TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

NIRAJAN PANT

DISCRETE SINE AND COSINE TRANSFORMS ON PARALLEL PROCESSORS

Master of Science Thesis

Examiner: Prof. Jarmo Takala
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 8th April 2015

ABSTRACT

NIRAJAN PANT: Discrete Sine and Cosine Transforms on Parallel Processors
Tampere University of Technology
Master of Science Thesis, 44 pages, 3 Appendix pages
June, 2015
Master's Degree Program in Information Technology
Major: Digital and Computer Electronics
Examiner: Prof. Jarmo Takala
Keywords: fixed-point number, floating-point number, discrete cosine transform, discrete sine transform, digital signal processing, processor configuration

Starting point of this master thesis is Discrete Cosine Transform (DCT) and Discrete Sine Transform (DST) algorithms for signal processing. Based on the number system used in DCT and DST application, they can be categorized as fixed-point and floating-point DCT/DST. Floating-point numbers have large dynamic range to represent very large and small numbers. However, floating-point operation requires more clock cycles than fixed-point operation. Specialized hardware can be used for floating-point operations for high performance, but it also increases hardware cost. So, for general applications, use of fixed-point number system would be a good choice provided that an optimum accuracy is guaranteed.

In this thesis, the existing floating–point DCT and DST of type-1 C-codes are first converted into fixed-point code. The fractional fixed-point representation is used for the fixed-point conversion for maximum possible accuracy. The choice of Q15 format provides highest precision for signed 16-bit fixed-point number. But in this format, the range of numbers has to be normalized between [-1, 1]. The conversion process introduces some error in the output which is calculated by signal to noise ratio (SNR). After designing the fixed-point DCT/DST code, the performance is evaluated in various Tensilica processor configurations. The configurations provided are generated for Tensilica's Diamond Standard Processor cores in Tensilica Xtensa Environment. The clock cycle counts of both fixed-point and floating-point DCT/DST code on four different configurations are recorded.

The results show that SNR of fixed-point DCT/DST is between (35-76dB) for different transform size of DCT/DST, which suggests that the fixed-point code is accurate enough. It is also observed that the fixed-point DCT/DST provides approximately 3 to 6 times performance improvement over floating-point code on Tensilica processors cores in terms of clock cycles. Furthermore, Tensilica's Diamond Standard 570T parallel processor configuration provides the best performance among all configurations used for designed fixed-point code. Results have shown that the fixed-point DCT/DST code offers a large performance improvement over floating-point code provided that the floating-point code has no added hardware support.

PREFACE

The Master of Science thesis was completed in Department of Pervasive Computing at Tampere University of Technology.

I would like to express my sincere gratefulness to my supervisor Prof. Jarmo Takala for providing me opportunity to work on this topic and examining my thesis. I am indebted to him for providing knowledge on the topic, answering my question, and correcting on my manuscript. I would also like to pay my gratitude to him for providing fund to carry out the thesis work.

Special thanks to my friends Bishwa Subedi and Prakash Subedi for patiently helping me to organize the work and sharing their helpful knowledge and experiences. I would also like to express my appreciation to friends Anil Baniya, Puskal Kunwar, Abhishekh Gupta and Prakash K.C. for their constant support and motivation.

I must acknowledge my wife and best friend; Jenny Maharjan Pant for her continues love, care and motivation. In particular, for the patience and understanding shown by her during the study year is greatly appreciated.

This thesis work is dedicated to my parents and my brother for their support, care, love and affection.

Nirajan Pant
Tampere, 28.07.2015

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABI	Application Binary Interface
DCT	Discrete Cosine Transform
DCT-I	DCT of type I
DCT-II	DCT of type II
DCT-III	DCT of type III
DCT-IV	DCT of type IV
DST	Discrete Sine Transform
DST-I	DST of type I
DST-II	DST of type II
DST-III	DST of type III
DST-IV	DST of type IV
DSP	Digital Signal Processing
FLIX	Flexible Length Instruction Extension
FPU	Floating-Point Unit
IDE	Integrated Development Environment
IP	Intellectual Property
ISA	Instruction Set Architecture
ISS	Instruction Set Simulator
GUI	Graphical User Interface
MAC	Multiply and Accumulate
MEX	MATLAB Executable
SDK	Software Development Kit
SNR	Signal-to-Noise Ratio
SoC	System on Chip
TIE	Tensilica Instruction Extension
VLIW	Very Long Instruction Word

1. INTRODUCTION

Digital signal processing is gaining more and more significance in daily life applications. The Discrete Cosine Transform (DCT) and Discrete Sine Transform (DST) algorithms are widely used in digital signal processing. The application ranges from image processing, speech processing, transform coding systems for data compression/decompression up to solution of differential equations [5]. Moreover, the DCT and DST of type 1 can even be used in calculation of the inverse of Circulant Hermitian matrices [10].

Based on data representation, Digital Signal Processors (DSP) are classified as fixed-point processors and floating-point processors. Typically a large number of computations are needed to perform a digital signal processing task. Therefore, a chosen numeric representation has a huge effect on the design and performance of a DSP processor. An application code has to be designed in respective number format to run over these DSP processors for better efficiency. A floating-point number has larger dynamic range than fixed-point numbers. In longer floating-point representations the dynamic range is so large that it is sufficient for many practical applications. However, the drawback of floating point number system is that every floating-point operation requires more clock cycles than fixed-point operation as the later uses integer operations [11]. Thus, floating-point systems are costly to implement in terms of clock cycles and hardware. The processor generally includes specialized hardware (floating-point unit) that performs floating-point arithmetic. The floating-point arithmetic is better suited to general-purpose computing, where the requirements of computing cannot be known at the time of developing the computing hardware. Fixed-point computing can be used in cases where the requirements of the applications can be exploited during the development of hardware. E.g., in application-specific solutions, fixed-point arithmetic can be exploited to produce more optimized computing structure for the given application.

Designing an accurate and efficient fixed-point code is the main scope of this thesis. Starting point is to select a reference floating-point code for DCT and DST of type 1 [5]. The choice of Q-format determines the accuracy and range of fixed-point numbers [14]. The Q15 format chosen for this thesis work has best possible accuracy for 16-bit numbers but the range has to be normalized between [-1,1]. The Signal-to-Noise Ratio (SNR) analysis is used to understand the accuracy of fixed-point design with respect to reference floating-point code.

Another part of the thesis work is to evaluate the performance of designed fixed-point code and referenced floating-point as in [5], on different target processor cores. The target processor cores for the generated fixed-point DCT and DST code are Tensilica Diamond Processor cores. The Diamond Standard Processor cores are high perfor-

mance preconfigured fixed-point DSP cores that employs 32- bit registers as base architecture [20]. The hosts of tools provided by Tensilica's Xtensa Environment are used to run the DCT/DST code in Tensilica Processors [17]. By using Tensilica software development tools, the floating-point DCT/DST arithmetic operations can also be emulated in 32-bit integer architecture. The Tensilica Xtensa tools can generate different processor configurations that can be used for a particular application. In this thesis work, four different processor configurations are used, which are pre-build inside the Xtensa Environment. The configurations are named DE_106micro, DE_108mini, DE_212GP and DE_570T associated with different Diamond Standard Processor Cores as mentioned in Tensilica white paper [20]. Among them, the DE_570T configuration uses very long instruction word (VLIW) instruction used for parallel processing. The DCT/DST application codes are profiled in Tensilica Environment to record the clock cycles in different configurations.

In this thesis, the initial chapters provide theoretical and technical background necessary for this research work. The chapter 2 explains discrete cosine and sine transforms, their mathematical properties and application domains. In chapter 3, a detailed explanation related to fixed-point number system and arithmetic along with a brief introduction about floating-point numbers is presented. Chapter 4 familiarizes with the tools and frameworks that are used to run the application DCT and DST code in Tensilica Xtensa Environment. Furthermore, different Tensilica Processor architectural features are described. In the chapter 5, the fixed-point design methodology is explained. Moreover, the chapter discusses MEX functions that facilitate the use of DCT/DST C-codes in MATLAB. This chapter also discusses about the processor configurations that are generated in Xtensa Environment. The chapter 6 discusses the results of the thesis. The SNR results of fixed-point code with respect to reference floating-point code and profiling result of DCT/DST are also presented. Finally, the last chapter includes the conclusion of this thesis work and recommendations for possible future work.

2. DISCRETE COSINE AND SINE TRANSFORMS

In this chapter, a general introduction about four even types of Discrete Cosine Transforms (DCT), Discrete Sine Transforms (DST) and their mathematical properties are discussed. A generalized signal flow graph for DCT and DST of type 1 is presented. Furthermore, general application areas of DCT and DST are also discussed.

2.1 Definitions of DCT and DST

A cosine/sine transform uses sum of cosine/sine functions oscillating at different frequencies to represent a waveform having relatively complex variation in signal amplitude. When the waveform and sine/cosine functions are sampled at certain intervals, they are known as discrete cosine/sine transforms [4].

The discrete cosine transform and discrete sine transform are associated with the family of sinusoidal unitary transform. The complete sets of DCT and DST are known as discrete trigonometric transform, which consists of eight versions of DCT and corresponding eight versions of DST [5]. These sets are identified as even or odd and of types I, II, III, and IV. Almost all DCT and DST digital and image processing signals application use only even types.

The four versions of even DCT matrices i.e. DCT type I, II, III and IV are defined as [5];

$$DCT-I:\ [C_{N+1}^{I}]_{mn} = \sqrt{\frac{2}{N}}\left[x_m x_n \cos\left(\frac{mn\pi}{N}\right)\right],\ \text{m, n=0,1,…….,N,} \qquad (2.1)$$

$$DCT-II:\ [C_{N}^{II}]_{mn} = \sqrt{\frac{2}{N}}\left[x_m \cos\left(\frac{m(n+\frac{1}{2})\pi}{N}\right)\right],\ \text{m, n = 0,1,…….,N-1,} \qquad (2.2)$$

$$DCT-III:\ [C_{N}^{III}]_{mn} = \sqrt{\frac{2}{N}}\left[x_n \cos\left(\frac{(m+\frac{1}{2})n\pi}{N}\right)\right],\ \text{m, n = 0,1,…….,N-1,} \qquad (2.3)$$

$$DCT-IV:\ [C_{N}^{IV}]_{mn} = \sqrt{\frac{2}{N}}\left[\cos\left(\frac{(m+\frac{1}{2})(n+\frac{1}{2})\pi}{N}\right)\right],\ \text{m, n = 0,1,…….,N-1,} \qquad (2.4)$$

where x_i is a scaling factor defined as:

$$x_{i(i=m\ or\ n)} = \begin{cases} \frac{1}{\sqrt{2}}, & if\ i = 0\ or\ i = N \\ 1, & otherwise \end{cases}.$$

The corresponding four types of even DST matrices denoted as DST type I, II, III, IV are defined as:

$$DST-I:\ [S_{N-1}^{I}]_{mn} = \sqrt{\frac{2}{N}}\left[\sin\left(\frac{\pi(m+1)(n+1)}{N}\right)\right],\ \text{m, n} = 0,1,\ldots,\text{N-2},$$

$$DST-II:\ [S_{N}^{II}]_{mn} = \sqrt{\frac{2}{N}}\left[x_n \sin\left(\frac{\pi(2m+1)(n+1)}{2N}\right)\right],\ \text{m, n} = 0,1,\ldots,\text{N-1},$$

$$DST-III:\ [S_{N}^{III}]_{mn} = \sqrt{\frac{2}{N}}\left[x_m \sin\left(\frac{\pi(2n+1)(m+1)}{2N}\right)\right],\ \text{m, n} = 0,1,\ldots,\text{N-1},$$

$$DST-IV:\ [S_{N}^{IV}]_{mn} = \sqrt{\frac{2}{N}}\left[\sin\left(\frac{\pi(2m+1)(2n+1)}{4N}\right)\right],\ \text{m, n} = 0,1,\ldots,\text{N-1},$$

where

$$x_{j(j=m\ or\ n)} = \begin{cases} \frac{1}{\sqrt{2}}, & if\ j = 0\ or\ j = N \\ 1, & otherwise \end{cases}.$$

In the above equations, *N* represents an integer, which is a power of 2. A superscript of a matrix represents its version number while a subscript represents the order.

Different authors have introduced different sets of Discrete Sine and Cosine transforms [5]. The DCT of type I (DCT-I), first introduced by Wang and Hunt, is defined for the order *N*+1 whereas, DST of type I (DST-I) defined for order *N*-1 is introduced by Jain. The first definitions of DCT of type II (DCT-II) and its inverse (DCT-III) were given by Ahmed et al. Kekre and Solanki first reported the DST of type II (DST-II) and its inverse (DST-III). Furthermore, Jain also introduced the DCT and DST of type IV.

2.2 Mathematical Properties

The mathematical properties of discrete cosine and sine transforms are basis for their application on practical domain. Different properties of DCT/DST such as shifting, convolution, scaling are extensively applied in the discrete transform field. In this section, the main mathematical properties of DCT and DST are described briefly.

2.2.1 Unitarity Property

The DCT and DST are separable transforms that allow decomposition of multidimensional transform into one-dimensional transform. As DCT and DST matrices both are orthogonal, its inverse transform matrices can be obtained by matrix transpose [5]. Furthermore, DCT/DST of type-I and type-IV are symmetric meaning the inverse transform is the transform of itself. On the contrary, the type II and type III of both DCT and DST are transposes of each other.

These relations can be formulated for inverse DCT matrices,

$[C_{N+1}^{I}]^{-1} = [C_{N+1}^{I}]^{T} = C_{N+1}^{I}$, $[C_{N}^{II}]^{-1} = [C_{N}^{II}]^{T} = C_{N}^{III}$,

$[C_{N}^{III}]^{-1} = [C_{N}^{III}]^{T} = C_{N}^{II}$, $[C_{N}^{IV}]^{-1} = [C_{N}^{IV}]^{T} = C_{N}^{IV}$.

Similarly for inverse DST matrices,

$[S_{N-1}^{I}]^{-1} = [S_{N-1}^{I}]^{T} = S_{N-1}^{I}$, $[S_{N}^{II}]^{-1} = [S_{N}^{II}]^{T} = S_{N}^{III}$,

$[S_{N}^{III}]^{-1} = [S_{N}^{III}]^{T} = S_{N}^{II}$, $[S_{N}^{IV}]^{-1} = [S_{N}^{IV}]^{T} = S_{N}^{IV}$.

2.2.2 Linearity Property

All DCT and DST hold the linearity property [5]. That is, for a matrix *M*,

$$M(aI+bJ) = aMI+bMJ,$$

where *a* and *b* are constants, and *I* and *J* are vectors.

2.2.3 Shift in Time Property

The relationship between discrete cosine and sine transforms of original sequence and its shifted sequence were first presented by P. Yip and K. R. Rao [2]. Shift property can be very useful for reducing the computational complexity of the discrete transform, when the transforms have to be applied on incoming continues data stream.

If the input sequence of data points is a vector,

$$\text{x} = [x(0), x(1), \ldots \ldots x(N)]^{T},$$

then the right shifted sequence of same vector is

$$\text{x+} = [x(1), x(2), \ldots \ldots x(N+1)]^{T}.$$

The minimum shift is one sample point in the given sequence. The corresponding DCTs are given by,

$$X_c = [C]\ \text{x and}\ X_c^{+} = [C]\ \text{x+}.$$

This shift in time property not only relates X_c^{+} to X_c but also it has a relation with the DST of *x* i.e. with X_s. The shift property for DCT-I is defined as,

$$X_{C_1}^{+}(m) = \cos\left(\frac{m\pi}{N}\right) X_{C_1}(m) + k_m \sin\left(\frac{m\pi}{N}\right) X_{S_1}(m)$$
$$+\sqrt{\frac{2}{N}} k_m \left[\left(-\frac{1}{\sqrt{2}}\right) \cos\left(\frac{m\pi}{N}\right) x(0) + \left(\frac{1}{\sqrt{2}}\right) - 1)\right] x(1)$$

$$+(-1)^m\left(1-\frac{1}{\sqrt{2}}\right)\cos\left(\frac{m\pi}{N}\right)x(N)+(-1)^m\frac{1}{\sqrt{2}}\,x(N+1)\,.$$

Here, $X_{C_1}(m)$ and $X_{S_1}(m)$ are m^{th} element of the DCT-I of vector $[x(0),x(1),\dots,x(N)]^T$ and DST-I of vector $[x(1),x(2),\dots,x(N+1)]^T$, respectively.

Similarly the shift property of DST-I is given by

$$X_{S_1}^{+}(m)=\cos\left(\frac{m\pi}{N}\right)X_{S_1}(m)-sin\left(\frac{m\pi}{N}\right)\,X_{S_1}(m)$$
$$+\sqrt{\frac{2}{N}}\sin\left(\frac{m\pi}{N}\right)\left[\frac{1}{\sqrt{2}}x(0)-\left(1-\frac{1}{\sqrt{2}}\right)(-1)^m x(N)\right]\,.$$

The shift property of other types of DCT and DST are explained by P. Yip and K. R. Rao in [2].

2.2.4 Scaling in Time Property

Since, DCTs and DSTs are the transforms that deal with discrete sample points and its resulting transform is in discrete frequency domain, a scaling in time has no effect on the overall transform. However, a scaling in time will cause an inverse scaling in the frequency domain [3].

If Δt and Δf are time and frequency units respectively, then

$$\Delta t.\,\Delta f=\frac{1}{2N}\,.$$

Thus if Δt is scaled by a factor a and it changes to $a\Delta t$, then the frequency unit Δf must change to $\frac{\Delta f}{a}$, provided the number of divisions N remains the same. There is no change in the overall magnitude of the transform.

2.2.5 Difference Property

This property is useful when differentiation of the adjacent samples is required in a signal; an application being differential pulse code modulation [3].

Considering a signal with a differences of adjacent samples $d(n)$ = x(n+1) - $x(n)$, n = 0,1,…, N-1. The difference vector can be defined as;

$$d = x+ - x$$

where, x+ is a right shifted version of x. So, the DCT and DST of vector d is given by

$$D_c=X_c^{+}-X_c \text{ and } D_s=X_s^{+}-X_s\,.$$

2.2.6 Convolution Multiplication Property

Convolution multiplication property is one of the most important properties of DCT and DST. It is used to perform digital filtering in the transfer domain. The convolution in transform domain, which is a result of an inverse transform of the product of forward

transform of two input data sequences, is equivalent to symmetric convolution of those sequences in the spatial domain [5].

If $\{a_n\}$ and $\{b_n\}$ are two input data sequences to be convolved, the relationship between transform domain convolution- multiplication property and symmetric convolution can be given as:

$$\{a_n\} < \text{sc} > \{b_n\} = T_c^{-1}[T_a\{a_n\} * T_b\ \{b_n\}]$$

where <sc> denotes the operator of symmetric convolution, * represents element by element multiplication of its operands, and $T_a\{a_n\}$ represents a transform T_a of the sequence $\{a_n\}$. For example, the convolution-multiplication property of type 2 DCT (DCT-II) can be obtained by substituting $T_a = T_b = [C_N^{II}]$ and $T_c = [C_{N+1}^{I}]^{-1}$ in the previous relation.

2.3 Basic Properties of DCT/DST

Signal Flow Graph

The signal flow graphs visualize the computational structure of DCT and DST and their inverse. The signal flow graphs of DCT/DST of type 1 describe the computation of DCT-I for any $N = 2^m + 1$, and DST-I for any $2^m - 1$where $m > 0$ and N is the length of data sequence. For DCT-I and DST-I computation, the generalized signal flow graphs for N=17 and N=15 are presented in the Figures 2.1 and 2.2, respectively. The details on DCT/DST computation and signal flow graphs are mentioned by K.R. Rao et.al. in [5].

Butterfly diagram

The butterfly is the simplest 2-point DCT/DST calculation and is a basic unit of DCT/DST calculation. It consists of one addition and one multiplication operation. The DCT/DST algorithm consists of many butterfly computations. The butterfly combines the results of smaller DCT/DSTs into larger DCT/DST, or vice-versa (breaking larger DCT/DSTs into sub transforms).

Radix

In general radix means that the entire algorithm is implemented with certain butterfly blocks, i.e,, radix-2 DCT means that the DCT is computed with the aid of 2-point DCTs. Here the radix can be interpreted as the length of the building block of the fast algorithm, i.e., 4-point DCT can be considered as radix-4. Butterfly and radix are interrelated.

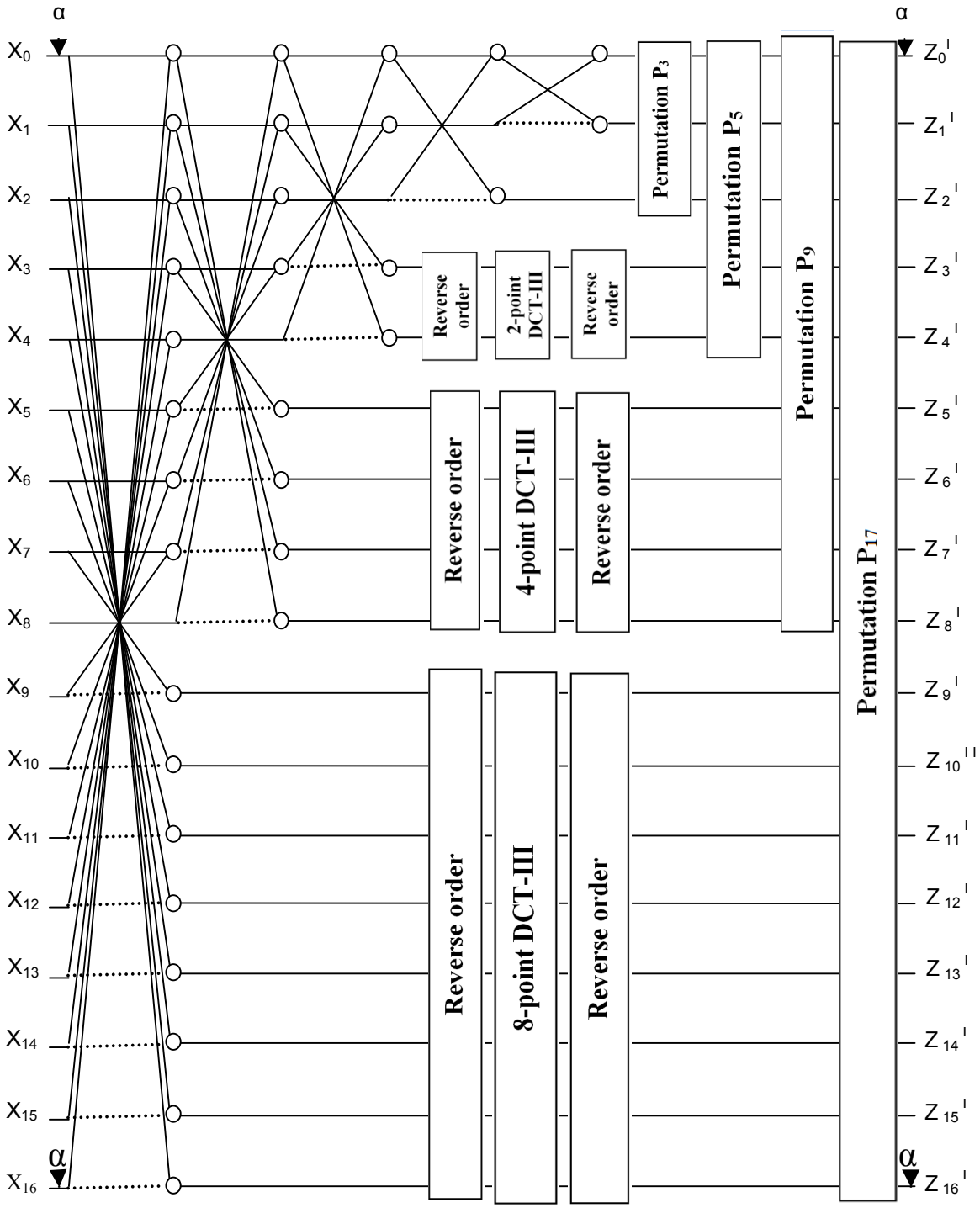

Figure 2.1. *Schematic diagram showing generalized signal flow graph of DCT-I and IDCT-I for N + 1 = 17, as in [5].*

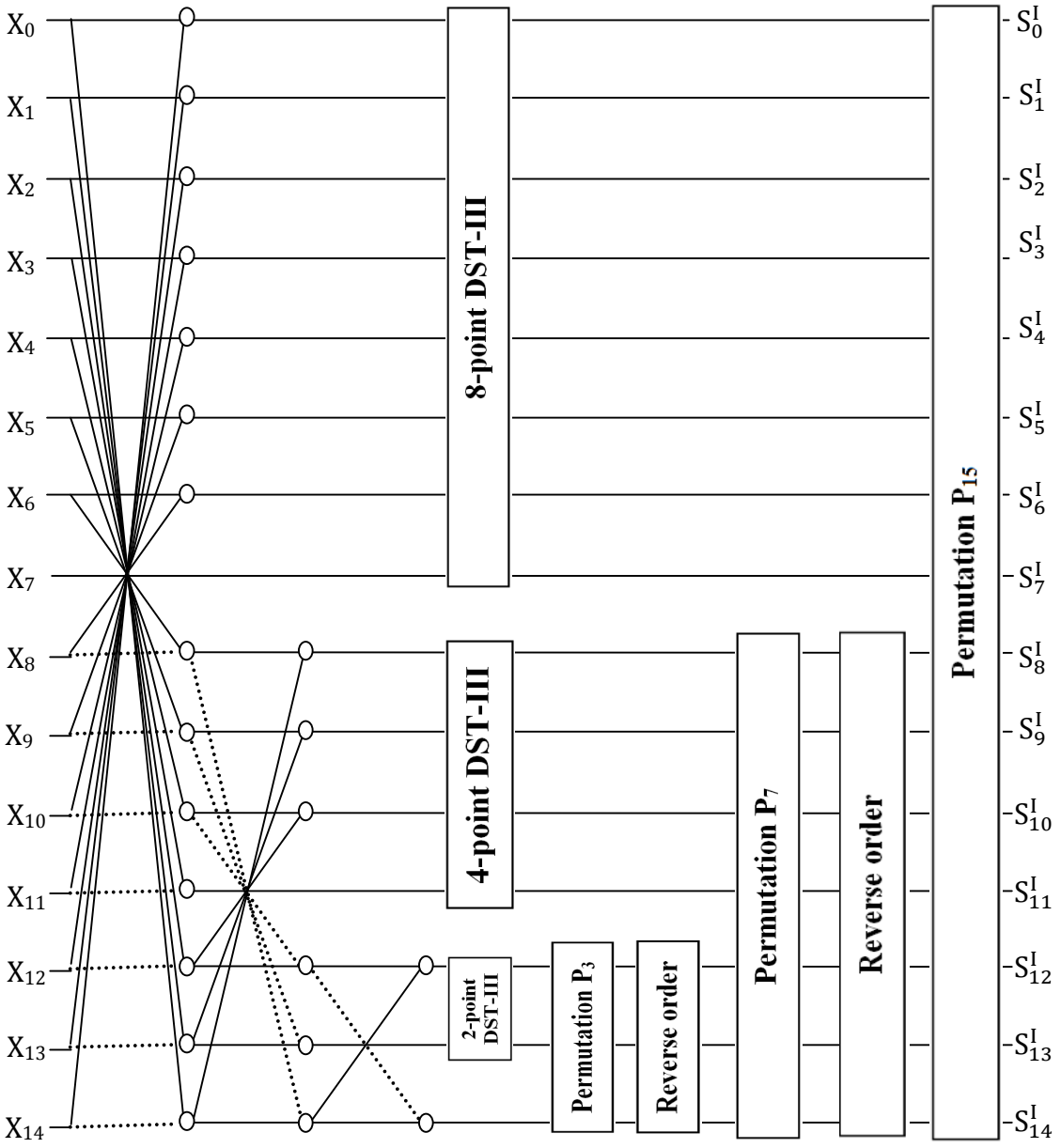

Figure 2.2. *Schematic diagram of DST-I and IDST-I generalized signal flow graph for $N - 1 = 15$, as in [5].*

2.4 Application of DCT/DST

The discrete cosine and sine transform have applications in various areas of digital signal and image processing. They are extensively used in transform coding systems for data compression and decompression [5]. The properties of DCT like decorrelation, energy compaction, separability, symmetry and orthogonality are very important in image processing applications [6]. Many international image and video coding standards have used DCT as main processing tool for data compression/decompression [7]. Simi-

larly, the international audio coding standards MPEG-1 and MPEG-2 use a modified form of DCT and DST [7] Furthermore, the DCT and DST are applicable on areas like solution of differential equations, Cepstral analysis in speech processing, and transform domain processing etc. [8]

In this thesis work, the DCT and DST of type 1 are used. Both types have similar application as general DCT. The DCT-I is as good as DCT-II in terms of computational requirements and its performance on energy compaction and digital filtering [9]. When the length of data sequence is increased, the DCT-I is competitive with DCT-II in terms of performance. At the same time, DCT-I requires less computations in comparison to DCT-II that makes it suitable even better than DCT-II for applications with larger data sequences having relatively low correlation coefficients. Furthermore, DCT and DST of type 1 are also used in calculating the inverse of circulant Hermitian matrices [10].

3. REPRESENTATION OF NUMBERS AND ARITHMETIC IN DIGITAL SIGNAL PROCESSING

In the field of digital signal processing, there are number of factors, which determine the type of processor, such as: computational efficiency, memory consumption, ease of implementation, precision requirement, time to market etc. [11]. For processor design, one of the significant criteria in decision processing is to determine the data representation by the processor for a particular application. To implement any digital signal-processing task, a large number of computations need to be performed. Therefore, a selected numeric representation has a huge influence on the design and performance of a DSP processor. The key for arithmetic representation is to represent dynamic range of numbers in less number of bits. The maximum size of an instruction and addressable memory is described by word length. Hence, a major characteristic required for choosing an arithmetic representation would be to represent a dynamic range of numbers in certain word size. In some cases, a large dynamic range is needed while in other cases, simplicity and computation efficiency is required. So, there is always a trade-off between them. There are two types of number representations: fixed-point and floating-point. According to data representation by DSP, they are classified as fixed-point processors and floating-point processors. The fig 3.1 shows how DSP are classified on the basis of number representation.

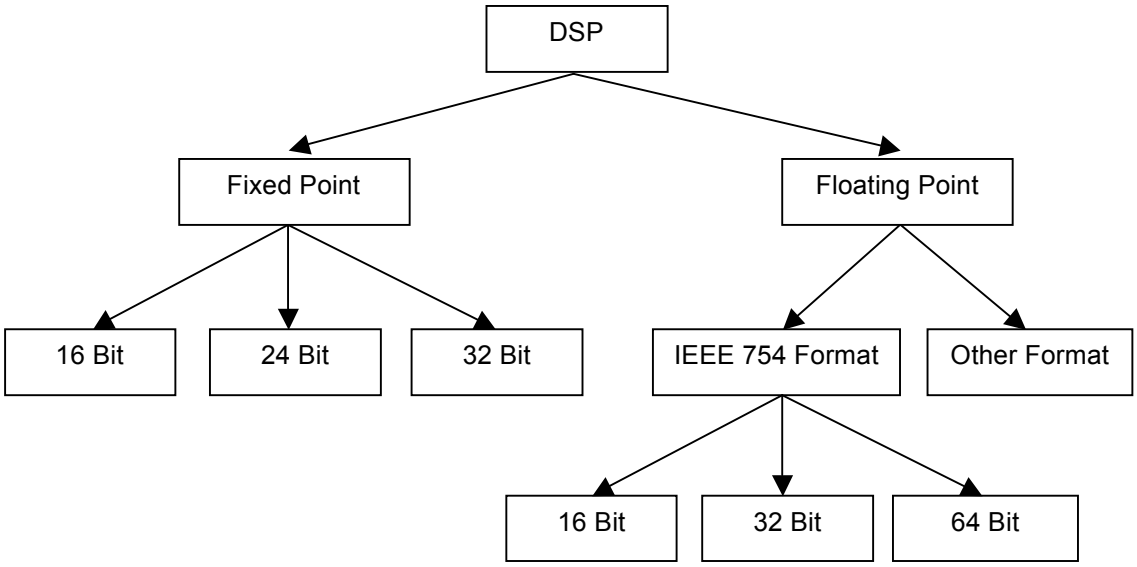

Figure 3.1. *Schematic diagram showing the DSPs on the basis of number representation [11]*

3.1 Fixed-Point Number Representation

Fixed-point number representation deals with both positive and negative numbers, and whole numbers. The key in fixed-point number representation is the concept of a binary point. The binary point divides a number between integer and fractional part, just like a decimal point in decimal system. The bits, which are left of the binary point carries a weight of 2^0, 2^1, and so on. On the other hand, the bits, which are right of the binary point, carry negative weights: $2^{-1}, 2^{-2}$ and so on.

As the name suggests, the binary point is fixed in this representation and there is a constant step between two representable numbers. The binary point could be located anywhere, e.g. in the beginning, in the end or at a certain location between the numbers. As an example, **xxx.xx** denotes fixed-point arithmetic with two bits after the binary point. The selection of binary point is done according to the precision requirements. The higher number of bits after the binary point, the higher will be the precision. The bits on the left hand side of the binary points (i.e. towards most significant bits) are known as integer bits while the bits after the decimal points are regarded as fractional bits. We need to remember that, the binary number is in fact an imaginary point that is not stored in the memory rather it is a way for the interpretation of stored binary bits. The data saved in the memory are always in the form of binary bits (i.e. 0 and 1) but this kind of representation simplifies the manipulation of those bits in different ways according to our necessities.

Fixed-point representation can be further divided into integer representation and fractional representation [11]. The integer arithmetic is used in a DSP for control operations, address calculations and other operations that are not related to signals [11]. On the other hand, fractional representation is useful in signal computations and they have values between -1 and +1.

3.1.1 Integer Representation

Integer representation is very simple and straightforward representation where the bit pattern is regarded such that the most significant bit (MSB) is the leftmost bit and the least significant bit (LSB) is the rightmost one. If the number represented is more than a byte, then the byte orientation is reliant on the endian of the representation. In big-endian representation, MSB is the leftmost bit, where as in little-endian representation, the leftmost bit is LSB, keeping same internal orientation of bits in every byte. Figure 3.2 shows the bit pattern of 16-bit binary integer representation.

x_{15}	....	....	....	x_2	x_1	x_0

Figure 3.2. *Bit format of integer representation*

The ranges of representable numbers are dependent on the number of bits and the weight of each bit is dependent on the bit position. In unsigned binary representation that shows numbers only in positive range, the representable range R for n number of bits is

$$0 \leq R \leq 2^n - 1.$$

The decimal value for n number of bit pattern can be calculated as,

$$X = 2^0 x_0 + 2^1 x_1 + \ldots + 2^{n-1} x_{n-1} = \sum_{i=0}^{n} 2^i x_i$$

where x is the bit position in the number.

The signed binary representation includes both positive and negative numbers. The MSB is called as sign bit and its value reflects whether the number is negative or positive. For a negative number, the sign bit is '1' whereas the sign bit is '0' for positive number. The representable range remains the same in signed binary representation; however, the maximum representable positive number gets reduced almost by half. The range R of any n number of bits can be found as,

$$-2^{n-1} \leq R \leq 2^{n-1} - 1.$$

The decimal value X for signed n numbers of bit can be calculated as,

$$X = -2^{n-1} x_{n-1} + \ldots\ldots + 2^1 x_1 + 2^0 x_0 = -(2^{n-1} x_{n-1}) + \sum_{i=0}^{n} 2^i x_i \,. \qquad (3.1)$$

3.1.2 Fractional Representation

In case of integer representation, double number of bits is required to store the result of multiplication operation. But, if the numbers can be normalized in the range of [-1, 1), the result will not overflow (exception is -1 x -1 = +1). This is because multiplying a fraction by a fraction always results in a fraction. (For example, 0.99999 x 0.99999 is always less than 1). This kind of representation is known as fractional representation [11]. Figure 3.3 shows the fractional representation for 16-bit number.

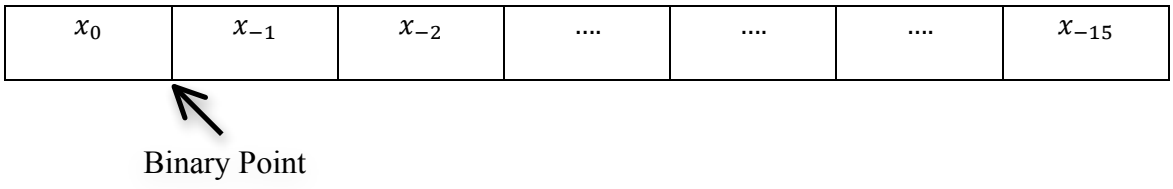

Figure 3.3. *Bit format of fractional representation*

In fractional representation, the range R for a number having n fractional bits can be calculated as,

$$-1 \leq R \leq 1 - 2^{-(n-1)} \,.$$

The decimal value X of the fractional number can be calculated as

$$X = -x_{n-1} + 2^{-1} x_{n-2} \ldots\ldots + 2^{-(n-1)} x_0 \,. \qquad (3.2)$$

3.1.3 Q-Format

Fixed-point numbers, combining both integer and fractional representation are generally represented by a well-defined Q-format [12]. The Q-format represents a fixed-point number in the form of $Qm.n$, where m represents the number of integer bits on the left hand side of the binary point known as integer word length and n represents the number of fractional bits on the right hand side of the binary point called fraction word length. The total number of bits in the format is called word length [13]. Hence, a fixed-point number is characterized by word length in bits, the location of binary point and sign of the number (signed or unsigned) [14].

Figure 3.4 shows different fixed-point format with imaginary binary point at different locations.

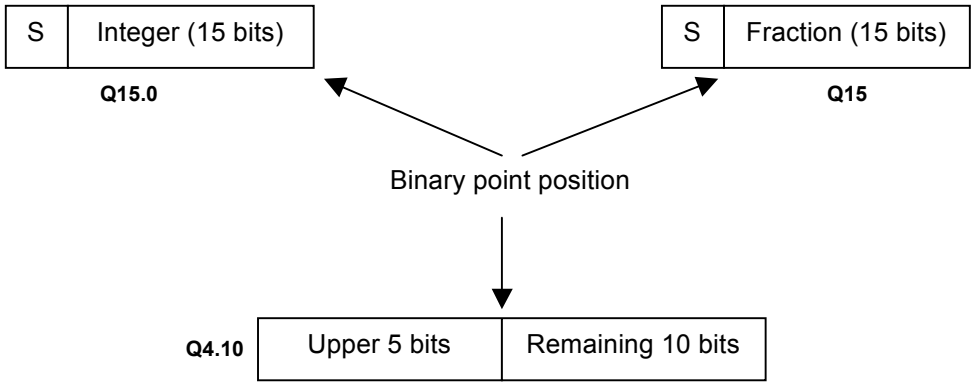

Figure 3.4. *Block diagram representing different Q-format*

There are no integer bits in case of fractional fixed-point representation. Therefore, this special format can be regarded as Qn format, where n is the number of fractional bits. For example, in a signed number, a $Q2.14$ format has 2 integer bits and 14 fractional bits and a sign bit. On the other hand, $Q15$ format has 15 fractional bits and 1 sign bit. However, some fixed-point designers may consider the sign bit while some do not; this is just a way to interpret a fixed-point number. A $Q5.5$, format can be interpreted such that total number of bits required are 10, without including the sign bit; while for the same format, some designer may consider total number of bits required are 11 considering a sign bit.

The location of binary point determines how fixed-point numbers are interpreted in decimal system. For example, combining equations (3.1) and (3.2), in signed two's complement arithmetic [15], the same 5-bit binary number can be interpreted as follows:

10110. Indicates $(-2^4 + 2^2 + 2)$ = -10 in decimal.

101.10 Indicates $(-2^2 + 2^0 + 2^{-1})$ = -2.5 in decimal.

1.0110 Indicates $(-2^{-0} + 2^{-2} + 2^{-3})$ = -2.5 in decimal.

3.1.4 Fixed-Point Range and Precision

The range of a fixed-point number is the minimum and maximum values, a Q-format can represent. For a fixed-point number with word length $'WL'$ and fractional word length $'FWL'$, the range of the format is from

$-2^{WL-FWL-1}$ to $2^{WL-FWL-1} - 2^{-FWL}$, for a signed number

0 to $2^{WL-FWL} - 2^{-FWL}$, for a unsigned number

The precision of fixed-point number is the distance between successive numbers within the range. For both signed and unsigned fixed-point numbers, the precision is 2^{-FWL}.

Table 3.1. *16-bit signed fixed-point range, precision and Q-formats [14]*

Q-Format	**Maximum Positive Value in Decimal**	**Maximum Negative value in Decimal**	**Quantization step/ Precision**
Q1.15 or Q15	0.999969482421875	-1	0.00003051757813
Q2.14	1.99993896484375	-2	0.00006103515625
Q3.13	3.9998779296875	-4	0.00012207031250
Q4.12	7.999755859375	-8	0.00024414062500
Q5.11	15.99951171875	-16	0.00048828125000
Q6.10	31.9990234375	-32	0.00097656250000
Q7.9	63.998046875	-64	0.00195312500000
Q8.8	127.99609375	-128	0.00390625000000
Q9.7	255.9921875	-256	0.00781250000000
Q10.6	511.984375	-512	0.01562500000000
Q11.5	1023.96875	-1024	0.03125000000000
Q12.4	2047.9375	-2048	0.06250000000000
Q13.3	4095.875	-4096	0.12500000000000
Q14.2	8191.75	-8192	0.25000000000000
Q15.1	16383.5	-16384	0.50000000000000
Q16.0	32767	-32768	1.00000000000000

Therefore, the fixed-point number has higher precision, if it has higher number of fractional bits. On the other hand, the range will decrease if we increase the number of fraction bits. Table 3.1 shows different Q-formats of signed 16-bit fixed-point numbers along with their range and precision.

3.1.5 Fixed-Point Arithmetic Operations

In fixed-point arithmetic calculations, special attention has to be taken to keep track of the binary point. Although, keeping track of binary point is easy and systematic, the scaling to avoid overflow is more problematic in arithmetic operations. The arithmetic operations are addition, subtraction, multiplication and division. Division operation is equivalent to multiplication by the multiplicative inverse, so it is not explained below.

Shifting is the key in a fixed-point representation [14]. It is used for addition/subtraction and multiplication. Therefore, a brief explanation about shifting is presented first.

Shifting

Shifting a number to the right by one bit is equivalent to the division of the number by 2^1 . Similarly, to the right by two bits is equivalent to division by 2^2 and so on. Conversely, shifting left acts as a multiplication by $2^1, 2^2,$ and so on. Shifting is also used for displacing the position of binary point, which is usually needed in addition, and multiplication operations. The shift to the right is denoted by >> and to the left by << symbol. If *x* is total number of shifts in a $Q(m,n)$ fixed point number, we have

$$\mathrm{Q(m,n)} \gg \mathrm{x} = \mathrm{Q(m-x,}n+\mathrm{x)}$$
$$Q(m,n) \ll x = Q(m+x,n-x)$$

Addition and Subtraction

In case the operands are of the same fixed-point data types, addition and subtraction operation are carried out just like integers. For example, the two fixed point numbers $Q(m1,n1)$ and $Q(m2,n2)$ has a correct result on a condition that $m1 = m2\ ;\ n1 = n2$. But, if the operands are of different data types, the variable having larger number of fractional bits is shifted to right by $n_{Larger} - n_{Smaller}$ bits to move its decimal place to align the binary points [16]. As an example, if we have to add two numbers $Q(0,7)$ and $Q(4,3)$, the $Q(0,7)$ number needs to be converted into the $Q(4,3)$ format by right shifting it 4 bits and sign extending it. Then, the addition operation can be done while keeping in mind that the operation does not overflow.

Multiplication

In multiplication, one needs to consider that the result of the operation requires a temporary storage of twice the size of the operands (assuming both operands have same storage size) so that there will be no loss of bits. The result then needs to be chopped to fit into the storage of the operands. If both the operands are of same Q-format, both the integer and fractional part have twice as much length in the temporary result. For the correct result and alignment of radix point, a right shift by the number of fractional bits is done. Rounding can be combined along with right shift to gain more accuracy.

Fixed-point additions and subtractions are performed by integer operation in a straightforward manner. For example, if we add two 16-bit numbers ($Q15$ numbers), the result will also be a $Q15$ number. But, in case of fixed-point multiplication, if we multiply two $Q15$ numbers, the result will be a $Q30$ number with two sign bit and 30 fractional bits. The extra sign bit in the result is known as a *sign extension bit*. This is further clarified by an example mentioned below;

Let us assume, we have to multiply 0.5 with 0.25. In $Q15$ format,

- 0.5 is represented as (0.5*2^15) = 16384 (decimal representation)
- 0.25 is represented as (0.25*2^15) = 8192 (decimal representation)
- On multiplication, the product is 134217728 (decimal representation)

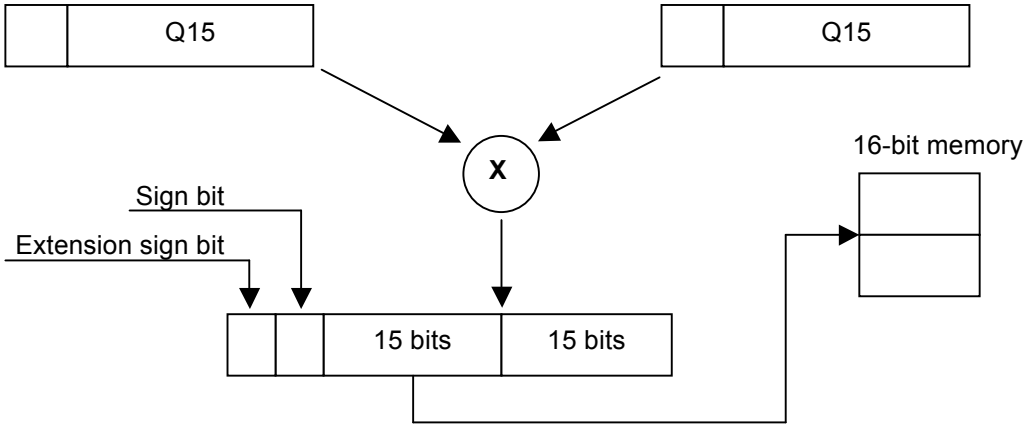

Figure 3.5. *Multiplication of two* ***Q15*** *numbers showing an extra sign extension bit*

The product is not a $Q15$ number as the number of bits required is more than 16. Our anticipated result is 0.125 i.e. 4096 in $Q15$ format. The result is in fact 0.125 times 2^{30}. In order to keep the same Q format of the result, we need to right shift the result by 15 bits (i.e. dividing by 2^{15}). Right shifting the result by 15 bits (division by 2^{15}) produces (134217728 / 2^{15}) = 4096, which is $Q15$ notation for 0.125.

3.2 Floating-Point Number Representation

DSPs generally need a large dynamic range to represent computation results. One way to accomplish this dynamicity is to use a large number of bits to represent the largest and smallest numbers. This can waste memory, if a wide range remains unused. To access a large memory area, the processing speed becomes slow. Large memory areas also increase the silicon size in a system [11]. The other way to achieve dynamicity is by using floating-point numbers, which introduce an exponent in the representation. The exponent increases the dynamic range that makes a very large and a very small numbers representable. The distance between two successive numbers (quantization step) does not remain as in the case of fixed-point number and it changes according to the exponent. The quantization step is the same for a number having the same exponent. The term floating point refers to the fact that the binary number can ‘float’, not like fixed-point where the binary point is fixed. The binary point can be placed anywhere and it changes with the exponent value.

The mantissa part of a floating-point number determines accuracy and the exponent part determines dynamicity. The accuracy increases with the increment of number of bits in mantissa part. On the other hand, increasing the number of bits in the exponent field will increase the dynamic range. Therefore, floating point number can be adjusted accordingly. The drawback of floating point number system is that, every floating-point operation requires more clock cycles than fixed-point operation [11]. The processor generally includes specialized hardware (FPU-floating point unit) that performs floating-point arithmetic.

A floating-point number n can be represented as follows

$$n = -1^{S} m.\, b^{e}$$

where *S* is the sign of the number, *m* is the mantissa, b is base of the floating-point system and e is exponent. The mantissa can be normalized as, $1 \leq m < b$. For a binary number, this determines the range of mantissa between [0.5, 1] on the positive side and between [-1, -0.5] on the negative side. To store a floating-point number, we need $x_m + x_e + 1$ bits, where, x_m is total number of bits in mantissa field, x_e is total number of bits in exponent field and an additional bit is required for a sign bit. A basic floating-point storage format in memory is shown below.

Sign (S)	Exponent Field (e)	Mantissa field (m)

Although, there are several floating-point representations that have been used in computers, the most commonly used representation is defined by IEEE 754 standard. Four different floating-point formats are defined in this standard and are mentioned below;

- Basic single precision floating-point
- Extended single precision floating-point
- Basic double precision floating-point
- Extended double precision floating-point

The number of bits in mantissa and exponent part is different in this format as shown in table 3.2.

Table 3.2. *IEEE floating-point standards [11]*

Parameter	Basic Single format	Extended Single format	Basic Double format	Extended Double format
Format width (bit)	32	43	64	79
Mantissa width (bit)	23	31	52	63
Exponent width (bit)	8	11	11	15
Maximum exponent	+127	+1023	+1023	+16383
Minimum exponent	-128	-1024	-1024	-16384

3.3 Fixed-Point Processors versus Floating-Point Processors

Fixed-point processors are used in high volume applications. They are comparatively less expensive as compared to its floating-point counterpart due to large scale of manufacturing. To compensate quantization noise, fixed-point arithmetic requires greater manipulation in algorithms. Although the development cost is higher for fixed point due

to difficult algorithm implementation, the final product will be cheaper. Moreover, the fixed-point implementation on a DSP allows lower power consumption, and smaller size on chip (reduced hardware complexity of fixed point circuit). Therefore, fixed point DSPs are used for high-volume general-purpose applications.

On the other hand, floating point DSP is optimized for computationally intensive and generalized tasks. Since the floating point has large dynamic range, there is practically no limitation on dynamic range for floating point designs. Floating point code development is less architecture dependent as well as high-level language friendly. Therefore, floating point DSP have cheaper and quicker development time than fixed-point DSP, however the final product cost is expensive (more complexity in silicon and also has wider buses to implement in design).

Hence, lower cost and higher speed of computation are trade off against added design effort for algorithm implementation in fixed-point algorithm. In the reverse manner, the ease of development process is trade off against the higher cost and hardware complexity in floating point applications.

4. FRAMEWORK AND TOOLS

Tensilica is a company founded in 1997 in Santa Clara, California based on semiconductor intellectual property core business, and is now part of Cadence Design System. Tensilica designed the first configurable and extensible processor core to address application specific microprocessor cores and software development tools. To implement the DCT/DST codes in Tensilica Processors, the host of tools provided by Tensilica's Xtensa Environment is used [17]. A detailed description about the Xtensa tools and Processors are discussed in the following sections.

4.1 Xtensa Xplorer Integrated Development Environment

Xtensa Xplorer IDE tool is a graphical user interface (GUI) design environment targeted for SoC modeling and software development for Tensilica processors. It provides software and hardware developers a common development tool to design Xtensa processor based systems. The Xplorer incorporates processor customization, software development and multi-processor SoC architecture tools, all together in a one common design environment. Xplorer is useful for the development of Tensilica Instruction Extension (TIE) [18], a Verilog like language used for custom instruction extensions to Xtensa Processors. The IDE is fully integrated with Xtensa Software Developer's Toolkit [19], where a developer can profile an application C-code, identify problems in the code and according to the necessity, make adjustment in the custom processor to speedup that code. Different features of Tensilica Processors can be added or removed to customize it according to the requirement of the SoC designer.

4.1.1 Processor Configurations

The Xtensa Xplorer tool provides different kinds of processor configurations options to use from the list for a particular application code. The processor configuration defines the type of Tensilica Processor. These configurations specifications can be either already built (and installed) in the Xtensa tool known as configuration build or that has not been built, simply known as configurations inside the Xplorer tool [17]. The Xtensa tools and configuration build are platform specific (Windows or Linux). Using a software configuration build, the Xtensa tools are adapted to a particular processor configuration. There might be one installation of Xtensa tools but many configurations of Tensilica processors. The target processor is selected using environment variables.

Using a configuration build, the Xtensa tools are adapted to a particular processor configuration. The Tensilica's Diamond Processor configuration builds are pre-build

inside the Xplorer tool and cannot be modified by a software developer (i.e. cannot be further configured or extended). While, on the other hand, Tensilica's Xtensa Processors can be configured and extended by using TIE as per requirements. Same technology is used for Tensilica's Diamond Processor builds as Xtensa Processor builds but the flow with which they are manipulated is more restrictive as they have predefined nature. Recent versions of Diamond Processor configuration builds are always included in Tensilica Xplorer tool, which are built for little-endian versions of the Diamond Processors. These configuration builds are used for evaluating Tensilica Processors as well as for developing software for a chosen processor.

4.1.2 Perspectives and Views

The Xplorer workbench can be dynamically rearranged according to different tasks such as editing, profiling or debugging any application code. A particular arrangement of the workbench interface to suit some set of tasks is known as 'Perspectives'. The 'Views' provides navigation of information in the workbench [17]. So, a perspective depicts how certain views are arranged, what kind of menus and set of toolbars are available and where the editor area is located inside the workbench. The Xplorer has many standard perspectives, which can be modified. Nevertheless, it is also possible to create and modify own perspectives. The key perspectives for Xtensa C/C++ project development are:

C/C++ editing Perspective: This Perspective is used for creating, editing and compiling any C/C++ projects, Xtensa configuration and tensilica instruction extension files. The Perspective has a View named 'Project Explorer', which displays C/C++ projects and its related files. Similarly, the View 'System Overview' displays various Xtensa configurations.

Debug Perspective: This Perspective shows a group of Views and a source code editor to debug program with Xtensa Xplorer. The views help to control the execution of the program by suspending or resuming the program, adding breakpoints, examining contents of memory and register etc.

Benchmark Perspective: The Benchmark Perspective is the main perspective to view the profiling results (Profile View) of the executed C/C++ application.

In Xplorer, there is a collection of 'Active set toolbar' consisting an active project, active configuration and active target. These toolbars display active set and use of them is the easiest way to build, run, profile and debug an application. Any task of building, running, profiling and debugging will be done for those chosen active sets.

The Benchmark Perspective and C/C++ Perspective along with Active set toolbar of the Xtensa workbench are shown in the figures below. In Figures 4.1 and 4.2, the P:

DCT_Fixed is the active project to be built, C: DE_570T is selected as an active configuration and T: Debug is a target toolbar option to select the target for which to build.

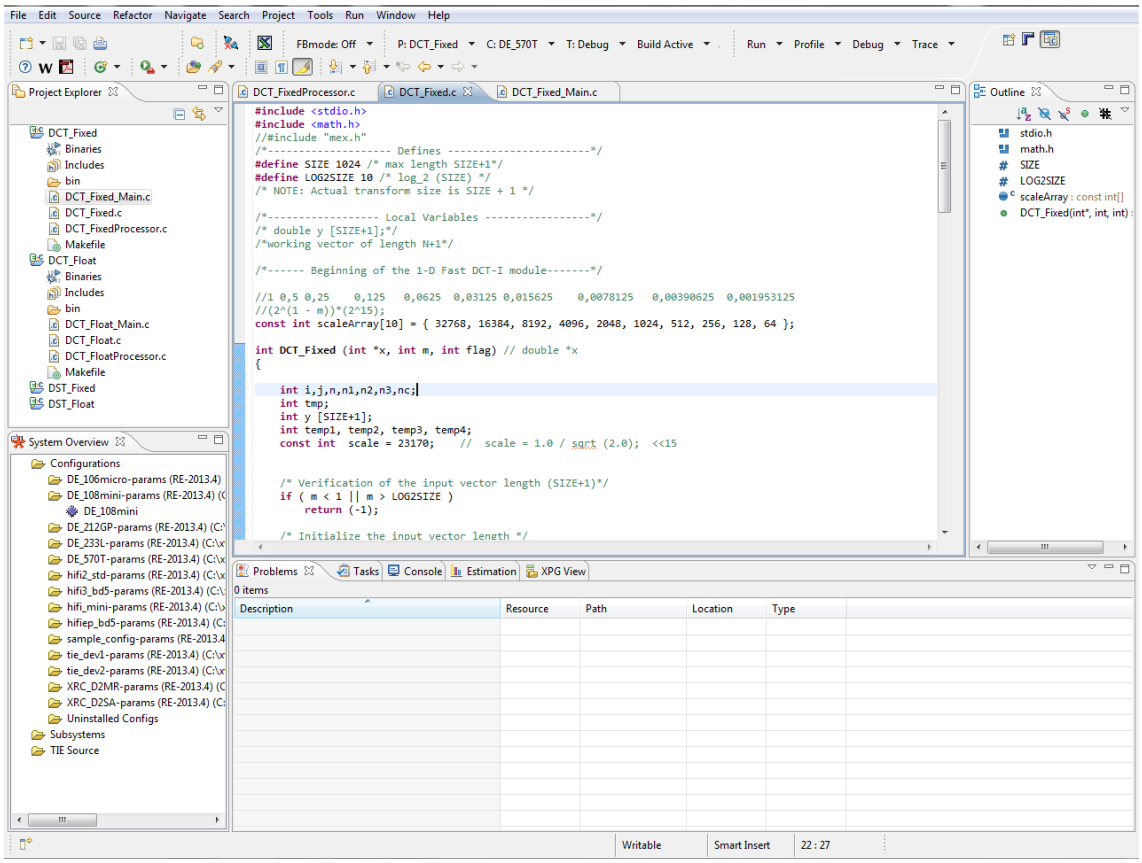

Figure 4.1. *C/C++ Perspective layout showing Active Set Toolbar of Xtensa Workbench*

4.1.3 Profile View

The Xtensa tools have numerous capabilities for profiling and benchmarking of various application behaviors [17]. The profiling task is to run the program using an appropriate launch that regulates execution and collects profile data. The profiling task also consists of navigating and analyzing of those profiled data using controls and views in the Benchmark perspective.

The cycle-accurate Instruction Set Simulator (ISS) included in Xtensa tool is used for profiling and can trace program execution at the lowest level. In addition to the cycle count profile, the ISS has other uses like collection of data on cache behavior and pipeline bubble; however this topic is out of the scope for the thesis. The Profile toolbar is used to launch the C/C++ project. After completion of profiling run, Xplorer will open the Benchmark Perspective to display the profiling results.

The Profile View inside Benchmark Perspective displays profile information of various functions in C/C++ program. The Profile View displays information up to twelve columns. Some of them are listed below.

- Function name: It displays the name of functions in the programs.
- Total (%): It displays the percentage of total profile count spent in executing this function.
- Function: It displays the clock cycle count only for this function.
- Children: It displays the total cycles spent in executing the functions called by this function plus the functions called by those functions.
- Total: It displays the total sum of both Function and Children results.
- Called: It displays the total number of times this function was invoked.
- Size (bytes): It displays the text size of the function.

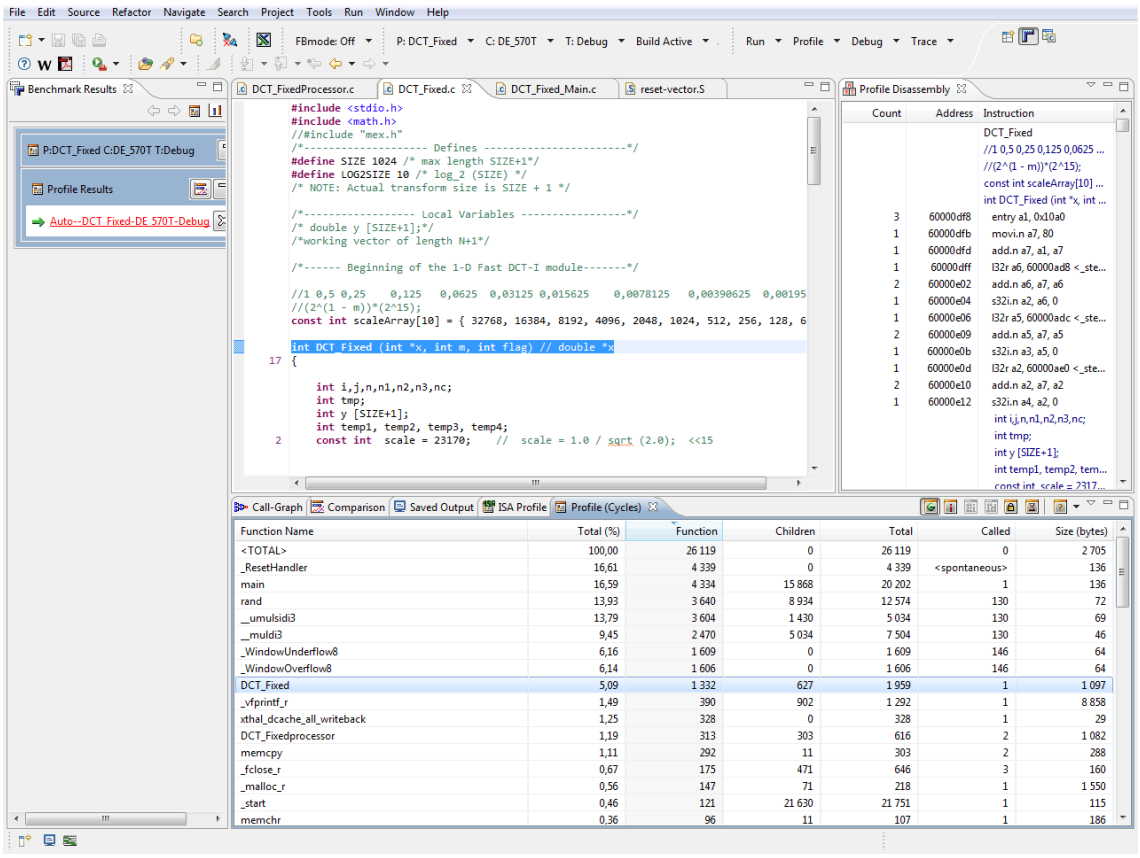

Figure 4.2. *Benchmark Perspective Layout showing the Profile View*

4.2 Processor Templates

There are mainly two sets of families of Tensilica processors, namely Diamond Standard Processors and Xtensa Processors. Both of the families of processors are described briefly in the following sections.

4.2.1 Diamond Standard Processors

The Diamond standard processor cores are preconfigured as 32- bit microprocessor and DSP Intellectual Property (IP) cores [20]. The basis of all Diamond standard processor cores is Tensilica's Xtensa Instruction Set Architecture (ISA) [21].

The Diamond Standard Processor core family comprises of three general-purpose controller cores, a Linux- compatible CPU core, a superscalar CPU core, an audio processor core and a DSP core. Figure 4.3 illustrates the performance of some of the Diamond Standard controllers/CPU in Dhrystone MIPS/MHZ and area consumed by those cores. Dhrystone is a computation benchmark representative of an integer processor performance [22]. The next section describes some of the Diamond processors briefly.

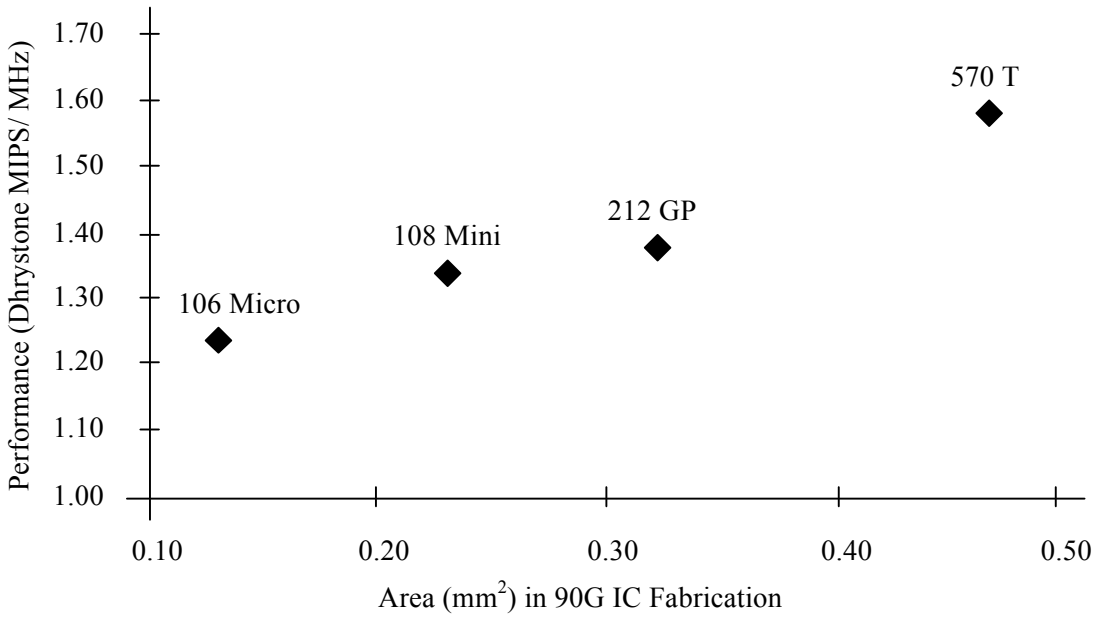

Figure 4.3. *Schematic diagram showing the performance of some of the Diamond Standard controllers/CPU in Dhrystone MIPS/MHZ against the area (mm^2) consumed by those cores [22].*

The Diamond Standard 106Micro Controller Core

The Diamond Standard 106Micro Controller Core is the smallest 32-bit RISC core among all the Diamond processor cores [23]. It has the smallest die area as well as lowest power consumption among the 32-bit Diamond processor family. It is a cache-less controller core and uses a 5-stage pipeline. Modeless switching between 24 and 16-bit instructions allows a good code density. To enhance performance of arithmetic and DSP code, the controller core has 32*32-bit multiplier. Furthermore, it consists of a 16-entry general-purpose register files known as AR register file to minimize area. The performance of 106Micro controller core is measured at 1.22 Dhrystone/MHz.

The Diamond Standard 108Mini Controller Core

The Diamond Standard 108Mini Controller Core is small cache-less fully synthesizable 32-bit RISC core [24]. Although 108Mini Core is small in area, it achieves high performance of 1.34 Dhrystone MIPS/MHz. Moreover, it is more useful for DSP application because of its integrated 32-bit integer divider along with 32*32-bit multiplier. The 32-entry general-purpose register file, with a 16-entry register window, facilitates fast context switching. It has a local, single cycle instruction memory interface and two local data memory interfaces.

The Diamond Standard 212GP Controller Core

The Diamond Standard 212GP Controller Core is a mid-range 32-bit RISC core of diamond family designed for high performance [20]. Similar to the Diamond 108Mini Controller core, the Diamond 212GP controller has interfaces for local instruction and data memories. In addition to that, it also has a cache controller that uses 8-Kbyte, 2-way set-associative instruction and data caches, efficient for large programs. The processor core itself has arithmetic and DSP hardware support, minimizing the need of individual DSP in the system. This includes a single cycle, 16*16 MAC unit adding four 32-bit registers and a 40-bit accumulator for DSP support. Additionally, a 32*32 bit multiplier and 32-bit integer divider is provided for arithmetic support. The performance of the 212GP core is measured at 1.38 Dhrystone MIPS/ MHz.

The Diamond Standard 570T Static-Superscalar CPU Core

The Diamond Standard 570T CPU core is one of the highest performance licensable processor cores available in market [20]. It combines 3-issue very long instruction word (VLIW) with 5-stage pipeline to provide high performance for both control and DSP code. The 16- bit and 24-bit instruction can be intermixed with 64-bit VLIW instruction in instruction stream to enhance performance with small code size. The Xtensa C/C++ compiler can create 64-bit VLIW instructions, if instructions can be issued simultaneously; otherwise, it selects 24 or 16-bit instructions, which is very effective to reduce the amount of memory required to store instruction. The Diamond 570T core has a 32*32- bit multiplier in addition to 32-bit integer divider. Furthermore, it has a single cycle 16*16-bit MAC unit. The 570T processor core can achieve high performance of 1.59 Dhrystone MIPS/MHz. Table 4.1 shows the memory types and sizes of Diamond standard processor cores discussed above.

Table 4.1. *Memory types and sizes of Diamond Standard Processor cores*

Memory Type (KB)	106Micro	108Mini	212GP	570T
Local Instruction RAM)	1-128	1-128	0-128	0-128
Local Data RAM0	0-128	0-128	0-128	0-128
Local Data RAM1	N/A	0-128	N/A	N/A
Instruction Cache	N/A	N/A	8	16
Data Cache	N/A	N/A	8	16

The following section briefly describes some of the features of Xtensa processors that are related to this thesis work.

4.2.2 Xtensa Processors

The Xtensa instruction set architecture (ISA) is the basis for Tensilica's Xtensa Processors. The 32-bit base architecture features modeless switching between 16- and 24-bit instruction set for maximum performance. The base architecture has 80 RISC instructions. There is a 32-bit Arithmetic Logic Unit (ALU) and maximum 64 general-purpose 32-bit registers along with six special-purpose registers.

The major difference of Xtensa Processor with Diamond processor is the fact that Xtensa Processors are customizable and can be configured with the integrated tool chain whereas Diamond Processor are pre-configured and cannot be extended or customized later. However, both processor families share same base architecture i.e. Xtensa ISA.

Tensilica HiFi Audio DSP

The HiFi Audio DSPs are add-on audio extension package for Xtensa LX processor [25]. HiFi mini is the smallest and lowest power audio core provided by Tensilica [26]. The HiFi 2 provides a low power MP3 audio processing and is accompanied by HiFi Extended Precision (HiFi EP) for further optimizations and improved performance [27]. The HiFi 3 offers higher performance of audio or voice processing than the rest [28]. Some of the features of various versions of HiFi audio DSPs are listed in Table 4.2.

Table 4.2. *Comparison of features of the different HiFi audio DSP [29]*

Architectural Comparison	HiFi mini	HiFi 2	HiFi EP	HIFi 3
Architecture (bits)	24	24	24	32
VLIW slots	2	2	2	3
MACs	Dual MACs 24*24 32*16	Dual MACs 24*24 32*16	3 MACs 24*24 32*16 32*24	4 MACs operating as: 4 24*24 or 4 32*16 or 2 32*32 or 2 32*24
Load/Store (bit)	64	64	64	64

Tensilica ConnX D2 DSP Engine

The ConnX D2 is an add-on option DSP engine for Xtensa LX processor [30]. The addition of ConnX D2 DSP to Xtensa LX core further adds dual 16-bit MACs and 8-entry 40-bit register file to its base RISC architecture. The ConnX D2 DSP supports various data types including 16, 32, and 40-bit integers; 16, 32, and 40-bit fixed points; 16-bit complex; 8 and 16-bit vectors. It employs two-way SIMD (single instruction, multiple data) instructions, 64-bit VLIW and a 5-stage pipeline.

5. SOFTWARE DESIGN AND IMPLEMENTATION

In this chapter, the design process to develop fixed-point C-code that is based on the technical background provided in the previous chapters is presented. The DSP algorithms are dependent on repeated number of multiplications and additions, so the designer needs to carefully consider the possibility of overflow and underflow after each arithmetic operation. The programmer has to understand the accumulation of quantization errors, signal levels during intermediate computations and needs to remember the scaling throughout the program. Several factors need to be pondered upon while converting floating-point codes to fixed-point domain. A comprehensive knowledge about the input data and the code flow is very important.

The starting point of the code development process is to find the appropriate reference floating point C-code for DCT/DST implementation. For this thesis work, the reference floating point C-code are taken from 'The Transform and Data Compression Handbook' by K.R. Rao et al. [5]. The choice of appropriate fixed-point format is crucial part in fixed-point design. If the fixed-point code output values are achieved with optimum accuracy, the code design process is completed.

5.1 Fixed-Point Code Design

For the fixed-point code design, the Q15 format, i.e. fractional-fixed point format is chosen for the reasonable accuracy. In that case, the input data has to be normalized in the range of [-1, 1). Multiplication overflows are handled by using fractional fixed-point format. Numerous code optimization strategies have to be considered to decrease the execution time e.g. substitution of calculations with look-up tables, and avoiding iterative divisions by computing the inverse factor and substituting with equivalent multiplication. Furthermore, divisions and multiplications can be replaced with bit-wise right shifting and left shifting respectively, whenever possible. Although there are many other optimization strategies, some basic strategies are used in this code development process.

Before the design of fixed point DCT/DST C-code, random input numbers were generated between [-1, 1] according to different DCT/DST length. The designed fixed point DCT-I and DST-I codes works for maximum transform size of N=129 and 127 respectively. Although, by adding the numbers of cosine coefficients in the defined cosine array of the code, it works for required transform. The generated inputs were first multiplied by 2^{15} and then rounded to nearest integer value to represent them in C-code, such that they can be used as Q15 format in C-code. Tensilica's 32-bit processors are considered as the target processor for fixed point DCT/DST code.

Following steps are taken to design the fixed point DCT/DST C-code:

1) All floating-point variables, which were represented as double data types in floating point DCT/DST C-code, are changed to integer data types. Since, fixed-point format represents numbers as integer data types.
2) The scaling functions used in DCT/DST code were changed to Q15 format and hard-coded as constant in the C-code.
   For example, scale = 1.0 / sqrt (2.0); is changed to Q15 format by multiplying with 2^{15}. So it is defined as
   const int scale = 23170; in C-code.
3) The cosine coefficients were calculated from reference floating point code manually. Those coefficients were changed to Q15 numbers and represented in a cosine coefficient array in the fixed point DCT/DST code.
4) As defined earlier in the fixed-point multiplication section, the resulting variables after multiplication between two operands were shifted right by 15 bits. This means dividing by 2^{15} after every multiplication such that the resulting value would remain in Q15 format.
5) As part of DCT-I/DST-I computation code, there is a function to compute DCT-III transform within the code. Before passing the input values to the DCT-III transform function, the input values need to be scaled down. This is necessary to control the overflow caused by repetitive addition/subtraction and multiplication inside the loop. By carefully examining the structure and output of the code, it was found out that the scaling that needs to be done is directly proportional to m (where, N= $2^m + 1\ and\ 2^m - 1$ for DCT-I and DST-I respectively (N is the transform size of DCT/DST). So, the input values that need to be passed in DCT-III function were divided by 2^m, which means right shifting the values by m bits. However, this shift increases quantization noise in the output.
6) The final part of the fixed-point code development process is to measure the fixed-point code output against the reference floating-point output with same set of input parameters. Therefore, MATLAB tool was used to find the output error in the designed fixed-point code and to calculate the signal to noise ratio for different sets of DCT/DST length. The flow chart in Figure. 5.1 explain the overall fixed-point code development process.

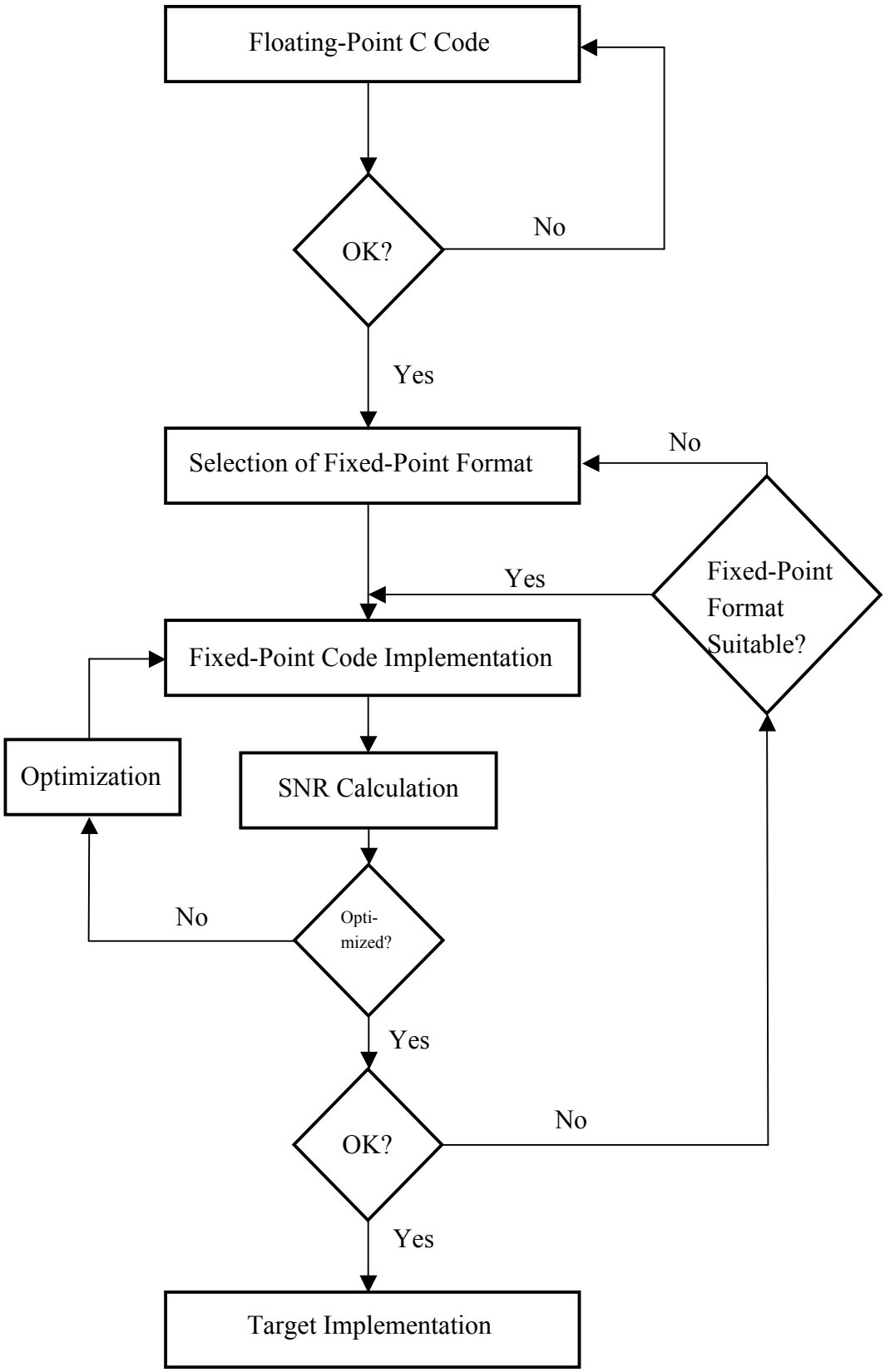

Figure 5.1. *Schematic flowchart diagram showing the fixed-point C-code design flow process.*

5.2 C/MEX Function

In this thesis, the Signal-to-Noise Ratio (SNR) is calculated in MATLAB. So, in order to integrate MATLAB and DCT and DST C-code, the C/MEX support of MATLAB was used [31]. A part of software design is to write a MEX C-code to invoke C-code from MATLAB. A brief discussion about how to use C/MEX function to call C-code is presented here.

MATLAB Executable (MEX) external interface function, or briefly a "MEX-function" allows compiling a C/C++ code, so that it can be called from MATLAB. MEX intact the high performance of C/C++ code while still working inside the MATLAB environment.

5.2.1 Using MEX File to Call C File

Mex function on the MATLAB command line is compiled using mex command as

mex *myfunc*.c *myfunc_mex*.c .

This command indicates, a mex file is required for every C file that needs to be compiled from MATLAB. After compiling C-code with mex command, we get a MEX binary, which then can be called by MATLAB like any other m-functions in MATLAB. The steps defined for a MEX file are; gateway functions creation, data structures declaration, Inputs and output management, Input Validation, Allocate and Free Memory, Data Manipulation, Displaying Messages to User and Error Handling [31].

5.2.2 MEX Files and MATLAB Interface

Figure 5.2 explains the interface between MEX files, C files and MATLAB. The *myfunc*.c file is compiled along with *myfunc_mex*.c by defined compiler using the 'mex' command as written in the previous section.

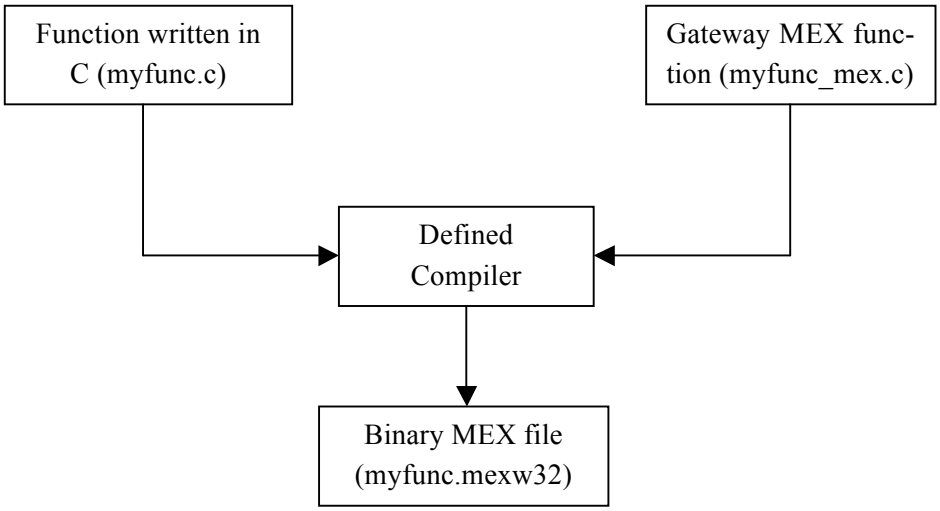

Figure 5.2. *Schematic diagram showing the interface between MEX files, C files and Gateway MEX function for MEX File Generation [31]*

The command generates a binary file named *myfunc.mexw*32 or *myfunc.mexw*64 depending on the operating system used (32 bit or 64 bit). The resulting binary MEX file can be called from MATLAB in the same way as MATLAB function.

Figure 5.3 shows the mechanism of calling the binary file from MATLAB. When the binary function is called, the gateway function first passes the parameters after executing its check routines to the given C function. After computation, the results are placed in the output vectors generated by gateway function and then they are available in MATLAB.

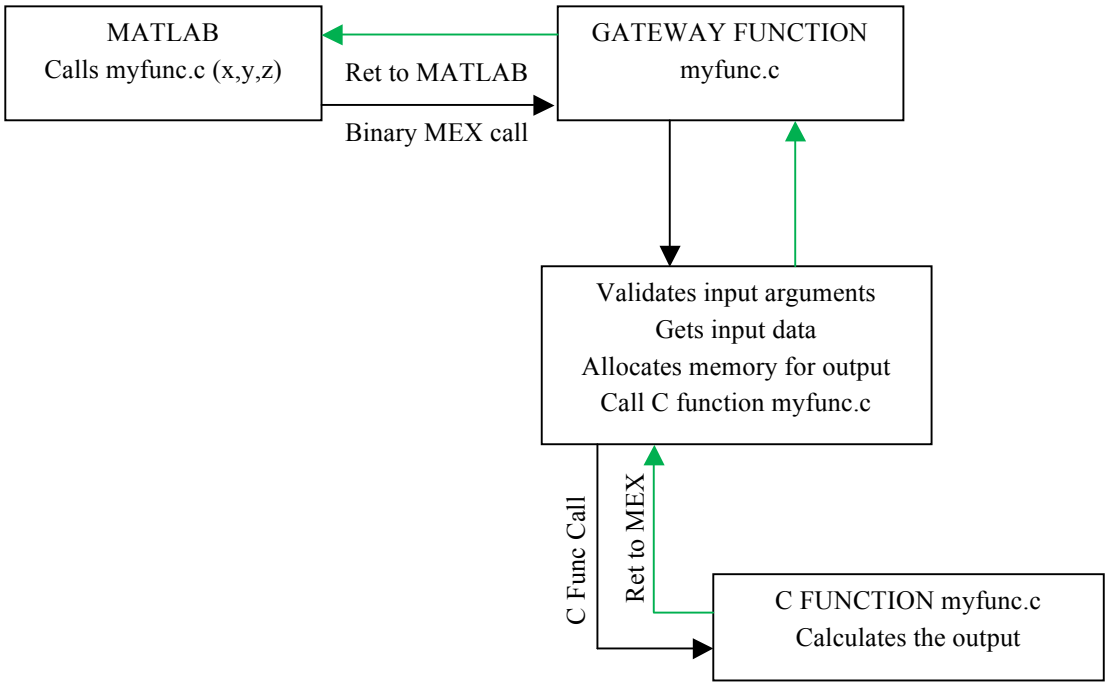

Figure 5.3. *Schematic diagram showing the mechanism of calling the binary file from MATLAB [31].*

5.3 Profiling the DCT/DST Code in Xtensa Environment

To analyze the performance (the number of cycles) of the fixed point DCT/DST code compared to the floating point, both codes are profiled in Tensilica's Xtensa Xplorer environment.

The first task is to create Xtensa C projects for sine and cosine transforms. The Xplorer has a C/C++ editor, which is used for creating and editing C/C++ code. The necessary codes are added as a C file in the project. The project is chosen along with necessary processor configuration option from the "Active Set" toolbar in Xplorer tool. The floating point DCT/DST code and fixed-point DCT/DST code are profiled to evaluate the performance of the sine and cosine transforms code in the targeted processor configurations. In the Benchmark Perspective, the profiling results are obtained where the total numbers of clock cycles taken to execute the codes are displayed.

Processor Configuration Summary

While running both the floating-point and fixed-point DCT/DST codes in target Tensilica processor builds, the entire base processor configuration settings that are provided by the Xtensa tool are retained. Since, the work is to evaluate the performance of both floating-point DCT/DST codes and the designed fixed-point DCT/DST code on Tensillica processors, there are no custom modifications done in these processors. There are various processor configuration build already installed in the Xtensa tool associated with various Tensilica Processors. The processor configuration name along with the configuration summary is presented below [17].

The table 5.1 presents the name of Processor configurations that are already built in Xtensa Xplorer tool and the base Tensilica processors from which the processor configurations are generated in the Xtensa tool. The base processors presented in Table 5.1 are Tensilica's Diamond Standard processor cores.

Table 5.1. *Processor configuration names and base Tensilica processors*

Configuration Name	**Processor Core**
DE_106micro	Diamond Standard 106 Micro Controller
DE_108mini	Diamond Standard 108 Mini Controller
DE_212GP	Diamond Standard 212GP Controller core
DE_570T	Diamond Standard 570T Static-Superscalar CPU core

In Table 5.2, the implementation options for processor configuration are presented. The clock gating is used to minimize power consumption of the processor. The two levels of clock gating features of Tensilica processors are; global clock gating and functional unit clock gating, both of which are selected in this configuration.

Table 5.2. *Implementation options (For configuration DE_106 micro, DE_108mini, DE_212GP, DE_570T)*

Option	**Selection**
Global clock gating	Selected
Functional unit clock gating	Selected

Table 5.3 shows different arithmetic configuration options that are chosen for the processor configuration, which are described below.

The MUL32 option selects a standard 32-bit multiplier. The compiler selects this option whenever there is a need of multiplying signed or unsigned variables of integer, short or character type. Without MUL32 option or any other multiplication option, the compiler will add emulation code for all multiplications, thus multiplications are realized by using shift and adds. However, emulation takes more clock cycles than multiplier option. In this case a 16-bit multiplier is included but not any 32-bit multiplier, the compiler emulates 32-bit multiplication using 16-bit operations. There are two types of MUL32 options that can be selected; iterative implementation or fully pipelined imple-

mentation. By using iterative and non-pipeline hardware, the Iterative Implementation option creates a multiply instruction that implements 32*32-bit multiplication into a 32-bit product. Depending on the bit-pattern being multiplied, this instruction can take anywhere from 1 to 6 clock cycles. On the other hand, the fully pipelined implementation creates a multiply instruction that implements a 32*32-bit multiplication with fully pipelined hardware, where multiplication instruction takes two clock cycles. As we can see in the table, for the configuration DE_106Micro, the iterative implementation is selected, while for other configurations (DE_108mini, DE_212GP and DE_570T), the fully pipelined implementation is selected.

The MUL16 option selects a 16-bit multiplier for both signed and unsigned 16-bit multiplication. On configurations with only MUL16 option, the compiler selects this option whenever it can ascertain that a 16-bit multiplication is equivalent to a 32-bit multiplication. That is, whenever both operands are 16-bit or less or results are 16-bit or less. If it is not the case, the compiler emulates a 32-bit multiplication using this option. Depending on the bit pattern of values being multiplied, emulation takes approximately 10 clock cycles. The table clearly shows the MUL16 option is selected for all processor configurations presented above.

Another option presented in the table is 32-bit integer divider. This option has four instructions that are used to perform 32-bit integer division. Depending on the bit-pattern, these instructions may take 2 to 13 cycles. These division instructions are implemented using non-pipelined or iterative hardware, which means that instructions after division operation will not execute until the division operation is complete. The compiler infers the use of these instructions for all 8-, 16- and 32-bit integer divisions. Table 5.3 shows that 32-bit integer divider option is not selected for DE_106micro configuration but it is selected for the rest of configurations.

The next option in the table displays the MAC16 DSP instruction family (16-bit multiply/Accumulate (MAC) with 40-bit accumulator). This instruction series allows a 16-bit MAC into a 40-bit accumulator paralleled with two 16-bit updating loads. It allows a full iteration of 16-bit dot product on every cycle. The instructions in this family are specialized and are not inferred by the compiler. However, the compiler can infer use of MAC instruction that does not execute parallel with a load. This instruction is normally not faster than the MUL16 option. In configurations presented Table 5.3, the 16-bit MAC with 40-bit accumulator option is selected for DE_212GP and DE_570T but it is not selected in case of DE_106micro and DE_108mini.

Finally, in the last option, no floating-point accelerator or processor is selected for any of the configurations presented above. Without this option, floating-point operations are supported using emulations at speeds that are generally between 50 to 200 cycles per base floating-point operation.

Table 5.3. *Arithmetic options and selections in processor configuration*

Option	Selection			
	DE_106micro	**DE_108mini**	**DE_212GP**	**DE_570T**
MUL32	Iterative	Pipelined	Pipelined	Pipelined
MUL16	Selected	Selected	Selected	Selected
32bit Integer divider	Not selected	Selected	Selected	Selected
16bit MAC with 40bit Accumulator	Not selected	Not selected	Selected	Selected
Floating point (single+double) Coprocessor/ Accelerator	Not selected	Not selected	Not selected	Not selected

Table 5.4 presents Instruction Set Architecture (ISA) configuration options and selection for mentioned processor configurations. The numbers of physical registers known as AR resisters are initially 16, which are directly assessable by instructions in the ISA. However, Tensilica windowed Application Binary Interface (ABI) allows more physical AR registers than the 16. This allows faster and smaller code. Tensilica has 16, 32, or 64 physical registers. The choice of these registers is trade-off between application performance and hardware area. There are 16 numbers of AR registers in case of DE_106micro configuration while 32 AR registers are selected in remaining configurations.

Another option used is the Maximum Instruction Width option. Xtensa core instructions are two or three bytes wide. Tensilica provide modeless intermixing of multiple instruction sizes and all configuration support 24-bit instructions. The 16-bit instructions are used to save code size. Tensilica also supports designer defined Flexible Length Instruction extension (FLIX) for multi-issue Xtensa Very Long Instruction word (VLIW) cores. Those instructions are 32, 64, 96 or 128 bits. A configuration can have at most two of 32-, 64-, 96-, 128- bit instructions. These 32-, 64-, 96-, 128- bit instructions can be partitioned into custom slots and each of them can execute one of a set of operations. The maximum instruction width option can be set maximum to utilize larger instructions. Since DE_570T allows FLIX instructions, the maximum width of instruction is set to 8 bytes (64 bit). For the rest of configurations, it is 3 bytes (24 bit).

The base Tensilica processor has a 5-stage pipeline micro-architecture. A single stage is dedicated to data memory and another stage is dedicated to instruction fetch. For large local memories, the memory speed limit can limit the processor core speed. For those configurations, Tensilica has an option to add two extra stages to the pipeline (one for instruction fetch and another for data memory). As in the ISA configurations option table, the pipeline is 5-stage for all of the configurations.

Table 5.4. *ISA configuration options and selections in processor configuration*

Option	Selection			
	DE_106micro	**DE_108mini**	**DE_212GP**	**DE_570T**
Number of AR register for call windows	16	32	32	32
Maximum Instruction width (bytes)	3	3	3	8
Pipeline length	5	5	5	5

Table 5.5 presents the interface width options and their selection for the configurations. The instruction fetch width option controls the number of bits that are fetched in a cycle from the Instruction cache or local memory into holding buffers. This option can be set to 32, 64 or 128 bit. For FLIX instructions, these parameters need to set to 64 or 128 bits. In general, wider width gives higher performance at a higher area cost. We can see in the table, for DE_570T, which use FLIX instructions, the interface fetch width is set to 64-bit. For others, it is set to 32-bit.

Another option presented in the table is data cache or memory width option. This option controls the number of transferred bits from external memory into the cache per cycle. It also provides the option to control the number of bits that can be loaded or stored from the cache or local data memory every cycle. It is a maximum width of data for a load/store instruction. The DE_570T configuration has 64-bit of data memory/cache interface. For other configurations, it is set to 32-bit.

In the table, the next option is interface to instruction cache width, which is 0 for DE_106micro and DE_108mini configuration. For DE_212GP configuration, it is 32-bit and for DE_570T configuration, it has 64-bit width.

Table 5.5. *Interface width Options and selections in Processor configuration*

Option	Selection			
	DE_106micro	**DE_108mini**	**DE_212GP**	**DE_570T**
Width of instruction fetch Interface	32	32	32	64
Width of data memory/ cache interface	32	32	32	64
Width of interface to instruction cache	0	0	32	64

Table 5.6 presents the sizes of instruction cache and data caches in bytes for different configurations. For DE_106micro and DE_108mini, there is no instruction cache, i.e., the size is zero. For DE_212GP, it is set to 8192 for both instruction and data cache. Finally, for DE_570T, it has larger size than other which is set to 16384 for both types of cache.

Table 5.6. *Instruction/ data cache option and selection in processor configuration*

Option	Selection			
	DE_106micro	**DE_108mini**	**DE_212GP**	**DE_570T**
Instruction cache size (Bytes)	0	0	8192	16384
Data Cache (Bytes)	0	0	8192	16384

Table 5.7 shows the amount of RAM and ROM memories in the systems for the listed configurations. The ROM and RAM size is the same for all the configurations as seen in the table.

Table 5.7. *System memories options and selections in processor configuration*

Option	Selection			
	DE_106micro	**DE_108mini**	**DE_212GP**	**DE_570T**
System RAM (byte)	64M	64M	64M	64M
System ROM (byte)	16M	16M	16M	16M

Besides all of above mentioned configuration options, there is another option provided for number of Load/Store units. This option is set to 1 for all of the configurations that we have used.

6. ANALYSIS AND RESULTS

This chapter discusses the results of the thesis work. First, the signal to ratio (SNR) analysis is discussed to understand the efficiency of the designed fixed-point DCT/DST codes with respect to the reference floating-point DCT/DST codes. In the next section, the performance of fixed-point code in different Tensilica processor configuration is compared with the floating-point version.

6.1 Signal-to-Noise Ratio

Signal to noise ratio is one of the key factors to be analyzed in digital signal processing domain. Signal to Noise Ratio (SNR) is a measure of signal strength relative to background noise [11]. The ratio is generally measured in decibels (dB). The fixed-point code generation process introduces some quantization error (noise) in the output, which can be expressed in terms of SNR. The SNR illustrates the output signal compared with the error. High value of SNR indicates less noise in output signal and vice-versa. Before describing the SNR graphs for fixed-point DCT-I and DST-I code, the experimental methodology of SNR analysis is discussed.

The stimuli are randomly generated in MATLAB between [-1,1](using rand () function of MATLAB) according to the DCT and DST input length. Let us define the input data length N of DCT/DST transforms such that, there exist a variable m, where N=$(2^m + 1)$ for DCT-I and N=$((2^m - 1)$ for DST-I. So, for m=2, the DCT-I and DST-I input length N would be 5 and 3 respectively. Similarly, for m =3, its length would be 9 and 7 for DCT-I and DST-I respectively and so on. In this experiment, the SNR for m =2,3,…,7 are calculated as the code is also designed to calculate DCT-I/DST-I up to m =7.

It is important to pass the same set of quantized input vectors to fixed-point and floating-point transform codes. Therefore, the random numbers generated between [-1,1) are first stored in MATLAB as a .mat file. For quantization, the generated inputs are first multiplied by 2^{15} and then rounded to nearest integer value. Then, same set of inputs is fed to both versions of codes and outputs are again stored as .mat file in MATLAB. We need to consider the scaling done in the fixed-point code while storing the output from the fixed-point codes. As described in earlier section, the scaling depends on the value of m.

Let us consider the output from fixed-point DCT-I is dct_fixed_out and floating-point DCT-I is dct_float_out. Similarly, the result of fixed-point DST-I is defined as dst_fixed_out and floating-point DST-I as dst_float_out. Then, the noise or error in the fixed-point signal can be defined by

Noise_DCT= dct_float_out- dct_fixed_out; for fixed-point DCT-I

Noise_DST= dst_float_out- dst_fixed_out; for fixed-point DST-I

Now, the SNR for fixed-point DCT-I/DST-I can be calculated in MATLAB as

$$SNR_{DCT} = 20 * log_{10}(\sigma(dct\_fixed\_out)/\sigma(Noise\_DCT))$$

$$SNR_{DST} = 20 * log_{10}(\sigma(dst\_fixed\_out)/\sigma(Noise\_DST))$$

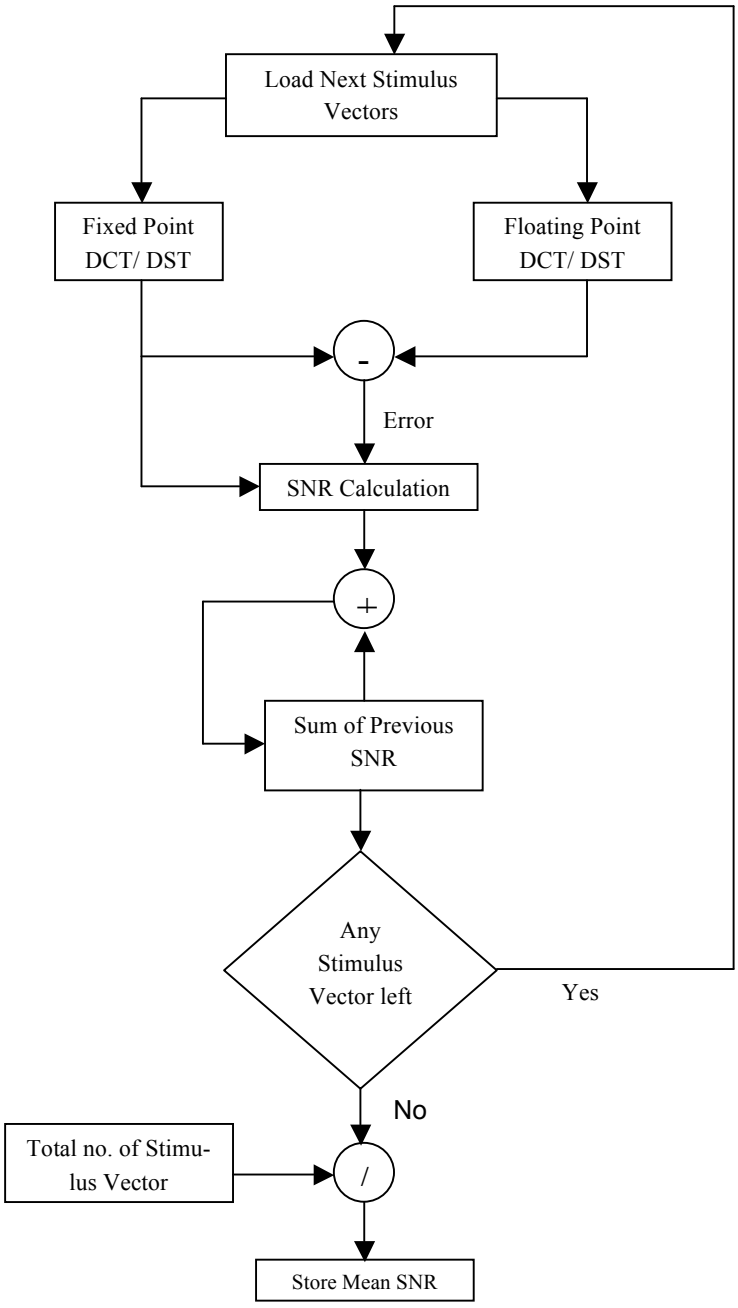

Figure 6.1. *A flowchart showing SNR calculation process*

All the variables used are stored in a structure array in MATLAB along with SNR. To calculate the average SNR, 50 different sets of random input stimuli are generated. Finally, the mean SNR is calculated by using *mean* function of MATLAB. The process of SNR calculation is illustrated in Fig. 6.1.

Figure 6.2 shows the SNR values of fixed-point DCT-I with reference to floating-point DCT-I according to the input length *N* of DCT-I where $N=(2^m + 1)$ as discussed earlier. The horizontal axis in the graph shows value of m and vertical axis shows the SNR value in dB.

As we can see, if we increase m, the SNR is decreasing accordingly. This is because to handle the overflow in fixed- point DCT code, we need to scale down the variables. The scaling done is indirectly proportional to the input length. With the increment in input length, variables in C-codes need to be scaled down more to compensate overflow. Therefore, the output has more noise with the increased input length, which is reflected in the SNR. As we can see in the figure 6.2, the maximum value of SNR is around 76dB for m =2 and minimum is for m = 7, where it is around 35dB.

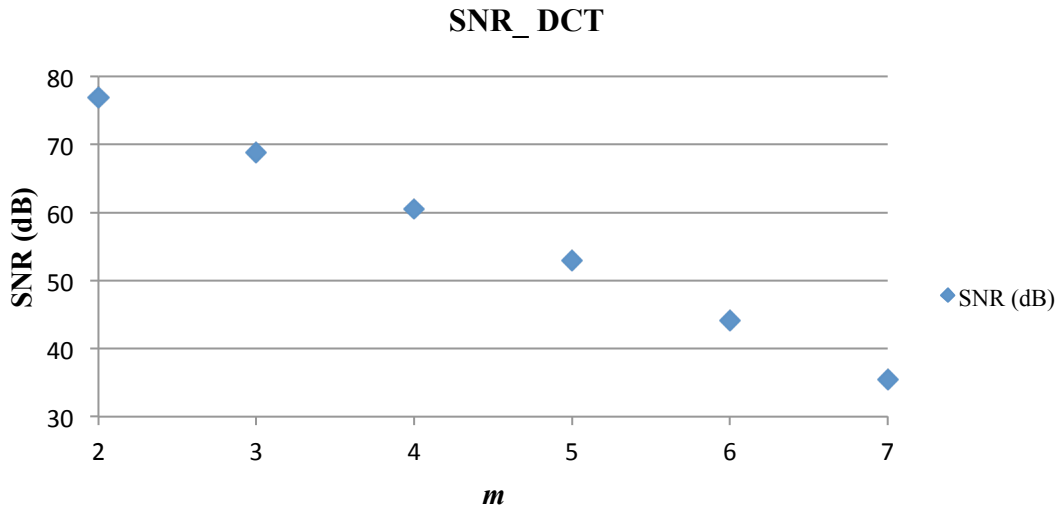

Figure 6.2. *SNR for fixed-point DCT of type 1*

Similarly, Fig. 6.3 demonstrates the value of m and SNR for DST-I. The horizontal and vertical axis of the graph represents values for m and SNR for DST-I respectively.

As expected, the results of DST-I are similar to that of the DCT-I. The DST-I length has similar effect on the SNR. As the length of input vector grows the SNR value declines. In this case, the maximum value of SNR is around 78dB for m =7 and minimum value is around 35dB for m = 2.

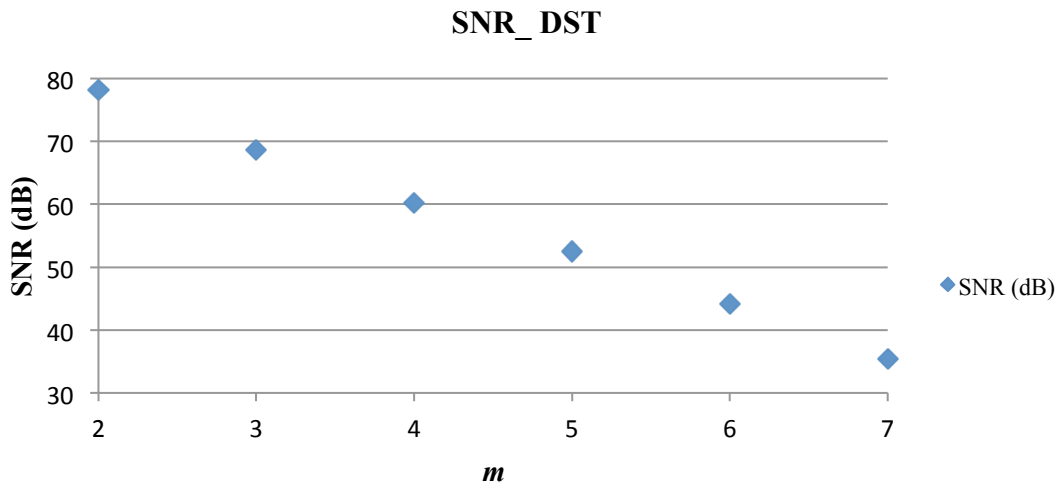

Figure 6.3. *SNR for fixed-point DST-I*

6.2 Performance on Tensilica Processors

While evaluating the performance of fixed-point and floating-point DCT/DST code, the configurations mentioned in the previous chapter are used. Both types of codes are profiled inside the Xtensa Xplorer IDE using Xtensa Instruction set simulator. The total number of clock cycles for both versions of DCT/DST codes in various Tensilica configurations is recorded. The results are discussed in the following sections.

Profiling results on different Processors

In the subsequent sections, DCT_Fixed and DST_Fixed represent the fixed-point DCT-I and DST-I code respectively. Similarly, DCT_Float and DST_Float define the floating-point DCT-I and DST-I code respectively. As in the previous section, m is related to length of DCT/DST such that the DCT-I length $N = 2^m + 1$ and DST-I length $N = 2^m - 1$.

Table 6.1 presents the recorded total number of clock cycles for the DCT-I/DST-I on DE_106micro configuration system. This includes both fixed-point and floating-point type of codes.

It can be seen that the configuration requires less clock cycles to execute the fixed-point DCT-I/DST-I than the floating-point code. Another observation is that the numbers of clock cycles are more than double than previous if we increase m by 1. The reason is doubling the transform size increases the number of arithmetic operations more than double and hence clock cycles.

The speedup in terms of clock cycles achieved by the fixed-point DCT/DST code system over the floating-point DCT/DST code system are calculated as

$$Performance\ gain = \frac{clock\ cycles\ of\ Float\ DCT/DST}{clock\ cycles\ of\ Fixed\ DCT/DST}$$

The speedup achieved by fixed-point DCT-I system over floating-point DCT-I system is 3.70, 4.69, 5.49, 5.94, 6.10 and 5.8 for m = 2,3,4,5,6 and 7 respectively. Simi-

larly, the speedup achieved by fixed-point DST-I over floating-point DST-I is 4.38, 4.91, 5.24, 5.29, 5.17 and 4.97 for m = 2, 3, 4, 5, 6 and 7 respectively. The speedup achieved by fixed-point code over floating-point code is due to the fact that, in floating-point code all mathematical operations are performed by emulation library, which has substantial computational overhead. For example, all cosines and sine coefficient functions that consume more time in the floating-point code are replaced with integer array of cosine/sine coefficients in fixed-point code. Similarly, other scaling functions are also hardcoded as constants. Moreover, the fixed-point code uses integer operation and takes lesser number of clock cycles. The slight variance on speedup is due to variance on arithmetic operations for different transform size.

Table 6.1. *Recorded clock cycles for configuration: DE_106micro*

m	Total Clock Cycles			
	DCT_Fixed	**DCT_Float**	**DST_Fixed**	**DST_Float**
2	2136	7906	1196	5244
3	4517	21176	3371	16575
4	9573	52531	8238	43187
5	20636	122562	19126	101222
6	44970	274552	43374	224439
7	98540	571446	96964	482185

Table 6.2 displays the number of clock cycles for both the fixed- and floating-point DCT/DST-I codes on DE_108mini configuration. As in the previous configuration system, the result shows the fixed-point system has lesser clock cycles than the floating-point system. Furthermore, the system takes fewer clock cycles comparing with the DE_106micro configuration system. Although being similar in terms of configuration with DE_106micro, the pipelined implementation of multiplication on DE_108mini resulted in slightly fewer clock cycles. The speedups are 3.45, 4.46, 5.29, 5.76, 5.94, 5.63 and 3.92, 4.60, 5.00, 5.09, 5.00, 4.81 for the fixed-point DCT-I and DST-I over floating-point DCT-I and DST-I for m = 2, 3, 4, 5, 6 and 7 respectively.

Table 6.2. *Recorded clock cycles for Configuration: DE_108mini*

m	Total Clock Cycles			
	DCT_Fixed	**DCT_Float**	**DST_Fixed**	**DST_Float**
2	2120	7325	1205	4719
3	4460	19901	3337	15341
4	9417	49816	8092	40488
5	20281	116888	18766	95525
6	44218	262815	42537	212598
7	97002	545775	95203	457747

The recorded clock cycle counts on the DE_570T configuration for both types of DCT/DST-I are presented in Table 6.3. This configuration has the fewest clock cycles recorded among all other configuration. The presence of 16-bit MAC with 40-bit accumulator option in this configuration as well cache, which was missing in previous configurations is the measure reason for the speed boost up. Moreover, the presence of widest bus in this configuration also helped to save some cycles as more data can be fetched at a time for computation. In this configuration, the speedup of fixed-point DCT-I system over floating-point DCT-I are 3.26, 4.14, 4.83, 5.20, 5.34, 5.01 and fixed-point DST-I system over floating-point DST-I are 3.59, 4.23, 4.56, 4.60, 4.51, 4.33 for for m = 2, 3, 4, 5, 6 and 7 respectively.

Table 6.3. *Recorded clock cycles for Configuration: DE_570T*

m	Total Clock Cycles			
	DCT_Fixed	**DCT_Float**	**DST_Fixed**	**DST_Float**
2	1959	6387	1124	4036
3	4217	17456	3169	13391
4	9081	43852	7826	35664
5	19820	103161	18382	84635
6	43550	232399	41993	189225
7	95929	481063	94419	409014

The clock cycles counts recorded on the DE_212GP configuration for both types of codes are presented in the table 6.4. Similar to DE_570T, the cache and MAC option is the factor for better performance than DE_106micro and DE_108mini. The speedup for fixed-point DCT-I over floating-point is measured 3.61, 4.60, 5.39, 5.83, 5.98, 5.60 for m = 2, 3, 4, 5, 6 and 7. Similarly, the improvement is 4.08, 4.73, 5.09, 5.14, 5.02, and 4.82 for fixed-point DST-I over floating-point DST-I for m = 2, 3, 4, 5, 6 and 7.

Table 6.4. *Recorded clock cycles for configuration DE_212GP*

m	Clock Cycles			
	DCT_Fixed	**DCT_Float**	**DST_Fixed**	**DST_Float**
2	2032	7339	1161	4733
3	4328	19922	3249	15362
4	9241	49844	7960	40516
5	20061	116923	18590	95560
6	43954	262857	42317	212640
7	96694	541856	94939	457654

Furthermore, we have also measured the clock cycle counts for different types of HiFI configurations and ConneX configurations. The results are provided in the appendix.

7. CONCLUSIONS

The objective of this thesis was to develop C-code for fixed-point DCT and DST applications. The selected floating-point DCT-I and DST-I C-codes [5] were first converted into fixed-point C-code and the performance was evaluated in Tensilica processor cores. The selected floating-point DCT and DST algorithms and codes were studied in detail and then the fractional fixed-point format (Q15) was chosen for the fixed-point code design.

The SNR calculation was the first task after designing the fixed-point DCT-I and DST-I codes. From the observation, we have found that the SNR of the designed fixed-point code with respect to floating-point was accurate (between 35-76dB range) for observed range. However, the scaling done to the variables to control overflow in fixed-point C-codes has direct effect on SNR. The scaling is proportional to the input length of DCT and DST transforms. We have found that with the increment in input length of DCT and DST, variables in C-codes need to be scaled down more to compensate overflow making the output more prone to quantization errors. This is because scaling reduced the resolution that was provided by Q15 format. This is a drawback of fixed-point code over floating-point code. The choice of Q format has an impact on the output error. The Q15 format chosen in the fixed-point development process has relatively very good accuracy. However, the truncation done while representing the numbers as integer in C-code and scaling done to handle any overflow that might occur, produced some error in the output.

Another part of the thesis work is to evaluate the performance of designed fixed-point code and reference floating-point code in Tensilica processor cores. The DCT and DST application codes were evaluated on four different types of Tensilica processor configurations (DE_106micro, DE_108mini, DE_212GP and DE_570T). The profiling results show that the fixed-point application codes have significant performance improvement in terms of clock cycles over floating-point application on all processor configurations. The total number of clock cycles taken to execute the fixed-point DCT and DST codes on various configurations is three to six times less than the reference floating-point DCT and DST codes for different sets of DCT/DST length. Another observation is that the configuration DE_106micro takes most clock cycles than the rest configuration for same code. The DE_108mini and DE_212GP configuration has better performance compared to DE_106micro and the DE_570T was fastest among all, which has VLIW architecture for parallel processing.

From the results presented in this work, we can conclude that the fixed-point DCT/DST code is very fast as compared to floating-point code. The fixed-point DCT/DST C-code uses the integer operation causing it to execute fast in hardware. On

the other hand, floating-point codes are emulated for arithmetic operation, as they do not have floating-point unit support. The emulation has a significant computation overhead slowing the execution down for DCT/DST application. Furthermore, the number of clock cycles is more than double when we double the transform size. Another conclusion can be drawn for the performance of codes on different configurations. Since, the DE_570T configuration has more resources, as well as it utilize data and instruction level parallelism, the performance is the best among all. The internal architecture of configurations like multiplier size, use of instruction and data width, memory and cache size etc. affected the performance of the application on configurations that we used.

In future work, the designed fixed-point DCT and DST codes could be further optimized. For example, in the codes some data from one array are copied to another, while better approach could be allocating two buffers and changing pointers rather than copy of array values. There are several code optimization strategies that could be used to optimize the code [32]. A design space exploration can be done to optimize the whole application and processor configuration provided added time. Moreover, the performance of code can be evaluated in multiple-processor pipeline architecture. In this thesis, the DCT/DST codes are only evaluated on Diamond processors, which are not customizable. It would be interesting to implement TIE instructions for customizable Xtensa processors and analyze the performance variation of the DCT/DST code with Diamond processors. Similarly, it would be better to understand whether the codes meet performance requirement of some industrial applications or not. Furthermore, the power and area are other main parameters that are not considered in this thesis. So, the codes can be analyzed in terms of power consumption and area covered in different processors.

REFERENCES

[1] Z. Wang and B.R. Hunt, ”The discrete W transform”, *Applied Mathematics and Computation*, vol. 16, no. 11, Jan. 1985.

[2] P. Yip and K.R. Rao, ”On the shift property of DCT’s and DST’s”, *IEEE Transactions on Acoustics, Speech and Signal Processig,* vol. 35, no. 3, pp. 404-406, March, 1987.

[3] V. Britanak, P.C. Yip, and K.R. Rao, “Discrete Cosine and Sine Transforms: General properties, Fast algorithms and Integer Approximations”, 1st Edition, 2007

[4] J. Nikara, “Application-Specific Parallel Structures for Discrete Cosine Transforms and Variable Length Decoding”, Ph.D Thesis, *Tampere University of Technology, Publication 481,* Tampere 2004

[5] K.R. Rao et al. (ed), The Transform and Data Compression Handbook, Published by Boca Raton, CRC Press LLC, InTech, 2001. V.Britanak, “Discrete Cosine ad Sine Transforms”, *book chapter 4 in K.R. Rao et al. (Ed),* “The Transform and Data Compression Handbook”, ISBN 0-8493-3692-9, Boca Raton, CRC Press LLC , 2001.

[6] S.A. Khayam, “The Discrete Cosine Transform (DCT): Theory and Application”, *Department of Electrical and Computer Engineering, Michigan State University,* March 2003.

[7] K.R. Rao, and J.J. Hwang, “Techniques and standards for image, video, and audio coding”, ISBN: 0-13-309907-5, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1996.

[8] A.D. Poularikas (2nd ed), The Transforms and Applications Handbook, Published by Boca Raton, CRC Press LLC,InTech, 2000. P.Yip, “Sine and Cosine Transforms”, *book chapter 3 in A.D. Poularikas (2nd ed)*, “The Transforms and Applications Handbook”, ISBN 0-8493-8595-4, Boca Raton, CRC Press LLC, 2000.

[9] Z. Wang, and B.R. Hunt, “The discrete cosine transform-A new version”, in *IEEE International Conference on Acoustics, Speech, and Signal Processing,,* pp. 1256-1259, April 1983.

[10] D. Guevorkian, K. Rounioja, and J. Takala, “Circulant Hermitian matrix inversion method based on discrete cosine and sine transforms”, in *IEEE Workshop on Signal Processing Systems,* pp. 306-311, Oct. 2012.

[11] A. Ghalib, "Analysis of fixed-point and floating-point quantization in fast Fourier transform", Master's Thesis*, Tampere University of Technology,* June, 2013.

[12] S.S. Saokar, R.M. Banakar, and S. Siddamal, "High speed signed multiplier for Digital Signal Processing applications", *Published in Signal Processing, Computing and Control (ISPCC), pp.:1-6,* Waknaghat Solan, March 2012.

[13] W.T. Padgett, and D.V. Anderson, "Fixed-Point Signal Processing", ISBN 9781598292589, Morgan and Claypool Publishers series, 2009.

[14] A. Haghparast, H. Penttinen, and A. Huovilainen, "Fixed-Point Algorithm Development", *Lab. of Acoustics and Audio Signal Processing, Helsinki University of Technology*, Finland, April, 2006.

[15] T. Finley, "Two's Complement", April 2000, http://www.cs.cornell.edu/~tomf/ notes/ cps104/ twoscomp.html, accessed on 25th April, 2015.

[16] A. Fisher, "Fixed-point Implementation of Acoustic Feature Generation Algorithms", *Diploma Thesis, Technische Universität Graz, Austria,* April 2008.

[17] Tensilica Inc., Xtensa processors and Xtensa Xplorer IDE, http://www.tensilica .com.

[18] "Xtensa Microprocessor", *Overview Handbook, A summary of the Xtensa Microprocessor Data Book,* Tensilica, Inc., USA, 2002.

[19] "Tensilica Sofware Development Toolkit (SDK)", *Tensilica Datasheet,* accessed on 25th April, 2015.

[20] "Diamond Standard Processor Cores", *white paper,* Oct. 2008, http://www.tensilica.com, accessed on 30th Nov. 2014.

[21] "Xtensa Instruction Set Architecture (ISA)", *Reference Manual,* Tensilica, Inc., Santa Clara, CA, 2010, http://www.tensilica.com, accessed on 15th Dec. 2014.

[22] A.R. Weiss, "Dhrystone Benchmark: History, Analysis, Scores and Recommendations", *white paper,* Oct. 2002, ECL, LLC, http://www.ebenchmarks.com, accessed on 10th March, 2015.

[23] "Diamond Standard 106Micro Controller", *Product Brief,* Tensilica, Inc., Santa Clara, CA, March 2010, http://www.tensilica.com, accessed on 20th Dec. 2014.

[24] "Diamond Standard 108Micro Controller", *Product Brief,* Tensilica, Inc., Santa Clara, CA, January 2010, http://www.tensilica.com, accessed on 20th Dec. 2014.

[25] "Xtensa LX Microprocessor", *Overview Handbook,* Tensilica, Inc., Santa Clara, CA, 2004, http://www.tensilica.com, accessed on 1st March 2015.

[26] *Press Release,* Santa Clara, January, 2013, http://ip.cadance.com, accessed on 1st March 2015.

[27] "HiFi 2 & HiFi EP Audio DSPs", *Product Brief,* Tensilica, Inc., Santa Clara, CA, 2012-3, http://www.tensilica.com, accessed on 1st March 2015.

[28] *Tensilica's HiFi 3 DSP Core: Audio Post-Processing Comes to the Fore,* Berkeley Design Technology, Inc., accessed on 1st March 2015.

[29] *HiFi Comparison Chart,* http://ip.cadence.com/ipportfolio/tensilica-ip/audio, accessed on 2nd March 2015.

[30] "Connx D2 DSP Engine", *Product Brief,* Tensilica, Inc., Santa Clara, CA, June, 2010, http://www.tensilica.com, accessed on 1st March 2015.

[31] P. Getreuer, *Writing Matlab C/Mex Code,* http://www.getreuer.info, April 2010, accessed on August, 2014.

[32] S. Ryoo, "Program Optimization strategies for Data-Parallel Many-core Processors", Ph.D. Thesis, *University of Illinois at Urbana-Champaign,* Urbana, Illinois, 2008.

APPENDIX A

The table below presents the clock cycles measured in various HiFi cofigurations of Tensilica Xlorer IDE for fixed-point and floating-point DCT/DST code.The configuration chosen are similar to that of Diamond configuration presented in the thesis work.

Table A.1: *clock cycle count for hifi2_std configuration (HiFi 2 Audio DSP)*

	Total Clock Cycles			
m	**DCT_Fixed**	**DCT_Float**	**DST_Fixed**	**DST_Float**
2	2029	7321	1159	4721
3	4335	19915	3254	15368
4	9276	49846	7992	40562
5	20128	116911	18667	95657
6	44014	262683	42442	212708
7	96593	552144	95078	457685

Table A.2: *clock cycle count for hifi3_bd5 (HiFI 3 Audio Processor)*

	Total Clock Cycles			
m	**DCT_Fixed**	**DCT_Float**	**DST_Fixed**	**DST_Float**
2	2692	29296	1406	17322
3	5572	85380	4081	61948
4	11821	220948	10005	167576
5	25829	527050	23968	394113
6	56630	1195645	55298	865644
7	124839	2561052	123826	1828799

Table A.3: *clock cycle count for hifi3_mini(HiFI mini Audio Processor)*

	Total Clock Cycles			
m	**DCT_Fixed**	**DCT_Float**	**DST_Fixed**	**DST_Float**
2	2715	28802	1406	17018
3	5624	83924	4107	60934
4	11919	217151	10075	164909
5	26008	518042	24113	388012
6	56957	1175438	55571	852736
7	125451	2114322	124316	1811915

Table A.4: *clock cycle count for hifiep_bd5(HiFi EP Audio Processor)*

m	Total Clock Cycles			
	DCT_Fixed	**DCT_Float**	**DST_Fixed**	**DST_Float**
2	2029	7318	1159	4718
3	4322	19863	3239	15316
4	9221	49687	7931	40403
5	19995	116571	18522	95317
6	43760	262076	42168	212101
7	96174	545047	94629	456763

APPENDIX B

The table below presents the clock cycles measured in various ConnX DSP cofigurations of Tensilica Xlorer IDE for fixed-point and floating-point DCT/DST code.The configuration chosen are similar to that of Diamond configuration presented in the thesis work.

Table B.1: *clock cycle count for XRC_D2MR*

	Total Clock Cycles			
m	**DCT_Fixed**	**DCT_Float**	**DST_Fixed**	**DST_Float**
2	2029	5564	1159	3558
3	4322	15150	3239	11790
4	9221	37939	7931	31498
5	19995	89247	18522	75168
6	43760	201302	42168	169266
7	96174	415618	94629	368314

Table B.2: *clock cycle count for XRC_D2SA*

	Total Clock Cycles			
m	**DCT_Fixed**	**DCT_Float**	**DST_Fixed**	**DST_Float**
2	2132	7413	1183	4785
3	4581	20287	3418	15751
4	9864	50953	8532	41779
5	21486	119693	19935	98801
6	47276	269118	45702	220132
7	103969	560103	102582	474274