexity AI designs (especially approaches 0 and 4, and the use of multiple constant multiplication) that will result in reduced circuit area, in comparison to previous AI designs for the 2D DCT. This thesis also contributes to the AI literature by highlighting, through the support of a large number and variety of designs, that the non-AI designs are generally superior to the AI designs, except for the case of using a single FRS within the AI design, which would be valuable for area-limited designs.

BIBLIOGRAPHY

[1] K. Wahid, V. Dimitrov, and G. Jullien, "New encoding of 8x8 DCT to make H.264 lossless," in *IEEE Asia Pacific Conf. Circuits and Systems*, 2006, pp. 780–783. ix, 4, 6, 7

[2] V. Dimitrov, K. Wahid, and G. Jullien, "Multiplication-free 8×8 2D DCT architecture using algebraic integer encoding," *IEE Electron. Lett.*, vol. 40, no. 20, pp. 1310–1311, 2004. ix, 3, 4, 5, 7, 8, 85, 138

[3] K. Wahid, V. Dimitrov, and G. Jullien, "Error-free computation of 8×8 2D DCT and IDCT using two-dimensional algebraic integer quantization," in *Proc. 17th IEEE Symp. Computer Arithmetic*, 2005, pp. 214–221. ix, 3, 4, 5, 7, 8, 21, 25, 27, 85, 138

[4] A. Pradini, T. Roffi, R. Dirza, and T. Adiono, "VLSI design of a high-throughput discrete cosine transform for image compression systems," in *2011 Int. Conf. Electrical Engineering and Informatics*, 2011, pp. 1–6. ix, 3, 5, 6, 8, 9, 10, 85, 125, 138

[5] A. Pradini, R. Dirza, and T. M. Roffi, "High throughput direct two dimensional discrete cosine transform DCT and inverse DCT with no transpositional buffer and no multiplier," Blog Entry. [Online]. Available: http://rvdirza.files.wordpress.com/2011/06/lsi-design-contest-2011-team-oasis-full-paper.pdf ix, 3, 5, 6, 8, 9, 10, 85, 125, 138

[6] K. Wahid, V. Dimitrov, and G. Jullien, "On the error-free realization of a scaled DCT algorithm and its VLSI implementation," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 54, no. 8, pp. 700–704, 2007. ix, 3, 4, 11, 27, 85, 138

[7] V. Dimitrov and K. Wahid, "On the error-free computation of fast cosine transform," *Int. J.: Information Theories and Applications*, vol. 12, no. 4, pp. 321–327, 2005. ix, 3, 4, 12

[8] A. G. Dempster and M. D. Macleod, "Constant integer multiplication using minimum adders," in *IEE Proc. Circuits, Devices and Systems*, vol. 141, no. 5. IEEE, 1994, pp. 407–413. xii, 4, 39, 40, 41, 42, 48, 51, 138

[9] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, *Synthesis and Optimization of DSP Algorithms*. Kluwer Academic Publishers, 2004. xii, 39, 59, 60

[10] H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity transform and quantization in H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 598 – 603, July 2003. 1

[11] N. Roma and L. Sousa, "Efficient hybrid DCT-domain algorithm for video spatial downscaling," *EURASIP J. Advances Signal Proc.*, no. 2, pp. 30–31, 2007. 1

[12] R. Westwater and B. Furht, *Real-time video compression: techniques and algorithms*, ser. Kluwer international series in engineering and computer science. Kluwer, 1997. 1

[13] A. Madisetti and A. N. Willson, "A 100 MHz 2-D 8×8 DCT-IDCT processor for HDTV applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, no. 2, pp. 158–165, Apr. 1995. 1

[14] D. F. Chiper, M. Swamy, M. O. Ahmad, and T. Stouraitis, "Systolic algorithms and a memory-based design approach for a unified architecture for the computation of DCT-DST-IDCT-IDST," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 6, pp. 1125–1137, Jun. 2005. 1

[15] P. K. Meher and M. N. S. Swamy, "New systolic algorithm and array architecture for prime-length discrete Fourier transform," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 3, pp. 262–266, 2007. 1

[16] T.-Y. Sung, Y.-S. Shieh, and H.-C. Hsin, "An efficient VLSI linear array for DCT/IDCT using subband decomposition algorithm," *Math. Probl. Eng.*, pp. 1–21, 2010. 1

[17] H. Huang, T.-Y. Sung, and Y.-S. Shieh, "A novel VLSI linear array for 2-D DCT-IDCT," in *2010 Proc. 3rd Int. Congr. Image and Signal Processing*. IEEE, 2010, pp. 3686–3690. 1

[18] Y.-H. Chen and T.-Y. Chang, "A high performance video transform engine by using space-time scheduling strategy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 4, pp. 655–664, Apr. 2012. 1

[19] P. K. Meher, J. C. Patra, and A. P. Vinod, "A 2-D systolic array for high-throughput computation of 2-D discrete Fourier transform," in *Proc. IEEE Asia Pacific Conf. Circuits and Systems*, 2006. 1

[20] P. K. Meher, "Unified DA-based parallel architecture for compting the DCT and the DST," in *Proc. 5th IEEE Int. Conf. Information, Communications, and Signal Processing*, 2005, pp. 1278–1282. 1

[21] S. Nayak and P. Meher, "3-dimensional systolic architecture for parallel VLSI implementation of the discrete cosine transform," *IEE Proc. Circuits Devices and Systems*, vol. 143, no. 5, pp. 255–258, 1996. 1

[22] P. K. Meher, "Highly concurrent reduced-complexity 2-D systolic array for discrete Fourier transform," *IEEE Signal Process. Lett.*, vol. 13, no. 8, pp. 481–484, 2006. 1

[23] P. K. Meher and J. Patra, "A new convolutional formulation of discrete cosine transform for systolic implementation," in *Proc. IEEE Int. Conf. Information, Communications, and Systolic Implementation*, 2007, pp. 1–4. 1

[24] A. Tumeo, M. Monchiero, G. Palermo, F. Ferrandi, and D. Sciuto, "A pipelined fast 2D-DCT accelerator for FPGA-based SoCs," in *Proc. IEEE Computer Society Annual Symp. VLSI*, 2007. 1

[25] J.-I. Guo, R.-C. Ju, and J.-W. Chen, "An efficient 2-D DCT/IDCT core design using cyclic convolution and adder-based realization," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 4, pp. 416–428, Apr. 2004. 1

[26] C.-C. Sun, P. Donner, and J. Gotze, "Low-complexity multi-purpose IP core for quantized discrete cosin and integer transform," in *Proc. 2009 IEEE Int. Symp. on Circuits and Systems*, 2009, pp. 3014–3017. 1

[27] A. M. Shams, A. Chidanandan, W. Pan, and M. A. Bayoumi, "NEDA: A low-power high-performance DCT architecture," *IEEE Trans. on Signal Process.*, vol. 3, no. 3, pp. 955–964, Mar. 2006. 1

[28] C.-Y. Huang, L.-F. Chen, and Y.-K. Lai, "A high-speed 2-D transform architecture with unique kernel for multi-standard video applications," in *Proc. IEEE 2008 Int. Symp. Circuits and Systems*, 2008, pp. 21–24. 1

[29] C. Inacio and D. Ombres, "The DSP decision: fixed point or floating?" *IEEE Spectr.*, vol. 33, no. 9, pp. 72–74, September 1996. 1

[30] O. Gustafsson, A. G. Dempster, K. Johansson, M. D. Macleod, and L. Wanhammar, "Simplified design of constant coefficient multipliers," *Circ. Syst. Signal Pr.*, vol. 25, no. 2, pp. 225–251, Apr. 2006. 1, 2, 4, 48, 138

[31] "PARI/GP, version `2.5.0`," http://pari.math.u-bordeaux.fr, The PARI Group, Bordeaux, 2011. 1, 51, 138

[32] A. V. Oppenheim and C. J. Weinstein, "Effects of finite register length in digital filtering and the fast Fourier transform," *Proc. IEEE*, vol. 60, no. 8, pp. 957–976, 1972. 2

[33] K. Ihsberner, "Roundoff error analysis of fast DCT algorithms in fixed point arithmetic," *Numer. Algorithms*, vol. 46, pp. 1–22, 2007. [Online]. Available: http://dx.doi.org/10.1007/s11075-007-9123-1 2

[34] H. A. Peterson, A. J. Ahumada, and A. B. Watson, "The visibility of DCT quantization noise," *SID Int. Symp. Dig. Tech. Papers*, vol. 24, p. 942, 1993. 2

[35] M. A. Robertson and R. L. Stevenson, "DCT quantization noise in compressed images," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 1, pp. 27– 38, Jan. 2005. 2

[36] V. S. Dimitrov, G. A. Jullien, and W. C. Miller, "A new DCT algorithm based on encoding algebraic integers," in *Proc. 1998 IEEE Int. Conf. Acoustics, Speech and Signal Processing*, vol. 3, 1998, pp. 1377–1380. 2, 3

[37] K. Wahid, "Error-free algorithms and architectures of discrete cosine transforms using multidimensional algebraic integer quantization," Ph.D. dissertation, Dept. Elect. and Comp Eng., Univ. of Calgary, Calgary, AB, CA, December 2007. 2, 3, 85, 138

[38] J. H. Cozzens and L. A. Finkelstein, "Computing the discrete Fourier transform using residue number systems in a ring of algebraic integer," *IEEE Trans. Inf. Theory*, vol. IT-31, no. 5, pp. 580–588, Sep. 1985. 3

[39] M. Fu, G. A. Jullien, V. S. Dimitrov, and M. Ahmadi, "A low-power DCT ip core based on 2D algebraic integer encoding," in *Proc. 2004 Int. Symp. Circuits and Systems*, vol. 2, 2004, pp. 765–768. 3

[40] V. Dimitrov and G. A. Jullien, "Multidimensional algebraic integer encoding for high performance implementation of the DCT and IDCT," *IEE Electron. Lett.*, vol. 29, no. 7, pp. 602–603, 2003. 3, 26

[41] R. Baghaie and V. Dimitrov, "Systolic implementation of real-valued discrete transforms via algebraic integer quantization," *Comput. Math. Appl.*, vol. 41, pp. 1403–1416, 2001. 3, 21, 22, 25, 33

[42] U. Meyer-Base and F. Taylor, "Optimal algebraic integer implementation with application to complex frequency sampling filters," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 11, pp. 1078–1082, 2001. 3, 26

[43] R. Games, D. Moulin, and J. Rushanan, "VLSI design of algebraic-integer signal processor," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, 1990, pp. 809–812. 3

[44] R. C. Agarwal and C. S. Burrus, "Number theoretic transforms to implement fast digital convolution," in *Proc. IEEE*, vol. 63, 1975, pp. 550–560. 3

[45] I. S. Reed and T. K. Truong, "Convolution over residue classes of quadratic integers," in *IEEE Trans. Inf. Theory*, vol. IT-22, July 1976, pp. 468–475. 3

[46] J. M. Pollard, "Implementation of number theoretic transforms," in *Electron. Lett.*, vol. 12, 1976, pp. 378–279. 3

[47] E. Dubois and A. N. Venetsanopoulos, "The discrete Fourier transform over finite rings with application to fast convolution," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 28, no. 2, Apr. 1980. 3

[48] ——, "The discrete Fourier transform over finite rings with application of fast convolution," *IEEE Trans. Comput.*, vol. c27, no. 7, July 1978. 3

[49] K. Wahid, S. B. Ko, and V. S. Dimitrov, "Low-area and low-power efficient video compressor for endoscopic capsules," in *Proc. 4th Int. Conf. Biomedical Engineering*, Kuala Lumpur, 2008. 3, 85, 138

[50] K. Wahid, S. B. Ko, and D. Teng, "Efficient hardware implementation of an image compressor for wireless capsule endoscopy applications," in *IEEE Int. Joint Conf. Neural Networks*, 2008, pp. 2761 –2765. 3, 85, 138

[51] K. Wahid, "An efficient IEEE-compliant 8×8 Inv-DCT architecture with 24 adders," *IEEJ Trans. Electron. Inf. Syst.*, vol. 131, pp. 1081–1082, 2011. 3, 4, 85, 138

[52] K. Wahid, V. Dimitrov, W. Badawy, and G. Jullien, "Error-free arithmetic and architecture for h.264," in *39th Asilomar Conf. Signals, Systems and Computers*, 2005, pp. 703 –707. 3, 85, 138

[53] O. Gustafsson, A. G. Dempster, and L. Wanhammar, "Extended results for minimum-adder constant integer multipliers," in *2002 IEEE Int. Symp. on Circuits and Systems*, vol. 1, 2002, pp. 73–76. 4, 51, 138

[54] Y. Arai, T. Agui, and M. Nakajima, "A fast DCT-SQ scheme for images," *IEICE Trans.*, vol. E71-E, no. 11, pp. 1095–1097, 1988. 7, 19, 79, 138

[55] A. Madanayake, R. J. Cintra, D. Onen, V. S. Dimitrov, N. T. Rajapaksha, L. T. Bruton, and A. Edirisuriya, "A row-parallel 8x8 2-D DCT architecture using algebraic integer based exact computation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. PP, no. 99, 2011. 11, 12, 13, 26, 48, 51, 68, 70, 85, 138

[56] A. Madanayake, R. J. Cintra, D. Onen, V. S. Dimitrov, and L. T. Bruton, "Algebraic integer based 8×8 2-D DCT architecture for digital video processing," in *Proc. 2011 Int. Symp. Circuits and Systems*, 2011. 12, 13, 26, 48, 51, 68, 70, 85, 138

[57] N. Rajapaksha, A. Edirisuriya, A. Madanayake, R. J. Cintra, D. Onen, I. Amer, and V. S. Dimitrov, "Asynchronous realization of algebraic integer-based 2D DCT using Achronix speedster SPD60 FPGA," *J. Electr. Comput. Eng.*, pp. 1–9, 2013. 12, 13, 48

[58] [Online]. Available: http://www.mathworks.com/help/toolbox/signal/ref/dct.html 17

[59] [Online]. Available: http://www.mathworks.com/help/toolbox/images/ref/dct2.html 18

[60] E. Feig and S. Winograd, "Fast algorithms for the discrete cosine transform," *IEEE Trans. Signal Process.*, vol. 40, no. 9, pp. 2174–2193, September 1992. 18

[61] H. Lim, C. Yim, and E. E. Swartzlander, "Finite word-length effects of an unified systolic array for 2-D DCT/IDCT," in *Proc. Int. Conf. ASAP*, 1996, pp. 35–44. 18

[62] M. Kovac and N. Ranganathan, "JAGUAR: a fully pipelined VLSI architecture for JPEG image compression standard," in *Proc. IEEE*, vol. 83, no. 2, 1995, pp. 247–258. 19

[63] E. Linzer and E. Feig, "New scaled DCT algorithms for fused multiply/add architectures," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.* IEEE, 1991, pp. 2201–2204. 20

[64] R. Dedekind, *Theory of Algebraic Integers*. Cambridge University Press, 1996. 20

[65] P. Cappello and K. Steiglitz, "Some complexity issues in digital signal processing," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 32, no. 5, pp. 1037 – 1041, October 1984. 34

[66] M. A. Soderstrand, L. G. Johnson, H. Arichanthiran, M. D. Hoque, and R. Elangovan, "Reducing hardware requirement in FIR filter design," Tech. Rep. 35

[67] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "The multiple wordlength paradigm," in *Proc. IEEE Symp. Field Programmable Custom Computing Machines*, 2001. 39, 59

[68] D. Onen, "Design of a constant coefficient multiplier CAD tool," University of Calgary, Tech. Rep., 2002. 46, 48, 49

[69] M. Puschel, J. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. Johnson, and N. Rizzolo, "SPIRAL: Code generation for DSP transforms," *Proc. IEEE*, vol. 93, no. 2, pp. 232–275, 2005. 46, 49, 138

[70] [Online]. Available: http://spiral.net/index.html 49

[71] Z. Wang and A. C. Bovik, "Mean squared error: love it or leave it - a new look at signal fidelity measures," *IEEE Signal Process. Mag.*, vol. 26, no. 1, pp. 98–117, January 2009. 58

[72] Y. Voronenko and M. Puschel, "Multiplierless multiple constant multiplication," *ACM Trans. on Algorithms*, vol. 3, no. 2, 2007. 138