EFFICIENT IMAGE COMPRESSION SYSTEM WITH A CMOS TRANSFORM IMAGER

A Dissertation
Presented to
The Academic Faculty

By

Jungwon Lee

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in
Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
December 2009

EFFICIENT IMAGE COMPRESSION SYSTEM WITH A CMOS TRANSFORM IMAGER

To my late father, my mother, and my wife Geunjung

ACKNOWLEDGMENT

I would like to thank, first and foremost, my thesis advisor Dr. David Anderson. I would not be able to finish my thesis without his constant support and guidance. I would also like to thank my thesis committee members, Dr. Paul Hasler, Dr. Justin Romberg, Dr. John Dorsey, Dr. Sung Ha Kang, and Dr. Joel Jackson, for their valuable advices and comments.

I would like to thank my colleagues, Abhishek Bandyopadhyay, Teahyung Lee, and Ryan Robucci, for their feedbacks and fruitful discussions that helped me improve my work. Likewise, I would also like to thank all the members of ESP Group, specially Faik Baskaya, Walter Huang, Michael Lo, and Sourabh Ravindran, for letting me have a lot of wonderful and unforgettable memories during my graduate studies at Georgia Tech.

Last, but not least, I would like to express my deep gratitude to my mother and my wife. Their constant and selfless love has always guided me like a lighthouse in my long voyage.

SUMMARY

This research focuses on the implementation of the efficient image compression system among the many potential applications of a transform imager system. The study includes implementing the image compression system using a transform imager, developing a novel image compression algorithm for the system, and improving the performance of the image compression system through efficient encoding and decoding algorithms for vector quantization.

A transform imaging system is implemented using a transform imager, and the baseline JPEG compression algorithm is implemented and tested to verify the functionality and performance of the transform imager system. The computational reduction in digital processing is investigated from two perspectives, algorithmic and implementation. Algorithmically, a novel wavelet-based embedded image compression algorithm using dynamic index reordering vector quantization (DIRVQ) is proposed for the system. DIRVQ makes it possible for the proposed algorithm to achieve superior performance over the embedded zero-tree wavelet (EZW) algorithm and the successive approximation vector quantization (SAVQ) algorithm. However, because DIRVQ requires intensive computational complexity, additional focus is placed on the efficient implementation of DIRVQ, and highly efficient implementation is achieved without a compromise in performance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Overview

Two trends are leading information technology in the 21st century, ubiquitous computing and technology convergence. There are many definitions and explanations of these terms, but in short, people want to make their devices more mobile and put more functionalities in one device. We can easily look at these trends in mobile phones, PDAs, MP3 players, and portable multimedia players (PMP). Digital camera or voice recording functions in mobile phones and PDAs are not optional anymore and considered standard features.

The prevalence of the ubiquitous and multi-functional systems in daily life creates a higher demand for more efficient signal processing systems in power consumption. Conventional digital-only signal processing has limitations in meeting this demand. On the other hand, analog computational circuits have various advantages, namely, power consumption and real-time signal processing. There is great potential for integrating both types of circuits into a cooperative analog-digital signal processing (CADSP) system [1].

CADSP is a new concept that explores advantages and potentials, which were impossible with digital-only systems, of combining analog processing and digital processing. This concept becomes feasible because of the new advances in analog VLSI circuits that have been improved for more accurate computations comparable to digital processing. In addition, the programmability, which is one of the biggest merits of digital processing, of the analog circuits is possible using floating-gate circuit technology. This programmability plays a key role in CADSP.

Conventional signal processing systems have to convert the all real-world analog signals to digital signals using analog-to-digital converters as shown in Figure 1.1(a) because signal processing is performed only in the digital domain. These systems need to allocate significant amounts of processing power and memory for the analog-to-digital conversion and digital-only processing. On the other hand, CADSP systems partition a signal processing algorithm and perform it both in the analog domain and in the digital domain as shown in Figure 1.1(b). In doing so, the processing power and memory can be efficiently reduced, and processing speed can be also improved.

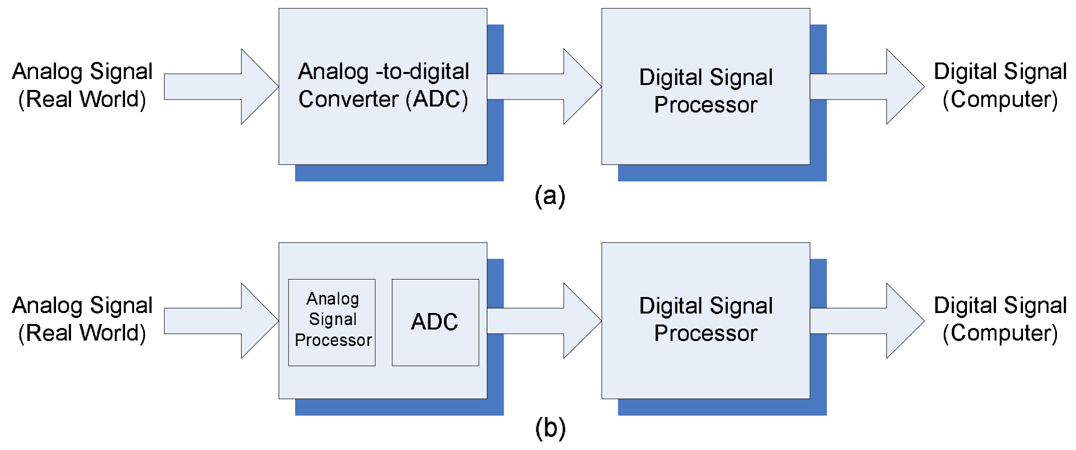


Figure 1.1. CADSP concept.

A transform imager has been designed and developed at Georgia Institute of Technology for CADSP image processing applications. The transform imager blends the advantage of focal-plane imagers and the advantage of CMOS pixel sensors, so it can do significant amounts of signal processing with large imaging arrays [2]. The programmability of the transform imager coefficients enables implementation of various signal processing in the transform imager. However, the transform imager must be complemented with digital processing to implement the full image processing algorithms in one system. Therefore, both analog and digital aspects of the design should be considered together in efficient signal processing system implementation.

This research focuses on the implementation of the efficient image compression system among the many potential applications of the transform imager system. The study includes implementing the image compression system using a CMOS transform imager, developing a novel image compression algorithm for the system, and improving the performance of the image compression system through efficient encoding and decoding algorithms for vector quantization.

A transform imaging system is implemented using a CMOS transform imager, and the baseline JPEG compression algorithm is implemented and tested to verify the functionality and performance of the transform imager system. The computational reduction in digital processing is investigated from two perspectives, algorithmic and implementation. Algorithmically, a novel wavelet-based embedded image compression algorithm using dynamic index reordering vector quantization (DIRVQ) is proposed for the system. Wavelets are widely used for image processing because of their great localization performance. In addition, wavelet-based embedded compression reduces the noise of images in some degree, and more denoising can be achieved using the thresholding technique [3],[4],[5]; therefore, transform imager noise can be effectively suppressed. DIRVQ makes it possible for the proposed algorithm to achieve superior performance over the embedded zero-tree wavelet (EZW) algorithm [6] and the successive approximation vector quantization (SAVQ) algorithm [7]. However, because DIRVQ requires intensive computational complexity, additional focus is placed on the efficient implementation of DIRVQ, and highly efficient implementation is achieved without a compromise in performance.

1.2 Contributions of the Research

The contributions of this research contains the followings:

• Implemented an imaging system using a CMOS transform imager. A full imaging system is implemented using a CMOS transform imager and a FPGA. Performance of the imager and the imaging system is evaluated by implementing a real-time hardware JPEG compression system in collaboration with my lab partners, Abhishek Bandyopadhyay, Ryan Robucci, and Jordan Gray.

• Characterized the noise of a CMOS transform imager, and denoising effects from various transform-based compression algorithms are shown.

• Developed a high performance wavelet-based image coding scheme. This scheme applies dynamic index reordering vector quantization (DIRVQ) to the newly defined temporal domain of embedded image coding. The compression performance improvement is evaluated by comparing the results to successive approximation vector quantization (SAVQ).

• Developed efficient encoding and decoding methods for D_4 and Λ_{16} lattice shell-based DIRVQ (L-DIRVQ). The proposed L-DIRVQ framework can be applicable to most gain-shape type vector quantization with significantly reduced computational complexity of DIRVQ.

CHAPTER 2
CMOS TRANSFORM IMAGER SYSTEM

Camera-on-a-chip systems have tried to include carefully chosen signal processing units for better functionality and performance, and also to broaden the range of applications they can be used for [8],[9],[10],[11]. These systems have been possible because of advances in CMOS active pixel sensor (APS) and neuromorphic focal-plane imagers [12],[13]. Some of the advantages of these systems are compact size, high speed, parallelism, low power dissipation, and dense system integration [14],[15],[16].

Neuromorphic focal-plane imagers have the capability of analog focal-level signal processing with high speed computation and low power dissipation. However, it requires a large amount of additional circuitry around pixels, and leads to very low pixel fill-factor. The fill-factor is defined as the percentage of photosensitive area in a pixel. A low pixel fill-factor makes it difficult to manufacture high-resolution and dense imagers. Implementing an analog-to-digital converter (ADC), digital computational units, and memory on a pixel is another approach for focal-plane image processing [17],[18]. One of the strength of this approach, when compared to the neuromorphic approach, is that the pixel can be programmable for different image processing algorithms. However, this method also suffers from low pixel fill-factor like the neuromorphic focal-plane imagers and consumes a considerable amount of power.

The solution is a transform imager architecture. The transform imager architecture is both modular and programmable, making it ideal for image dataflow computation. This approach allows for retina and high-level bio-inspired computation in a programmable architecture that still possesses the high fill-factor pixel characteristics of APS imagers. The system architecture for the transform imager is described in this chapter.

2.1 Pixel Structure and Imager Architecture

Each pixel is composed of a photodiode sensor element and an analog multiplier [19]. Figure 2.1 shows that an nFET differential pair performs this multiplication. For the differential pair operating with subthreshold bias currents, which should always be the case because of the low-level image sensor current, the differential output current can be expressed as

$$I_1^+ - I_1^- = I_{sensor_1} tanh\left(\frac{\kappa(V_1^+ - V_1^-)}{2U_T}\right), \tag{2.1}$$

where κ is the gate-coupling efficiency into the transistor surface potential (typically 0.6-0.8), and U_T is the thermal voltage, $\kappa T/q$. T is an analysis temperature (Kelvin) and q is an electron charge. If the difference $V_1^+ - V_1^-$ is small, then equation 2.1 becomes

$$I_1^+ - I_1^- = I_{sensor_1}\left(\frac{\kappa(V_1^+ - V_1^-)}{2U_T}\right). \tag{2.2}$$

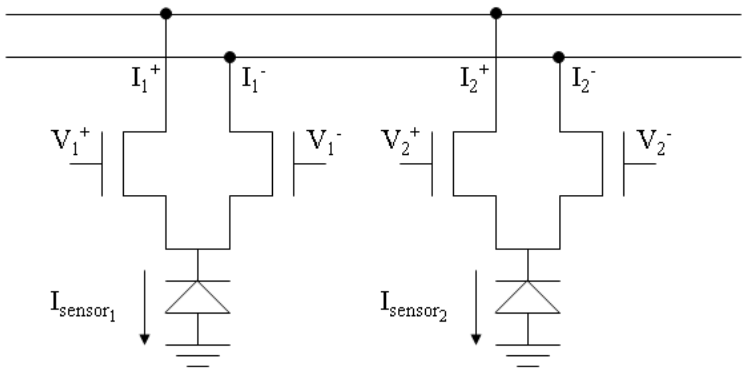


Figure 2.1. Pixel structure: The chip performs arbitrary separable block transforms. The basis functions are programmed on-chip using floating-gate circuits.

Therefore, the difference of the output current, I_{diff}, is the product of the sensor output current, I_{sensor}, and the differential input voltage, V_{diff}:

$$I_{diff} = C \cdot I_{sensor} \cdot V_{diff}, \tag{2.3}$$

where C is $\kappa/2U_T$, which is a constant.

Matrix multiplication has two parts: element-by-element multiplication and subsequent addition. In this pixel, the former is performed at the pixel level, while the latter is performed by Kirchoff's law, since the outputs are currents. A vector of input voltages comes from the circuitry to the image array and is shared by each column of the image array. Along each column, outputs are tied together to obtain a summation of currents as shown in Figure 2.1. This means that the output of each column is a sum of products; in particular, it is the inner product of the voltage vector with the light intensities along the column.

This imager architecture can compute arbitrary separable 2D linear operations. These operations are expressed as two matrix multiplications on the image:

$$Y = A^T PB, \tag{2.4}$$

where P is the image matrix of pixels, Y is the computed output matrix, and A and B are the transform kernels [20]. The values of A and B are stored in analog floating-gate arrays for on-chip processing.

The transform kernel has the following structure:

$$A = \begin{bmatrix} a[0,0] & a[0,1] & \cdots & a[0,N-1] \\ a[1,0] & a[1,1] & \cdots & a[1,N-1] \\ \multicolumn{4}{c}{\dotfill} \\ a[N-1,0] & a[N-1,1] & \cdots & a[N-1,N-1] \end{bmatrix}, \tag{2.5}$$

where $a[n,k] = C_k cos[(2n+1)kn/2N]$ for discrete cosine transform (DCT) type-II, and $a[n,k] = \sqrt{2/(N+1)} sin[\pi(k+1)(n+1)/(N+1)]$ for discrete sine transform (DST). The Hadamard transform can be written in the matrix form as

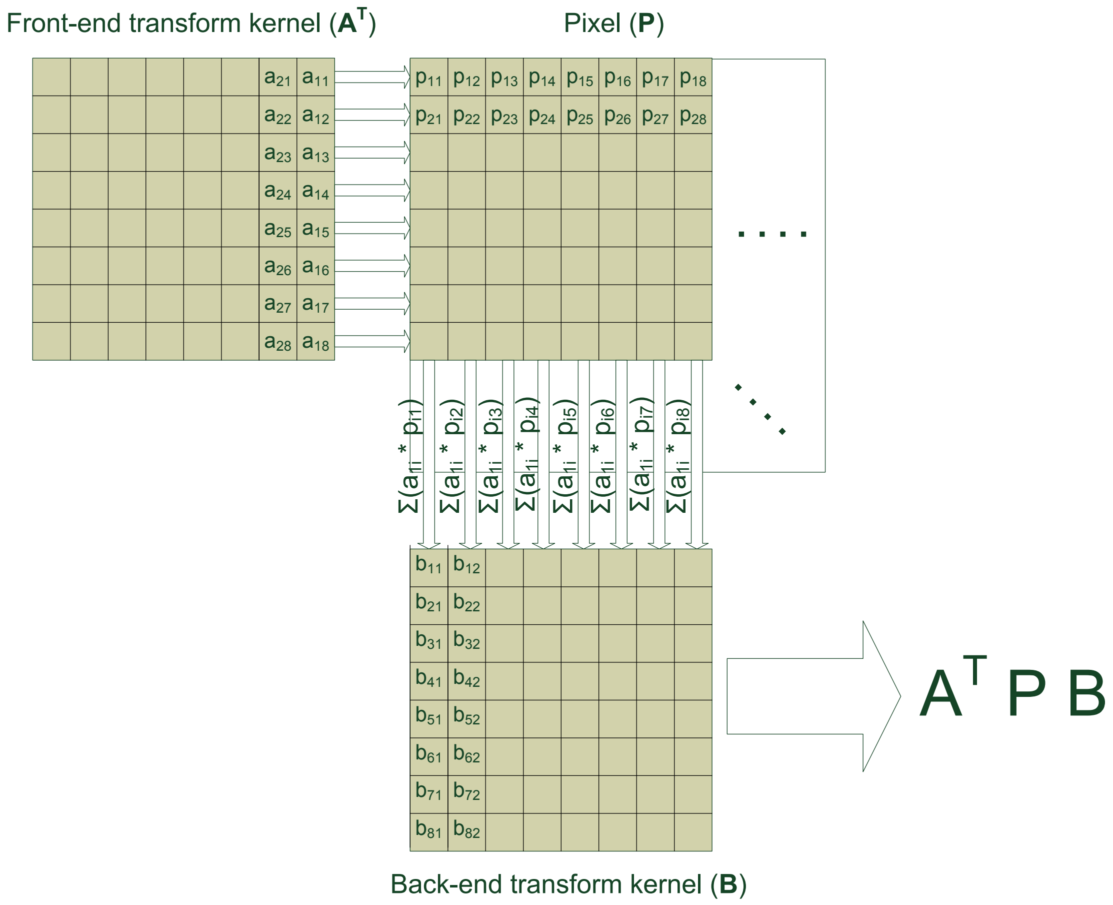
$$A_{2^n} = A_{2^{n-1}} \bigotimes A_2 = A_2 \bigotimes A_{2^{n-1}}, \tag{2.6}$$

where

$$A_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{2.7}$$

Similarly, a Haar transform can also be written in the matrix format for different sizes. The matrix coefficients are stored on-chip using floating gates. A floating gate is a polysilicon gate surrounded by silicon dioxide. The charge on the floating gate is stored permanently as it is surrounded by high-quality insulator.

The first matrix multiplication, $Y = A^T P$, takes place at the pixel array itself in a parallel manner, and the output currents from the pixel array are supplied to the multiplier array, which contains the back-end transform kernel, for the second multiplication (with B) as shown in Figure 2.2.



Front-end transform kernel ($\mathbf{A^T}$)
Pixel ($\mathbf{P}$)
a_{21} a_{11}
a_{22} a_{12}
a_{23} a_{13}
a_{24} a_{14}
a_{25} a_{15}
a_{26} a_{16}
a_{27} a_{17}
a_{28} a_{18}
p_{11} p_{12} p_{13} p_{14} p_{15} p_{16} p_{17} p_{18}
p_{21} p_{22} p_{23} p_{24} p_{25} p_{26} p_{27} p_{28}
$\Sigma(a_{1i} * p_{i1})$
$\Sigma(a_{1i} * p_{i2})$
$\Sigma(a_{1i} * p_{i3})$
$\Sigma(a_{1i} * p_{i4})$
$\Sigma(a_{1i} * p_{i5})$
$\Sigma(a_{1i} * p_{i6})$
$\Sigma(a_{1i} * p_{i7})$
$\Sigma(a_{1i} * p_{i8})$
b_{11} b_{12}
b_{21} b_{22}
b_{31} b_{32}
b_{41} b_{42}
b_{51} b_{52}
b_{61} b_{62}
b_{71} b_{72}
b_{81} b_{82}
$A^T P B$
Back-end transform kernel ($\mathbf{B}$)

Figure 2.2. How to compute 2D separable transform in a transform imager.

The block diagram of the single-chip CMOS imager is shown in Figure 2.3. This imaging architecture is made possible largely by advancements in analog floating-gate circuit technology. These circuits have the added advantage that they can be built in standard CMOS or double-poly CMOS processes. In this approach, the floating-gate circuits store and reproduce arbitrary analog waveforms for image transforms, allow for correction factors to account for device mismatch, and perform matrix-vector computations.

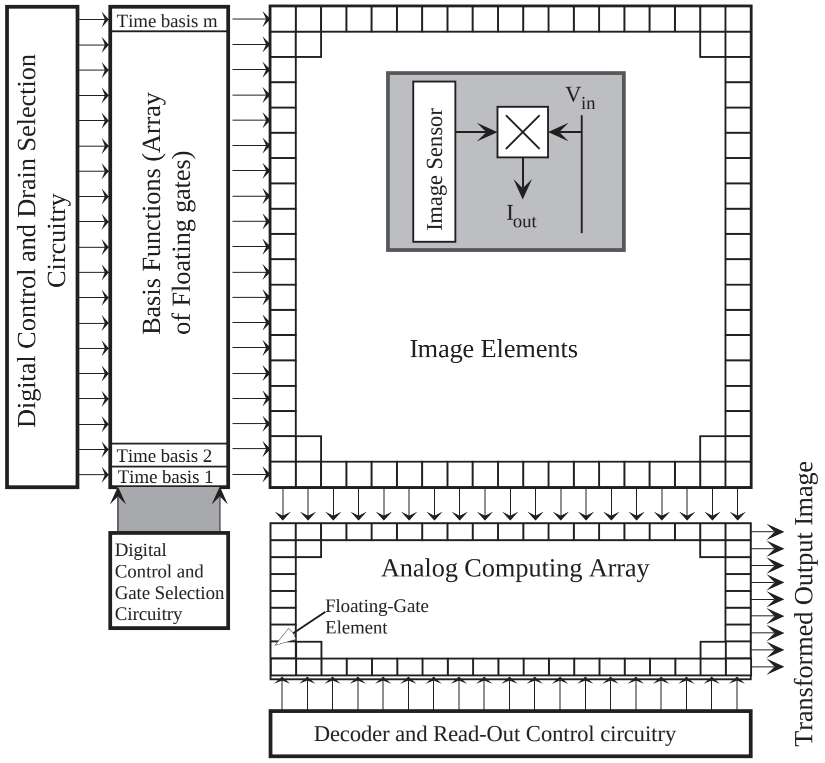

Digital Control and Drain Selection Circuitry
Time basis m
Basis Functions (Array of Floating gates)
Time basis 2
Time basis 1
Image Sensor
V_{in}
I_{out}
Image Elements
Digital Control and Gate Selection Circuitry
Analog Computing Array
Floating-Gate Element
Decoder and Read-Out Control circuitry
Transformed Output Image

Figure 2.3. Transform imager architecture: The chip performs arbitrary separable block transforms. The basis functions are programmed on-chip using floating-gate circuits.

For an indication of multiplication accuracy, a gain error measurement of the imager array was performed by illuminating the chip with nearly uniform light. The mean gain was $2.12nA/V$, which should be constant across the imager for constant illumination. The average error was 1.58%. This measurement is a conservative measure of expected performance since currents were measured off-chip in a noisy environment. On-chip processing will have the benefit of even better signal-to-noise ratio.

The DCT coefficients that were programmed on-chip are shown in Figure 2.4 (a), while Figure 2.4 (b) shows the percentage error associated with each coefficient. The coefficients can be programmed in both subthreshold and above-threshold regions. The programming algorithm used allows for programming accuracies up to 99.8% for 3.5 decades of currents. The retention loss is less than $1V$ at $27°C$ over 10 years, and less than $1mV$ at $90°C$ over 10 years for floating gates fabricated in 0.5-μm N-well CMOS process [21].

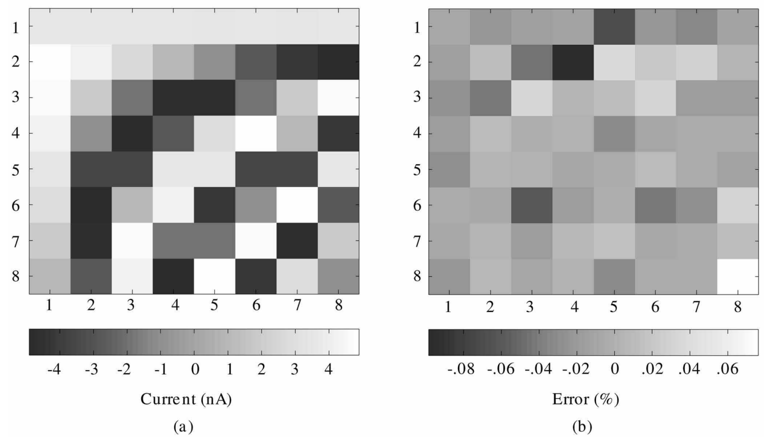


Figure 2.4. Programmed DCT values: an 8×8 DCT kernel was programmed onto an 8×8 array. The values were programmed around a DC of $10nA$. The DC was subtracted for clarity of display. The maximum percentage deviation for the array was 0.7%.

A traditional system would have to read out each pixel value, perform an analog-to-digital conversion, and store the image. The data would then be processed by digital circuitry to perform the matrix multiplication. This implementation of the matrix multiplication is much more efficient. The CMOS imager is based on the earlier work that demonstrated the viability of the pixel element, which computed simple single-block transforms on a fabricated IC, and that a single-block DCT could be computed [2].

2.2 System Implementation

The floating-gate elements are programmed to arbitrary values using an external programming board that only requires an external power supply and field programmable gate array (FPGA) interface. The schematic of the printed circuit board (PCB) used to control the imager is shown in Figure 2.5. The analog biases (drain voltage, gate voltages, tunneling voltages, and power supply) required by the imager chip are provided by DACs, which are controlled by the FPGA. There is additional circuitry for generating the tunnel voltage, and all the analog voltages are buffered before they are presented to the imager chip. Since a high power supply is required for programming, the digital signals to be used during that phase go through level shifters. The board has 14-bit 10-MS/s ADCs for image capture. For testing, an optical test bench was built, which projects an LCD image using a directed light source for illumination. During experiments, there was little noticeable movement of the floating-gate elements from their respective programmed values.

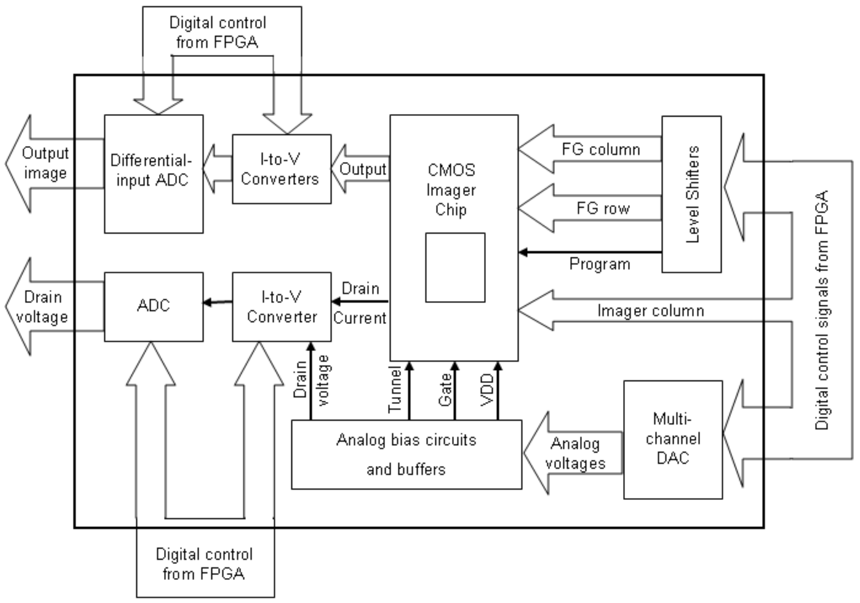


Figure 2.5. Schematic of the PCB: A PCB was designed for providing the analog biases and the digital control required for the chip. All the digital control is provided by an FPGA. The digital outputs are acquired and stored using the FPGA for further processing.

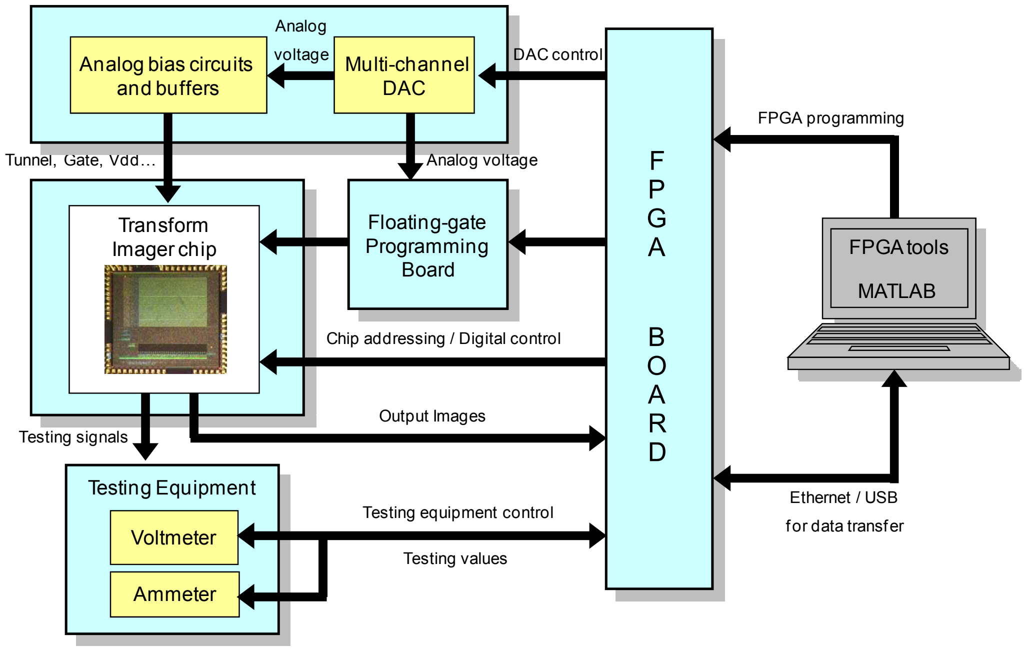


Figure 2.6. Imager system overview.

The timing sequence for image readout is shown in Figure 2.7. The different rows of the basis function are presented to the chosen block using kernel column selection. The outputs are read out using column parallel I-to-V converters and open-loop sample-and-hold circuits. After all the rows of the basis function have been presented for in-pixel multiplication and the output have been read out, the block selection is changed and the above process is repeated for reading that block.

The measured DCT transformed images and Haar transformed images are shown in Figure 2.8 and Figure 2.9, respectively. Figure 2.8 shows the original image as acquired by the transform imager, the 8×8 block DCT output of the imager, and the corresponding reconstructed image without compression. As expected, energy compaction is observed in each block. For the top image the DC components of the blocks have similar values since it has uniform contrast variation, whereas, for the bottom image the DC components near the face are higher than the ones corresponding to the background. The image was reconstructed without compression to verify the operation of the chip. The reconstructed images from the DCT coefficients show

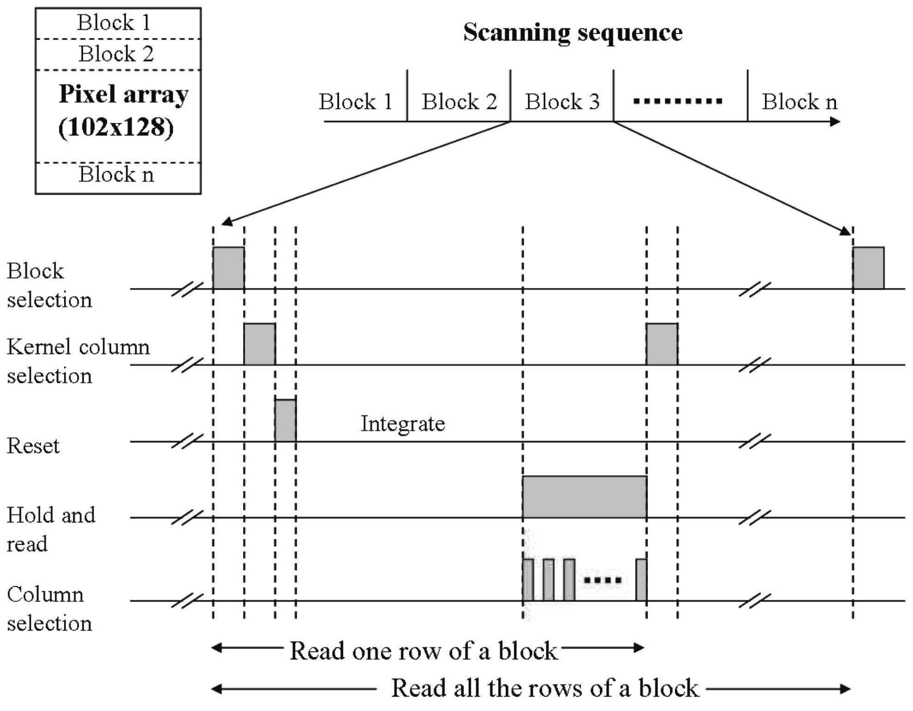


Figure 2.7. Timing sequence for image readout. The images are read out in a column parallel fashion. Random access is possible for reading parts of the image. The timing sequence is provided using an FPGA.

that the computation process has introduced minimal sources of error. The difference between reconstruction from the compressed image and reconstruction starting from the original image is negligible. Figure 2.9 shows the Haar transform results. The transform imager is programmed with an 8×8 Haar transform. Figure 2.9(b) shows a Haar transformed image before reordering, and Figure 2.9(c) is a reconstructed image to verify the operation of the chip.

MATLAB is used to communicate with a computer for more convenient development and verification of algorithms. Since controlling the transform imager and other hardware such as DACs and ADCs requires indepth knowledge of the entire imager system, several different levels of MATLAB application programming interface (API) are designed so that high-level algorithm developers do not necessarily need to know hardware details. For example, algorithm developers can read an image just by typing a ReadImage command in MATLAB without knowing how to control DACs and

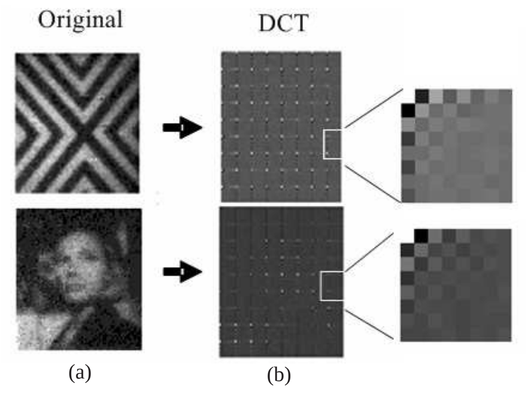


Figure 2.8. DCT results: (a) Original images (b) 8×8 block 2D DCT and DCT coefficients for one block.

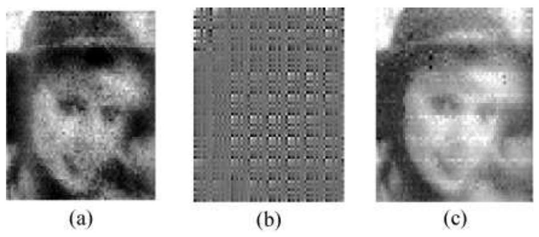


Figure 2.9. Haar transform results: (a) Original image (b) Haar transform (before reordering) (c) Reconstructed image.

ADCs on the imager PCB board.

2.3 Measured Noise Characteristics

The potential sources of error in the pixel are offset mismatches, gain mismatches, and mismatches across the pixel array. The transform imager pixel was characterized for dark current by measuring the current under reset conditions with no chip illumination. The distribution of dark currents from a 16×16 block of a larger imager is shown in Figure 2.10. The average dark current was measured to be 14.9 nA/cm. Edge effects are not observed in this case as this block was from the middle of the imager. The variations have been observed to follow no trends and are random in nature. These measurements were taken under no illumination conditions with the transform imager configured to read an image. Conventional dark current methods would give a lower value than measured here, as extra parasitics exist in the signal path when using the known method for measuring dark currents. A characterization chip was fabricated for testing the variations of the above. The voltage offset variations are plotted in Figure 2.11(a).

The voltage offset can be accounted for by programming the incoming bias voltages accordingly. The gain and mismatches are plotted in Figure 2.11(b) and (c). The gain is the slope of the tanh curve in the linear region. Since the first multiplication is performed at the pixel, the linearity of the in-pixel multiplier is important. The variation of linearity across a 40×48 array is plotted in Figure 2.11(d). The performance of the pixel is summarized in Table 2.1. The used photodiode has square edges, and reducing the edges might reduce mismatch.

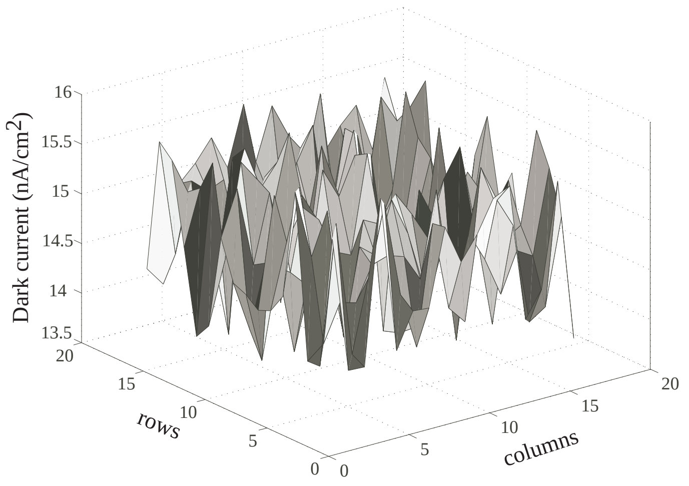


Figure 2.10. Dark current distribution: this figure shows the dark current distribution in a 16×16 block taken from the middle of the imager. The average dark current was measured to be 14.9 nA/cm^2.

Table 2.1. Summary of the transform imager pixel.

Parameter	Mean	Std. dev.
Gain	2.12 nA/V	0.34 nA/V
Linear range	54.4 mV	4.3 mV
Voltage offset	8.9 mV	6.7 mV
Kappa	0.7149	0.0072

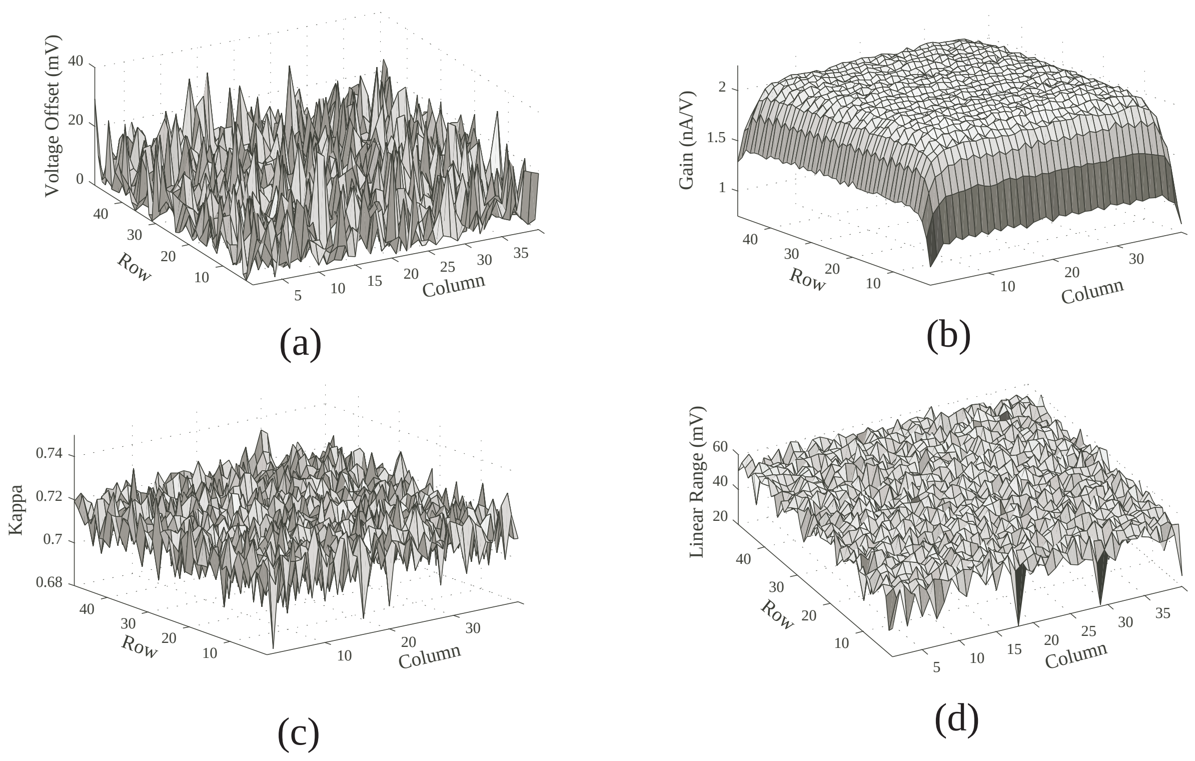


Figure 2.11. Pixel characterization measurements for uniform illumination: (a) the variation of voltage offsets for a 48×40 array is shown above. The mean and the standard deviation are 8.9 mV and 6.7 mV, respectively. (b) The variation of gain for a 48×40 array is shown above. The mean and the standard deviation are 2.12 nA/V and 0.0336 nA/V, respectively. This mismatch is primarily because of the mismatch of κ. (c) The variation of κ for a 48×40 array is shown above. The mean and the standard deviation are 0.7149 and 0.0072, respectively. This mismatch is primarily because of the mismatch of the gate coupling efficiency into the transistor surface potential. (d) The variation of linearity for a 48×40 array is shown above. The mean and the standard deviation are 54.4 mV and 4.3 mV, respectively.

2.4 Transform Imager Simulator

For algorithm developers, setting up the imager system, reprogramming the imager, and adjusting the imager biases could be onerous tasks, especially at the initial development stage when more frequent imager programming is required. A transform imager simulator is designed to make the development process more efficient [22]. The simulator is designed to share the same high-level MATLAB API as in Figure 2.12. Therefore, the developed MATLAB code using the simulator can be directly used with the actual imager system without any major change. There are many other advantages of the imager simulator. One advantage of using the simulator is that it makes the algorithm verification simpler and faster. Another advantage is that it also allows developers to develop algorithms remotely from the imager system, which is not always portable with the optical test bench. Another advantage of using the simulator is that it does not interrupt algorithm development when the imager system needs frequent hardware revision or maintenance.

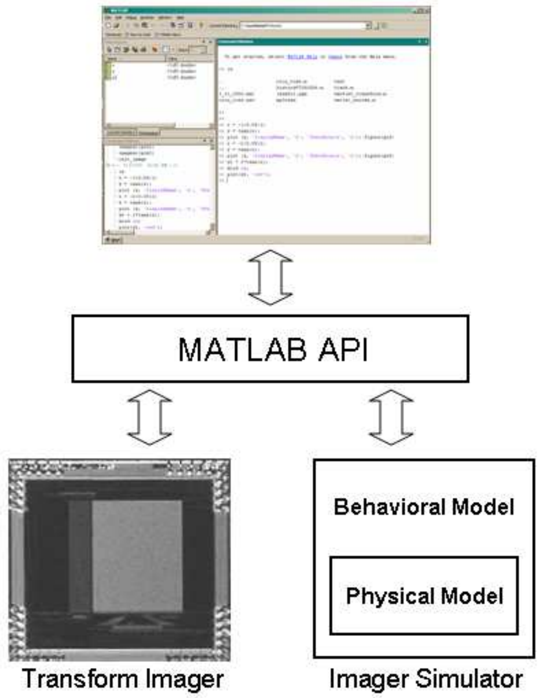


Figure 2.12. Imager simulator is designed to share the same MATLAB API as what the transform imager system uses.

The pixel structure and the architecture of the imager are simplified in the simulator with the implementation of basic noise characteristics. The simulator can be used in two different ways. When the feasibility of algorithms in the transform imager needs to be checked quickly, the behavioral simulation is suitable. The behavioral model, which sets the internal physical model as ideal without any noise, can be used for the behavioral simulation. When the analysis of noise characteristics is required in the simulation, the internal physical model can be set to the desired noise level. The physical model, which represents the physical behavior of the pixels, is mostly about the noise characteristics such as the fixed pattern noise (FPN) and the noise from the device mismatches.

CHAPTER 3

JPEG COMPRESSION SYSTEM USING A TRANSFORM IMAGER

For decades, transform image coding techniques have gotten the most attention in the field of image compression because of their simplicity and efficiency. They especially became more popular after the Joint Photographic Experts Group (JPEG) selected the DCT-based transform coding technique as a standard in 1988. The general strategy of transform image coding is shown in Figure 3.1 and can be described as follows:

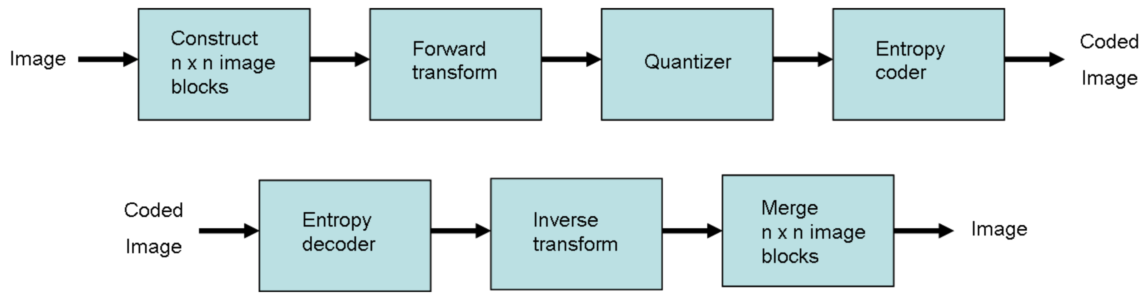


Figure 3.1. General strategy of image transform coding

1. Partitioning the images into $n \times n$ subimages. The size of image blocks is a significant factor affecting the performance of transform image coding. Different block sizes have been tested for popular transforms such as a discrete Fourier transform (DFT), DCT, Haar, Walsh-Hadamard; the results show that the root-mean-square error does not reduce much for sizes larger than 8×8 [20],[23]. Considering the computational complexity and performance increase, the JPEG standard uses the 8×8 block size [24].

2. Performing a forward transform for each image block. The main purpose of performing a forward transform is to de-correlate the pixels of the block. There are

many unitary block transforms that are capable of compacting the signal energy into a few transform coefficients such as DFT, DCT, Walsh-Hadamard, and so on. Among them, DCT is the most popular transform because of its superior compacting ability and periodic property. The periodic property is particularly important for reducing the block artifact [20].

3. Quantizing the transform coefficients. Major compression of DCT-based image coding is performed in the quantization stage. The significant coefficients are quantized finely, and the least significant coefficients are quantized coarsely. In the JPEG standard, transform coefficient matrices are divided by quantization matrices, and the divided values are rounded to the nearest integer values [24].

4. Entropy coding the quantized symbols. Quantized coefficients are converted to symbols, and the symbols are entropy coded. Huffman coding and arithmetic coding are widely used for entropy coding, and the JPEG standard specifies both.

Since the JPEG standard is the most successful and widely used among the transform image coding algorithms, extensive research has been done not only for the algorithm itself but also for the hardware/software implementation [25] [26]. DCT computation is considered the most intensive in the JPEG system [27]. Therefore, the implementation of dedicated DCT computational hardware also has been actively studied to achieve faster and more power efficient performance [28],[29],[30].

3.1 JPEG Compression System using a Transform Imager

Baseline JPEG compression requires computation of a 2D DCT, quantization, and run-length followed by Huffman coding [20]. A conventional JPEG implementation is shown in Figure 3.2 (a), in which most of the computations are processed in the digital chip except the imager and the analog-to-digital converter (ADC). The proposed system is shown in Figure 3.2 (b). A DCT block is one of the most computationally intensive blocks in the JPEG system [27]. This block is moved to the programmable transform imager to reduce the digital computations and power consumption. The tested system to prove the CADSP approach and concept is shown in Figure 3.2 (c). The analog computing array is modeled in MATLAB. An FPGA is used to interface between the transform imager and MATLAB, and also provides control signals to the imager. Figure 3.2 (d) is an ideal low-power system. The ADC is merged into the transform imager, so that all analog computations are implemented in a single chip. This solution is currently under development and will be presented in a future publication.

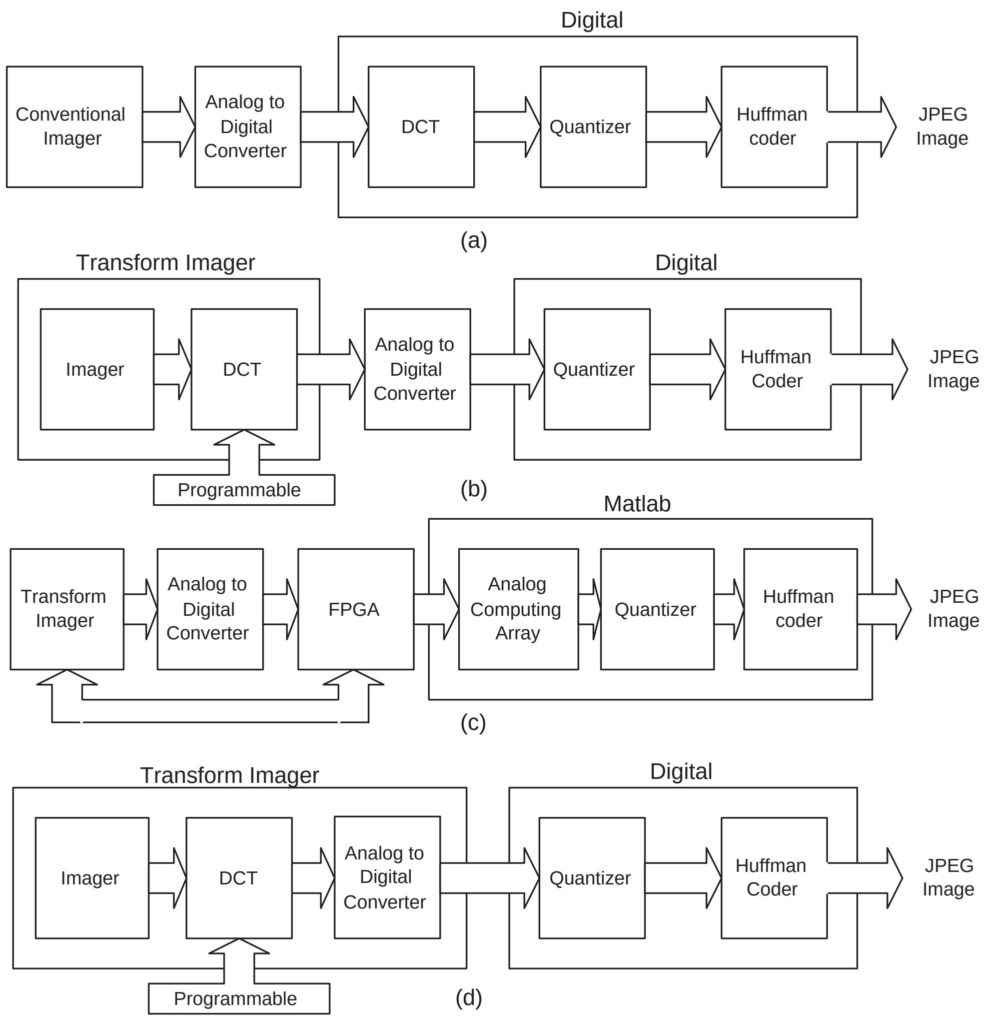


Figure 3.2. Top level view of the proposed JPEG compression system used as an application for signal processing. (a) Conventional approach (b) proposed system (c) Tested system (d) Ideal single chip system.

3.1.1 System Implementation

A printed circuit board (PCB) is designed for programming the imager and capturing images. This board is controlled by a Xilinx Virtex FPGA with a 32.768Mhz clock. Because the computation in the analog transform imager is very fast compared to the digital system, the readout speed is constrained by the test setup, such as the speed of interface and FPGA clock rate. The test images are focused onto the pixel array using a multiple lens system, an LCD (1024×768 resolution), and a DC regulated light source. Other imagers have been built that range from 14×14 resolution to 512×480 resolution and use an imager with 48×40 resolution and an imager with 72×64 resolution to easily analyze the transform results. The implemented hardware on the optical test bench is shown in Figure 3.3.

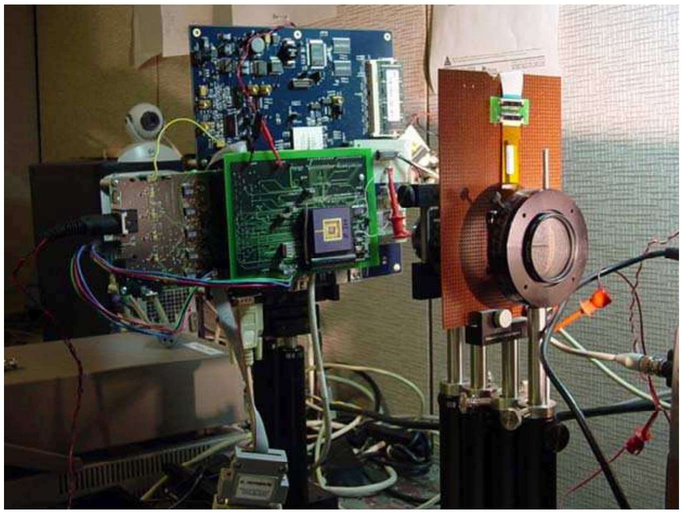


Figure 3.3. Implemented JPEG compression system using a transform imager on the optical bench.

The implementation of an 8×8 2D DCT based on the fast symmetry-based 1D DCT needs 208 multiplications and 464 additions, and even the fast FFT-based 1D DCT-based method requires 80 multiplications and 464 additions [31]. By using the transform imagers, the number of digital functional units is significantly reduced. The key concern here is the amount of power that could be saved by this reduction. It is not easy to estimate the total power of 2D DCT digital implementations directly from the number of reduced operations because there are several ways to implement it with various architectures.

The fully digital JPEG compression system was implemented in an FPGA using VHDL for power consumption comparison with the transform-imager-based system. Total power consumption of the fully digital implementation was estimated at $183mW$ by the Altera FPGA power estimate worksheet [32]. Total $146mW$ using the same estimation method by removing the DCT computation part, which is processed in the transform imager, was saved. Note that a generic DCT module, which is not designed for a low power solution, is used. However, even DCT chips optimized for low power consume a considerable amount of power, which ranges from about $10mW$ to $140mW$ [28],[29]. Usually, fixed implementations consume less power, and reconfigurable implementations consume more. A transform imager consumes less than $5mW$. Considering that the transform imager includes pixel elements, the power consumption reduction is significant.

Three-dimensional DCT implementation can be a good candidate for utilizing the system. A video compression based on the 3D DCT is not widely used since the 3D DCT is highly computationally intensive. Direct implementation of the $8 \times 8 \times 8$ 3D DCT requires 12,288 multiplications and 12,096 additions, and 960 multiplications and 5,568 additions are required if the Fourier-based 1D DCT is used [31]. The transform imager can reduce two-thirds of the total operations because the 2D DCT is already computed. Another problem of the 3D DCT-based compression algorithm is the memory requirement. To compute one block of an $8 \times 8 \times 8$ 3D DCT, eight frames of data should be stored in a buffer, which requires a large amount of memory. After DCT, most energy is concentrated on low frequencies, which have few DCT coefficients, depending on the compression ratio. Therefore, if only the low-frequency coefficients of the 8×8 block are read and transferred to the digital processing chip,

Table 3.1. Reduced digital computation of 8×8 2D DCT by using a transform imager. Note that the fast FFT-based 1D DCT is considered for both 2D and 3D DCT implementation in this table. 208 multiplications and 464 additions are required when the fast symmetry-based 1D DCT is used.

operations	2D DCT	3D DCT
additions	80	3712
multiplications	464	640

the buffer memory and the bandwidth between the imager and the digital chip can be significantly reduced. Because reading images is simply sending address signals to the imager, this selective reading is straightforward to implement.

The transform imager can be used for various image processing applications because it is reconfigurable. In addition to this flexibility, this reconfigurability makes it easy for the analog chip to calibrate its parameters. Since the transform imager performs multiplication of its inputs, calibrating the parameters, which is used for the multiplication coefficients, is important to avoid calculation errors.

3.1.2 Measured Images and Analysis

The PSNR of the images with different compression ratios is given in Figure 3.4. The following PSNR formula was used: $PSNR = 20log(1/\sqrt{MSE})$, where images are normalized to [0,1], and MSE is the mean squared error between them. The 16×16 Walsh-Hadamard transform output sequence of the imager with 20 frames per second (fps) and the corresponding reconstructed sequence are shown in Figure 3.5 (a), and the 16×16 DCT output sequence and the reconstructed sequence are shown in Figure 3.5 (b). Still images are used to make input sequences to the imager. Therefore, all movement is global translation.

Based on the bandwidth of a pixel, the peak computing power can be calculated [33]. To calculate one 8×8 DCT coefficient, 16 multiplications and 16 additions are required. Therefore, a total of $48 \times 40 \times (16 + 16)$ operations is needed for one 48×40

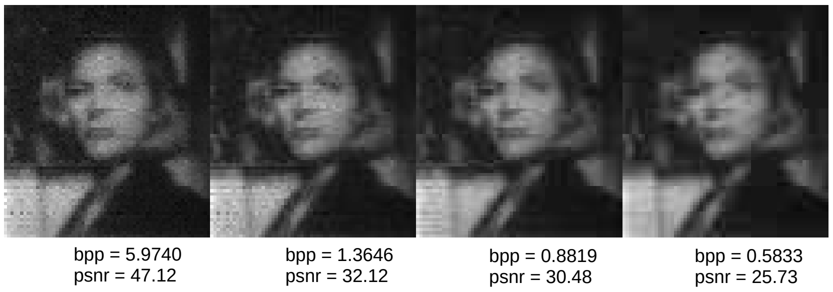


Figure 3.4. JPEG compression results: JPEG compression using a transform imager. The PSNR of the images with different compression ratio are given below each frame.

image. Considering the maximum bandwidth of a pixel to be $100Khz$, $10\mu sec$ is needed to process one column, and $400\mu sec$ is needed for processing the full image. This results in 153 $MOPS/s$, which means 153 million analog operations per second. Using this calculation, the peak computing power of the 128×128 transform imager can be easily computed. The required total operations for the 128×128 transform imager is $128 \times 128 \times (16+16)$, and it takes $1.28msec$ to finish the DCT calculation for one image. Therefore, the peak computing power is $204MOPS/s$. The fill factor of the imager is 46%, which is much higher than that of other focal-plane imagers such as a neuromorphic imager [34] and a SIMD-based digital focal-plane imager [17],[18].

3.2 Noise in the Transform Imager-Based JPEG Compression System

A transform imager is different from other CMOS imagers in the sense that the transform imager is designed for both capturing images and performing block transforms on the focal plane. For the focal-plane reprogrammable computation, the pixel structure of the transform imager is designed differently from that of other conventional CMOS imagers. This design difference makes a different image data flow. Compared to the conventional CMOS imagers, most noticeable different part of the transform imager is multiplying kernel coefficients to a pixel data. Because this operation is performed in the differential mode, the accuracy of the kernel values is important. However, thanks to the floating-gate technology, the kernel values can be reprogrammed, which

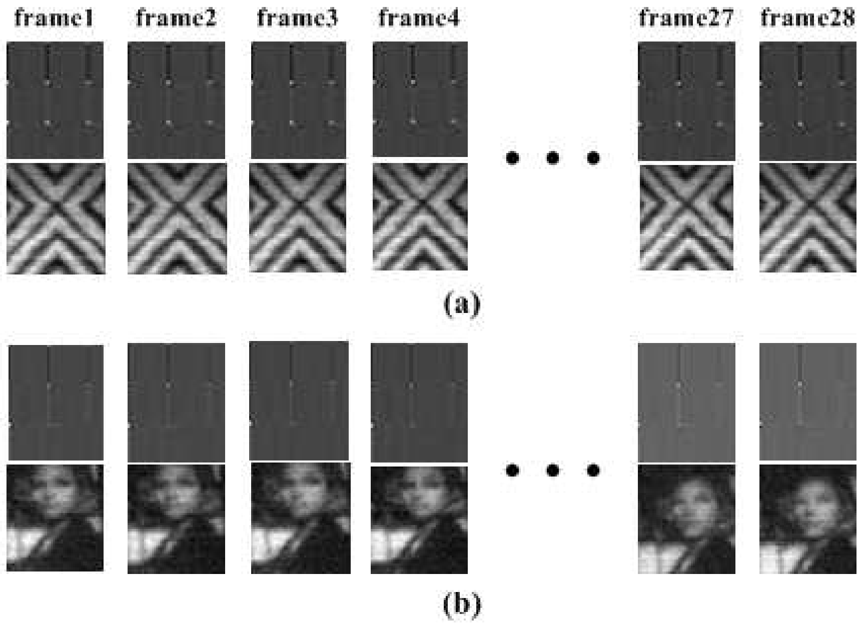


Figure 3.5. Image sequences (20 fps): (a) A 16×16 Walsh-Hadamard transformed image sequence and a reconstructed image sequence (b) A 16×16 DCT transformed image sequence and a reconstructed image sequence.

means the values can be precisely adjustable.

The noise model of a transform imager can be simplified as follows:

$$y(i_1, i_2) = (1 + m(i_1, i_2))x(i_1, i_2) + n(i_1, i_2) \tag{3.1}$$

where $x(i_1, i_2)$ is an input image, $y(i_1, i_2)$ is an output image, $m(i_1, i_2)$ is multiplicative noise, and $n(i_1, i_2)$ is additive noise. Fixed pattern noise (FPN) caused by device mismatches is one of the main sources of the multiplicative noise $m(i_1, i_2)$. Pixel noise and column amplifier noise also contribute to the multiplicative noise. The additive noise $n(i_1, i_2)$ is usually modeled as additive Gaussian noise, and the main sources of additive noise are dark current, shot noise, thermal noise, and quantization noise of ADCs.

The Result from one of the most recent transform imagers represent the effect of noise in DCT-based compression [35], and it is shown in Figure 3.6. It is shown that PSNR drops when more high-frequency coefficients are used for reconstruction in Figure 3.6. This is because the image contains a quite amount of noise, so the noise power in the high-frequency domain dominates the signal power. Therefore, the more noise is injected when more high-frequency DCT coefficients are used for reconstruction. It is also considered that throwing out more high-frequency coefficients works to filter out noise so that PSNR is improved.

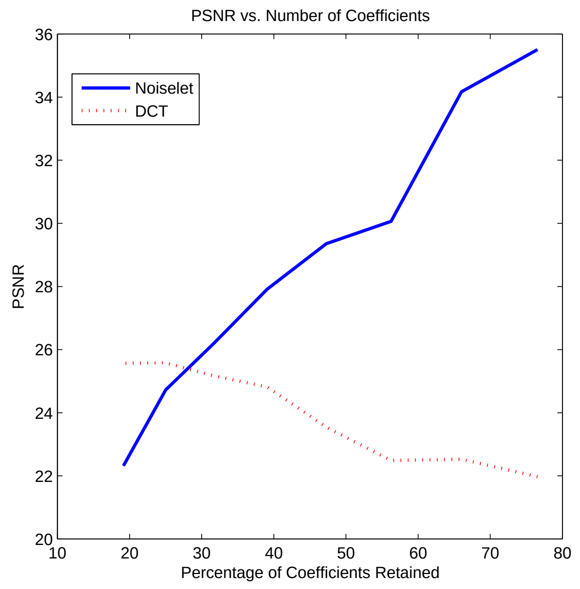


Figure 3.6. PSNR performance of DCT-based compression using a transform imager system [35]. PSNR drops when more coefficients are used for reconstruction.

The additive white Gaussian noise is a good candidate to model this case. The simplified noise model with an additive white Gaussian noise is used, and PSNRs corresponding to different compression ratios are shown in Figure 3.7. The mean and the standard deviation of the used additive white Gaussian noise are 0 and 0.9, respectively, and multiplicative noise is not applied in this case.

The additive Gaussian noise in the transform imager system can be removable by averaging multiple consecutive reconstructed images in the digital domain. This temporal averaging effectively suppresses time-varying additive noise, and the experimental results are depicted in Figure 3.8. The same 256×256 Lena image is used, and three images are averaged. PSNR performance is noticeably improved compared

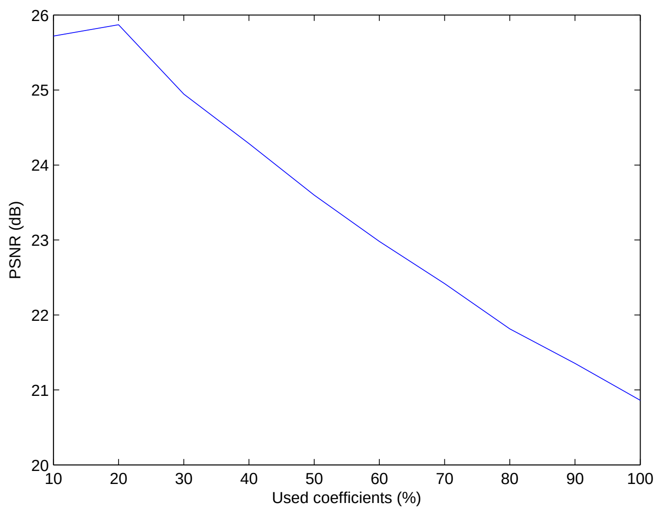


Figure 3.7. PSNR of DCT-based compression using the simplified noise model with additive white Gaussian noise. A 256×256 Lena image is used in this experiment.

to the results in Figure 3.7. PSNR still drops in Figure 3.7 when more than 80% of DCT coefficients are used. PSNR performance can be improved by averaging more images at the expense of increased computation and lower frame rates.

Time-invariant temporal FPN cannot be removed by averaging multiple images because time-invariant temporal FPN does not change over time. However, temporal FPN is computed before the imaging system is used, so removing temporal FPN is simply subtracting the pre-computed DC values from each pixel.

3.3 Zonal Mask Implementation in a Transform Imager System

In JPEG coding, a quantization weighting matrix, which is generally called a Q-matrix or Q-table, is applied to the DCT-transformed image blocks. The DCT-transformed block is divided by the predefined Q-matrix, and the elements of the divided matrix are rounded to the nearest integer values. This process makes most high frequency elements zeros, the number of which depends on the compression ratio. Therefore, a JPEG coder allocates more bits for the low frequency elements with higher magnitudes, and the low frequency elements are located around the left top

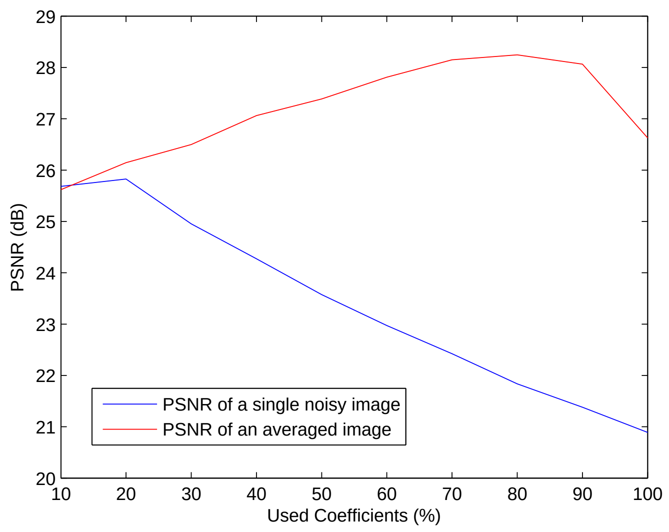


Figure 3.8. PSNR improvement of an averaged image of multiple reconstructed images. 256×256 **Lena image is used, and three noisy images are averaged.**

corner. A zonal mask is introduced based on this property of DCT-based image coding [36], [37]. Instead of applying a Q-matrix to the all elements in a DCT-transformed block, a mask is applied to the DCT-transform block, and only the filtered elements in the predefined zone are selected for encoding. Therefore, by using the zonal masking makes, applying a Q-matrix and rounding process are not required, and the number of elements that are coded is significantly reduced. This implementation is a compromise between the efficient implementation and the optimal bit allocation.

The examples of zonal masks are shown in Figure 3.9. The typical zonal mask, which is shown in Figure 3.9 (a), masks the low frequency elements on the left top corner, which is shown as a grey zone. The zonal mask is static, so the same zonal mask is applied to the all DCT-transformed blocks in an image [36]. Since the transform imager can perform a DCT in the imager, this zonal masking is simply a selective reading of the imager data. This selective image reading is straightforward to implement by adjusting the chip addressing scheme. The chip addressing scheme can be changed on the fly if it is necessary, so different zonal masks can be applied in the middle of reading the DCT-transformed image. Figure 3.9 (b) shows an example of different zonal masks, which can be designed for the specific target applications.

The other benefit of implementing this zonal mask in the transform imager compression system is that the amount of data reading from the imager is reduced based on the zonal mask. For example, if the zonal mask shown in Figure 3.9 (b) is used, only 25 elements out of 64 is required to be read from the imager. That is more than 60% reduction of data. Therefore, this makes the transform imager compression system suitable for the applications that require high frame rates or very low power consumption.

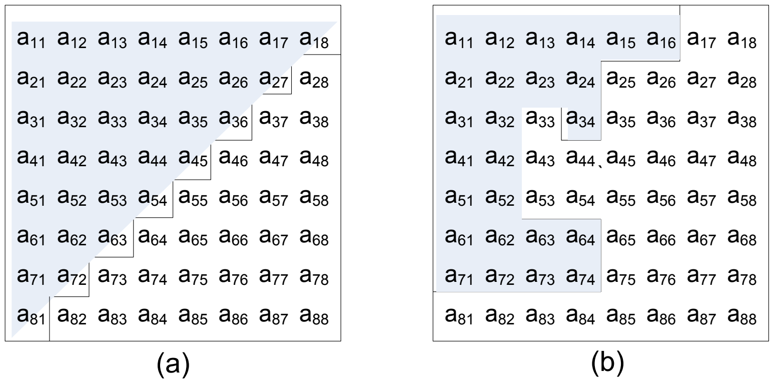


Figure 3.9. Zonal masks: (a) typical zonal mask (b) application-specific zonal mask.

CHAPTER 4

WAVELET-BASED EMBEDDED IMAGE COMPRESSION USING DYNAMIC INDEX REORDERING VECTOR QUANTIZATION

There have been several attempts to dynamically change the order of codebook indices or map onto the different index set to reduce entropy in image and video coding [38],[39],[40],[41]. Most of the algorithms are usually applied to the non-transform-based coding schemes for exploiting spatial redundancy among blocks. However, the majority of state-of-the art image coders are transform-based embedded coders such as EZW [6], SPIHT [42], and JPEG2000; and vector-quantization-based algorithms are no exception [7],[43].

In [7], da Silva *et al.* introduced a vector quantization (VQ) concept that is similar to EZW called a successive approximation vector quantizer (SAVQ) based on wavelet transforms. In SAVQ, codevectors represent the angles or patterns of wavelet coefficient vectors, and the wavelet coefficient vector is successively approximated. This VQ method achieved good performance compared to EZW and SPIHT, however, poses problems for index reordering or remapping algorithms to wavelet-based vector quantizers because of a lack of spatial redundancy. After an image is transformed and partitioned to vectors, the spatial correlation is significantly weakened. Furthermore, the irregular scanning order of embedded coding makes it more difficult to exploit the remaining spatial redundancy.

This chapter introduces a new concept of the temporal domain in embedded image coding to overcome the lack of spatial redundancy of SAVQ. After carefully analyzing the angle transitions of SAVQ, dynamic index reordering vector quantization

(DIRVQ) can be successfully applied in this temporal domain and improve the coding performance. Dynamic index reordering vector quantization (DIRVQ) [44] is one method of signal compression that provides excellent coding performance. However, it also requires significant computation and memory. This shortcoming will be addressed in Chapter 5.

4.1 Wavelet Transforms and Wavelet-Based Image Coding

A Fourier transform has been one of the most valuable tools in signal analysis, but it gives only frequency information. This means that the locations of the analyzed frequency components cannot be tracked in the time domain. A short-time Fourier transform (STFT)[45], also known as a time-dependent Fourier transform, is a Fourier-transform-based tool that overcomes this problem, and it is defined as [46]:

$$X[n, \lambda) = \sum_{m=-\infty}^{\infty} x[n+m]w[m]e^{-j\lambda m}, \tag{4.1}$$

where $w[n]$ is a window sequence. The STFT becomes very popular for time-frequency analysis in speech and audio processing. The visual representation of the magnitude of STFT $|X[n, \lambda)|^2$ is called a spectrogram, and it is now a indispensable tool in speech processing. However, the STFT uses the same-sized filters through all frequencies, so it has a fixed resolution. This fixed resolution is not desired in many applications especially in image processing since most information is concentrated in low frequencies. Therefore, more resolutions are demanded in the low frequency analysis.

Fourier-transform-based approaches for time-frequency analysis have been changed toward wavelet-based approaches since wavelets are introduced as an efficient tool for signal processing in 1987 by Mallat [47]. Unlike the STFT, wavelets use narrow filters in high frequencies and wide filters in low frequencies. Therefore, this new approach of wavelets makes it possible to analyze signals with various resolutions as shown in Figure 4.1.

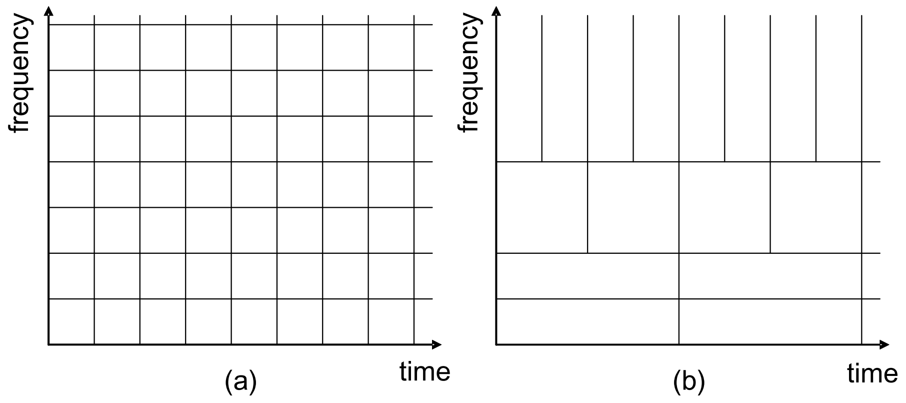


Figure 4.1. Time-frequency tiles of (a) short-time Fourier transform (STFT) (b) wavelet.

4.1.1 Wavelet Transforms

A continuous wavelet transform (CWT) W_f can be considered as mapping a function $f(x)$ onto time-scale space using the following wavelet $\psi_{ab}(t)$, which is dilations and translations of a mother function $\psi(t)$ [48],

$$\psi_{ab}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-b}{a}) \tag{4.2}$$

$$W_f(a,b) = \int_{-\infty}^{\infty} \psi_{ab}(t)f(t)dt =< \psi_{ab}(t), f(t) > . \tag{4.3}$$

A discrete wavelet transform (DWT) is the sampled version of the continuous wavelet transform with the sampling lattice, $a = a_0^m, b = nb_0a_0^m$, where $m, n \in Z$ [48]. Then the wavelet and the function $f(x)$ can be expressed as follows:

$$\psi_{mn}(t) = a_0^{-m/2}\psi(a_0^{-m}t - nb_0) \tag{4.4}$$

$$f(t) = \sum_m \sum_n d_{m,n}\psi_{mn}(t), \tag{4.5}$$

where $d_{m,n} =< f(t), \psi_{mn}(t) >= \frac{1}{a_0^{m/2}} \int f(t)\psi(a_0^{-m}t - nb_0)dt$.

Filter banks, which were developed in the early 1980's, are closely related to wavelets, and the major difference is the iteration process of wavelets [49]. Wavelets can be implemented using the filter banks for multi-resolution signal analysis. The scale function $\phi(t)$, which is defined as $\phi_{mn}(t) = 2^{-m/2}\phi(2^{-m}t - n)$, of wavelets and the band-pass wavelet function $\psi(t)$ can be expressed as follows: [48]

$$\phi(t) = 2\sum_{n} h_0(n)\phi(2t-n) \tag{4.6}$$

$$\psi(t) = 2\sum_{n} h_1(n)\phi(2t-n), \tag{4.7}$$

The coefficients $h_0(n)$ and $h_1(n)$ can be considered as the low-pass filter and the high-pass filter, respectively in the two-channel filter bank implementation as shown in Figure 4.2. In 2D discrete wavelet transforms that are used in image processing, these filter bank operations are performed along the rows and columns and shown in Figure 4.3. The three-level 2D discrete wavelet transform (DWT) result of a Lena image is shown in Figure 4.4.

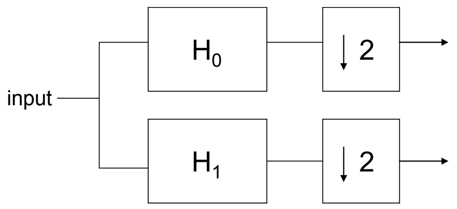


Figure 4.2. Two-channel filter bank.

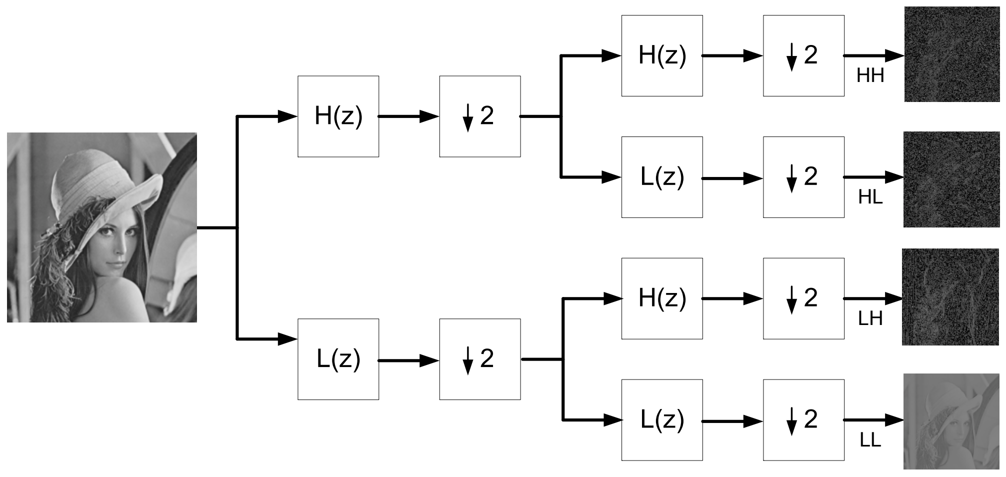


Figure 4.3. The one-level 2D filter bank results of a Lena image.

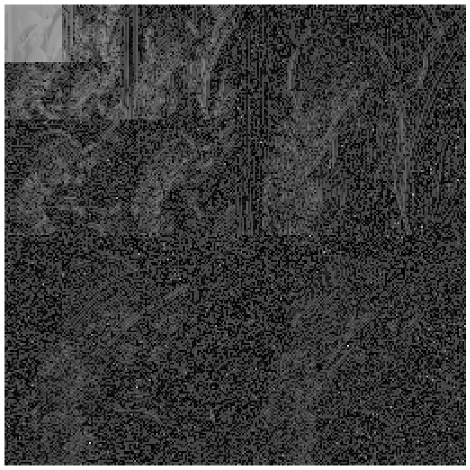


Figure 4.4. The three-level wavelet result of a Lena image.

4.1.2 Wavelet-Based Image Coding

In image coding, a discrete cosine transform (DCT) has played the key role in transform-based coding because of its simplicity and good performance. Nevertheless, the DCT suffers from a block artifact in low bit-rate coding. This block artifact is not a serious issue when the amount of data in image and video processing is small because the bit-rate can be maintained relatively high enough to avoid the artifact. However, modern image and video processing requires handling a vast amount of data to keep pace with the demand of higher resolution images and videos from consumers. This trend increases the necessity of very low bit-rate coding because of the limited storage space and transmission bandwidth especially in a mobile multimedia environment. Wavelets meet this expectation without the block artifact.

Like other transforms such as a DCT and a discrete Fourier transform (DFT), wavelets decorrelate the image data and concentrate the energy of image data into a few low-frequency coefficients. The distinct property to make wavelets different from other transforms is the capability to rearrange the image data structure efficiently. This rearranged data structure of wavelets gives more redundancy among different frequency subbands. Most wavelet-based image coding algorithms such as EZW, SPIHT, and JPEG2000 exploit this inter-subband redundancy.

4.1.3 Successive Approximation Vector Quantization (SAVQ)

Successive approximation vector quantization [7] can be considered as the vector version of EZW, which is briefly described in Section 4.1.2. Instead of successively approximating a target scalar value in EZW, SAVQ approximates a target vector using a VQ codebook as shown in Figure 4.5.

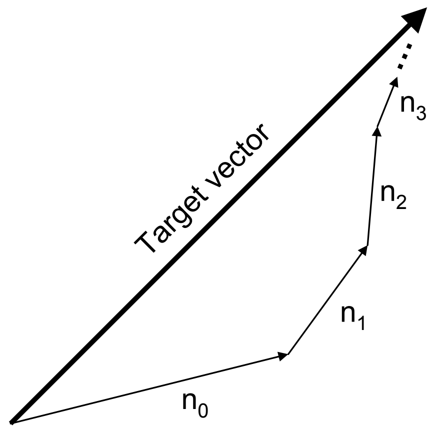


Figure 4.5. Successive approximation in a vector space.

The wavelet transformed image is divided with $M \times N$ blocks or vectors, and the magnitude of the each vector is calculated. The maximum magnitude T is used for the first approximation magnitude αT, where α is a constant scaling factor that has been determined based on the codebook. The symbols in SAVQ includes a zero (Z), a zero tree(ZT), and a coded value (C). They are similar to those of EZW except that differentiating the signs of the significant nodes are not necessary. Only positive significant symbol is used because the magnitude of a vector is always not negative.

One of the most difficult tasks associated with using SAVQ is to determine the approximation scaling factor, α. The performance is highly sensitive to the choice of α, but there is no analytic method to find the optimum. The experimental finding of the optimal α is well explained in [7], but it also slightly varies based on the target images. Improved SAVQ introduces a zero-vector and an escape code to alleviate the sensitivity issue by satisfying the following condition [50]:

$$\alpha^{n+1} X_{max} \leq \|\vec{r}_n\| \leq \alpha^n X_{max} \tag{4.8}$$

where X_{max} is the maximum magnitude of all the wavelet coefficient vectors, and $\|\vec{r}_n\|$ is the norm of the residual vector at pass n.

The basic idea of the improved SAVQ is sending zero-vector when the residual vector is smaller than the current threshold. Therefore, refinement process is omitted in the pass, and the estimated vector never goes beyond the target vector. When the residual vector is larger than the previous threshold, an escape code is sent instead. Once the decoder finds the escape code,the current threshold value is updated to the previous larger threshold value, and the next coded value, which contains the actual codebook value, is read and processed. This escape code makes it possible to track the target point more quickly.

Originally this improved SAVQ is introduced to overcome the sensitivity of choosing the α value and to give more robustness in the algorithm. In addition, it is found that the improved SAVQ gives more predictability of angle transition. Since the predictability gives more redundancy, wavelet-based embedded coding is applied to explore this redundancy. The details will be discussed in Section 4.3.

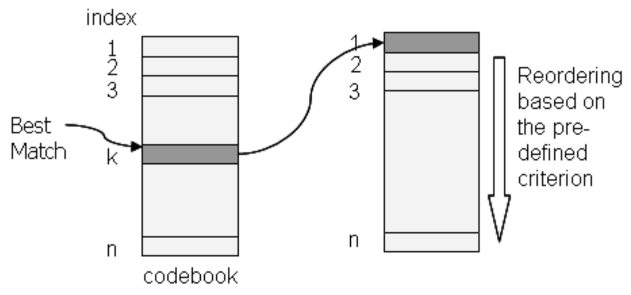
4.2 Dynamic Index Reordering Vector Quantization (DIRVQ)

Vector quantization is a compression scheme that encodes a set of data, which forms a vector, into a scalar index in the predefined codebook. Depending on the way of determining the indices, VQ can be categorized as either memoryless or memory-based. If the indices are determined only by the current input data set, the VQ is memoryless. On the other hand, if the indices are determined not only by the current input data set but also by the previous encoded indices, the scheme is considered VQ with memory. Since the VQ with memory exploits the redundancy between or among the pre-encoded indices, the coding performance is better than the memoryless VQ at the expense of increased computational complexity. VQ with memory has been intensively studied in many applications that the block data are highly correlated as in image processing such as address VQ, predictive VQ, finite-state VQ, and so on [39], [39], [41], [51].

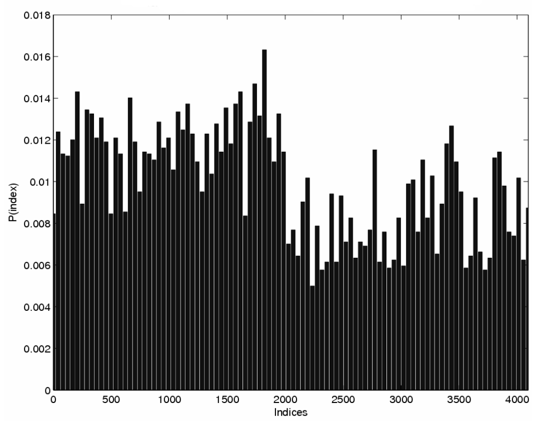
The concept of dynamic index reordering vector quantization [44] is shown in Figure 4.6. After choosing the best-matched codevector, the entire codebook is reordered based on the predefined reordering criterion as depicted in Figure 4.6 (a). For example, if the L_2 norm (Euclidean distance) is chosen for the criterion, every distance between the best-matched codevector and the other codevectors is calculated, and the codebook is reordered based on the distance. The chosen best-matched codevector is re-indexed to the index '0', and the remaining codevectors are re-indexed by their distance from the best-matched codevector, with closer codevectors having lower indices. The reordering process can be performed for the all codevectors in a codebook for the best performance or for only portion of the codevectors for compromise between the performance and the computational complexity. If the consecutive samples of the source have high correlation, the lower index has a higher probability of being chosen. Therefore, the indices are highly concentrated near index '0', as shown in Figure 4.6 (c). Since the probability of choosing the lower indices increases, the entropy is reduced, which means there is more room for increasing coding efficiency. The degree of concentration depends on how closely the samples are correlated.

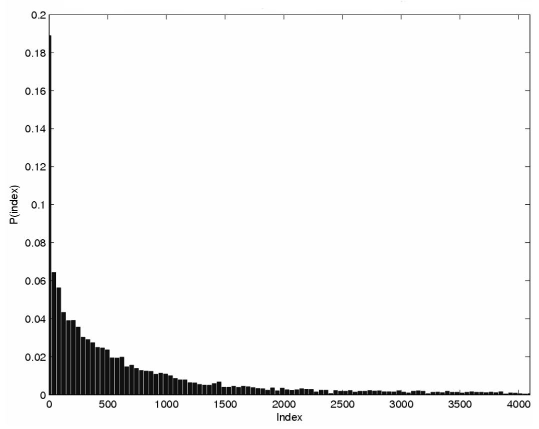
4.3 Temporal Dynamic Index Reordering Vector Quantization

A temporal dynamic index reordering vector quantization (TDIRVQ) is proposed for wavelet-based embedded coding. Note that the successive refinement process is defined as a temporal process here. The temporal updates are performed in every



(a)





(b) (c)

Figure 4.6. Dynamic index reordering: (a) concept; (b) the index probability distribution of 12-bit vector quantization with a computer generated Gaussian Markov source; (c) the index probability distribution after reordering with the L_2 norm criterion

refinement pass, and the updates of codevectors reflect the updates of angles for vector approximation. Because the approximation trajectory of SAVQ [7] is similar to that of the least-mean-square algorithm, temporal redundancy does not seem to be obvious. However, the redundancy becomes more clear in the improved SAVQ [50]. By carefully analyzing the angle transitions, dynamic index reordering vector quantization (DIRVQ) is successfully applied to the temporal domain, and the coding performance is improved.

4.3.1 Temporal Domain in Embedded Image Coding

In embedded image coding algorithms such as EZW, SPIHT, SAVQ, and vector SPIHT [43], the image data are usually called nodes because the data form a tree structure based on the relationship among subbands. The nodes, pixels in scalar quantization or vectors in vector quantization, are scanned to find which nodes have higher values than a predefined threshold. The found nodes are considered to be significant. Once one node is marked significant, the value of the node is continuously updated or approximated in every refinement pass. The values are binary numbers in scalar quantization and indices in vector quantization. Figure 4.7 shows the refinement process. Since the value corresponding to one node changes in each pass instead of location of the image, the passes are defined as a temporal domain in embedded image coding.

As shown in in Figure 4.7, the significant nodes A, B, C, D, E, and F are differently located in the spacial domain. However, one significant node is updated in every pass at the same spacial location as A_0, A_1, A_2, and so on. If this process is compared to a video playback, a vector in a significant node can be considered as a pattern in a video frame, and the vector updates in each pass can be considered as pattern changes in time.

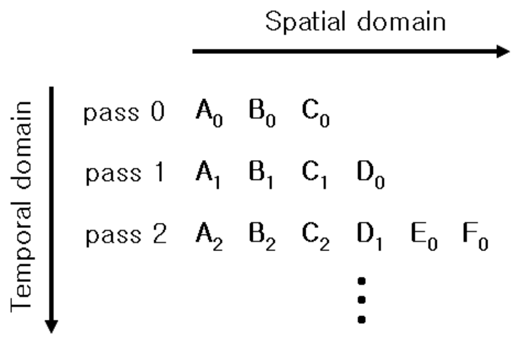


Figure 4.7. Temporal domain in embedded image coding: A, B, C, D, E, and F correspond to the significant nodes. The values of the nodes are successively approximated with refinement passes such as A_0, A_1, A_2, and so on.

4.3.2 Temporal Dynamic Index Reordering Vector Quantization

Improved SAVQ was designed to alleviate the sensitivity of the update coefficient α in SAVQ [50] as briefly mentioned in Section 4.1.3. However, improved SAVQ not only resolves the problem of SAVQ but also makes the successive approximation trajectory simpler. By introducing the zero-vector concept, the approximated vector never passes the final target vector, which is a wavelet coefficient vector, because the refinement procedure stops when the case happens. This prevents the algorithm from oscillating around the target point. Therefore, the angle transition becomes more predictable, that is to say, the angle transition has redundancy to exploit.

The following equation, which is simply the delayed version of Equation 4.8, can help one understand the simplified angle transition with Figure 4.8.

$$\alpha^n X_{max} \leq \|\vec{r}_{n-1}\| \leq \alpha^{n-1} X_{max} \tag{4.9}$$

If the current threshold, $\alpha^n X_{max}$, is greater than the magnitude of the current target vector, $\|\vec{r}_{n-1}\|$, a zero-vector index is coded. Therefore, no refinement update occurs at the pass. If $\|\vec{r}_{n-1}\|$ is greater than the previous threshold, $\alpha^{n-1} X_{max}$, an escape code is coded, and the previous larger threshold is used for the refinement update at the pass

The current threshold, $\alpha^n X_{max}$, is always smaller than the magnitude of the target

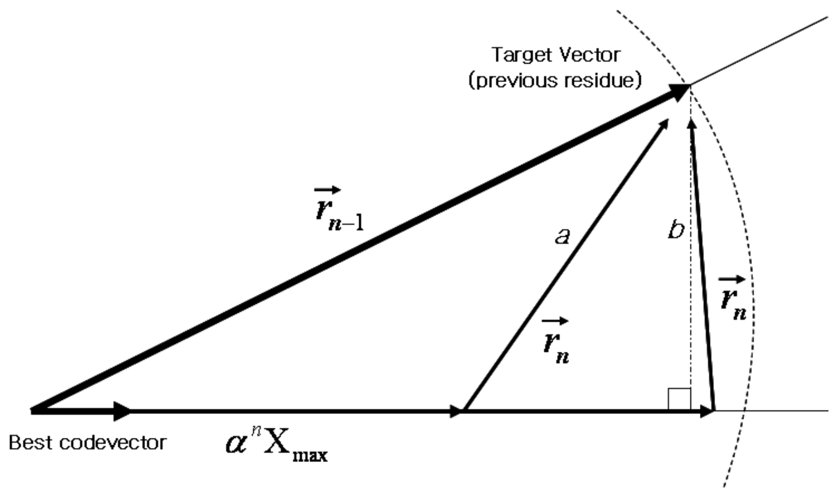


Figure 4.8. Vector successive approximation: The best-matched codevector with unit length is chosen based on the angle difference from the target vector $\vec{r}_{n-1}$, the previous residue, in pass n. The current threshold $\alpha^n X_{max}$ is multiplied to the chosen best-matched codevector. Then the residual vector $\vec{r}_n$ is calculated, and it becomes the target vector in the next pass

vector when the next residual vector, $\vec{r}_n$, is calculated. Therefore, the possible angles of the next residual vector are either a or b in Figure 4.8. The reason why the possible vectors are categorized as a and b is that a has a high probability of making an acute angle with the current codevector, and b has a high possibility of making an obtuse angle with the current codevector in the next pass. Through the experiments, the frequency of case a is generally much higher than the frequency of case b.

Even though the angle transition is not fully predictable, the trend of the angle transition can be estimated since the probability of acute angles is much higher. Therefore, the redundancy can be exploited by applying DIRVQ in the temporal domain. In this case, the reordering criterion is the angle difference, $\theta = cos^{-1}\frac{\vec{x}\cdot\vec{y_i}}{\|\vec{x}\|\|\vec{y_i}\|}$, where $\vec{x}$ is the best-matched codevector and $\vec{y_i}$s are other codevectors in the codebook. The algorithmic flow of TDIRVQ is described in Table 4.1.

4.3.3 Implementation of a TDIRVQ-Based Image Coder

TDIRVQ-based embedded image coding follows the conventional transform-based image compression procedure as described in Chapter 3 since it is also a transform-based

Table 4.1. The algorithmic flow of temporal dynamic index reordering vector quantization (TDIRVQ)

Encoder

Step 1: find the best-matched codevector from the initial codebook (absolute index)

Step 2: update the residual vector using the absolute index

Step 3: reorder the codebook based on the stored previous absolute index

Step 4: find the best-matched codevector from the reordered codebook and code the index (relative index)

Step 5: store the current absolute index

Decoder

Step 1: reorder the codebook based on the stored previous absolute index

Step 2: find the best-matched codevector using the decoded relative index

Step 3: update the wavelet coefficient vector using the best-matched codevector (vector addition)

Step 4: find the absolute index from the initial codebook

Step 5: store the current absolute index

algorithm. The diagram of TDIRVQ-based embedded image coding is shown in Figure 4.9. After the input image is transformed with a wavelet, the transformed coefficients are vectorized to the predefined block size such as 2×2 or 4×4. The converted vectors are preconditioned for the next TDIRVQ process; the magnitude of each vector is calculated, and the mean of each vector is subtracted. TDIRVQ is applied to the preconditioned vectors, which is normalized in the previous stage, and the chosen codevectors from the TDIRVQ process are entropy coded for the final output. The TDIRVQ process consists of multiple functional blocks; searching significants and running refined passes, encoding input vectors to generate the corresponding codevectors, and reordering the codebook based on the predefined criteria. Since the TDIRVQ-based image coding is an embedded image coding, the TDIRVQ process is recursive until reaching the target bit rates.

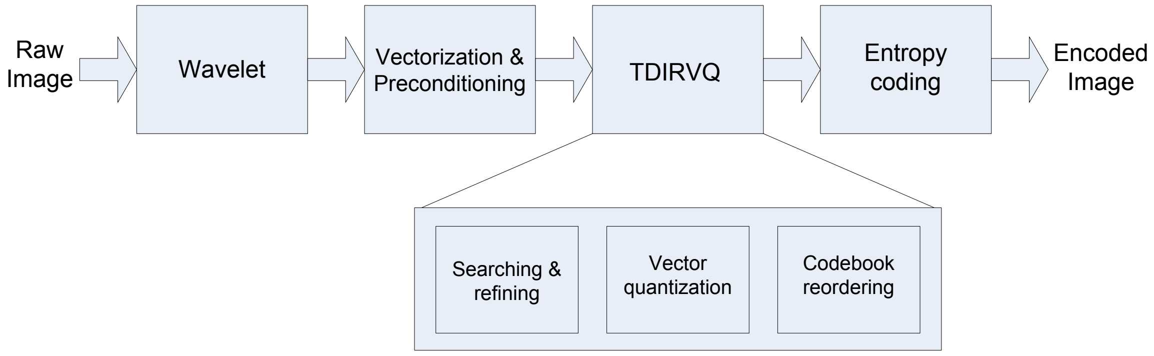


Figure 4.9. Diagram of TDIRVQ. The full procedure consists of wavelet-based embedded image coding and VQ with dynamic reordering.

4.3.3.1 Implementation of a TDIRVQ-Based Image Coder on a Transform Imager System

The efficient implementation of DWT coefficient computation is one of the most important topics in real-time wavelet-based image processing systems, so it has been intensively researched, especially on efficient VLSI implementations [52], [53] [54]. Even though some implementations are highly optimized for a specific wavelet, they require a considerable amount of either computational units or registers. They also show lack of flexibility because they are optimized only for one specific wavelet computation. Since the transform imager is capable of computing transforms, the block for computing the discrete wavelet transform (DWT) coefficients of input images, which is shown in Figure 4.9, is moved into the transform imager. Therefore, the digital computational unit for computing DWT coefficients is completely unnecessary. In addition, various wavelets can be applied in the system because the programmability of the transform imager. Other function blocks in Figure 4.9 are performed digitally. Since wavelets are not block-based transforms, rearrangement of the wavelet coefficients is required for programming the transform imager. More details about this programming will be discussed in Section 4.3.3.2.

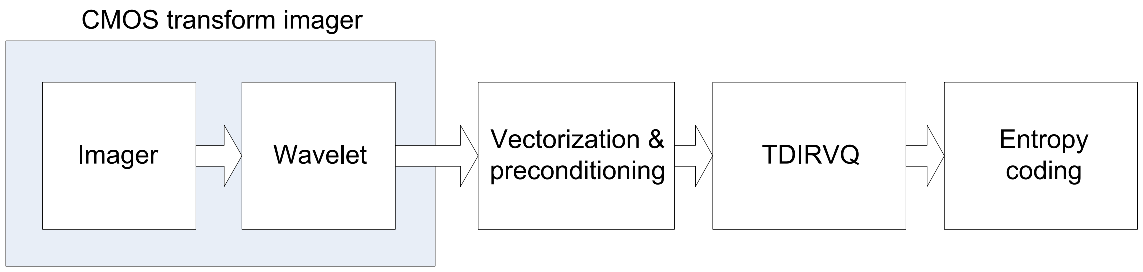


Figure 4.10. TDIRVQ implementation using a transform imager system. Full wavelet computation is performed in the transform imager, therefore, the outputs of the transform imager are wavelet coefficients. The other procedures are performed in the digital part of the system.

4.3.3.2 Wavelet Implementation using a Transform Imager

It is well known that wavelets can be implemented by filter banks, so the multi-level wavelets can be also implementable with tree-structured filter banks as shown in Figure 4.11 [55], [49]. Implementing filter banks can be divided into two sequential operations; finite impulse response (FIR) filtering for signal separation in the frequency domain and down-sampling to avoid increasing the size of signals. Since both FIR filtering and down-sampling are linear operations, the two cascaded operations can be combined to one operation.

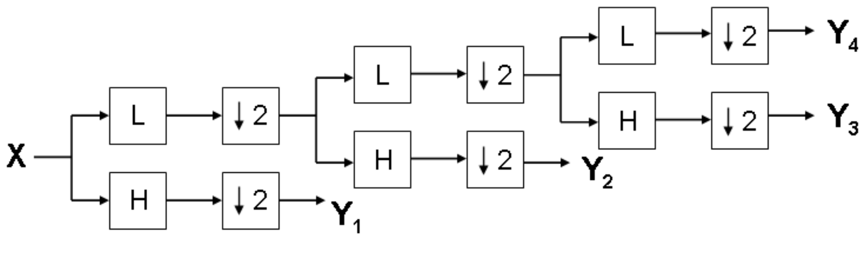


Figure 4.11. Three-level wavelet transform implementation using tree-structured filter banks.

If input signal x with length N, a low-pass FIR filter $L = [a_1, a_2]$, and a high-pass FIR filter $H = [b_1, b_2]$ are assumed, the following filters represent the combined low-pass and high-pass filters with the down-sampling operation, respectively [49]:

$$V = \begin{bmatrix} a_1 & a_2 & 0 & 0 & 0 & \cdots \\ 0 & 0 & a_1 & a_2 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} (\frac{N}{2} \times N), \tag{4.10}$$

$$W = \begin{bmatrix} b_1 & b_2 & 0 & 0 & 0 & \cdots \\ 0 & 0 & b_1 & b_2 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} (\frac{N}{2} \times N). \tag{4.11}$$

The above two filters can be represented in one square matrix, and the filter bank is fully described with the following matrix:

$$W = \begin{bmatrix} a_1 & a_2 & 0 & 0 & 0 & \cdots \\ 0 & 0 & a_1 & a_2 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ b_1 & b_2 & 0 & 0 & 0 & \cdots \\ 0 & 0 & b_1 & b_2 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} (N \times N). \tag{4.12}$$

Normalization is omitted in the above matrix representations for convenient understanding, but proper normalization is required to compensating the loss by down-sampling.

Multi-level wavelet can be considered as applying different FIR-filtering kernels to the image. Figure 4.11 shows that a three-level wavelet can be implemented by applying four FIR filtering kernels, Y_1, Y_2, Y_3, and Y_4, to the image. FIR filtering is basically a convolution operation with a finite-length kernel. Since the basic operation of the transform imager is the inner product of the programmed kernel coefficients with the pixel intensity values, performing FIR filtering using the transform imager is straightforward. But the size of the kernel is limited by the size of the kernel block implemented in the transform imager, and the output could suffer from the blocking effect. To overcome this blocking effect, a half-shifting scheme is implemented in the transform imager as in Figure 4.12. By overlapping half of the kernel block and arranging the kernel coefficients carefully using the modified multiplexer, the blocking effect is effectively removed.

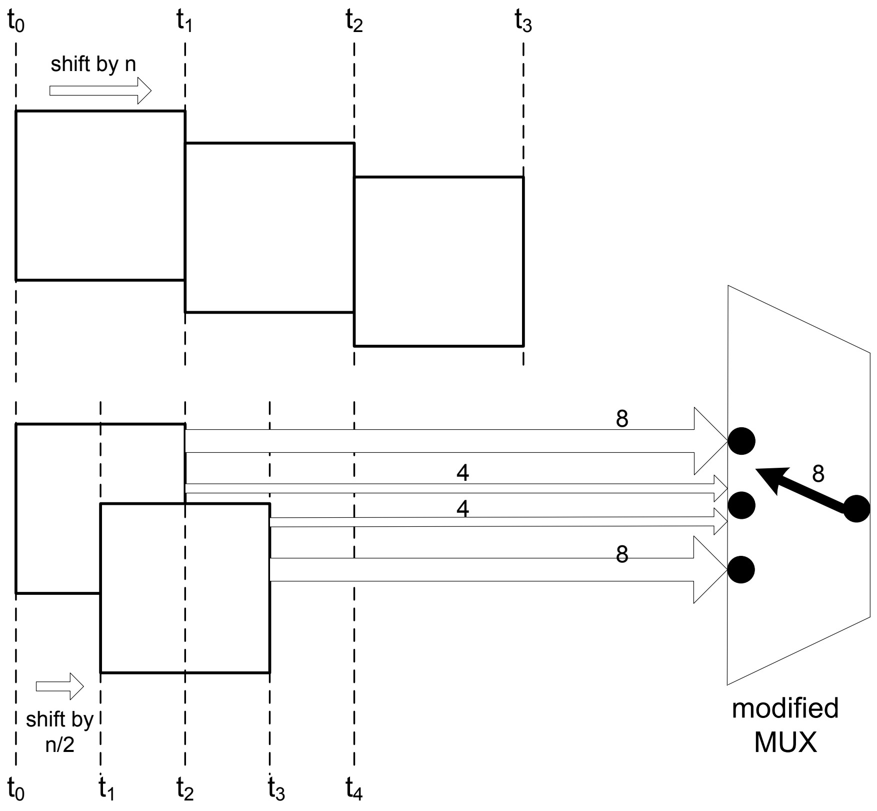


Figure 4.12. Half-shifting scheme to overcome block boundary

4.3.3.3 Wavelet-Based Denoising of TDIRVQ in a Transform Imager System

EZW is an embedded image coder, so only the wavelet coefficients above predefined thresholds, which is determined based on the desired bit rate, are coded. This is much like the wavelet-based thresholding technique, which is proposed by Donoho and Johnstone [56]. Once the wavelet coefficients above a threshold are chosen, those coefficients are successively refined in each pass. Therefore, applying EZW to an image can be considered as wavelet-based denoising with a soft thresholding.

On the other hand, the denoising property of TDIRVQ is different from that of EZW even though the basic idea of TDIRVQ comes from EZW. TDIRVQ is based on vector quantization, so the threshold of TDIRVQ means the magnitude of a wavelet coefficient block, not the magnitude of a wavelet coefficient. Therefore, the soft thresholding is performed on a block. TDIRVQ can still remove some noise in an image, but the block-based soft thresholding is not as effective as the coefficient-based soft thresholding in EZW. To overcome this limitation, threshoding can be performed after the transform imager as shown in Figure 4.13. The various thresholding techniques can be implemented in the thresholding unit such as a SureShrink [3] and a BayesShrink [57], but more complicated techniques add more complexity in the digital domain.

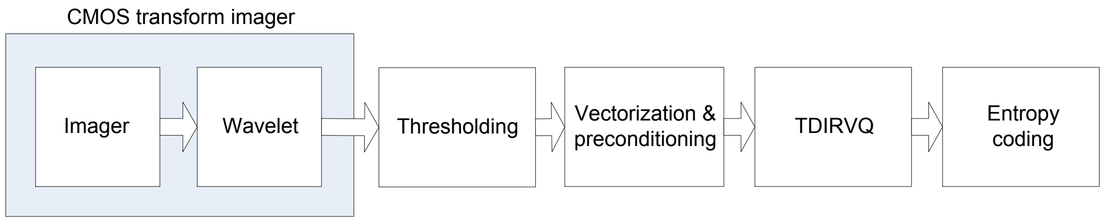


Figure 4.13. Denoising in TDIRVQ implementation: Adding a threshold unit after the transform imager can effectively remove the pixel-level noise.

4.3.4 Experimental Results of TDIRVQ

The five-stage Antonini wavelet transform is used for compressing an 8-bit gray scaled 512×512 images. The reason why the Antonini wavelet transform is used is for easy comparison with SAVQ [7], which also uses the same wavelet transform. The approximation scaling factor, α, is chosen as 0.547 for 2×2 successive approximation and 0.641 for 4×4 successive approximation. Experiments are performed with both the 2×2 vector size and the 4×4 vector size. For the four-dimensional case, the codebook is generated based on the D_4 lattice-shell, which makes 24 codevectors, to compare the results with improved SAVQ. For the sixteen-dimensional case, a modified Λ_16 lattice-shell is used for the codebook. This modified Λ_16 lattice-shell is designed for reducing the computational complexity of DIRVQ, and it will be explained thoroughly in Chapter 5. Extending the size to other dimensions like 4×2 is straightforward.

4.3.4.1 Index Probability Distribution and Entropy Reduction

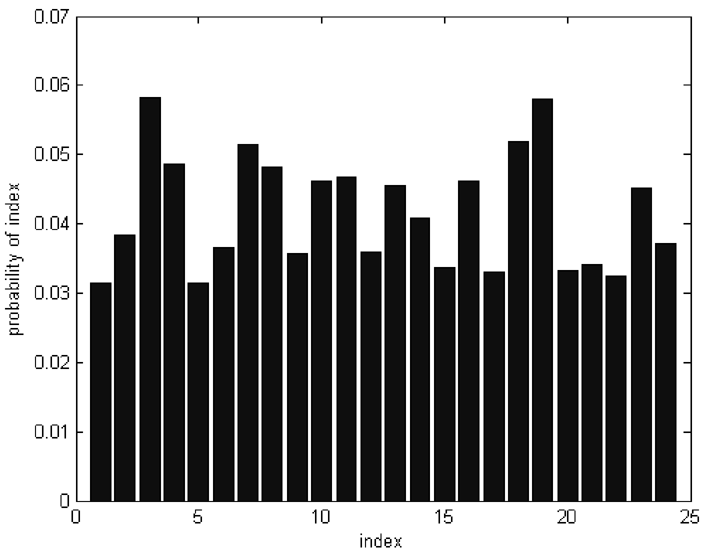
The probability distribution of indices for SAVQ is close to the uniform distribution as shown in Figure 4.14 (a). Figure 4.14 is for the case of 0.1 bit per pixel (bpp) of the Lena image, but the probability distribution of indices is almost constant for other compression ratios. Figure 4.14 (b) shows the probability distribution change of indices after applying TDIRVQ. The shape of the distribution is determined by two factors: The first is the relationship between the reordering reference vector, which is the previous best-matched codevector, and the approximated vector. The direction of the approximated vector is not usually the same as that of the reference vector, but they are related with angle offsets. The second is the maximum angle error caused by the vector quantizer. The maximum angle error of D_4 lattice-shell 1 is 45°.

Because entropy is maximized when the distribution is uniform, it is obvious that the entropy of indices is reduced. Therefore, performance improvement is expected. The probability of two extra indices, a zero-vector and an escape code, is omitted in the figures because they are the same in both cases. Entropy comparison for several tested images with a 2×2 block size is shown in Table 4.2. The first-order entropy is used and can be expressed as follows [58]:

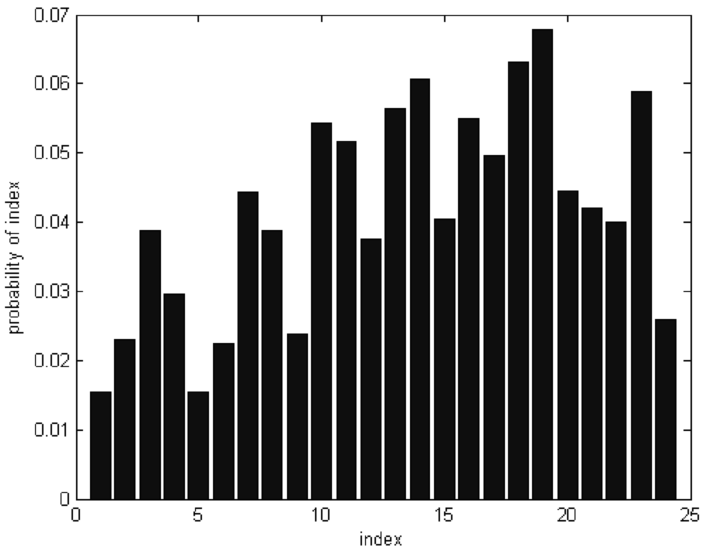
$$H(X) = -\sum_{x \in \chi} p(x) \log p(x), \tag{4.13}$$

where X is a discrete random variable with alphabet χ and p(x) is a probability mass function.

The probability of two extra symbols, a zero-vector and an escape code, are outliers in the index distribution, so entropy comparison for the 24 symbols excluding the extra



(a)



(b)

Figure 4.14. Index probability distribution for the case of 0.1 bit per pixel and 2×2 case: (a) Index probability distribution without TDIRVQ; (b) Index probability distribution with TDIRVQ.

symbols is shown in Table 4.2 as well to see the difference from the 26-symbol case. Note that TDIRVQ can be applied only to the vectors that are already chosen as significant. Newly chosen significant vectors are not approximated through TDIRVQ, which limits the amount of performance gain.

Table 4.2. Entropy reduction of test images by using TDIRVQ: Total number of used symbols for a 2×2 block is 26 including a zero-vector and an escape symbol. Entropy excluding the special symbols is also calculated and compared.

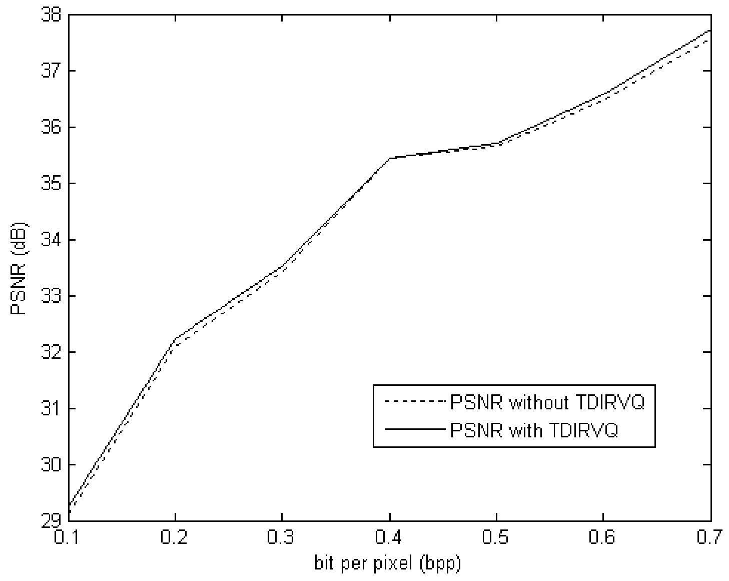
images	26 symbols		24 symbols	
(0.4 bpp, 512×512)	conventional VQ	TDIRVQ	conventional VQ	TDIRVQ
Lena	4.5988	4.5404	4.5619	4.4891
Baboon	4.6308	4.6167	4.5806	4.5635
Barbara	4.5327	4.4655	4.5705	4.4880
Peppers(256×256)	4.4964	4.4627	4.5390	4.4968
Boats	4.5881	4.5510	4.5668	4.5191
Fingerprint	4.4659	4.4326	4.5707	4.5285

4.3.4.2 Compression Results

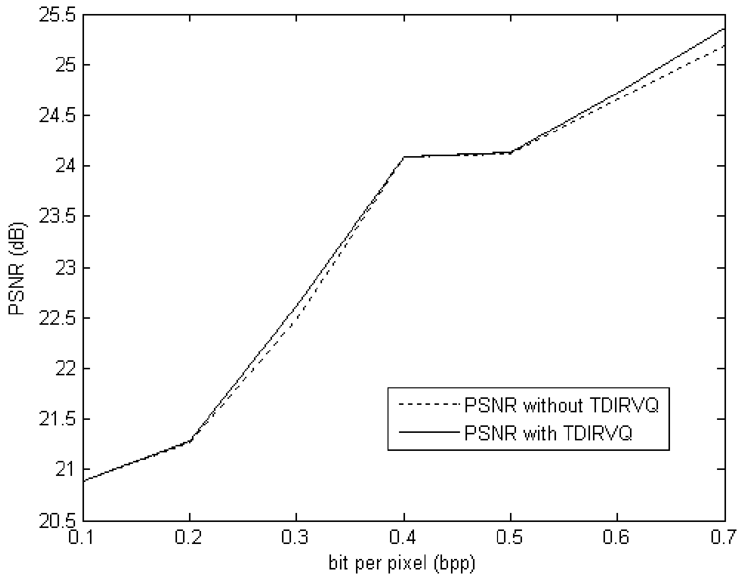
The PSNR performance improvement of the proposed algorithm is shown in Table 4.3, Figure 4.15, and Figure 4.16. PSNR improvement of D_4-based TDIRVQ, compared to the improved SAVQ with the Antonini wavelet transform, for the Lena image can be seen in Table 4.3, and the improvement ranges from about 0.05 dB to 0.18 dB. Here, PSNR follows the following definition:

$$PSNR = 10log_{10}\frac{\sum_{n_1}\sum_{n_2} 255^2}{\sum_{n_1}\sum_{n_2}(x(n_1,n_2)-\hat{x}(n_1,n_2))^2}, \qquad (4.14)$$

where $x(n_1, n_2)$ is the original image and $\hat{x}(n_1, n_2)$ is the coded image. Improvement for the highly detailed baboon image of D_4-based TDIRVQ and λ_{16}-based TDIRVQ are shown in Figure 4.15 (b) and Figure 4.16 (b), respectively. For all the tested bpps, the PSNR performance of the proposed algorithm outperforms that of SAVQ.

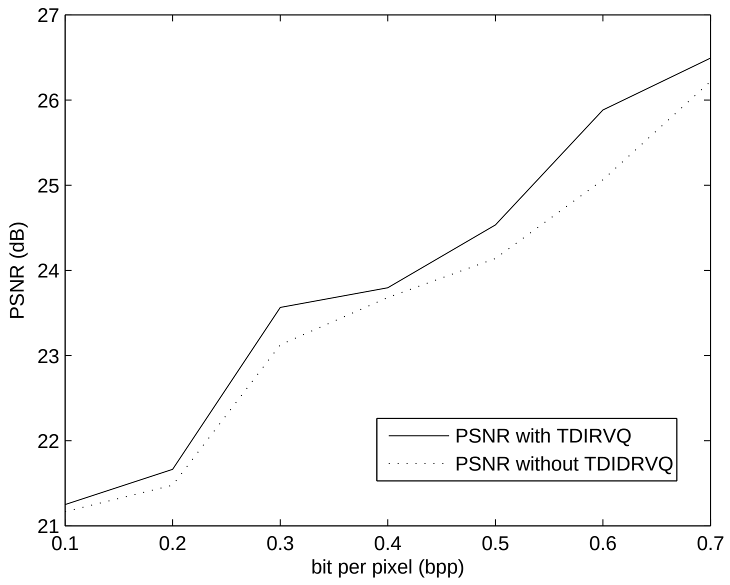


(a)

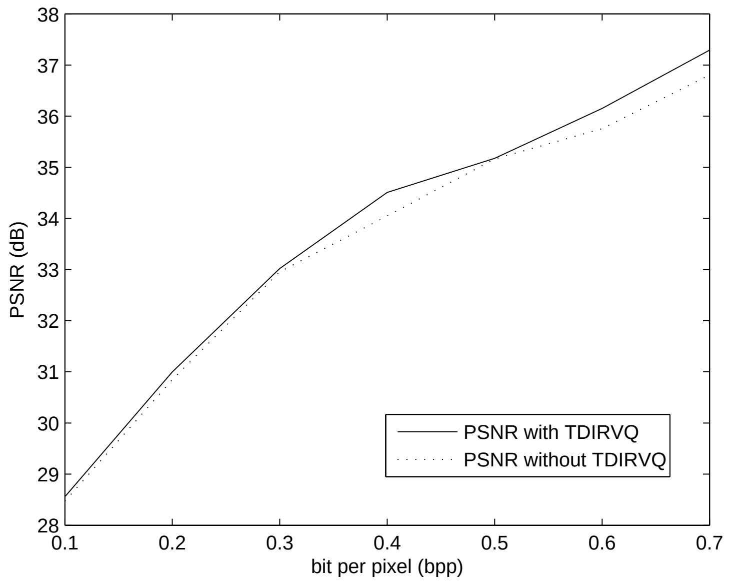


(b)

Figure 4.15. PSNR performance improvement of D_4-based TDIRVQ: (a) PSNR comparison for the Lena image; (b) PSNR comparison for the baboon image.



(a)



(b)

Figure 4.16. PSNR performance improvement of λ_{16}-based TDIRVQ: (a) PSNR comparison for the Lena image; (b) PSNR comparison for the baboon image.

Table 4.3. Performance comparison for Lena image (512×512)

	PSNR without TDIRVQ	PSNR with TDIRVQ
(bit/pixel)	(dB)	(dB)
0.1	29.0479	29.1528
0.2	31.9041	32.0852
0.3	33.3643	33.4195
0.4	35.3056	35.4788
0.5	35.6433	35.6929
0.6	36.4366	36.5200
0.7	37.3965	37.5491

CHAPTER 5

EFFICIENT IMPLEMENTATION OF LATTICE SHELL-BASED DIRVQ

In Chapter 4, dynamic index reordering vector quantization (DIRVQ) is applied to the temporal domain of wavelet-based embedded image coding. The performance improvement over the static vector quantization is noticeable and promising. However, as briefly mentioned in Chapter 4, DIRVQ requires a vast amount of computation to reorder the entire codebook.

One of the biggest merits of lattice VQ is that fast encoding and decoding is possible without searching a codebook [59],[60],[61],[62]. It is highly desirable in the case that the codebook size is very large. Storing the codebook in the memory and accessing it during each encoding or decoding step is not only computationally intensive but also power consuming and even more power consuming when the codebook is stored in the off-chip memory because the on-chip memory capacity of the processor is rarely sufficient. The off-chip memory is noticeably slower and more power-consuming than the on-chip memory. However, the fast encoding and decoding algorithms of lattice VQ become unfeasible when the dynamic index reordering scheme is applied, since indices should be encoded or decoded based on the reordered codebook whose order keeps being changed. Therefore, the fast encoding and decoding schemes in [59],[60],[61],[62] cannot be directly applicable to DIRVQ. However, abandoning the efficient schemes and performing encoding and decoding in the traditional fashion with the entire codebook cause too much additional cost both in more frequent memory accesses and in excessive computation because of reordering. This chapter is about how to overcome this computational overhead by efficient encoding and decoding algorithms.

5.1 Efficient Implementation of Lattice Shell-Based DIRVQ

The main reason for the high complexity of DIRVQ is the reordering of the entire codebook, which dynamically changes the order of the codebook in the middle of the encoding or decoding process. Since the existing efficient encoding and decoding schemes [59],[60] for lattice VQ are for static codebooks, DIRVQ cannot take advantage of them.

When the reordering criterion is angular distance as in TDIRVQ, reordering a codebook can be considered as rotating the coordinates of the lattice points. In addition, lattice-based codebooks are highly structured. For example, the D_4 lattice-based codebook has only three kinds of elements, -1, 0, and -1, and each codevector has two non-zero elements and two zero elements. The concept of coordinate rotation of angular-distance-based DIRVQ and the highly-structured codevectors in lattice shells play key roles in the proposed efficient implementation of lattice shell-based dynamic index reordering vector quantization (L-DIRVQ).

5.1.1 Optimal Lattices

When VQ is used in the gain-shape fashion as in SAVQ and TDIRVQ, the codevectors only represent the pattern or direction of the block without considering the magnitude that has already been extracted, so using the codevectors with iso-norm is a natural choice. The iso-norm forms a shell in the lattice, and the problem of finding the optimal lattice becomes the well-known kissing number problem. The kissing number problem can be defined as how many spheres can be arranged so that they all just touch another sphere of the same size [63]. There has been in-depth research on the optimal lattice, which make the maximum kissing number, in many dimensions, and the found lattices are listed in Table 5.1[63]. For the dimension $n \leq 4$, $n = 8$, and $n = 24$, optimum lattices are found with proofs. No proof is found for the other dimensions. However, the found lattices are widely accepted as optimal lattices. In the image processing, the popular block sizes of VQ are 2×2 and 4×4, which correspond to the dimension 4 and 16, respectively. The optimal lattices for those dimensions are D_4 and Λ_{16}. Therefore, efficient encoding and decoding algorithms and implementation for D_4 and Λ_{16} lattice VQ are presented.

Table 5.1. Optimum lattices

n	Lattice	Kissing number
1	A_1	2
2	A_2	6
3	D_3	12
4	D_4	24
5	D_5	40
6	E_6	72
7	E_7	126
8	E_8	240
$\vdots$	$\vdots$	$\vdots$
16	Λ_{16}	4320
$\vdots$	$\vdots$	$\vdots$
24	Λ_{24}	196560

5.1.2 Cyclic codes

Cyclic linear codes C can be defined as follows: [64]

$$\forall_{(c_0,c_1,\ldots,c_{n-1})\in C}[(c_{n-1},c_0,c_1,\cdots,c_{n-2})\in C]. \tag{5.1}$$

C is cyclic if C contains all right-shifted codewords of its codewords. For example, a code $C_1 = \{(000),(110),(011),(101)\}$ is cyclic. Any shifted codeword exists in the code C_1. On the other hand, a code $C_2 = \{(000),(100),(010),(001),(101)\}$ is not cyclic since the shifted codewords of (101) such as (110) and (011) are not included in the code C_2.

The cyclic property of lattice-based codebooks is the main idea of the proposed efficient encoding and decoding algorithms for DIRVQ. As mentioned earlier, the reordering process of DIRVQ can be considered as the coordinate rotation. Therefore, if a set of codevectors forms a cyclic code and has the same weight or magnitude, which is always true since the codevectors have an iso-norm in L-DIRVQ, the coordinates of the rotated codevectors are re-aligned exactly with the non-rotated coordinates they were located before. One example of this is depicted in Figure 5.1. Since the codebook is cyclic, the structure of all the codevectors from the view of the top-indexed codevector (#1) remains the same even after reordering.

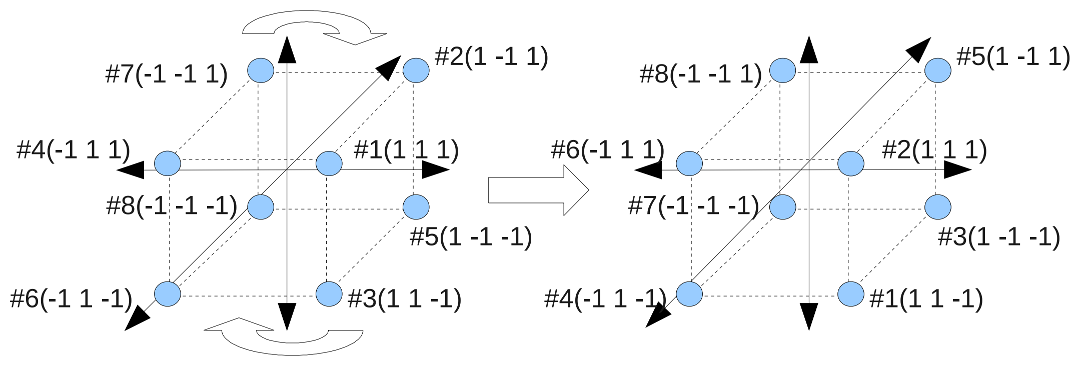


Figure 5.1. Cyclic codes: If the codevectors have an iso-norm and cyclic property, the reordered codevectors are re-aligned like the way they were located before reodering. $\#1, \#2, ...$, **and** $\#8$ **indicate index numbers.**

5.2 Efficient Implementation of 4-Dimensional L-DIRVQ

A novel encoding and decoding algorithms for L-DIRVQ based on D_4 sphere packing intensively use the property of cyclic codes. The algorithm includes the masking technique and inner product-based grouping. Note that this algorithm can be usable for gain-shape type VQ such as L-DIRVQ, whose reordering criterion is angular distance. Since the algorithm significantly reduces the necessity of storing the entire codebook, they can be considered as either a codebookless approach or a reduced codebook approach.

A 4-dimensional lattice D_4 can be defined by the following generator matrix [63]:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/2 & 1/2 & 1/2 & 1/2 \end{bmatrix}, \tag{5.2}$$

The combination of the rows of M generates all possible codewords. Minimal vectors are all permutations of $(\pm1, \pm1, 0, 0)$. The D_4 lattice-based codebook with minimal vectors is shown in Table 5.2, and this codebook is used for the TDIRVQ experiments.

Finding the best-matched codevector from the input vector is straightforward in the D_4 case. For example, since only two elements in the codevector are non-zero in the codevectors, two elements with the largest absolute values in the input vector are picked, and then change them to 1 or -1 depending on their original signs. The other elements are changed to 0 like the following:

$$[3826, \ -34, \ -565, \ 89] \Longrightarrow [\ 1, \ 0, \ -1, \ 0] \tag{5.3}$$

This simple process generates the codevector that is best matched to the input vector. However, finding the index corresponding to the found codevector from the reordered codebook is not straightforward at all, since the codebook order does not remain the same. Therefore, the procedure on how to find the index from the reordered codebook is the most difficult obstacle to overcome. Decoding from the received index is exactly the opposite of the encoding process, so the explanation of the detailed decoding procedure will be omitted.

5.2.1 Inner Product-Based Grouping

When the angular distance is used for the reordering criterion in L-DIRVQ, the distance between two codevectors can be characterized by the inner product. The angle between two vectors, $\vec{x}$ and $\vec{y}$, can be expressed as follows:

$$\theta = cos^{-1}\frac{\vec{x}\cdot\vec{y}}{\|\vec{x}\|\|\vec{y}\|}. \tag{5.4}$$

Assuming the magnitude of the vectors is normalized, the angle θ fully depends on the inner product of the vectors. The farther the distance is, the larger the theta is. The codevectors of D_4 L-DIRVQ have an iso-norm, which means the magnitudes of the codevectors are normalized. The codevectors that are composed from a shell of D_4 sphere are symmetrically structured, so the inner product value between the codevectors forms only a limited number of cases. For example, only four inner product values, which are 3, 1, -1, and -3, can be found in the codevectors shown in Figure 5.1.

The 24 codevectors that are generated from a shell of D_4 sphere are shown in Table 5.2. The previous best-matched vector is set to [0 0 1 1], which is indexed as 0, and the other codevectors are ordered based on the inner product values. As seen in Table 5.2, all the codevectors are categorized in group 0, 1, 2, or 3. Therefore, once the inner product value is calculated and the corresponding group is found, the index search region is significantly reduced. After the search region is found, the masking technique is applied to the region.

5.2.2 Masking Technique

As discussed, the main difficulty of applying the codebookless encoding algorithm to DIRVQ is that the codebook order keeps being changed. However, if only the values in the position of the non-zero elements of the previous best-matched vector are considered, and the other elements are processed separately, the coordinate changes caused by reordering are nullified. The previous best-matched codevector without considering the signs of its elements is defined as a mask. The masking procedure is shown in Figure 5.4. This procedure is possible because the codebook generated from the D_4 lattice shell forms hyperspheres with equally distanced points and the

Table 5.2. The codebook generated from the first shell of the D_4 sphere

index	codevector	inner product with the best-matched codevector (currently index 0)	group
0	[0, 0, 1, 1]	2	
1	[1, 0, 1, 0]	1	0
2	[0, 1, 1, 0]	1	
3	[0, -1, 1, 0]	1	
4	[-1, 0, 1, 0]	1	
5	[1, 0, 0, 1]	1	
6	[0, 1, 0, 1]	1	
7	[0, -1, 0, 1]	1	
8	[-1, 0, 0, 1]	1	
9	[0, 0, 1, -1]	0	1
10	[1, 1, 0, 0]	0	
11	[1, -1, 0, 0]	0	
12	[-1, 1, 0, 0]	0	
13	[-1, -1, 0, 0]	0	
14	[0, 0, -1, 1]	0	
15	[1, 0, 0, -1]	-1	2
16	[0, 1, 0, -1]	-1	
17	[0, -1, 0, -1]	-1	
18	[-1, 0, 0, -1]	-1	
19	[1, 0, -1, 0]	-1	
20	[0, 1, -1, 0]	-1	
21	[0, -1, -1, 0]	-1	
22	[-1, 0, -1, 0]	-1	
23	[0, 0, -1, -1]	-2	3

codebook is cyclic. The masking technique is performed with predefined comparison templates. These templates can be considered as another form of a codebook even though the size is a lot smaller than that of the original codebook. Therefore, the proposed encoding algorithm can be also considered as a reduced codebook algorithm instead of a codebookless algorithm.

After finding the group of the input vector based on inner product computation, the mask is applied to the input vector, and successive comparisons are performed

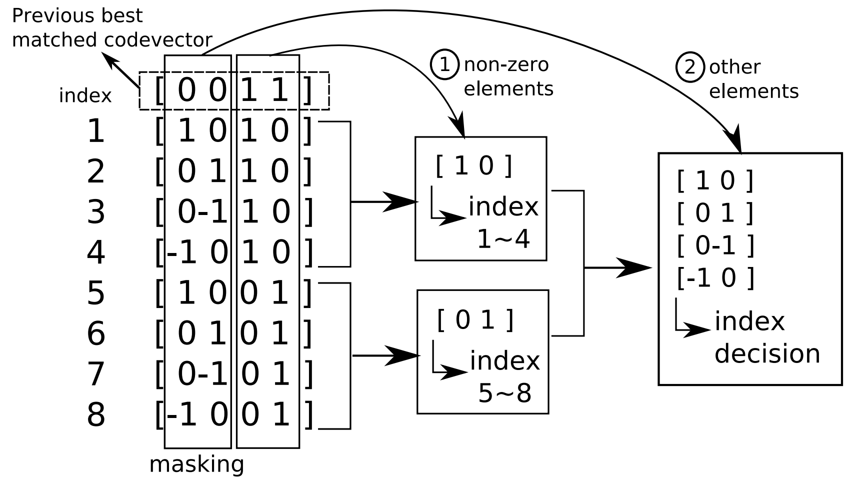
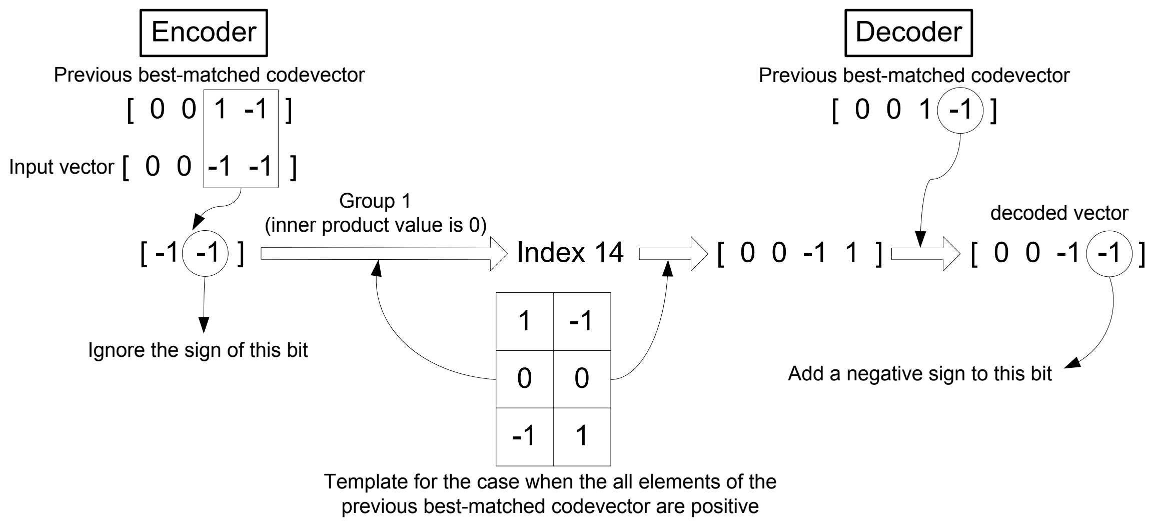


Figure 5.2. Masking the position of zero elements - mask the position of zero elements of the previous best-matched vector and calculate the inner product to decide the group, and then refine the decision based on the elements at the previously masked position

with predefined templates. For example, assume the previous best-matched codevector is [0, 0, 1, 1] as in Figure 5.4. If the group is determined as 0, which means the inner product value is 1, the possible cases of the masked bits are {[1, 0],[0, 1]}. The set {[1, 0],[0, 1]} is a template for this case. However, if the previous best-matched codevector is [0, 0, -1, 1], the possible cases of the masked bits for the group 1 are changed to {[-1, 0],[0, 1]}. This change means that multiple templates are necessary for the different sign combinations of the previous best-matched codevector. Using multiple templates for each possible combination is not desirable because it significantly increases memory consumption. The solution is ignoring the signs in the masked bits if the corresponding bits of the best-matched codevector are negative. Then only the templates for positive sign combination are necessary. In a decoding process, these ignored sign bits need to be added to the found vector based on the previous best-matched codevector. This additional step does not add much computational complexity. Figure 5.3 explains this procedure with an example.



Encoder
Previous best-matched codevector
[0 0 1 -1]
Input vector [0 0 -1 -1]
[-1 -1]
Ignore the sign of this bit
Group 1
(inner product value is 0)
Index 14
[0 0 -1 1]
1 -1
0 0
-1 1
Template for the case when the all elements of the previous best-matched codevector are positive
Decoder
Previous best-matched codevector
[0 0 1 -1]
decoded vector
[0 0 -1 -1]
Add a negative sign to this bit

Figure 5.3. Omitting and recovering the signs of the previous best-matched codevector: Input vector [0 0 -1 -1] is categorized as the group 1 if the previous best-matched codevector is [0 0 1 -1]. The masked bits are [-1 -1], and the sign of the last bit is ignored because the corresponding bit of the previous best-matched vector is negative. Therefore, the masked bits become [-1 1]. Using this modified bits and the template, which is for the case when all elements of the previous best-matched codevector are positive, the input vector is encoded as a index 14. The decoder will generate a vector [0 0 -1 1] using the same template. Reverse process of ignoring the sign of the last bit can recover the minus sign of the last bit, and the final decoded vector is [0 0 -1 -1]. The location of the ignored sign bit can be found from the location of the negative number in the previous best-matched codevector.

5.2.3 Example

The following example will help understanding the full encoding algorithm. Assume the input vector is [3826, -34, -565, 89] as in Equation 5.3, and the current codebook status is shown in Table 5.2, which means the previous best-matched codevector is [0, 0, 1 1] and indexed as 0. Since the two elements with the largest absolute values in the input vector is 3826 and -565, the corresponding codevector is [1, 0, -1, 0] as in Equation 5.3,

$$[3826, \ -34, \ -565, \ 89] \Longrightarrow [\ 1, \ 0, \ -1, \ 0]. \tag{5.5}$$

Two non-zero elements become either 1 or -1 based on the original signs of the chosen elements with the largest absolute values. The other elements become 0. The next step is finding the codebook index corresponding to the found codevector. The inner product of the found codevector [1, 0, -1, 0] and the previous best-matched codevector

[0, 0, 1 1] results in -1. The inner product result determines the group of the target index, and the group corresponding to -1 is the group 2 in Table 5.2. Since the codevector is located in the group 2, the possible indices should be located between 15 and 22. After finding this region, the masking technique is applied. The non-zero elements of the previous best-matched codevector [0, 0, 1 1] are the third and the forth elements. The location of the non-zero elements is defined as a nz-zone. The third and the fourth elements of the found codevector [1, 0, -1, 0] is processed as a two-bit vector [-1, 0] after masking out the first and the second elements.

$$[\ 1,\ 0,\ -1,\ 0] \Longrightarrow [-1,\ 0] \tag{5.6}$$

$[-1,\ 0]$ makes the search region reduced more to between 19 and 22 as shown in Figure 5.4. The unmasked two-bit vector [1, 0] is used for the final search. The location of the zero elements is defined as a z-zone. The vector [1, 0] is compared with the two-bit vectors in the z-zone and determines the final index. In this example, the target index is 19 as shown in Figure 5.4. The full encoding procedure is shown in Figure 5.5.

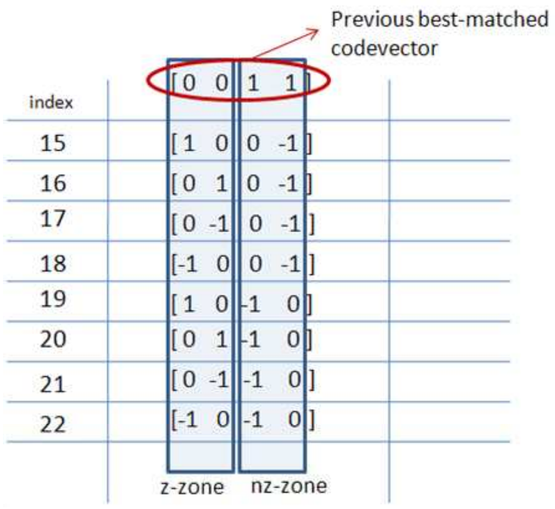


Figure 5.4. Template matching in nz-zone and z-zone: The locations of the non-zero elements in the previous best-matched codevector are used for masking the codevectors, and the masked zone is defined as a nz-zone. Following the same method, the zone from the locations of the zero elements is defined as a z-zone.

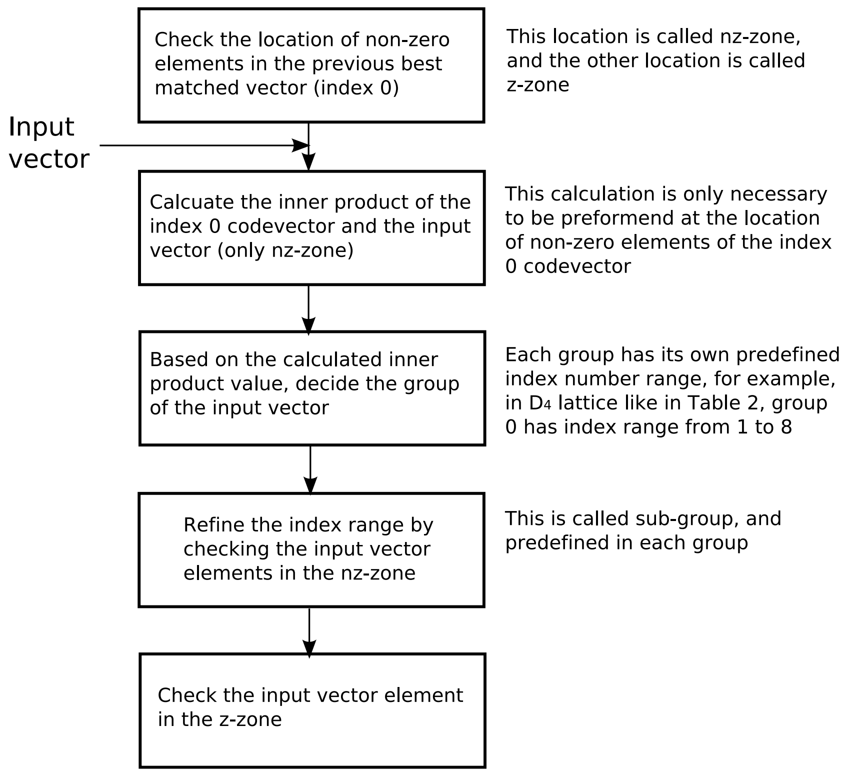

Check the location of non-zero elements in the previous best matched vector (index 0)
This location is called nz-zone, and the other location is called z-zone
Input vector
Calcuate the inner product of the index 0 codevector and the input vector (only nz-zone)
This calculation is only necessary to be preformend at the location of non-zero elements of the index 0 codevector
Based on the calculated inner product value, decide the group of the input vector
Each group has its own predefined index number range, for example, in D4 lattice like in Table 2, group 0 has index range from 1 to 8
Refine the index range by checking the input vector elements in the nz-zone
This is called sub-group, and predefined in each group
Check the input vector element in the z-zone

Figure 5.5. The diagram of the full encoding procedure

5.2.4 Computational Reduction

Instead of the brute-force search, the proposed algorithm, which finds the two elements with the largest absolute values and substitutes the values with 1 or -1 based on the original sign, significantly reduces the amount of computation. The brute-force search requires 24 inner products in the worst case. In fact, even in the best case, the 24 inner products are required to performed for reordering. The entire codebook is reordered based on the 24 inner product values. Considering one inner product operation requires 4 multiplications and 3 addition, 24 inner products need 96 multiplications and 72 additions. In addition, all inner product values should be stored in memory because the reordering process is based on them. On the other hand, the proposed encoding algorithm needs one inner product between the found codevector and the previous best-matched codevector, one masking operation, and two two-bit comparisons. This computational reduction is summarized in Table 5.3.

Table 5.3. Computational reduction of the proposed D_4 lattice-based codebook encoding and decoding algorithm. Note that one inner product requires 4 multiplications and 3 additions.

	Brute-force method	Proposed algorithm
Codebook search	24 inner products, 23 comparisons (96 multiplications and 72 additions)	2 max functions
Reordering	sorting	two-bit masking 2 two-bit comparisons

5.3 Efficient Implementation of 16-Dimensional L-DIRVQ

The Barnes-Wall lattice Λ_{16} can be constructed based on the first-order Reed-Muller code of length 16 with the following generator matrix [63]:

$$M = \frac{1}{\sqrt{2}} \left[\begin{array}{cccccccccccccccc}
4 & & & & & & & & & & & & & & & \\
2 & 2 & & & & & & & & & & & & & & \\
2 & 0 & 2 & & & & & & & & & & & & & \\
2 & 0 & 0 & 2 & & & & & & & & & & & & \\
2 & 0 & 0 & 0 & 2 & & & & & & & & & & & \\
2 & 0 & 0 & 0 & 0 & 2 & & & & & & & & & & \\
2 & 0 & 0 & 0 & 0 & 0 & 2 & & & & & 0 & & & & \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & & & & & & & & \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & & & & & & & \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & & & & & & \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & & & & & \\
1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & & & & \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & & & \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & & \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array} \right]. \tag{5.7}$$

The kissing number is 4320. The minimal codevectors are 480 vectors of all permutations of $2^{-1/2}(\pm 2^2, 0^{14})$ and 3840 vectors of the form $2^{-1/2}(\pm 1^8, 0^8)$, in which the ± 1s are located in the 1's positions of the first-order Reed-Muller code with weight 8, and the total number of minus signs in a codevector should be even. For example, one of the first-order Reed-Muller codewords with weight 8 is [1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 0]. The following two example vectors meet the requirements of the Λ_{16} lattice-based codevector:

$$\begin{gathered}[1,\ 1,\ -1,\ -1,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0] \\ [1,\ 1,\ -1,\ -1,\ 0,\ -1,\ 0,\ 1,\ -1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0].\end{gathered} \tag{5.8}$$

Note that the number of negative elements in the codevectors should be even.

Unlike the D_4 lattice-based codevectors, the Λ_{16} lattice-based codevectors are not cyclic because the 3840 vectors of the form $2^{-1/2}(\pm 1^8, 0^8)$ are based on the Reed-Muller code, which is not a cyclic code. The remaining 480 vectors of $2^{-1/2}(\pm 2^2, 0^{14})$ are cyclic. Therefore, it is not straightforward to apply the masking technique, which is used for the 4-dimensional case, to this 16-dimensional case because the masking technique takes advantage of the cyclic property. Even the inner product-based grouping is not as simple as the case of the D_4 lattice because 480 vectors of the form $2^{-1/2}(\pm 2^2, 0^{14})$ have different structure from 3840 vectors of the form $2^{-1/2}(\pm 1^8, 0^8)$.

5.3.1 Cyclic Reed-Muller Codes

Even though Reed-Muller codes are not cyclic, it is well known that the punctured general Reed-Muller codes are cyclic [65],[66],[67]. For this reason, Reed-Muller codes are here defined as pseudo-cyclic codes. The first-order Reed-Muller code of length 16, which is used to generate the Λ_{16} lattice-based codebook, is not cyclic as in the example codes shown in Table 5.4 [68]. However, if one bit of the code is dropped, the remaining 15-bit codes become cyclic codes. By properly adding zeros to this 15-bit cyclic codes, 16-bit modified Λ_{16} ($M - \Lambda_{16}$) lattice-based codevectors can be obtained. These $M - \Lambda_{16}$ lattice-based codevectors are used for the 16-dimensional

TDIRVQ experiments.

Table 5.4. Fifteen codes based on the first-order Reed-Muller code (weight 8). To generate the Λ_{16} lattice-based codebook, 30 codes are required, and the other 15 codes can be obtained by complimenting the following codes.

0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 1 0 1 1 1 0 0 1 0 1 0 1 0 0 1
0 1 1 0 1 1 0 1 0 0 0 0 1 1 1 0
0 1 1 1 0 1 0 0 0 1 0 1 0 0 1 1
0 1 1 1 1 0 1 0 0 0 1 1 0 1 0 0
1 0 0 1 1 1 0 0 1 0 0 1 0 1 1 0
1 0 1 0 1 1 0 1 0 0 1 1 0 0 0 1
1 0 1 1 0 1 0 0 0 1 1 0 1 1 0 0
1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1
1 1 0 0 1 1 1 0 0 1 0 1 1 0 0 0
1 1 0 1 0 1 1 1 0 0 0 0 0 1 0 1
1 1 0 1 1 0 0 1 0 1 1 0 0 0 1 0
1 1 1 0 0 1 1 0 1 0 1 0 0 0 1 0
1 1 1 0 1 0 0 0 1 1 0 0 0 1 0 1
1 1 1 1 0 0 0 1 1 0 0 1 1 0 0 0

5.3.2 Constructing a Modified Λ_{16} ($\boldsymbol{M}-\Lambda_{16}$) Lattice-Based Codebook

The pseudo-cyclic property of Reed-Muller codes and the procedure to make a pseudo-cyclic Reed-Muller code are introduced in [65], [67], and the same method is followed to construct pseudo-cyclic Reed-Muller codes for an $M-\Lambda_{16}$ lattice-based codebook. The maximum-length sequence is the first period of the output of a length m linear shift feedback register (LSFR) and the length of the maximum-length sequence is $2^m - 1$. For example, a length 4 or 4-stage LSFR is shown in Figure 5.6 with the following feedback polynomial:

$$F(X) = X^4 + X^3 + 1. \tag{5.9}$$

If the initial state is set to 1111, the first period of the output is as follows:

$$m = [1,\ 1,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0]. \tag{5.10}$$

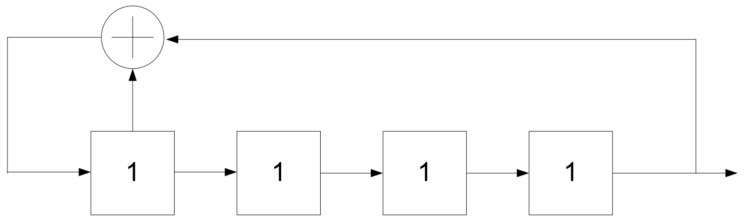


Figure 5.6. 4-bit linear shift feedback registers (LSFR) for generating maximal length sequences. The initial state is set to 1111 **as shown.**

This output is the maximum-length sequence of the 4-stage LSFR, and it is here defined as a seed vector. Using this seed vector m, the following generator matrix can be obtained:

$$P = \begin{bmatrix} m \\ Rm \\ R^2m \\ R^3m \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}, \qquad (5.11)$$

where R is a right cyclic shift operator.

The generator matrix P generates 15-bit binary cyclic codes, and the generated cyclic codes are obviously different from the Reed-Muller codes. However, if the generator matrix P is compared with the 12th, 13th, 14th, and 15th rows of the generator matrix M in Equation 5.7, the generator matrix P is the same as the 12th, 13th, 14th, and 15th rows of the generator matrix M without the last bits. The four rows of the generator matrix M are based on Reed-Muller codes. The 15-bit binary codes that are generated from the generator matrix P are shown in Table 5.5, and the generated codes are cyclic. The generated codes are simply shifted versions of the maximum-length sequence or seed vector of the 4-stage LSFR shown in Figure 5.6 with the feedback polynomial in Equation 5.10.

$M - \Lambda_{16}$ lattice-based codevectors are based on the 15 codes in Table 5.5. By adding a zero in the end of the first seed vector and shifting the modified seed vector, 16 codes can be obtained as shown in Table 5.6. These 16 codes are here names as seed codes. Since the length of the codes increases to 16 from 15, the total number of

Table 5.5. Fifteen codes are generated from the generator matrix P, which is based on the 4-stage LSFR. The generated codes are simply shifted versions of the maximum-length sequence or seed vector of the LSFR.

1	1	1	1	0	1	0	1	1	0	0	1	0	0	0
0	1	1	1	1	0	1	0	1	1	0	0	1	0	0
0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
1	0	0	0	1	1	1	1	0	1	0	1	1	0	0
0	1	0	0	0	1	1	1	1	0	1	0	1	1	0
0	1	0	0	0	0	1	1	1	1	0	1	0	1	1
1	0	1	0	0	0	0	1	1	1	1	0	1	0	1
1	1	0	1	0	0	0	0	1	1	1	1	0	1	0
0	1	1	0	1	0	0	0	0	1	1	1	1	0	1
1	0	1	1	0	1	0	0	0	0	1	1	1	1	0
0	1	0	1	1	0	1	0	0	0	0	1	1	1	1
1	0	1	0	1	1	0	1	0	0	0	0	1	1	1
1	1	0	1	0	1	1	0	1	0	0	0	0	1	1
1	1	1	0	1	0	1	1	0	1	0	0	0	0	1

codes also should be increased to 16 to make the entire codebook cyclic. Therefore, an $M-\Lambda_{16}$ lattice-based codebook requires 32 codes, which include the 16 codes in Table 5.6 and their complements. A $M-\Lambda_{16}$ lattice is not optimal because all codes are not uniformly distributed. The nearest neighbor angle, which is defined as the angle between the two nearest codes [7], is not always 60° as in a Λ_{16} lattice. The nearest neighbor angle of a $M-\Lambda_{16}$ lattice varies from 51.3178° to 67.9757°. However, no performance degradation is observed in the experiments.

The 32 codes provide the locations of ±1s in an $M-\Lambda_{16}$ lattice-based codebook as in a Λ_{16} lattice-based codebook. The ±1s are located in the 1's positions of the 32 codes, and the number of -1s should be even. This combination makes 4096 codevectors, in which there are 128 codevectors per one code. Therefore, the $M-\Lambda_{16}$ lattice-based codebook has total 4576 codevectors including the 480 vectors of all permutations of $2^{-1/2}(\pm 2^2, 0^{14})$. The number of total codevectors in the $M-\Lambda_{16}$ lattice-based codebook increases by 256, when compared to the Λ_{16} lattice-based codebook, but the complexity of encoding and decoding is significantly reduced because of the cyclic property of the $M - \Lambda_{16}$ lattice-based codebook. This reduced complexity will be discussed in the end of this chapter.

The two sets of codevectors, 480 vectors of all permutations of $2^{-1/2}(\pm 2^2, 0^{14})$ and 4096 vectors of the form $2^{-1/2}(\pm 1^8, 0^8)$, should be treated differently in the encoding and decoding process because their structures are different. Therefore, the 4096 vectors of the form $2^{-1/2}(\pm 1^8, 0^8)$ and the 480 vectors of the form $2^{-1/2}(\pm 2^2, 0^{14})$ are grouped and named as class-1 codevectors and a class-2 codevectors, respectively. In addition, the class-1 codevectors are subdivided into two subclass, codevectors from the seed codes in Table 5.6 and codevectors from their complements. The combinations of these two classes and two subclasses generate different distributions of inner product values. The subclass of the codevectors from the seed codes and the subclass of the complement codes are names as a SEED subclass and a COMP subclass, respectively.

Table 5.6. Sixteen codevectors based on a pseudo-cyclic Reed-Muller code (weight 8). $M - \Lambda_{16}$ lattice-based codebook requires 32 codevectors, and the other codevectors can be obtained by complimenting the shown 16 codevectors.

1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	0
0	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0
0	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0
0	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
0	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
1	0	0	0	0	1	1	1	1	0	1	0	1	1	0	0
0	1	0	0	0	0	1	1	1	1	0	1	0	1	1	0
0	0	1	0	0	0	0	1	1	1	1	0	1	0	1	1
1	0	0	1	0	0	0	0	1	1	1	1	0	1	0	1
1	1	0	0	1	0	0	0	0	1	1	1	1	0	1	0
0	1	1	0	0	1	0	0	0	0	1	1	1	1	0	1
1	0	1	1	0	0	1	0	0	0	0	1	1	1	1	0
0	1	0	1	1	0	0	1	0	0	0	0	1	1	1	1
1	0	1	0	1	1	0	0	1	0	0	0	0	1	1	1
1	1	0	1	0	1	1	0	0	1	0	0	0	0	1	1
1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	1

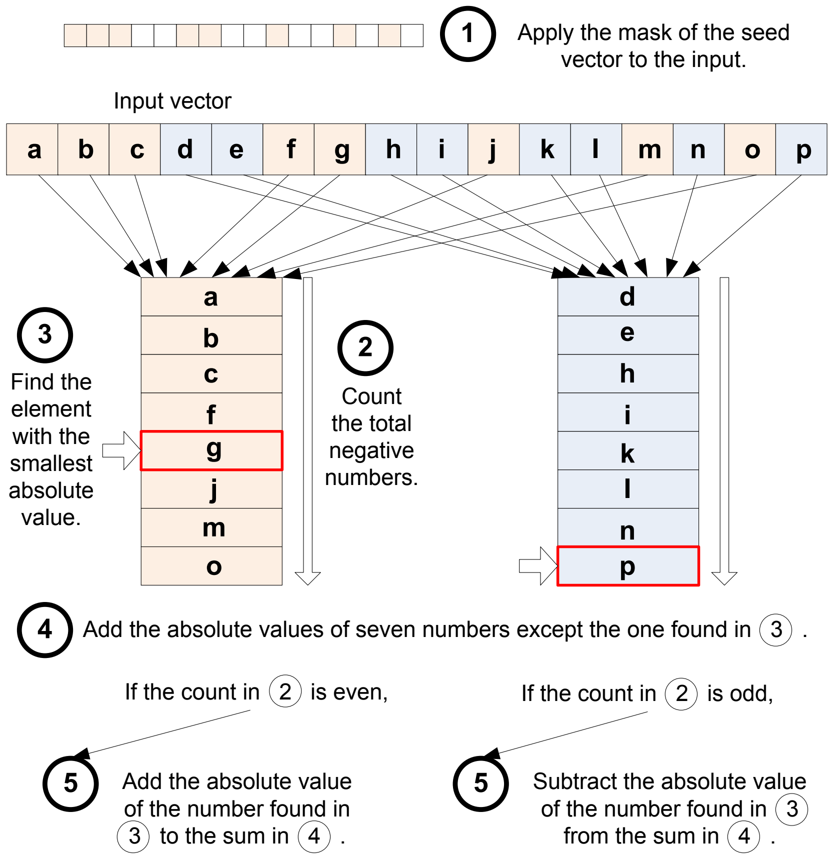
5.3.3 Finding an $M - \Lambda_{16}$ Lattice-Based Codevector

Finding codevectors corresponding to input vectors in an $M - \Lambda_{16}$ lattice-based codebook is more complicated than finding codevectors in a D_4 lattice-based codebook because the structure of the $M - \Lambda_{16}$ lattice-based codebook is more complex. A D_4 lattice-based codebook is all permutations of $(\pm 1, \pm 1, 0, 0)$. Therefore, finding the best-matched codevector corresponding to an input vector is simply finding the two elements with the largest absolute values in an input vector and assigning 1 or -1 to the location of the found elements based on their original signs. However, class-1 codevectors in the $M - \Lambda_{16}$ lattice-based codebook have a limited number of permutations of $(\pm 1^8, 0^8)$. The locations of ± 1s are determined by the 32 codes that are generated from a seed vector. Therefore, finding eight elements with the largest absolute values in an input vector for the location of ± 1 assignment does not work because the locations of the eight elements should match to those of the 32 code.

$M - \Lambda_{16}$ lattice-based codevectors have two different classes as mentioned in the previous section, and these two different classes have different structures. For that reason, two different methods are required to be performed for each class, and the results need to be compared to find the best-matched codevector. The inner product value of an input vector and each codevector is maximized when the codevector is best matched. Therefore, finding a codevector maximizing the inner product with the input vector is the goal. Since the codevectors contains only ± 1, this goal can be rephrased as how to find the maximal sum of the masked elements under the condition that an even number of signs is allowed to be changed.

Only limited combinations of non-zero elements are acceptable for a valid codevector in an $M - \Lambda_{16}$ lattice-based codebook, and the total number of minus signs should be considered as well. The first 16 codes of the 32 codes are generated from a seed vector and its shifted vectors. Since only the locations of 1s of a seed vector and its shifted vectors should be evaluated when finding the best-matched codevector, a seed vector and its shifted vectors are used as a mask to find the locations of 1s. This mask is applied to an input vector, and only masked elements are evaluated. After the evaluation, this mask is right-shifted and applied to the input vector again. It is noted that the complements of the first 16 codes are also parts of the 32 codes. Therefore, when the mask is applied to an input vector, the unmasked elements form another code. Unnecessary masking processes can be omitted so that the 32 masking processes reduce to 16 masking processes by evaluating these unmasked elements at the same time when the masked elements are evaluated. Once the masked elements or unmasked elements are obtained, the elements are searched to find the element with the smallest absolute value, and the number of negative elements is counted. The sign of the element with the smallest absolute value is determined by the count of negative numbers when the total sum is calculated. This is because only an even number of minus signs is allowed for a valid codevector. If the count is even, the absolute values of all masked elements are summed up. All negative elements in the masked elements can be changed to positive because an even number of sign changes is allowed. The sum is the maximum inner product value of the masked elements and all possible codevectors using the code that used for the mask. If the count is odd, not all negative elements can be changed to positive. One of the negative elements should stay negative, and choosing the element with the smallest absolute value as the negative number maximizes the sum. These processes are repeated for the 32 codes. The best-matched codevector in the class-1 can be found by comparing the 32 sums and finding the maximum out of them. The full procedure is depicted in Figure 5.7.

The best-matched codevector in the class-1 should be compared with the one in the class-2. Since the class-2 codevectors are all possible permutations of $2^{-1/2}(\pm 2^2, 0^{14})$, the maximal sum can be obtained by finding the two elements with the largest absolute values as in the D_4 case. The found two elements are multiplied by ± 2 based on their original signs. The best-matched codevector in the class-2 is again compared with the best-matched codevector in the class-1, and the bigger one becomes the final best-matched codevector in an $M - \Lambda_{16}$ lattice-based codebook.



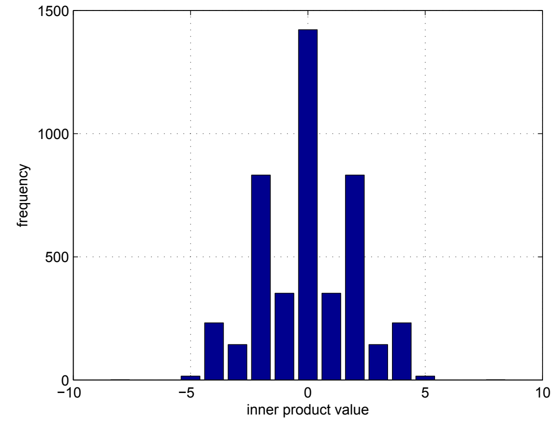
1
Apply the mask of the seed vector to the input.
Input vector
a b c d e f g h i j k l m n o p
a b c f g j m o
d e h i k l n p
3
Find the element with the smallest absolute value.
2
Count the total negative numbers.
4
Add the absolute values of seven numbers except the one found in 3 .
If the count in 2 is even,
If the count in 2 is odd,
5
Add the absolute value of the number found in 3 to the sum in 4 .
5
Subtract the absolute value of the number found in 3 from the sum in 4 .

Figure 5.7. Efficient algorithm for finding a codevector in the class-1: The seed vector that are found from the LSFR is used to generate a mask. The locations of non-zero elements are set to 1 (colored as red) in the mask, and other locations are set to 0 as shown in ①. After the masking operation, the resultant 8 numbers are processed following ②, ③, ④, and ⑤. Since only even number of negative numbers is allowed as a codevector, counting the total negative numbers is necessary. The sign of the element with the smallest absolute value is determined based on this count. Both the shifted 15 vectors of the seed vector and their complements are used to generate the codebook, so the unmasked elements are processed exactly the same way as the masked elements. The unmask elements are colored as blue in the figure. This entire procedure need to be performed 32 times for a seed vector, the 15 shifted versions of the seed vector, and their complements. After the best-matched codevector is found in the class-1, it is compared with the best-matched codevector in the class-2. The two best-matched codevector in each class are the candidates for the best-matched codevector in the full codebook. An inner product is calculated between each candidate and the input vector, and the one with the bigger inner product value becomes the final answer.

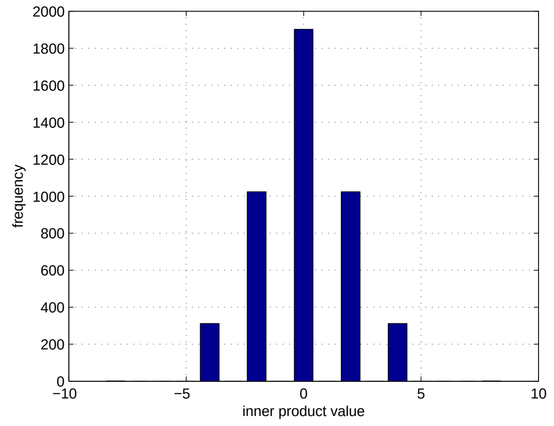
5.3.4 Inner Product-Based Grouping in $M - \Lambda_{16}$ L-DIRVQ

The inner product-based grouping method in Section 5.2.1 is similarly applicable to a $M - \Lambda_{16}$ lattice-based codebook with some modifications. The major difference between the $M - \Lambda_{16}$ lattice-based codebook and the D_4 lattice-based codebook is that the $M - \Lambda_{16}$ lattice-based codebook contains two different classes, which result in different structures. The inner product-based grouping method is based on the inner product of the previous best-matched codevector and all codevectors, and the different structures of the previous best-matched codevector affect the organization of the grouping in the reordered codebook. In addition, a limited number of permutations of $(\pm 1^8, 0^8)$ of class-1 codevectors also influences the distributions of the inner product values.

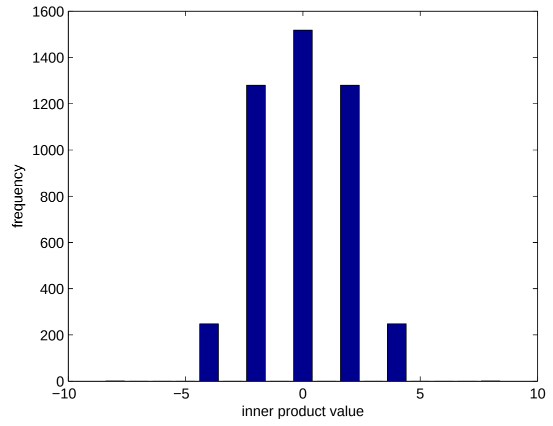
Distributions of the inner product values are shown in Figure 5.8. Figure 5.8 (a) shows the distribution when the previous best-matched codevector is in the class-1. Figure 5.8 (b) shows the distribution when the previous best-matched codevector is in the class-2 and the distance of non-zero elements of the previous best-matched codevector is either 1, 2, 3, 5, 6, 8. These class-2 codevectors form a subclass, which is named as a Dist1. Figure 5.8 (c) shows the distribution when the previous best-matched codevector is in the class-2 and the distance of non-zero elements of the previous best-matched codevector is either 4, 7. These class-2 codevectors form a subclass, which is named as a Dist2. Here the distance is defined as the number of zeros between non-zero elements plus one. Since codevectors in the class-2 have a cyclic property, distance can be measured circularly, and the shorter distance is chosen. For example, the distance of non-zero elements of the previous best-matched vector [0, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0] is 6 instead of 10. Codevectors in the class-2 contain only ± 2, so the inner product values are always even numbers. There are two distributions when the previous best-matched codevector is in the class-2, and the selection of distributions is determined by the distance of non-zero elements of the previous best-matched codevector. More details will be explained in Section 5.3.5.3.



(a)



(b)



(c)

Figure 5.8. Distribution of inner product values in a $M-\Lambda_{16}$ lattice: (a) when the previous best-matched codevector was in the class-1; (b) when the previous best-matched codevector was in the class-2 and the distance of non-zero elements of the previous best-matched codevector is either 1, 2, 3, 5, 6, 8.; (c) when the previous best-matched codevector was in the class-2 and the distance of non-zero elements of the previous best-matched codevector is either 4, 7.

The detailed organizations of each group are shown in Table 5.7 and in Table 5.8. Table 5.7 shows the organization of the codebook when the previous best-matched codevector is in the class-1 and SEED subclass. When the previous best-matched codevector is in a class-1 and COMP subclass, this organization remains same except that the order of subclasses is reversed. For example, the subclasses of the group 3 in Table 5.7 are reversed so that 116 codevectors and 88 codevectors belong to a

COMP subclass and a SEED subclass, respectively. Table 5.7 shows the organization of the codebook when the previous best-matched codevector is in the class-2. After all, there are three inner product-based groupings, and they are as follows:

- Codebook that is reordered based on the previous best-matched codevector in the class-1 and SEED subclass (SEED codebook).
- Codebook that is reordered based on the previous best-matched codevector in the class-1 and COMP subclass (COMP codebook).
- Codebook that is reordered based on the previous best-matched codevector in the class-2 and Dist1 subclass (C2D1 codebook).
- Codebook that is reordered based on the previous best-matched codevector in the class-2 and Dist2 subclass (C2D2 codebook).

The selection of the codebook depends only on the previous best-matched codevector, and the encoder and the decoder already know this information. Therefore, uniqueness of encoding and decoding is guaranteed in spite of multiple complicated codebook organizations.

5.3.5 Encoding and Decoding procedure in $M - \Lambda_{16}$ L-DIRVQ

Four sub-codebooks, a SEED codebook, a COMP codebook, a C2D1 codebook, and a C2D2 codebook, are differently ordered and structured in $M - \Lambda_{16}$ L-DIRVQ. Because of this difference among codebooks, different encoding and decoding methods should be applied to each codebook separately.

5.3.5.1 Encoding and Decoding procedure for a SEED codebook

The best-matched codevector of the input vector is obtained based on the method described in Section 5.3.3. After that, the inner product value is computed, and the number of index candidates is considerably reduced. For example, assume the found best-matched codevector is in the COMP codebook. If the inner product value of the found codevector and the previous best-matched codevector is 3, the

Table 5.7. **Organization of the codebook reordered based on the previous best-matched codevector in the class-1 and SEED subclass codebook (SEED codebook).**

Inner Product	Class	Sub-class	Number of codevectors	Group
8	1	SEED	1	1
5	1	COMP	16	2
4	1	SEED	116	3
		COMP	88	
	2		28	
3	1	SEED	64	4
		COMP	80	
2	1	SEED	352	5
		COMP	352	
	2		128	
1	1	SEED	192	6
		COMP	160	
0	1	SEED	598	7
		COMP	656	
	2		168	
-1	1	SEED	192	8
		COMP	160	
-2	1	SEED	352	9
		COMP	352	
	2		128	
-3	1	SEED	64	10
		COMP	80	
-4	1	SEED	116	11
		COMP	88	
	2		28	
-5	1	COMP	16	12
-8	1	SEED	1	13

Table 5.8. Organization of the codebook reordered based on the previous best-matched codevector in the class-2 (C2 codebook). Two distributions exist as in Figure 5.8 (b) and (c).

Inner Product	Class	Number of codevectors			Group
		Subclass	Dist_1	Dist_2	
8	2		1	1	1
4	1	SEED	128	96	2
		COMP	128	96	
	2		56	56	
2	1	SEED	512	640	3
		COMP	512	640	
0	1	SEED	768	576	4
		COMP	768	576	
	2		366	366	
-2	1	SEED	512	640	5
		COMP	512	640	
-4	1	SEED	128	96	6
		COMP	128	96	
	2		56	56	
08	2		1	1	7

index candidates should be located between index number 314 and index number 393 as shown in Table 5.7. The classification of the class and the subclass of the codevector corresponding to the input vector efficiently narrows down the number of index candidates. However, finding the correct index among these reduced index candidates is real challenging.

The templates for the masked bits in D_4 L-DIRVQ are all possible combinations of bits to make the desired inner product value for the group. However, class-1 codevectors have only a limited number of vectors of the form $2^{-1/2}(\pm 1^8, 0^8)$, so not all possible combinations of bits to make the desired inner product value for the group can be valid templates. Therefore, additional information is necessary for reducing index candidates.

Table 5.9 shows all possible combinations of overlaps when both the previous best-matched codevector, which is located in the first row, and the current best-matched codevector are in a SEED codebook. There are four consecutive non-zero elements in the codevectors in a SEED codebook. The four bits that are masked by these four consecutive non-zero elements are interpreted as a binary number, and it is unique for each possible combination. This binary number is named as an identifier, and the first column in Table 5.9 shows the identifiers. Since these identifiers are unique for each possible combination, they can provide an additional information, which was provided from the templates in D_4 L-DIRVQ, for reducing index candidates in a group. The last column shows the number of non-zero overlaps between the previous best-matched codevector and the possible current best-matched codevectors.

Table 5.9. All possible combinations of overlaps when both the previous best-matched codevector, which is located in the first row, and the current best-matched codevector are in a SEED codebook. The first column shows unique identifiers for each codevector, and the last column shows the number of non-zero overlaps between the previous best-matched codevector and the possible current best-matched codevectors. Circled identifiers indicates that the number of non-zero overlaps is 3.

Identifier	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	Number of overlaps
15	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	8
7	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	4
3	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0	4
1	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0	4
0	0	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1	3
8	1	0	0	0	0	1	1	1	1	0	1	0	1	1	0	0	4
4	0	1	0	0	0	0	1	1	1	1	0	1	0	1	1	0	4
2	0	0	1	0	0	0	0	1	1	1	1	0	1	0	1	1	3
9	1	0	0	1	0	0	0	0	1	1	1	1	0	1	0	1	4
12	1	1	0	0	1	0	0	0	0	1	1	1	1	0	1	0	3
6	0	1	1	0	0	1	0	0	0	0	1	1	1	1	0	1	4
11	1	0	1	1	0	0	1	0	0	0	0	1	1	1	1	0	4
5	0	1	0	1	1	0	0	1	0	0	0	0	1	1	1	1	3
10	1	0	1	0	1	1	0	0	1	0	0	0	0	1	1	1	4
13	1	1	0	1	0	1	1	0	0	1	0	0	0	0	1	1	4
14	1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	1	4

Table 5.10 shows all possible combinations of overlaps when the previous best-matched codevector and the current best-matched codevector are in a COMP codebook. As seen in Table 5.10, the number of non-zero overlaps and the corresponding identifiers are different from those of the SEED codebook. Therefore, encoder and decoder should have information for both cases. Switching between these two set of information is trivial because it is automatically determined once the best-matched codevector corresponding to the input vector is found in the beginning of the encoding process.

Table 5.10. All possible combinations of overlaps when the previous best-matched codevector, which is located in the first row, and the current best-matched codevector are in a COMP codebook. Circled identifiers indicates that the number of non-zero overlaps is 5.

Identifier	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	Number of overlaps
0	0	0	0	0	1	0	1	0	0	1	1	0	1	1	1	1	0
8	1	0	0	0	0	1	0	1	0	0	1	1	0	1	1	1	4
12	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1	1	4
14	1	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1	4
15	1	1	1	1	0	0	0	0	1	0	1	0	0	1	1	0	5
7	0	1	1	1	1	0	0	0	0	1	0	1	0	0	1	1	4
11	1	0	1	1	1	1	0	0	0	0	1	0	1	0	0	1	4
13	1	1	0	1	1	1	1	0	0	0	0	1	0	1	0	0	5
6	0	1	1	0	1	1	1	1	0	0	0	0	1	0	1	0	4
3	0	0	1	1	0	1	1	1	1	0	0	0	0	1	0	1	5
9	1	0	0	1	1	0	1	1	1	1	0	0	0	0	1	0	4
4	0	1	0	0	1	1	0	1	1	1	1	0	0	0	0	1	4
10	1	0	1	0	0	1	1	0	1	1	1	1	0	0	0	0	5
5	0	1	0	1	0	0	1	1	0	1	1	1	1	0	0	0	4
2	0	0	1	0	1	0	0	1	1	0	1	1	1	1	0	0	4
1	0	0	0	1	0	1	0	0	1	1	0	1	1	1	1	0	4

Identifiers and the number of overlaps are used to narrow down the search region in a codebook. Table 5.11 shows all possible inner product values for different numbers of non-zero overlaps when the current best-matched codevector is either in a SEED codebook or a COMP codebook. The number of non-zero overlaps determines possible

Table 5.11. All possible inner product values for the number of non-zero overlaps when the current best-matched codevector is either in a SEED codebook or a COMP codebook.

Number of overlaps	Possible inner product values
8	8, 4, 0, -4, -8
4	4, 2, 0, -2, -4
5	5, 3, 1 ,-1, -3, -5
3	3, 1, -1, -3
0	0

inner product values, and the inner product value also determines possible numbers of non-zero overlaps. For example, if the inner product value is 4, possible numbers of non-zero overlaps are [8, 4], and the corresponding identifiers help to narrow down the index candidates. The corresponding identifiers are [1 3 4 6 7 8 9 10 11 13 14] if the number of non-zero overlaps is 4. Figure 5.9 shows this detailed organization.

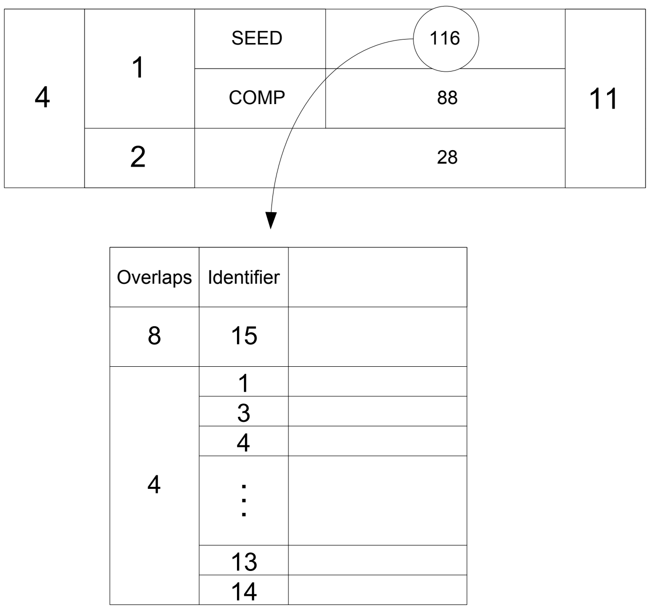


Figure 5.9. Detailed organization in a subclass. A number of non-zero overlaps and the corresponding identifer determine the narrowed region of index candidates.

After reducing the number of index candidates using the identifiers, the template-based searching is applied as in D_4 L-DIRVQ. For example, assume the followings; the previous best-matched codevector is [1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0], and the inner product value of the previous best-matched codevector and the current best-matched codevector is 2, and the number of non-zero overlaps is 4. Then the possible combinations of the non-zero overlaps are {[1, 1, 1, -1],[1, 1, -1, 1],[1, -1, 1, 1],[-1, 1, 1, 1]}. This set is the template for the non-zero overlaps. As in D_4 L-DIRVQ, dropping signs of the previous best-matched vector makes it possible to have only one set of template instead of multiple templates for all sign combinations.

The final step to find the index corresponding to the input vector is applying another template to the unmasked non-zero elements. Table 5.10 shows the templates for a SEED codebook and a COMP codebook. This set of templates is designed to be used for the both codebooks rather than different sets of templates for each codebook. This approach reduces the memory consumption for storing templates. The number of unmasked non-zero elements and the number of minus signs in the non-zero overlaps determine which template should be used. The number of minus signs in the non-zero overlaps is considered here because $M - \Lambda_{16}$ lattice-based codevectors are required to have only even number of negative elements.

When the current best-matched codevector is in the class-2, the template-based search can be directly followed without finding identifiers. Therefore, the template size is considerably larger than that of class-1 cases because there is no identifier to reduce the template size. If the amount of memory is very limited, instead of storing all templates, a tree search algorithm can be applied to this case with the possible combinations, [-2, 0, 2].

5.3.5.2 Encoding and Decoding procedure for a COMP codebook

The encoding and decoding procedures for a COMP codebook are not much different from the procedures for a SEED codebook. They share the exactly same methods such

5-element-odd-minus template

-1				
	-1			
		-1		
			-1	
				-1
-1	-1	-1		
-1	-1		-1	
-1	-1			-1
-1		-1	-1	
-1		-1		-1
-1			-1	-1
	-1	-1	-1	
	-1	-1		-1
	-1		-1	-1
		-1	-1	-1
-1	-1	-1	-1	-1

5-element-even-minus template

-1	-1			
-1		-1		
-1			-1	
-1				-1
	-1	-1		
	-1		-1	
	-1			-1
		-1	-1	
		-1		-1
			-1	-1
-1	-1	-1	-1	
-1	-1	-1		-1
-1	-1		-1	-1
-1		-1	-1	-1
	-1	-1	-1	-1

4-element-odd-minus template

-1			
	-1		
		-1	
			-1
-1	-1	-1	
-1	-1		-1
-1		-1	-1
	-1	-1	-1

4-element-even-minus template

-1	-1		
-1		-1	
-1			-1
	-1	-1	
	-1		-1
		-1	-1
-1	-1	-1	-1

3-element-odd-minus template

-1		
	-1	
		-1
-1	-1	-1

3-element-even-minus template

-1	-1	
-1		-1
	-1	-1

* Blanks are filled with 1's.

Figure 5.10. Templates for a SEED codebook and a COMP codebook: The number of unmasked non-zero elements and the number of minus signs in the non-zero overlaps determine which template is used.

as the identifiers and the template-based search except that the codebook organization is slightly different as mentioned in Section 5.3.4. Therefore, the explanation of the detailed procedure is omitted.

5.3.5.3 Encoding and Decoding procedure for a C2D1 codebook and a C2D2 codebook
When the previous best-matched codevector is in class-2, two inner product distributions exist as mentioned in Section 5.3.4. The selection of these distributions is fully determined by the distance of the non-zero elements in the previous best-matched codevector. Table 5.13 and Table 5.12 shows the templates for the encoding and decoding procedure for a C2 codebook. These templates are only for the reference to locations of the non-zero elements in the best-matched codevector, so the signs of the codevector are ignored.

The distance of the non-zero elements in the previous best-matched codevector in Table 5.13 is 2. If the distance is either 1, 2, 3, 5, 6, or 8, the frequency of overlapping two non-zero elements is 3, and the frequency of no overlap is also 3. For the other 10 cases, the number of overlaps is 1. This case generates the distribution shown in Figure 5.8 (c).

The distance of the non-zero elements in the previous best-matched codevector in Table 5.12 is 3. If the distance is either 4 or 7, the frequency of overlapping two non-zero elements is 4, and the frequency of no overlap is also 4. For the other 8 cases, the number of overlaps is 1. This case generates the distribution shown in Figure 5.8 (b).

Once the codebook corresponding to the distance of the non-zero elements in the previous best-matched codevector is determined, the number of index candidates can be considerably narrowed down. For example, assume the previous best-matched codevector is [0, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] and the current best-matched codevector is [1, -1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0], which is in the SEED codebook. Since the distance of non-zero elements in [0, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] is 4,

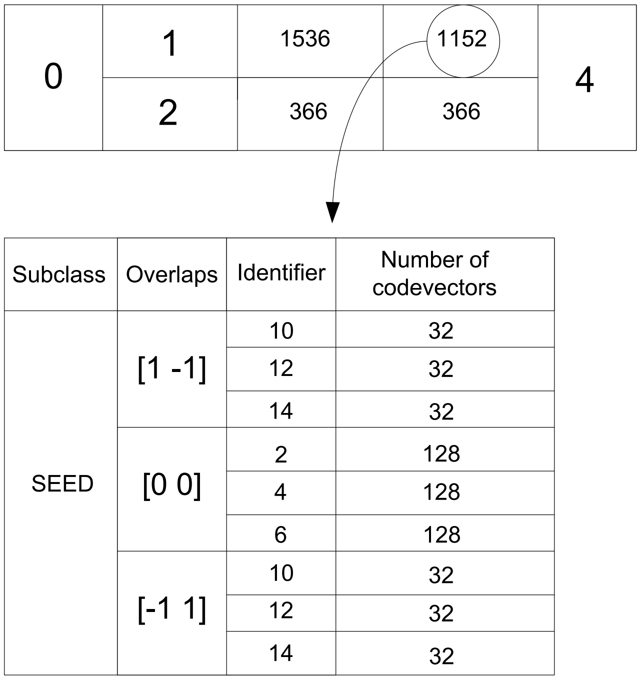
Table 5.12. Template for the encoding and decoding procedure for a C2 codebook when the distance of the non-zero elements in the previous best-matched codevector, which is at the top row, is either 1, 2, 3, 5, 6, 8.

Identifier		2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	Number of overlaps
15	←	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	2
7		0	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1
3		0	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0	1
1		0	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0	1
0		0	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
8		1	0	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
4		0	1	0	0	0	0	1	1	1	1	0	1	0	1	1	0	0
2		0	0	1	0	0	0	0	1	1	1	1	0	1	0	1	1	0
9		1	0	0	1	0	0	0	0	1	1	1	1	0	1	0	1	2
12		1	1	0	0	1	0	0	0	0	1	1	1	1	0	1	0	1
6		0	1	1	0	0	1	0	0	0	0	1	1	1	1	0	1	0
11		1	0	1	1	0	0	1	0	0	0	0	1	1	1	1	0	2
5		0	1	0	1	1	0	0	1	0	0	0	0	1	1	1	1	1
10		1	0	1	0	1	1	0	0	1	0	0	0	0	1	1	1	1
13		1	1	0	1	0	1	1	0	0	1	0	0	0	0	1	1	2
14		1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	1	1

Table 5.13. Template for the encoding and decoding procedure for a C2 codebook when the distance of the non-zero elements in the previous best-matched codevector, which is at the top row, is either 4 or 7.

Identifier	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	Number of overlaps
15	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	1
7	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1
3	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0	1
1	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0	1
0	0	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1	1
8	1	0	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
4	0	1	0	0	0	0	1	1	1	1	0	1	0	1	1	0	0
2	0	0	1	0	0	0	0	1	1	1	1	0	1	0	1	1	0
9	1	0	0	1	0	0	0	0	1	1	1	1	0	1	0	1	1
12	1	1	0	0	1	0	0	0	0	1	1	1	1	0	1	0	2
6	0	1	1	0	0	1	0	0	0	0	1	1	1	1	0	1	0
11	1	0	1	1	0	0	1	0	0	0	0	1	1	1	1	0	1
5	0	1	0	1	1	0	0	1	0	0	0	0	1	1	1	1	1
10	1	0	1	0	1	1	0	0	1	0	0	0	0	1	1	1	2
13	1	1	0	1	0	1	1	0	0	1	0	0	0	0	1	1	1
14	1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	1	2

the codebook organization follows the Dist2 column in Table 5.8. The inner product of [0, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] and [1, -1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0] is 0. The subclass of the current best-matched codevector and the overlapped elements [-1, 1] efficiently narrowed down the index candidates as shown in Figure 5.11. The identifier of [1, -1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0] is determined by the four consecutive elements from the location of the first non-zero element in the previous best-matched codevector. In this example, the four consecutive number is [-1, 1, 1, 0], and it is interpreted as binary number 1110, which is 14 in decimal representation. The number of index candidates is now reduced to 32. The templates for the final step are similar to those in a SEED codebook or a COMP codebook. Difference is that the total number of elements is 6, 7, and 8 instead of 3, 4, and 5 in the class-1 case shown in Figure 5.10.



Subclass	Overlaps	Identifier	Number of codevectors
SEED	[1 -1]	10	32
		12	32
		14	32
	[0 0]	2	128
		4	128
		6	128
	[-1 1]	10	32
		12	32
		14	32

Figure 5.11. Detailed organization of a subclass in the class2: Information about subclass, templates for overlaps, and the corresponding identifer determine the narrowed region of index candidates.

5.3.6 Example

Assume the previous best-matched codevector is [1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0], and the input vector is [22, -3, 33, 34, -56, 4, 2, -9, 25, -320, 57, 8, 1, 220, -77, 16]. The first mask [1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0] is applied to the input vector, and the result is as follows:

$$Masked\ output = [22,\ -3,\ 33,\ 34,\ 4,\ -9,\ 25,\ 8]. \tag{5.12}$$

The number with the smallest absolute value is -3, and the number of negative numbers is 2. Since the number of negative numbers is even, all the negative numbers can be converted to positive. Therefore, the maximum inner product value for this input vector with the applied mask is the sum of the absolute values of all masked output elements, which is 138. The same procedure is performed for the other masks, and the results are compared each other to find the best-matched codevector in a class-1 codebook. The resultant codevector and the inner product value corresponding to the found codevector is as follows:

$$B_1 = [0,\ 0,\ 0,\ 0,\ -1,\ 0,\ 1,\ 0,\ 0,\ -1,\ 1,\ 0,\ -1,\ 1,\ -1,\ 1]\ (747). \tag{5.13}$$

The next step is to find the best-matched codevector in a class-2 codebook. The two elements with the largest absolute value is -320 and 220 in the input vector, so the best-matched codevector and the maximum inner product value of the input vector and all codevectors in a class-2 codebook are as follows:

$$B_2 = [0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ -1,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0]\ (540). \tag{5.14}$$

B_1 and B_2 are compared to find the final best-matched codevector. Since the inner product value of B_1 is bigger than that of B_2, B_1 is chosen as the final best-matched codevector. The inner product value of the found codevector and the previous best-matched codevector is 0. The found best-matched codevector B_1 gives additional information for encoding. B_1 is in a COMP codebook, and the overlaps with the previous best-matched codevector is [0, 0, 0, 0]. Considering the inner product value and the information about the previous best-matched codevector and B_1, the index should be located between 2176 and 2831, and it is shown in Figure 5.12.

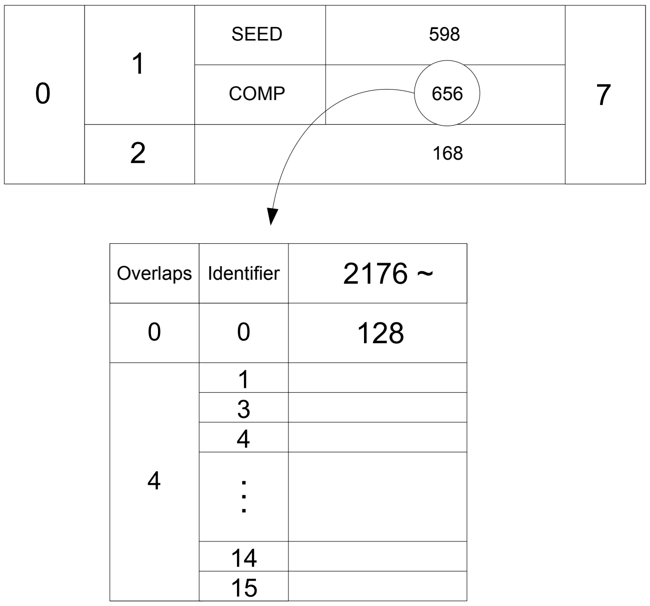


Figure 5.12. The detailed organization of the codebook is shown when the inner product value is 0. Index starts from 2173 in this case.

Since the overlap is [0, 0, 0, 0], the search region is again reduced to between 2176 and 2303. The next step is to find the final index using the template that have 8 elements and an even number of negative numbers. Non-overlaps are [-1, 1, -1, 1, -1, 1, -1, 1], and it is the 50th vector in the template. Therefore, the final index is 2225.

5.3.7 Computational Reduction

The brute-force search for the best-matched codevector requires 4576 inner products and 4575 comparisons in the worst case. For the 16-dimensional case, one inner product computation contains 16 multiplications and 15 additions. Therefore, total 73216 multiplications and 68640 additions have to be performed in the worst case. This number of multiplications and additions is enormous for encoding one block of an image. To make this worse, to reorder the entire codebook, the 4576 inner products and 4575 comparisons are always required to be performed regardless of whether it is the worst case or not. In addition, if the elements of the input vector are transformed coefficients, floating-point multipliers and adders are required.

On the other hand, the proposed algorithm for finding the best-matched codevector corresponding to the input vector needs 32 16-bit masking processes, finding the minimum in a vector with length 8 (min function, 32 times), counting the total negative numbers for a vector with length 8 (count function, 32 times), finding absolute value (abs funtion, 256 times), 96 comparisons, and finding the maximum (max function, twice). This computational reduction for finding the best-matched codevoctor is summarized in Table 5.14.

Table 5.14. Computational reduction of the proposed $M-\Lambda_{16}$ lattice-based codebook encoding and decoding algorithm.

	Brute-force method	Proposed algorithm
Codebook search	4576 inner products, 4575 comparisons (73216 multiplications and 68640 additions)	2 max functions, 32 16-bit maskings, 256 abs functions, 32 min functions, 32 count functions, 96 comparisons

The required number of computations for finding the corresponding index from the best-matched codevector varies based on the class and subclass combinations. Therefore, it is difficult to count the total number of operations precisely. Instead, the total run-time of coding a 64×64 Lena image is compared as shown in Table 5.15. Shown run-times include the run-time of both codebook searching and dynamic index reordering. The run-time is measured 10 times and averaged. The experiments are performed with the following environment:

Hardware: Pentium Core2duo CPU (T9600), 4 Gigabytes RAM.

Software: MATLAB R2008b on Windows XP (32-bit)

Table 5.15. Run-time reduction of the proposed $M-\Lambda_{16}$ lattice-based codebook encoding and decoding algorithm. A 64×64 Lena image is coded with MATLAB on Windows XP.

	Brute-force method	Proposed algorithm
total run-time (second)	38.4983	1.0212

CHAPTER 6
CONCLUSIONS AND DISCUSSIONS

6.1 Conclusions

This research focuses on the implementation of efficient image compression systems. To achieve this system-wise goal, three domains of research are performed; in imaging system implementation with an analog CMOS transform imager, in image compression algorithm development for the imaging system ,and in the efficient digital implementation of the compression system.

First, an imaging system using a CMOS transform imager is proposed based on the CADSP concept, and the system has been implemented with a FPGA board. Printed circuit boards (PCB) are designed for operating the transform imager and interfacing with the FPGA board. Since both the transform imager and the FPGA board are highly programmable, the implemented imaging system can perform various image processing algorithms. In addition, the transform imager is capable of computing block-based transforms in the analog domain. This can reduce the computational loads of the digital implementation, so more efficient algorithm implementation is possible. To verify the functionality of the system, the JPEG image compression algorithm is realized in the system, and the performance is analyzed. 2D DCT operations are moved from the digital implementation into the transform imager. This JPEG implementation shows higher power efficiency compared to the digital-only implementation.

Second, the state-of-the-art image compression algorithm based on wavelets and dynamic index reordering vector quantization (DIRVQ) is proposed. Transform-based embedded coders highly decorrelate the correlation among the image pixels or blocks in the spatial domain, so it is hard to achieve the better performance by applying

VQ with memory such as DIRVQ. To overcome this difficulty, the refinement pass of the embedded coders is defined as the temporal domain, and the DIRVQ is applied to this newly-defined temporal domain. The experimental results show the PSNR performance improvement at almost all compression ratios, and it proves that the proposed algorithm successfully exploits the redundancy in the temporal domain. The discrete wavelet transform (DWT) computation is partitioned and moved to the transform imager.

Third, for the efficient implementation of DIRVQ, novel encoding and decoding scheme is proposed. The improvement of the proposed wavelet-based image compression algorithm comes from DIRVQ. However, DIRVQ is computationally intensive because of the full codebook reordering. To alleviate this high computational load in digital implementation, the property of lattice-based codebooks of 2×2 and 4×4 are thoroughly investigated. It is found that the lattice-based codebook is highly structured and cyclic or pseudo-cyclic. Using these properties, the proposed encoding and decoding algorithm significantly reduces the computation and the memory requirement.

Modern image compression algorithms become more and more complicated not only to implement but also to analyze. The validity of algorithms can be easily assessed by computer simulations, but the implementation in real world as a system can pose a lot of difficulties when trying to achieve the target performance such as high speed operations and low power consumption. The CADSP approach can give more degrees of freedom in designing systems and achieving the target performance. In this research, a wavelet-based embedded image coding algorithm is proposed, and the algorithm is studied in different aspects to implement on a transform imager-based compression system. The system has more degrees of freedom for finding the desired solution compared to the conventional digital-only system with a conventional CMOS or CCD imager.

6.2 Discussions - DIRVQ for Videos

Videos have a considerable amount of redundancy between frames. The temporal redundancy can be exploited by applying DIRVQ as shown in Figure 6.1. Several video sequences are coded using DIRVQ to verify how much efficiency DIRVQ can achieve, and index changes and the entropy reduction are examined. In the experiments, the Lloyd algorithm is used for generating codebook. The codebook size is 1024, and the dimension of the vector is 16 (4×4 block). Three 512×512 images in Figure 6.2 are spatially applied for codebook training.

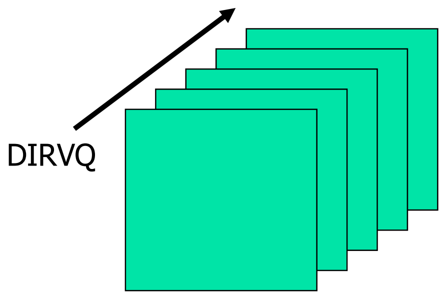


Figure 6.1. DIRVQ for video processing

Figure 6.2. Sample figures used for codebook training

Index changes of one sampled block in a Missa sequence are shown in Figure 6.3 (a). As expected, a strong resemblance is seen between frames. The constant region of index changes in normal VQ reflects this fact. Index changes after applying DIRVQ are shown in Figure 6.3 (b). Considering the concept of DIRVQ, Figure 6.3 (b) is simply the derivative form of Figure 6.3 (a). Total index count for the full video coding is shown in Figure 6.4. The distribution is concentrated in lower indices, and the entropy reduces from 8.2483 bits to 5.316 bits.

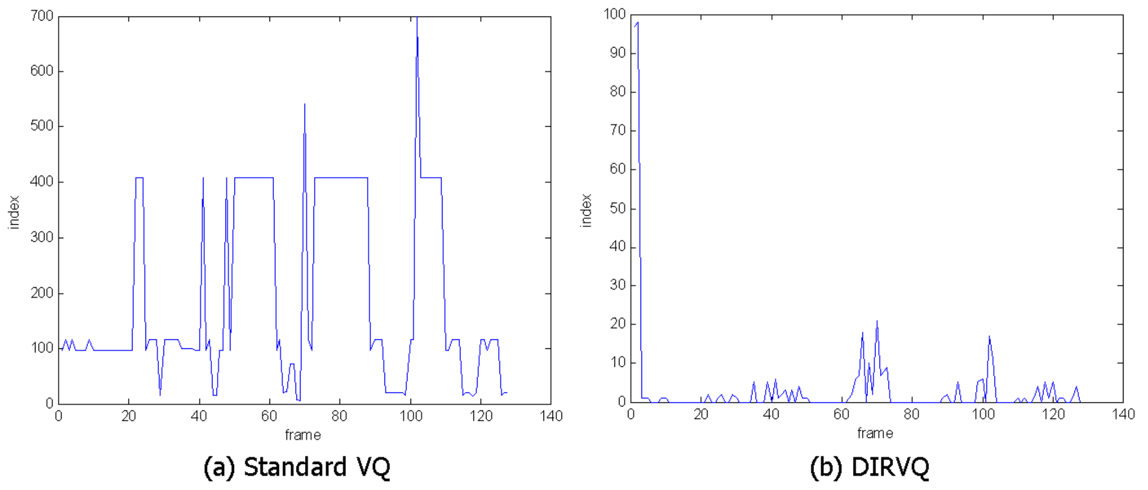


Figure 6.3. Index changes of one sampled block in the missa sequence

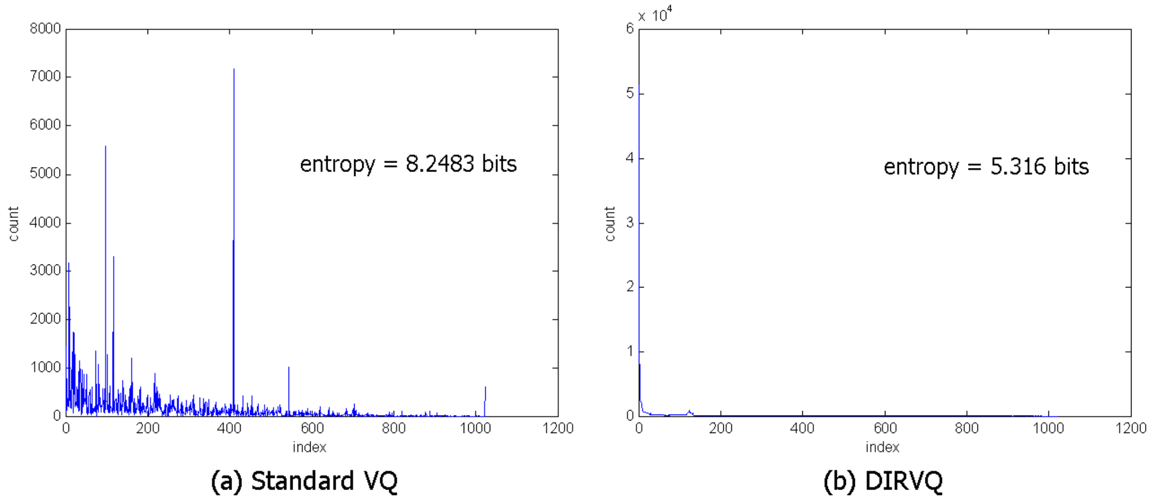


Figure 6.4. Total index count for the full video of the missa sequence

The test is performed for more sequences such as tennis, Susie, and Foreman. The entropy reduction results are shown in Table 6.1. The amount of entropy reduction varies with sample sequences and codebook sizes, but overall about a 40% entropy reduction is achieved. The following works will be interesting to study further:

- Developing more refined video coding algorithms using L-DIRVQ instead of using training-based codebooks.

• Comparing the output performance and computational complexity of L-DIRVQ-based video coding with conventional motion estimation-based video coding algorithms.

Table 6.1. Entropy reduction of the test video samples after applying DIRVQ

Sample movies	Codebook size	Standard VQ PSNR	Standard VQ Entropy	DIRVQ Entropy
Tennis(SIF,150frame)	1024		7.67	4.18
Susie (720×480,150frame)	1024	34.92	6.83	3.70
	2048	35.36	7.53	4.32
Foreman (CIF,300frame)	1024	29.30	7.95	4.73
	2048	29.82	8.74	5.42
	4096	30.30	9.49	6.12
Block size: 4×4 Average improvement: around 40% of entropy reduction				

REFERENCES

[1] P. Hasler and D. V. Anderson, "Cooperative analog-digital signal processing," in *2002 IEEE International Conference on Acoustic, Speech, and Signal Processing*, 2002, vol. 4, pp. 3972–3975.

[2] Paul Hasler, Abhishek Bandyopadhyay, and D. V. Anderson, "High-fill factor imagers for neuromorphic processing enabled by floating-gate circuits," *International Journal of Signal Processing*, vol. 103, pp. 2273–2281, 2003.

[3] Iain M. Johnston and David L. Donoho, "Adapting to smoothness via wavelet shrinkage," *Journal of the Statistical Association*, vol. 90, pp. 1200–1224, Dec. 1995.

[4] David L. Donoho, "De-noising by soft thresholding," *IEEE Transactions on Information Theory*, vol. 41, pp. 613–627, May 1995.

[5] Maarten Jansen, *Noise Reduction by Wavelet Thresholding*, vol. 161, Springer, 2001.

[6] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transaction on Signal Processing*, vol. 41, pp. 3445–3462, Dec. 1993.

[7] Eduardo A. B. da Silva, Demetrios G. Sampson, and Mohammad Ghanbari, "A successive approximation vector quantization for wavelet transform image coding," *IEEE Transaction on Image Processing*, vol. 5, pp. 299–310, Feb. 1996.

[8] O. Yadid-Pecht and E. R. Fossum, "Wide intrascene dynamic range cmos aps using dual sampling," *IEEE Transactions on Electron Devices*, vol. 44, pp. 1721–1723, Oct. 1997.

[9] E. R. Fossum, "Cmos image sensors: electronic camera-on-a-chip," *IEEE Transactions on Electron Devices*, vol. 44, pp. 1689–1698, Oct. 1997.

[10] M. Cohen, G. Cauwenberghs, M. Vorontsov, and G. Carhart, "Focal-plane image and beam quality sensors for adaptive optics," in *2001 Conference on Advanced Research in VLSI*, 2001, pp. 224–237.

[11] A. Aslam-Siddiqi, W. Brockherde, and B. Hosticka, "A 128-pixel cmos image sensor with integrated analog nonvolatile memory," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 1497–1501, Oct. 1998.

[12] C. Kwang-Bo, A. Krymski, and E. R. Fossum, "A 1.2v micropower cmos active pixel image sensor for portable applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2000, pp. 114–115.

[13] D. X. D. Yang, A. E. Gamal, B. Fowler, and H. Tian, "A 640 x 512 cmos image sensor with ultrawide dynamic range floating-point pixel level adc," *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 1821–1832, Dec. 1999.

[14] J. C. Gealow and C. G. Sodini, "A pixel-parallel image processor using logic pitch-matched to dynamic memory," *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 65–73, June 1999.

[15] M. Schwarz, R. Hauschild, B. Hosticka, J. Huppertz, T. Kneip, S. Kolnsberg, L. Ewe, and H. Trieu, "Single-chip cmos image sensors for a retina implant system," *IEEE Transaction on Circuits System II, Analog Digital Signal Processing*, vol. 46, pp. 870–977, July 1999.

[16] J. C. Gealow, F. P. Herrmann, L. T. Hsu, and C. G. Sodini, "System design for pixel-parallel image processing," *IEEE Transaction on VLSI System*, vol. 4, pp. 32–41, Mar. 1996.

[17] R. H. Robinson and D. S. Wills, "Design of an integrated focal plane architecture for efficient image processing," in *15th International Conference on Parallel and Distributed Computing Systems*, 2002, vol. 171, pp. 128–135.

[18] William H. Robinson and D. Scott Wills, "Efficiency analysis for a mixed-signal focal plane processing architecture," *Journal of VLSI Signal Processing*, vol. 41, pp. 65–80, 2005.

[19] Abhishek Bandyopadhyay, Jungwon Lee, Ryan W. Robucci, and Paul Hasler, "Matia: A programmable 80uw/frame cmos block matrix transform imager architecture," *IEEE Journal of Solid-State Circuits*, vol. 41, pp. 663–672, Mar. 2006.

[20] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Prentice Hall, 2002.

[21] V. Srinivasan, G. J. Serrano, J. Gray, and P. Hasler, "A precision cmos amplifier using floating-gates for offset cancellation," in *Proc. IEEE Custom Integrated Circuit Conference (CICC)*, 2005, pp. 734–737.

[22] Teahyung Lee, Leung Kin Chiu, David V. Anderson, Ryan Robucci, and Paul Hasler, "Rapid algorithm verification for cooperative analog-digital imaging systems," in *IEEE International Midwest Symposium on Circuits and Systems*, 2007 to be published.

[23] A. N. Netravali and J. O. Limb, "Picture coding: A review," in *Proc. IEEE*, 1980, vol. 68, pp. 366–406.

[24] Gregory K. Wallace, "The jpeg still picture compression standard," *Communication of the ACM*, vol. 34, no. 4, pp. 30–44, April 1991.

[25] K. Ogawa, T. Urano, N. Mori, S. Moriai, H. Yamamoto, and S. Kato, "A single chip compression/decompression lsi based on jpeg," *IEEE Transaction on Comsumer Electronics*, vol. 38, pp. 703–710, Aug. 1992.

[26] M. Kovac and P. Ranganathan, "Jaguar: a high speed vlsi chip for jpeg image compression standard," in *Processings of the 8th international conference on VLSI design*, Jan. 1995, pp. 220–224.

[27] N. Narasimhan, V. Srinivasan, M. Vootukuru, J. Walrath, S. Govindarajan, and R. Vemuri, "Rapid prototyping of reconfigurabe coprocessors," in *Acoustics, Speech, and Signal Processing, 2002 IEEE International Conference on*, 1996, pp. 303–312.

[28] J. Park, S. Kwon, and K. Roy, "Low power reconfigurable dct design based on sharing multiplication," in *Acoustics, Speech, and Signal Processing, 2002 IEEE International Conference on*, 2002, vol. 3, pp. 3116–3119.

[29] M. Martina, A. Molino, and F. Vacca, "Reconfigurable and low power 2d-dct ip for ubiquitous multimedia streaming," in *Multimedia and Expo, 2002. ICME '02. Proceedings. 2002 IEEE International Conference on*, 2002, vol. 2, pp. 26–29.

[30] Liang-Gee Chen, Juing-Ying Jiu, Hao-Chieh Chang, Yung-Pin Lee, and Chung-Wei Ku, "Low power 2d dct chip design for wireless multimedia terminals," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1998, vol. 4, pp. 41–44.

[31] R. Westwater and B. Furht, *Real-time Video Compression: Techniques and Algorithms*, Kluwer Academic Publishers, 1997.

[32] *http://www.xilinx.com/cgi-bin/powerweb.pl.*

[33] Ri. Carmona-Galan, F. Jimenez-Garrido, C. M. Dominguez-Mata, R. Domingguez-Castro, S. E. Meana, I. Petras, and A. Rodriguez-Vazquez, "Second-order neural core for bioinspired focal-plane dynamic image processing in cmos," *IEEE Transactions on Circuits and Systems I*, vol. 51, no. 5, pp. 913–925, May 2004.

[34] A. A. Stocker, "Analog vlsi focal-plane array with dynamic connections for the estimation of piecewise-smooth optical flow," *IEEE Transactions on Circuits and Systems I*, vol. 51, no. 5, pp. 963–973, May 2004.

[35] R. Robucci, Leung Kin Chiu, J. Gray, J. Romberg, P. Hasler, and D. Anderson, "Compressive sensing on a cmos separable transform image sensor," in *IEEE International Conference on Acoustics, Speech and Signal Processing, 2008. ICASSP 2008*, 2008, pp. 5125–5128.

[36] Anil K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, 1988.

[37] Cheng-Hsiung Hsieh, "A jonal jpeg," in *International Conference on Coding and Computing, ITCC 2005*, 2005, vol. 2, pp. 756–757.

[38] Chia-Lung Yeh, "Color image-sequence compression using adaptive binary-tree vector quantization with codebook replenishment," in *IEEE International Conference on Acoustic, Speech, and Signal Processing*, 1987, vol. 12, pp. 1059–1062.

[39] Nasser M. Nasrabadi and Yushu Feng, "Image compression using address-vector quantization," *IEEE Transactions on Communications*, vol. 12, pp. 2166–2173, Dec. 1990.

[40] Nasser M. Nasrabadi, Chang Y. Choo, and Yushu Feng, "Dynamic finite-state vector quantization of digital images," *IEEE Transactions on Communications*, vol. 42, pp. 2145–2154, May 1994.

[41] Seung Jun Lee, Kyeong Ho Yang, Chul Woo Kim, and Choong Woong Lee, "Efficient lossless coding scheme for vector quantization using dynamic index mapping," *Electronics Letters*, vol. 31, pp. 1426–1427, Aug. 1995.

[42] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transaction on Circuits and Systems for Video Technology*, vol. 6, pp. 243–250, June 1996.

[43] Debargha Mukherjee and Sanjit K. Mitra, "Vector SPIHT for embedded wavelet video and image coding," *IEEE Transaction on Circuits and Systems for Video Technology*, vol. 13, pp. 231–246, Mar. 2003.

[44] V. Krishnan, *A framework for low bit-rate speech coding in noisy environments*, Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, 2005.

[45] S. H. Nawab and T. F. Quatieri, "Short-time fourier transform," in *Advanced Topics in Signal Processing*. 1988, Prentice Hall.

[46] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck, *Discrete-time Signal Processing*, Prentice Hall, 1999.

[47] S. Mallat, "A compact multiresolution representation: The wavelet model," in *Proc. IEEE Computer Society Workshop on Computer Vision*. 1987, pp. 2–7, IEEE Computer Society Press.

[48] Ali N. Akansu and Richard A. Haddad, *Multiresolution Signal Decomposition*, Academic Press, 2001.

[49] Gilbert Strang and Truong Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press, 1996.

[50] M. Craizer, E. A. B. da Silva, and E. G. Ramos, "Convergent algorithms for successive approximation vector quantisation with applications to wavelet image compression," in *IEE Proceedings - Vision, Image, and Signal Processing*, 1999, vol. 146, pp. 159–164.

[51] T. Lookabaugh, E. A. Riskin, P. A. Chou, and R. M. Gray, "Variable rate vector quantization for speech, image, and video compression," *IEEE Transaction on Communications*, vol. 41, pp. 186–199, Jan. 1993.

[52] K. Andra, C. Chakrabati, and T. Acharya, "A vlsi architecture for lifting-based forward and inverse wavelet transform," *IEEE Transactions on Signal Processing*, vol. 50, pp. 966–977, April 2002.

[53] Khan Wahid, Vassil Dimitrov, and Graham Jullien, "Multiplication-free architecture for daubechies wavelet transforms using algebraic integers," in *Advanced Signal Processing Algorithms, Architectures, and Implementations XIII*. 2003, pp. 597–606, SPIE- International Society for Optical Engineering.

[54] Chin-Fa Hsieh, Tsung-Han Tsai, Chih-Hung Lai, Shu-Chung Yi, and Mao-Hsu Yen, "Implementation of an efficient dwt using a fpga on a real-time platform," in *Second International Conference on Innovative Computing, Information and Control, 2007. ICICIC*, 2007, pp. 235–235.

[55] Stephane Mallat, *A Wavelet Tour of Signal Processing*, Academic Press, 1999.

[56] D. L. Donoho, I. M. Johnstone, G. Kerkyacharian, and D. Picard, "wavelet shrinkage: Asymptopia?," *Journal of the Royal Statistics Society*, vol. 57, pp. 301–369, 1995.

[57] Martin Vetterli, S Grace Chang, and Bin Yu, "Adaptive wavelet thresholding for image denoising and compression," *IEEE Transactions on Image Processing*, vol. 9, pp. 1532–1546, Sep. 2000.

[58] Thomas M. Cover and Joy A. Thomas, *Elements of Information Theory*, Wiley-Interscience Publication, 1991.

[59] J. Conway and N. Sloan, "Fast quantizing and decoding and algorithms for lattice quantizers and codes," *IEEE Transactions on Information Theory*, vol. 28, pp. 227–232, Mar. 1982.

[60] J. Conway and N. Sloan, "A fast encoding method for lattice codes and quantizers," *IEEE Transactions on Information Theory*, vol. 29, pp. 820–824, Nov. 1983.

[61] C. Lamblin, J. P. Adoul, D. Massaloux, and S. Morissette, "A compact multiresolution representation: The wavelet model," in *IEEE International Conference on Acoustic, Speech, and Signal Processing*, 1989, vol. 1, pp. 61–64.

[62] P. Rault and C. Guillemot, "Indexing algorithms for zn, an, dn, and dn++ lattice vector quantizers," *IEEE Transactions on Multimedia*, vol. 3, pp. 395–404, Dec. 2001.

[63] J. Conway and N. Sloan, *Sphere Packings, Lattices, and Groups*, Springer, 1998.

[64] J. H. van Lint, *Introduction to Coding Theory*, Springer-Verlag, 1982.

[65] T. Kasami, S. Lin, and W. W. Peterson, "New generalizations of the reed-muller codes - part i: Primitive codes," *IEEE Transaction on Information Theory*, vol. IT-14, Mar. 1968.

[66] E. J. Weldon, "New generalizations of the reed-muller codes - part ii: Non-primitive codes," *IEEE Transaction on Information Theory*, vol. IT-14, Mar. 1968.

[67] James L. Massey, "The ubiquity of reed-muller codes," in *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, 2001, pp. 1–12.

[68] William Cook, Laszlo Lavasz, and Paul D. Seymour, *Combinatorial optimization: papers from the DIMACS Special Year*, AMS Bookstore, 1995.