AN ADAPTIVE RECONFIGURABLE ARCHITECTURE FOR IMAGE DENOISING

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

B.Tech.

in

ELECTRICAL AND ELECTRONICS ENGINEERING

By

BADRI NARAYANAN N (107113016)

SREYAS SRIRAM (107113091)

SUSHMITHA M (107113098)

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
TIRUCHIRAPPALLI-620015

MAY 2017

BONAFIDE CERTIFICATE

This is to certify that the project titled **An Adaptive Reconfigurable Architecture for Image Denoising** is a bonafide record of the work done by

BADRI NARAYANAN N (107113016)

SREYAS SRIRAM (107113091)

SUSHMITHA M (107113098)

in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in **ELECTRICAL AND ELECTRONICS ENGINEERING** of the **NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI**, during the year 2016-2017.

Dr. S. Moorthi Head of the Department

Project Guide

Project Viva-voce held on _____

Internal Examiner **External Examiner**

ABSTRACT

This project presents an architectural approach for the design of a low power Finite Impulse Response (FIR) Filter with efficient trade-off between the power consumption and filter performance. This architecture is most suited for fixed filter order for particular applications. Theoretical analysis of this approach show significant power savings without seriously compromising the filter performance. We also propose to enhance the FIR filter co-efficients using the Artificial Bee Colony (ABC) Algorithm. Simulation results indicate a significant denoising image in comparison to other algorithms.

Keywords: Denoising reconfigurable filter approximate filtering

ACKNOWLEDGEMENT

We would like to thank the following people for their support and guidance without whom the completion of this project in fruition would not be possible.

Dr.S. Moorthi, our project guide, for helping us and guiding us in the course of this project.

Dr.K.Sundareswaran, the Head of the Department, Department of Electrical and Electronics Engineering.

Our internal reviewers, **Dr P.S.Nayak** , **Dr S.Sudha** ,for their insight and advice provided during the review sessions.

We would also like to thank our individual parents and friends for their constant support.

TABLE OF CONTENTS

Title **Page No.**

ABSTRACT ..... **ii**
ACKNOWLEDGEMENT ..... **iii**
TABLE OF CONTENTS ..... **iv**
LIST OF TABLES ..... **vi**
LIST OF FIGURES ..... **vii**
ABBREVIATIONS ..... **ix**

1 INTRODUCTION **1**
1.1 Image Noise 1
1.1.1 Sources of Image Noise 1
1.1.2 Types of Image Noise 1
1.2 Image Denoising 4
1.2.1 Averaging filter 4
1.2.2 Median Filter 5
1.2.3 Order Statistics Filter 6
1.2.4 Adaptive Filter 7
1.2.5 Infinite Impulse Response Filters 8
1.2.6 FINITE IMPULSE RESPONSE FILTERS 10
1.2.7 FIR Filter vs IIR Filter 11

2 METHODOLOGY **15**
2.1 Reconfigurablility 15
2.1.1 Methods of Reconfiguration 15
2.2 Proposed Architecture 17

2.2.1 Reconfigurable FIR Filter 17
2.2.2 FIR Filtering to trade off filter performance and computation energy 18
2.2.3 Working Method 20
2.2.4 Extending a 1-d filter to 2-d 21

3 CO-EFFICIENT OPTIMIZATION ALGORITHM 22
3.1 Optimization 22
3.1.1 Genetic Algorithms for optimization 22
3.1.2 Particle Swarm Optimization 24
3.2 Artificial Bee Colony Algorithm 25
3.2.1 Properties in ABC Algorithm 27

4 RESULTS AND CONCLUSION 31
4.1 Fast Fourier Transform 31
4.2 Verilog Output for audio signal 32
4.3 Results from Verilog for images 33

REFERENCES 46

APPENDICES 47

A.1 Code Attachments 47

LIST OF TABLES

Table No. **Title** **Page No.**

4.1 Verilog Results for Audio signal 33

4.2 Verilog Output Specifications 35

4.3 Verilog Output Specifications 36

4.4 Verilog Output Specifications 37

LIST OF FIGURES

Fig. No.	Title	Page No.
1.1	Impulse Image Noise	2
1.2	Gaussian Image noise	2
1.3	Poisson Image Noise	3
1.4	Speckle Image Noise	4
1.5	Averaging Filter used on salt pepper and Gaussian Noise	5
1.6	Averaging Filter used on Poisson and Speckle Noise	5
1.7	Median Filter used on salt pepper and Gaussian Noise	5
1.8	Median Filter used on Poisson and Speckle Noise	6
1.9	Order Statistics Filter used on salt pepper Noise	6
1.10	Order Statistics Filter used on Gaussian Noise	6
1.11	Order Statistics Filter used on Poisson Noise	7
1.12	Order Statistics Filter used on Speckle Noise	7
1.13	Adaptive Filter used on salt pepper and Gaussian Noise	8
1.14	Adaptive Filter used on Poisson and Speckle Noise	8
2.1	Direct Form of a Reconfigurable FIR Filter	17
2.2	Proposed Form of a Reconfigurable FIR Filter	19
2.3	Amplitude Detector	20
2.4	Multiplier Control Signal Window	20
3.1	ABC Algorithm working	28
4.1	Input with Noise	32
4.2	Verilog Output	33
4.3	Input sample	34

Fig. No.	Title	Page No.

4.4 Output from Verilog row wise method 35

4.5 Output from Verilog row and column wise method 36

4.6 Output from MATLAB 37

4.7 Input sample 38

4.8 Output from Verilog row wise method 39

4.9 Output from Verilog row and column wise method 40

4.10 Output from MATLAB 41

4.11 Input sample 42

4.12 Output from Verilog row wise method 43

4.13 Output from Verilog row and column wise method 44

4.14 Output from MATLAB 45

ABBREVIATIONS

FIR	Finite Impulse Response
ABC	Artificial Bee Colony
FPGA	Field Programmable Gate Array
VLSI	Very Large Scale Integration
IIR	Infinite Impulse Response
DSP	Digital Signal Processor Radio
RF	Frequency
ASIC	Application Specific Integrated Circuit
PSO	Particle Swarm Optimisation
LUT	Look up Table Biological Solids
DF	Direct Form
LSB/MSB	Least/Most Significant Bit
FFT	Fast Fourier Transform
MCSD	Multiplier Control Signal Decision
AD	Amplitude Detection

Chapter 1

INTRODUCTION

1.1 Image Noise

Image noise is a random variation of image Intensity and visible as grains in the image. It may arise in the image as effects of basic physics-like photon nature of light or thermal energy of heat inside the image sensors. It can also be produced by the sensor and circuitry of a scanner or digital camera. In image noise the pixels in the image show different intensity values instead of true pixel values.

1.1.1 Sources of Image Noise

Noise is generally introduced in the image either during image acquisition or transmission. Several factors contribute for introduction of noise in the image. The quantification of the noise is determined by the number of pixels corrupted in the image. The major sources of noise in a digital image are: (1) The imaging sensor may be affected by environmental conditions during image acquisition. (2) Insufficient Light levels and sensor temperature may introduce the noise in the image. (3) Interference in the transmission channel may also corrupt the image. (4) If dust particles are present on the scanner screen, they can also introduce noise in the image.

1.1.2 Types of Image Noise

Impulse Noise (Salt and Pepper Noise)

The term impulse noise is also used for this type of noise. Other terms are spike noise, random noise or independent noise. Black and white dots appear in the image as a result

of this noise and hence salt and pepper noise. This noise arises in the image because of sharp and sudden changes of image signal. Dust particles in the image acquisition source or over heated faulty components can cause this type of noise. Image is corrupted to a small extent due to noise. The figure below shows the effect of this noise on the original image.

Figure 1.1: Impulse Image noise

Gaussian Noise (Amplifier Noise)

The term normal noise model is the synonym of Gaussian noise.

This noise model is additive in nature and follow Gaussian distribution. Meaning that each pixel in the noisy image is the sum of the true pixel value and a random, Gaussian distributed noise value. The noise is independent of intensity of pixel value at each point.

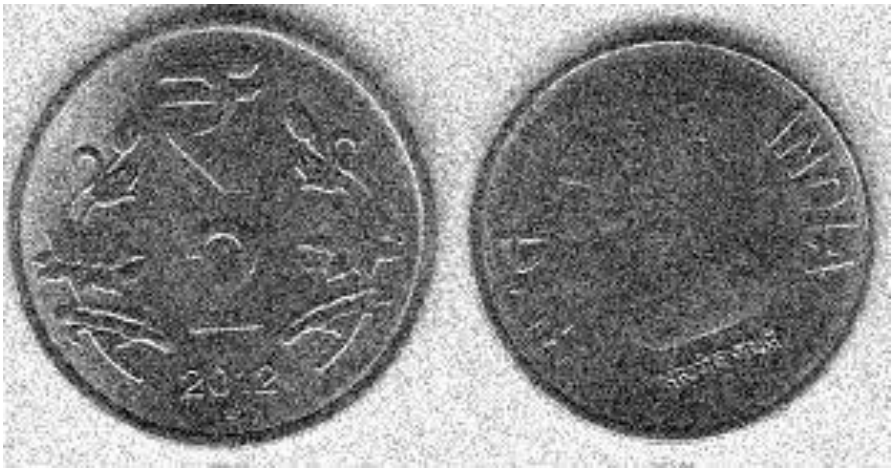

Figure 1.2: Gaussian Image Noise

Poisson Noise (Photon Noise)

Poisson or shot photon noise is the noise that can cause, when number of photons sensed by the sensor is not sufficient to provide detectable statistical information. This noise

has root mean square value proportional to square root intensity of the image. Different pixels are suffered by independent noise values. At practical grounds the photon noise and other sensor based noise corrupt the signal at different proportions. The figure shows the result of adding Poisson noise.

Figure 1.3: Poisson Image Noise

Speckle Noise

Radar waves can interfere constructively or destructively to produce light and dark pixels known as speckle noise. Speckle noise is commonly observed in radar (microwave or millimetre wave) sensing systems, although it may appear in any type of remotely sensed image utilizing coherent radiation. Like the light from a laser, the waves emitted by active sensors travel in phase and interact minimally on their way to the target area. After interaction with the target area, these waves are no longer in phase because of the different distances they travel from targets, or single versus multiple bounce scattering. Once out of phase, radar waves can interact to produce light and dark pixels known as speckle noise. Speckle noise in radar data is assumed to have multiplicative error model and must be reduced before the data can be utilized otherwise the noise is incorporated into and degrades the image quality. Ideally, speckle noise in radar images must be completely removed. However, in practice it can be reduced significantly. Reducing the effect of speckle noise permits both better discrimination of scene targets and easier automatic image segmentation.

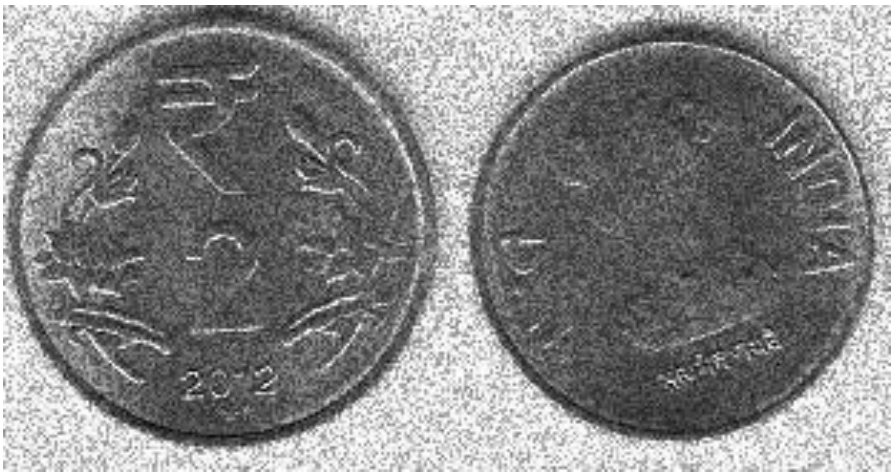

Figure 1.4: Speckle Image Noise

1.2 Image Denoising

Removing noise from the original signal is still a challenging problem for researchers. This project aims at providing an efficient architecture to achieve image denoising. Image denoising is one of the major image processing tasks. It is either an independent task or a component of other tasks. It is a process by which we reconstruct a signal from a noisy and distorted signal. The goal of image denoising is to remove noise and preserve useful information about the image. It is a pre-processing step for image analysis.

De-noising filters can be categorized in the following categories:

- Averaging filter
- Median filter
- Order Statistics filter
- Adaptive filter

1.2.1 Averaging filter

Averaging filtering is a simple, intuitive and easy to implement method of smoothing images, i.e. reducing the amount of intensity variation between one pixel and the next. It is often used to reduce noise in images. The idea of averaging filtering is simply to replace each pixel value in an image with the average (‘mean’) value of its neighbors, including itself. This has the effect of eliminating pixel values which are

unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighborhood to be sampled when calculating the


mean. Often a square with dimensions kernel is used, although larger kernels can be used for more severe smoothing.

Figure 1.5: Averaging Filter used on salt pepper and Gaussian Noise


Figure 1.6: Averaging Filter used on Poisson and Speckle Noise

1.2.2 Median Filter

Median filtering is a nonlinear method used to remove noise from images. It is widely used as it is very effective at removing noise while preserving edges. It is particularly effective at removing ‘salt and pepper’ type noise. The median filter works by moving through the image pixel by pixel, replacing each value with the median value of neighbouring pixels. The pattern of neighbours is called the ”window”, which slides, pixel by pixel over the entire image The median is calculated by first sorting all the pixel values from the window into numerical order, and then replacing the pixel being considered with the middle (median) pixel value.


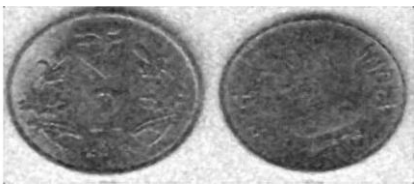

Figure 1.7: Median Filter used on salt pepper and Gaussian Noise

Figure 1.8: Median Filter used on Poisson and Speckle Noise

1.2.3 Order Statistics Filter

Order-Statistics filters are non-linear filters whose response depends on the ordering of pixels encompassed by the filter area. When the center value of the pixel in the image area is replaced by 100th percentile, the filter is called max-filter. On the other hand, if the same pixel value is replaced by 0th percentile, the filter is termed as minimum filter.

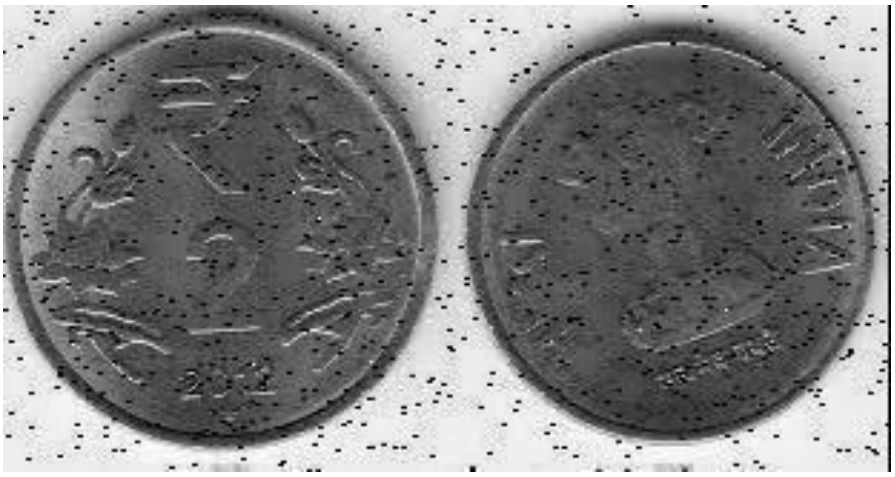

Figure 1.9: Order Statistics Filter used on salt pepper Noise

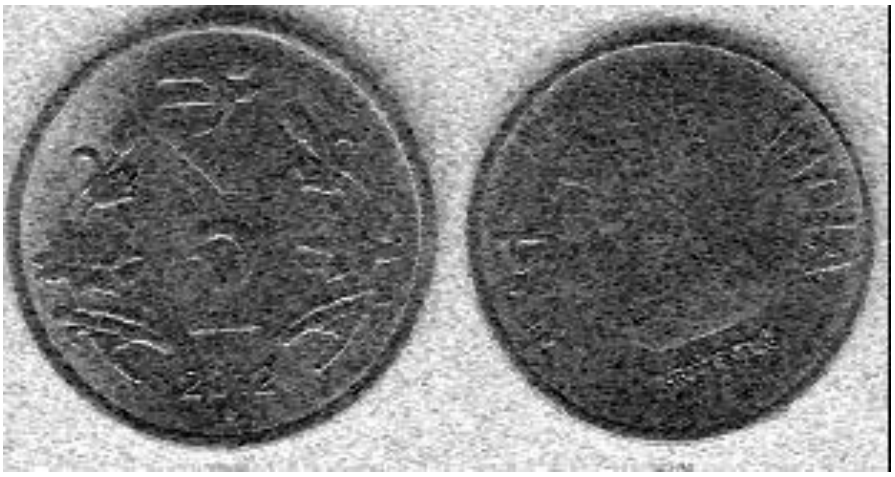

Figure 1.10: Order Statistics Filter used on Gaussian Noise

Figure 1.11: Order Statistics Filter used on Poisson Noise

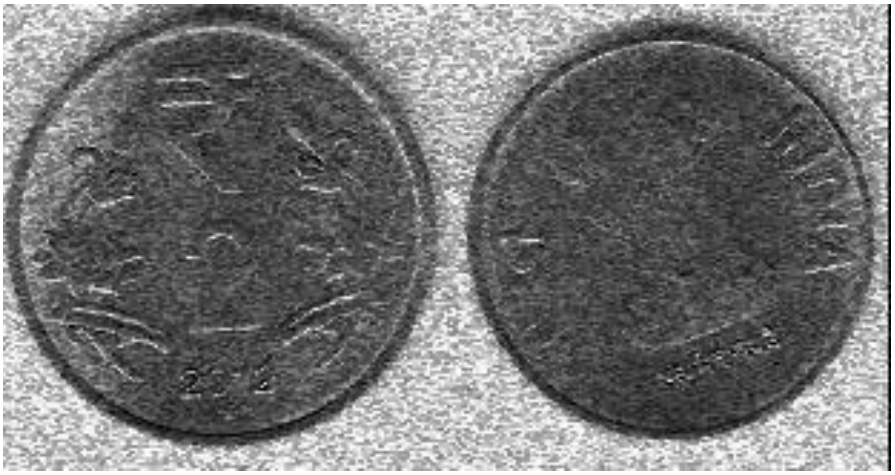

Figure 1.12: Order Statistics Filter used on Speckle Noise

1.2.4 Adaptive Filter

An adaptive filter is a computational device that attempts to model the relationship between two signals in real time in an iterative manner. Adaptive filters are often realized either as a set of program instructions running on an arithmetical processing device such as a microprocessor or DSP chip, or as a set of logic operations implemented in a field-programmable gate array (FPGA) or in a semi-custom or custom VLSI integrated circuit. However, ignoring any errors introduced by numerical precision effects in these implementations, the fundamental operation of an adaptive filter can be characterized independently of the specific physical realization that it takes.

This de-noising algorithm can be divided in three steps:

1. Analysis. Firstly similar image blocks are collected in groups. Blocks in each group are stacked together to form 3-D data arrays, which are de-correlated using an invertible 3D transform.

2. Processing. The obtained 3-D group spectra are filtered by hard thresholding.

3. Synthesis. The filtered spectra are inverted, providing estimates for each block in the group. These block-wise estimates are returned to their original positions and the final image reconstruction is calculated as a weighted average of all the obtained block-wise estimates.

Figure 1.13: Adaptive Filter used on salt pepper and Gaussian Noise

Figure 1.14: Adaptive Filter used on Poisson and Speckle Noise

1.2.5 Infinite Impulse Response Filters

IIR filters are digital filters with infinite impulse response. Unlike FIR filters, they have the feedback (a recursive part of a filter) and are known as recursive digital filters. Infinite impulse response (IIR) is a property applying to many linear time-invariant systems. Common examples of linear time-invariant systems are most electronic and digital filters. Systems with this property are known as IIR systems or IIR filters, and are distinguished by having an impulse response which does not become exactly zero past a certain point, but continues indefinitely. This is in contrast to a finite impulse response (FIR) in which the impulse response h(t) does become exactly zero at times t is greater than T for some finite T, thus being of finite duration. In practice, the impulse response, even of IIR systems, usually approaches zero and can be neglected past a certain point. However the

physical systems which give rise to IIR or FIR responses are dissimilar, and therein lies the importance of the distinction.

For instance, analog electronic filters composed of resistors, capacitors, and/or inductors (and perhaps linear amplifiers) are generally IIR filters. On the other hand, discrete-time filters (usually digital filters) based on a tapped delay line employing no feedback are necessarily FIR filters. The capacitors (or inductors) in the analog filter have a "memory" and their internal state never completely relaxes following an impulse. But in the latter case, after an impulse has reached the end of the tapped delay line, the system has no further memory of that impulse and has returned to its initial state; its impulse response beyond that point is exactly zero.

Although almost all analog electronic filters are IIR, digital filters may be either IIR or FIR. The presence of feedback in the topology of a discrete-time filter (such as the block diagram shown below) generally creates an IIR response. The z domain transfer function of an IIR filter contains a non-trivial denominator, describing those feedback terms. The transfer function of an FIR filter, on the other hand, has only a numerator as expressed in the general form derived below. All of the coefficients with i greater than 0 (feedback terms) are zero and the filter has no finite poles.

The transfer functions pertaining to IIR analog electronic filters have been studied and optimized for their amplitude and phase characteristics. These continuous time filter functions are described in the Laplace domain. Desired solutions can be transferred to the case of discrete-time filters whose transfer functions are expressed in the Z domain, through the use of certain mathematical techniques such as the bilinear transform, impulse invariance, or pole–zero matching method. Thus digital IIR filters can be based on well-known solutions for analog filters such as the Chebyshev filter, Butterworth filter, and Elliptic filter, inheriting the characteristics of those solutions.

1.2.6 Finite Impulse Response Filters

Finite impulse response (FIR) filters are the most popular type of filters implemented in software. This introduction will help you understand them both on a theoretical and a practical level.

Filters are signal conditioners. Each functions by accepting an input signal, blocking prespecified frequency components, and passing the original signal minus those components to the output. For example, a typical phone line acts as a filter that limits frequencies to a range considerably smaller than the range of frequencies human beings can hear. That's why listening to CD-quality music over the phone is not as pleasing to the ear as listening to it directly.

A digital filter takes a digital input, gives a digital output, and consists of digital components. In a typical digital filtering application, software running on a digital signal processor (DSP) reads input samples from an A/D converter, performs the mathematical manipulations dictated by theory for the required filter type, and outputs the result via a D/A converter.

An analog filter, by contrast, operates directly on the analog inputs and is built entirely with analog components, such as resistors, capacitors, and inductors.

There are many filter types, but the most common are lowpass, highpass, bandpass, and bandstop. A lowpass filter allows only low frequency signals (below some specified cutoff) through to its output, so it can be used to eliminate high frequencies. A lowpass filter is handy, in that regard, for limiting the uppermost range of frequencies in an audio signal; it's the type of filter that a phone line resembles.

A highpass filter does just the opposite, by rejecting only frequency components below some threshold. An example highpass application is cutting out the audible 60Hz AC power "hum", which can be picked up as noise accompanying almost any signal in the U.S.

A finite impulse response (FIR) filter is a filter structure that can be used to implement almost any sort of frequency response digitally. An FIR filter is usually implemented by using a series of delays, multipliers, and adders to create the filter's

output. The delays in the block diagram of a FIR Filter result in operating on prior input samples. The hk values are the coefficients used for multiplication, so that the output at time n is the summation of all the delayed samples multiplied by the appropriate coefficients.

The designer of a cell phone or any other sort of wireless transmitter would typically place an analog bandpass filter in its output RF stage, to ensure that only output signals within its narrow, government-authorized range of the frequency spectrum are transmitted.

Engineers can use bandstop filters, which pass both low and high frequencies, to block a predefined range of frequencies in the middle.

The process of selecting the filter's length and coefficients is called filter design. The goal is to set those parameters such that certain desired stopband and passband parameters will result from running the filter. Most engineers utilize a program such as MATLAB to do their filter design. But whatever tool is used, the results of the design effort should be the same. Engineers also use evolution strategies to optimize co-efficients as told by [1] in his paper.

1.2.7 FIR Filter vs IIR Filter

Because of the way FIR filters can be synthesized, virtually any filter response you can imagine can be implemented in an FIR structure as long as tap count isn't an issue. For example, Butterworth and Chebyshev filters can be implemented in FIR, but you may need a large number of taps to get the desired response. So we generally use prototypes other than the s domain polynomials as prototypes for FIR filters. As another example, if you want a band pass filter with two pass bands, or a band pass filter with specific phase properties, an FIR can do the job with no problem.

IIR filters on the other hand are essentially restricted to the well defined responses that can be achieved from the s domain polynomials such as the Butterworth. It is quite difficult to synthesize a user defined filter in an IIR structure.

It is quite possible however that the most important difference between an IIR and

FIR isn't the filter itself, but rather how the filter is implemented. The beauty of FIR filters, and quite possibly their most important feature, is that they can be implemented with integer math. As you are surely aware, everyone wants small, low power, low cost, portable devices. These devices typically use a processor similar to the Texas Instruments MSP430, or an FPGA, or an ASIC.

These types of processors work great and are as common as dirt, but seldom have a floating point math core. Integer math is the standard because its easy, cheap, fast, and low power. (Floating point math is then done in code, which is very slow.) So the only reasonable way to implement a digital filter in one of these devices is to use FIR. Thus it is the hardware, not the characteristics of the filter that determines whether FIR or IIR will be used.

IIR filters can't be implemented in integer math because the IIR coefficients can't be scaled to integer values without having the filter's math calculations explode (the problem is with the feedback coefficients). For example, scaling the coefficients by 16, which won't create nearly enough significant digits for filtering, will cause the calculations to go exponential. Try it.

The rest of this page shows some of the basic differences between an FIR and IIR filters. In the first plot below, we compare the frequency responses of a12 pole Butterworth IIR and a typical FIR filter (with emphasis on the word typical). The reader should understand that if we want to, we can implement the 12 pole Butterworth as an FIR and obtain an identical response, but that isn't typically done simply because of the excessive number of taps required to achieve the Butterworth response in an FIR.

These two filters have comparable magnitude responses. Notice the differences in the knee of the pass band at Omega = 0.2. In general, an IIR filter will have a sharper knee than a comparable FIR filter. Also, the slope of IIR filter's transition band will be 6 dB / octave / pole. A typical FIR transition band slope changes with frequency as shown here (again, emphasis on the word typical).

This FIR requires 50 multiplies + 50 adds while the IIR requires 30 multiplies + 36 adds for its 6 second order sections. The FIR has a constant group delay of 24.5 (equal

to (NumTaps-1)/2) while the IIR has a much lower, but non-constant group delay.

The group delay and impulse response are the most basic differences between an FIR and IIR filter. An IIR filter has, at least in principle, an infinitely long impulse response while an FIR filter's impulse response is as long as its tap count. If an infinitely long impulse response sounds problematic, remember that all analog filters have the same type of response.

An important difference between IIR and FIR filters is the potential for the IIR filter to be unstable. An FIR filter will be stable no matter how it is synthesized or implemented (it has no feedback). On the other hand, an IIR filter with improperly placed poles can't be made stable no matter how it is implemented. Please remember however that OpAmp filters are the same as IIR filters in this respect. Both IIR and OpAmp filters need to be carefully synthesized and implemented or they risk being unstable. It is difficult to get a stable IIR filter designed and implemented. on discussing IIR filter pole locations, comparing the frequency response of IIR filters to known stable filters, there are methods by which the engineer can assess the margin of stability the IIR filter has.

For the sake of comparing an FIR to an IIR, we will assume we are using a fixed point processor. Both the number of bits to left and to the right of the decimal place are important. The number of bits to the right determine the maximum attenuation one is able to achieve with the filter. The number of bits to the left determine the register's maximum value (overflow must never occur).

An important difference between FIR and IIR filters is the peak math values generated by the filter as a signal is processed. Because of the overshoot in the filters step response, square waves typically create the largest math values in a filter.

For the two filters shown on this page, the peak math register value for the FIR is only 1.3, and this is fairly typical of an FIR filter. In fact, FIR filters never generate math values greater than 2 (for a signal with plus or minus 1 amplitude and unity filter gain).

IIR filters are quite different in this respect. The IIR filter shown above has a peak

math value of 6.5 while filtering a square wave, but there is no such thing as a typical peak math value for IIR filters. They can range from 1.0 to 10,000 or even higher. The filter's peak math value is a strong function of the polynomial it is based on and its selectivity. The more selective the filter, the higher the math values. If working with relatively low distortion filters (moderate selectivity), you can expect the peak math value to be less than 32, but even this small value would require 4 more bits than an FIR filter.

Implementation Structures An important difference between IIR and FIR filters is in the implementation structure. FIR filters are usually implemented as Nth order polynomials. IIR filters on the other hand, can be implemented this way, but only if a floating point processor is available. If using a fixed point processor, an IIR filter must be implemented as a series of second order sections (biquads).

The reason for this lies with the magnitude of the math values generated by the filter, which we show in the following table. If using fixed point, the larger math values require more register bits to prevent overflow, which is strictly forbidden. Here we give the peak math values for an IIR filter implemented both ways. For example, if implemented in 2nd order sections, the Butterworth filter needs 3 integer bits to prevent overflow, but would require 18 integer bits if implemented as an Nth order poly. The number of fractional bits required would be the same in both cases. (Integer bits are needed to accommodate math values greater than 1, while the fractional bits are needed to achieve the desired stop band performance. These are completely separate problems.)

Chapter 2

METHODOLOGY

2.1 Reconfigurablility

Denotes the reconfigurable capability of a system, so that its behavior can be changed by reconfiguration, i. e. by loading different configuration. The ability to rearrange components; in the context of embedded systems, it is the ability to dynamically being able to change the hardware or software after the system has been deployed without manually reprogramming it.

Reconfigurability is often required in a variety of systems that use FIR Filters. The reconfigurable filter with change in co-efficients dynamically are some of the general requirements of FIR filter based applications. Apart from this reconfigurability of a filter reduces the complexity of the system thereby aiding better understanding and debugging.

2.1.1 Methods of Reconfiguration

The available reconfigurable methods can be classified as below:

1. Integrating multiplexers into the design
2. Partial reconfiguration (e.g., using ICAP)
3. Reconfigurable LUTs

Integrating Multiplexers into the design

The add and shift networks present in a typical FIR Filter circuit are replaced and integrated with multiplexers to support reconfigurability. Advantage of this method

is its fast reconfiguration within a single clock cycle. As good as the fast reconfiguration, there is a major drawback which is not of much use for present day applications. The change of co-efficients is not possible in this reconfigurable configuration method.

Partial Reconfiguration

It is implemented on a FPGA where partial regions are reconfigured via ICAP. This method of reconfiguring requires minimal or least resources. The filter can be used with arbitrary co-efficients. The major problem associated with this are that synthesis is required for each set of co-efficients. This results in slow reconfiguration in the range of milliseconds.

Reconfigurable LUTs

FPGAs implement combinational logic with the help of Look Up Tables (LUTs).This method of reconfiguration involves changing the contents of the Look Up Tables. The routing presented should be fixed. Similar to partial reconfiguration this involves arbitrary co-efficients and it has to be synthesized for every new set of co-efficients. The synthesis can be avoided if a general architecture is converted to fixed routing. The major advantage of this reconfiguration method is its speed which is in the range of nanoseconds. FPGA components for realizing reconfigurability are Shift-Register LUT for older versions and CFGLUT5 for newer versions. There are two methods suitable for FIR Filters that employ LUTs in a fixed structure. They are as below: Distributed Arithmetic and LUT based multipliers

Name	Logic	Routing
Integrating Multiplexers into the design	**Fixed**	**Flexible**
Partial Reconfiguration	**Flexible**	**Flexible**
Reconfigurable LUTs	**Flexible**	**Fixed**

2.2 Proposed Architecture

Many previous efforts for reducing power consumption of FIR filter generally focus on the optimization of the filter co-efficients while maintaining a fixed filter order. In those approaches, FIR filter structures are simplified to add and shift operations, and minimizing the number of additions/subtractions is one of the main goals of the research. However, one of the drawbacks encountered in those approaches is that once the filter architecture is decided, the coefficients cannot be changed; therefore, those techniques are not applicable to the FIR filter with programmable coefficients.

2.2.1 Reconfigurable FIR Filter

A direct form (DF) architecture of the reconfigurable FIR filter is shown below. In order to monitor the amplitudes of input samples and cancel the right multiplication operations, amplitude detector (AD) is used. When the absolute value of x(n) is smaller than the threshold x th , the output of AD is set to "1". The design of AD is dependent on the input threshold x th , where the fan-in's of AND and OR gate are decided by x th . If x th and c th have to be changed adaptively due to designer's considerations, AD can be implemented using a simple comparator.

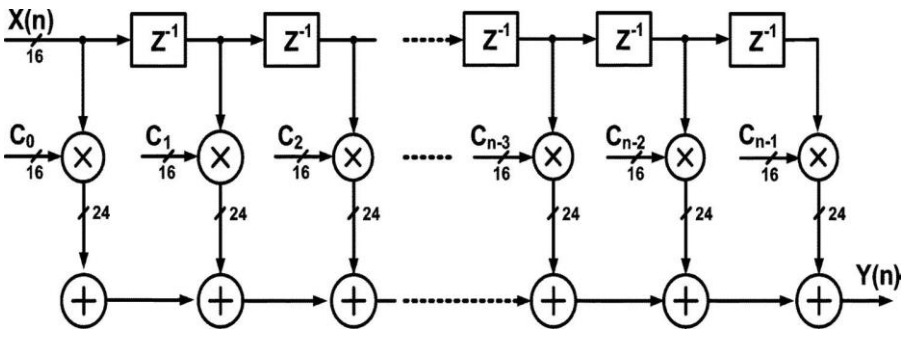

Figure 2.1: Direct Form of a Reconfigurable FIR Filter

2.2.2 FIR Filtering to trade off filter performance and computation energy

FIR filtering operation performs the weighted summations of input sequences, called as convolution sum, which are frequently used to implement the frequency selective low-pass, high-pass, or band-pass filters. Generally, since the amount of computation and the corresponding power consumption of FIR filter are directly proportional to the filter order, if we can dynamically change the filter order by turning off some of multipliers, significant power savings can be achieved. However, performance degradation should be carefully considered when we change the filter order.

Figure below exemplary shows the coefficients of a typical 25-tap low-pass FIR filter. The central coefficient has the largest value the coefficient has the largest value in the 25-tap FIR filter—and the amplitude of the coefficients generally decreases as becomes more distant from the center tap. The data inputs

of the filter, which are multiplied with the coefficients, also have large variations in amplitude. Therefore, the basic idea is that if the amplitudes of both the data input and filter coefficient are small, the multiplication of those two numbers is proportionately small; thus, turning off the multiplier has negligible effect on the filter performance. For example, since two's complement data format is widely used in the DSP applications, if one or both of the multiplier input has negative value, multiplication of two small values gives rise to large switching activities, which is due to the series of 1's in the MSB part. By canceling the multiplication of two small numbers, considerable power savings can be achieved with negligible filter performance degradation.

In the fixed point arithmetic of FIR filter, full operand bit-widths of the multiplier outputs is not generally used. In other words, when the bit-widths of data inputs and coefficients are 16, the multiplier generates 32-bit outputs. However, considering the circuit area of the following adders, the LSBs of multipliers outputs are usually truncated or rounded off, which incurs quantization errors. When we turn off the multiplier in the FIR filter, if we can carefully select the input and coefficient amplitudes such that the multiplication of those two numbers is as small as the quantization error, filter

performance degradation can be made negligible.

In the following, we denote threshold of input and threshold of coefficient as x th and c th , respectively. By threshold, we mean that when the filter input x (n) and coefficient c k are smaller than x th and c th , respectively, the multiplication is canceled in the filtering operation. When we determine x th and c th, the trade-off between filter performance and power savings should be carefully considered.

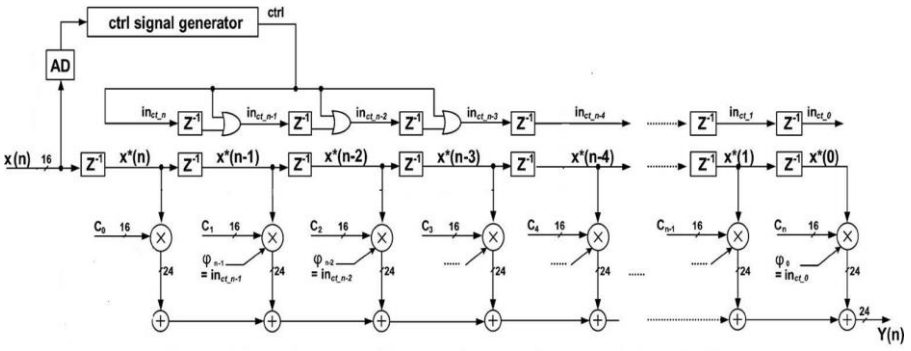

Figure 2.2: Proposed Form of a Reconfigurable FIR Filter

Dynamic power consumption of CMOS logic gates is a strong function of the switching activities on the internal node capacitances. In the proposed reconfigurable filter, if we turn off the multiplier by considering each of the input amplitude only, then, if the amplitude of input abruptly changes for every cycle, the multiplier will be turned on and off continuously, which incurs considerable switching activities. This is similar to the method proposed by [2] in his work on dynamic power consumption. Multiplier control signal decision window (MCSD) is used to solve the switching problem. Using ctrl signal generator inside MCSD, the number of input samples consecutively smaller than are counted and the multipliers are turned off only when consecutive input samples are smaller than . Here, means the size of MCSD is equal to 4.

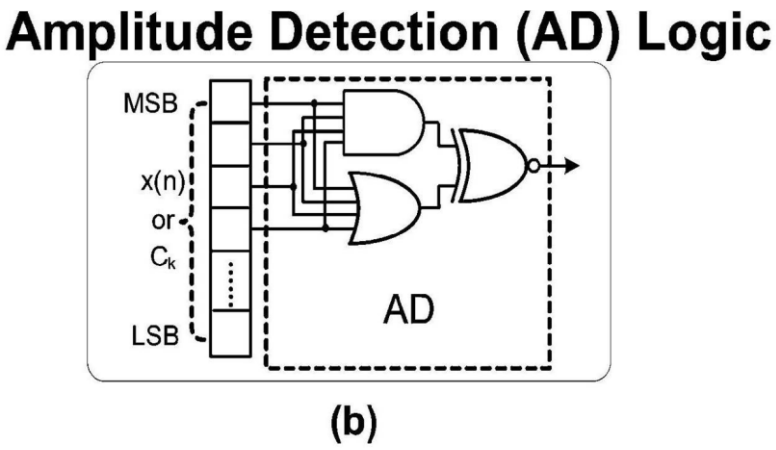

Figure 2.3: Amplitude Detector

In the ctrl generator design, as an input smaller than comes in and AD output is set to "1", the counter is counting up. When the counter reaches , the ctrl signal in the figure changes to "1", which indicates that consecutive small inputs are monitored and the multipliers are ready to turn off. One additional bit, in Figure below, is added and it is controlled by ctrl. The accompanies with input data all the way in the following flip-flops to indicate that theinput

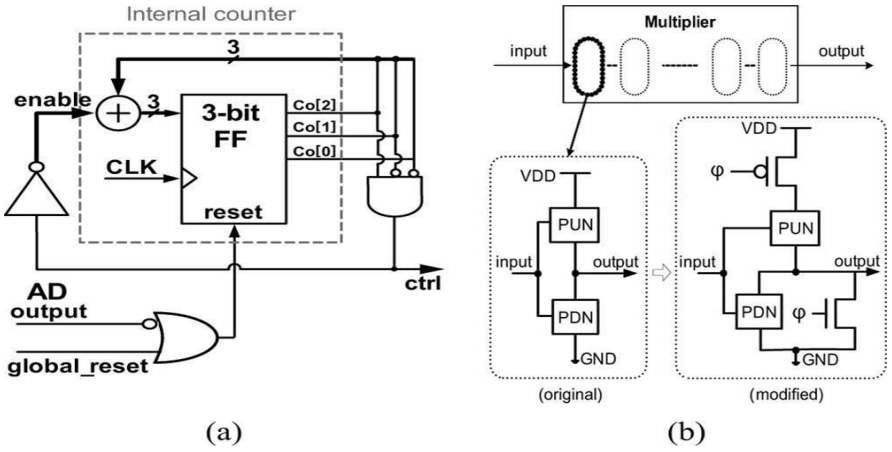

Figure 2.4: Multiplier Control Signal Window

2.2.3 Working Method

The proposed Architecture for one dimensional filtering was initially applied to an audio signal. A high frequency noise was added to an already existing audio recording and the corresponding. The results were obtained and the architecture was extended to 2-D and the co-efficients were optimized using the Artificial Bee Colony Algorithm.

2.2.4 Extending a 1-D filter to 2-D

There are three types of methods to modify a one dimensional filter to a two dimensional filter. The methods available are:

1. Either row or column
2. Both rows and columns
3. 2-D Filter design

With the motive to reduce the power consumption, this project uses the both rows and columns method. A trade off between the power consumption and performance is drawn between both the methods.

Chapter 3

CO-EFFICIENT OPTIMIZATION ALGORITHM

3.1 Optimization

Optimization is an applied science which explores the most efficient values of the parameters of a given problem statement under specific conditions. Optimization, in simple words, aims to obtain the relevant parameter values which enable an objective function to generate the minimum or maximum value. The design of an optimization problem generally starts with the design of an objective function. The objective function must correctly define the related problem mathematically and clearly express the relation between the parameters of the problem. Furthermore, if any pre-defined constraints are to be used in respect of the parameters of the related problem, these should be considered during the design of the optimization problem. Generally, the optimization problems are classified under many subtitles; Linear Programming, Integer Programming, Quadratic Programming, Combinatorial Optimization and Metaheuristics are the terms often used to classify the optimization methods.

3.1.1 Genetic Algorithms for optimization

Genetic Algorithm (GA) works on the theory of Darvin's theory of evolution and the survival-of-the fittest. Genetic algorithms guide the search through the solution space by using natural selection and genetic operators, such as crossover, mutation and the selection. GA encodes the decision variables or input parameters of the problem into solution strings of a finite length. While traditional optimization techniques work

directly with the decision variables or input parameters, genetic algorithms usually work with the coding. Genetic algorithms start to search from a population of encoded solutions instead of from a single point in the solution space. The initial population of individuals is created at random. Genetic algorithms use genetic operators to create Global optimum solutions based on the solutions in the current population. The most popular genetic operators are

(1) selection

(2) crossover

(3) mutation.

The newly generated individuals replace the old population, and the evolution process proceeds until certain termination criteria are satisfied.

1. Selection

The selection procedure implements the natural selection or the survival-of-the fittest principle and selects good individuals out of the current population for generating the next population according to the assigned fitness. A generic selection procedure is as follows: The fitness function is evaluated for each individual, providing fitness values, which are then normalized. Normalization means dividing the fitness value of each individual by the sum of all fitness values, so that the sum of all resulting fitness values equals 1. The population is sorted by descending fitness values. Accumulated normalized fitness values are computed (the accumulated fitness value of an individual is the sum of its own fitness value plus the fitness values of all the previous individuals). The accumulated fitness of the last individual should be 1 (otherwise something went wrong in the normalization step). A random number R between 0 and 1 is chosen. The selected individual is the first one whose accumulated normalized value is greater than R. For a large number of individuals the above algorithm might be computationally quite demanding. A simpler and faster alternative uses the so-called stochastic acceptance. If this procedure is repeated until there are enough selected individuals, this selection method is called fitness proportionate selection or roulette-wheel selection. If instead of a single pointer spun multiple times, there are

multiple, equally spaced pointers on awheel that is spun once, it is called stochastic universal sampling. Repeatedly selecting the best individual of a randomly chosen subset is tournament selection. Taking the best half, third or another proportion of the individuals is truncation selection. There are other selection algorithms that do not consider all individuals for selection, but only those with a fitness value that is higher than a given (arbitrary) constant. Other algorithms select from a restricted pool where only a certain percentage of the individuals are allowed, based on fitness value. Retaining the best individuals in a generation unchanged in the next generation, is called elitism or elitist selection. It is a successful (slight) variant of the general process of constructing a new population.

2. Crossover

Crossover, also called the recombination operator, exchanges parts of solutions from two or more individuals, called parents, and combines these parts to generate new individuals, called children, with a crossover probability. There are a lot of ways to implement a recombination operator. The well-known crossover operators include one-point crossover.

3. Mutation

Mutation usually alters some pieces of individuals to form perturbed solutions. In contrast to crossover, which operates on two or more individuals, mutation operates on a single individual. One of the most popular mutation operators is the bitwise mutation, in which each bit in a binary string is complemented with a mutation probability.

3.1.2 Particle Swarm Optimization

Particle swarm optimization (PSO) is an evolutionary computation technique developed by Kennedy and Eberhart. It exhibits common evolutionary computation attributes including initialization with a population of random solutions and searching for optima by updating generations. Potential solutions, called particles, are then “flown” through the problem space by following the current optimum particles. The particle swarm

concept was originated as a simulation of a simplified social system. The original intent was to graphically simulate the graceful but unpredictable choreography of a bird flock. Each particle keeps track of its coordinates in the problem space, which are associated with the best solution (fitness) it has achieved so far. This value is called ‘pBest’. Another “best” value that is tracked by the global version of the particle swarm optimization is the overall best value and its location obtained so far by any particle in the population. This location is called ‘gBest’. The particle swarm optimization concept consists of, at each step, changing the velocity (i.e. accelerating) of each particle toward its ‘pBest’ and ‘gBest’ locations (global version of PSO). Acceleration is weighted by a random term with separate random numbers being generated for acceleration toward ‘pBest’ and ‘gBest’ locations.

Unlike genetic algorithm, PSO algorithm does not need complex encoding and decoding process and special genetic operator. PSO takes real number as a particle in the aspect of representation solution and the particles update themselves with internal velocity. In this algorithm, the evolution looks only for the best solution and all particles tend to converge to the best solution.

3.2 Artificial Bee Colony Algorithm

The Artificial Bee Colony (ABC) algorithm is a population-based numeric optimization algorithm. It is based on the simplified mathematical models of the food searching behaviors of the bee-swarms. Bonabeau has defined swarm intelligence as “any at- tempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies” The term swarm in general refers to any restrained collection of interacting agents or individuals. The two fundamental concepts involved in this algorithm are self organization and division of labor. They are necessary and sufficient properties to obtain swarm intelligent behavior. The process of forage selection starts when bees leave the hive of a forage to search for a food source (nectar). After finding nectar, the bees store it in their stomach. After coming back to the hive, the bees unload the nectar and perform a waggle dance to share their information about the food source

(nectar quantity, distance and direction from black the hive) and recruit new bees for exploring most rich food sources. The three essential components of forage selection:

- Food Sources: The value of a food source depends on many factors such as its proximity to the nest, its richness or concentration of its energy, and the ease of extracting this energy.
- Employed Foragers: They are associated with a particular food source which they are currently exploiting or are "employed" at. They carry with them information about this particular source, its distance and direction from the nest, the profitability of the source and share this information with a certain probability.
- Unemployed Foragers: They are continually at look out for a food source to exploit. There are two types of unemployed foragers: onlooker bees or scout bees.

Onlooker bees: onlooker bees receive information from employed bees about the quality of food sources and choose food sources with better quality to explore the neighbor- hood. At the moment that onlooker bees choose a food source to explore, they become employed bees.

Scout bees: employed bees become scout bees when a food source is exhausted. In other words, the employed bees explored a food source neighborhood MAX LIMIT times; however, they did not find any food source with better quality. Scout bees try to find new food sources. The model defines two leading modes of the behavior:

(1) recruitment to a nectar source

(2) the abandonment of a source

The exchange of information among bees is the most important occurrence in the formation of collective knowledge. The most important part of the hive with respect to exchanging information is the dancing area Communication among bees related to the quality of food sources takes place in the dancing area. This dance is called a Waggle dance. By performing waggle dance honey bees communicate:

1. The direction of flower patches (Angle between the sun and patch)
2. The distance from the hive (Duration of the dance)
3. The quality rating (Frequency of the dance)

Employed foragers share their information with a probability proportional to the profitability of the food source, and the sharing of this information through waggle dancing is longer in duration. An onlooker on the dance floor, decides to employ herself at the most profitable source. There is a greater probability of onlookers choosing more profitable sources since more information is circulated about the more profitable sources. The first half of the colony consists of the employed artificial bees and the second half includes the onlookers. The number of employed bees is equal to the number of food sources around the hive. The employed bee whose food source has been exhausted by the bees becomes a scout.

3.2.1 Properties in ABC Algorithm

The two fundamental concepts involved in this algorithm are self-organization and division of labor. They are necessary and sufficient properties to obtain swarm intelligent behavior.

(1) The self-organization properties are:

- Positive Feedback: The nectar amount of food sources is proportional to the number of onlookers. As the nectar amount of food sources increases, the number of onlookers visiting them increases, too.
- Negative Feedback: The exploitation process of poor food sources is stopped by bees.
- Fluctuations: The scouts carry out a random search process for discovering new food sources.
- Multiple interactions: Bees share their information about food sources with their nest mates on the dance area.

(2) The Division of labor properties are:

- Different tasks are performed simultaneously by specialized cooperating individuals
- Enable the swarm to react to changed conditions in the search space

Working

Each cycle of search consists of three steps: moving the employed and onlooker bees onto the food sources and calculating their nectar amounts; and determining the scout

bees and directing them onto possible food sources. A food source position represents a possible solution to the problem to be optimized. The amount of nectar of a food source corresponds to the quality of the solution Onlookers are placed on the food sources by using a probability based selection process. As the nectar amount of a food source increases, the probability value with which the food source is preferred by onlookers increases, too. The scouts are characterized by low search costs and a low average in food source quality. The selection is controlled by a control parameter called ”limit”. If a solution representing a food source is not improved by a predetermined number of trials, then that food source is abandoned and the employed bee is converted to a scout.

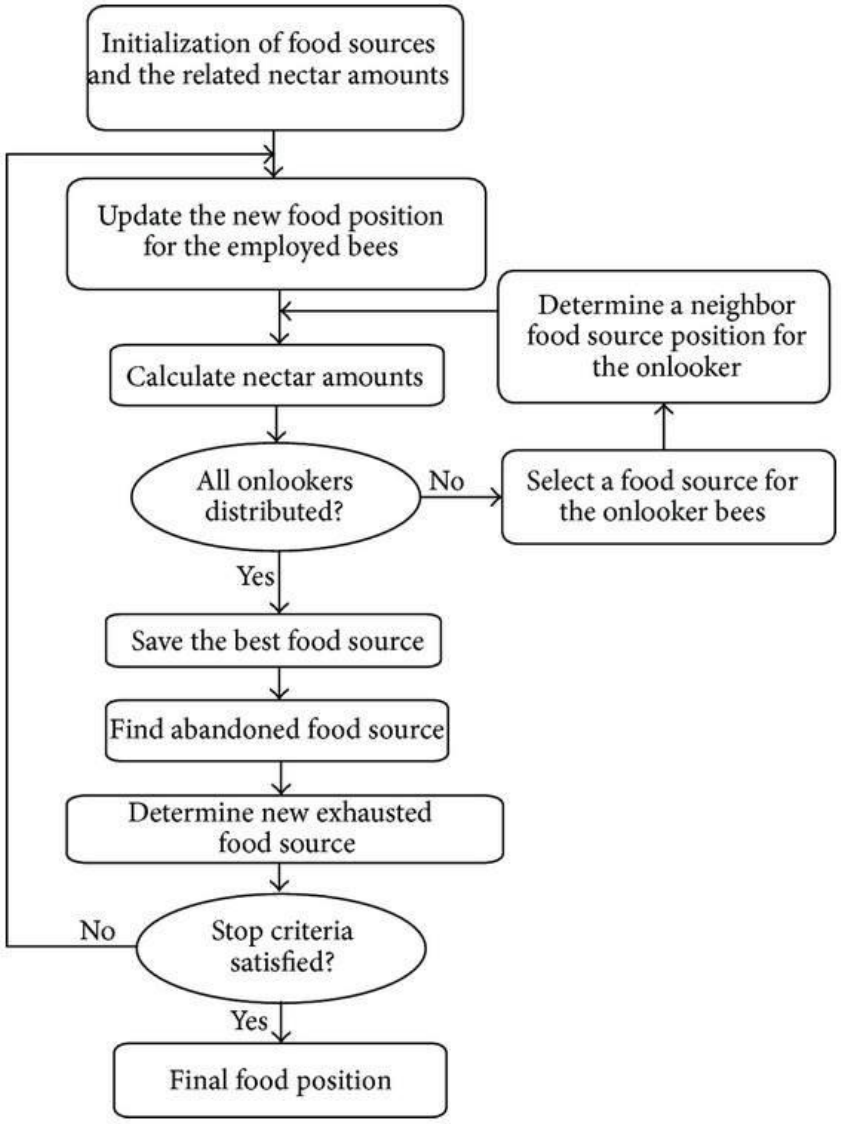

Figure 3.1: ABC Algorithm working

Step-wise Algorithm

Step 1: Initialize the population of solutions i=1 to SN, j=1 to D (SN: number of solutions in the colony), (D: the number of optimization parameters of 2D filter)

Step 2: Evaluate the population by using equation MSE

Step 3: cycle=1, Repeat

Step 4: Produce a new solution for each employed bee

Step 5: Apply the greedy selection process for the employed bees

Step 6: Calculate the probability values for the solutions

Step 7: Produce the new solutions v for the onlookers from the solutions x selected depending on p values and evaluate them

Step 8: Apply the greedy selection process for the onlookers

Step 9: Determine the abandoned solution for the scout, if exists, and replace it with a new randomly produced solution

Step 10: Memorize the best solution achieved so far

Step 11: cycle = cycle + 1

Advantages of ABC Algorithm

The success of the ABC algorithm in finding the global optimum is sensitive to the control parameters of the algorithm (i.e, the limit value, the number of the employed-bees, the population size, and the maximum cycle value). The local-search ability of the ABC algorithm is sufficiently strong for various problem types. The ABC algorithm selects the pattern used in defining the search-direction using the probability values and the roulette-selection rule used in the genetic-algorithm. In the ABC algorithm, the probability values are calculated using the fitness values. The mathematical model used in the standard ABC algorithm while calculating the fitness values causes it to produce equal probability values for the local solutions that have equal absolute values but different signs. This decreases the probability of a pattern that provides a relatively better solution being selected to define the search-direction. The modified-ABC algorithm calculates the fitness values for different local-solutions

by using different method. Therefore, it is appropriate to sort the patterns in the probability values of pattern matrix acquired in the end just nonlinearly according to the quality of the solution they acquire. Thus, the strategies used to calculate the probability values in the ABC algorithm produce the pseudo-probability value that is suitable to grade the patterns in the pattern matrix just nonlinearly. This affects the problem solving success of the ABC algorithm significantly, and decreases its local-search ability. As stated by [3] in his paper, this algorithm can be used for image denoising by using two dimensional design.

Chapter 4

RESULTS AND CONCLUSION

4.1 Fast Fourier Transform

The FFT is a complicated algorithm, and its details are usually left to those that specialize in such things. This section describes the general operation of the FFT, but skirts a key issue: the use of complex numbers. If you have a background in complex mathematics, you can read between the lines to understand the true nature of the algorithm. Don't worry if the details elude you; few scientists and engineers that use the FFT could write the program from scratch.

In complex notation, the time and frequency domains each contain one signal made up of N complex points. Each of these complex points is composed of two numbers, the real part and the imaginary part. For example, when we talk about complex sample X[42], it refers to the combination of ReX[42] and ImX[42]. In other words, each complex variable holds two numbers. When two complex variables are multiplied, the four individual components must be combined to form the two components of the product. The following discussion on "How the FFT works" uses this jargon of complex notation. That is, the singular terms: signal, point, sample, and value, refer to the combination of the real part and the imaginary part.

The FFT operates by decomposing an N point time domain signal into N time domain signals each composed of a single point. The second step is to calculate the N frequency spectra corresponding to these N time domain signals. Lastly, the N spectra are synthesized into a single frequency spectrum.

4.2 Verilog Output for audio signal

Noise above 2000 Hz was added to the original sound sample to obtain the noisy sample, whose FFT is shown in Fig 1. This sample was used as the input to the 1D FIR Filter, which is a 16-tap FIR filter. The input being 1-dimensional in nature was sampled at 8000 Hz and fed one value a time to the filter. The FIR filter was simulated for the input sample and the resultant output is as shown in Figure (FFT).

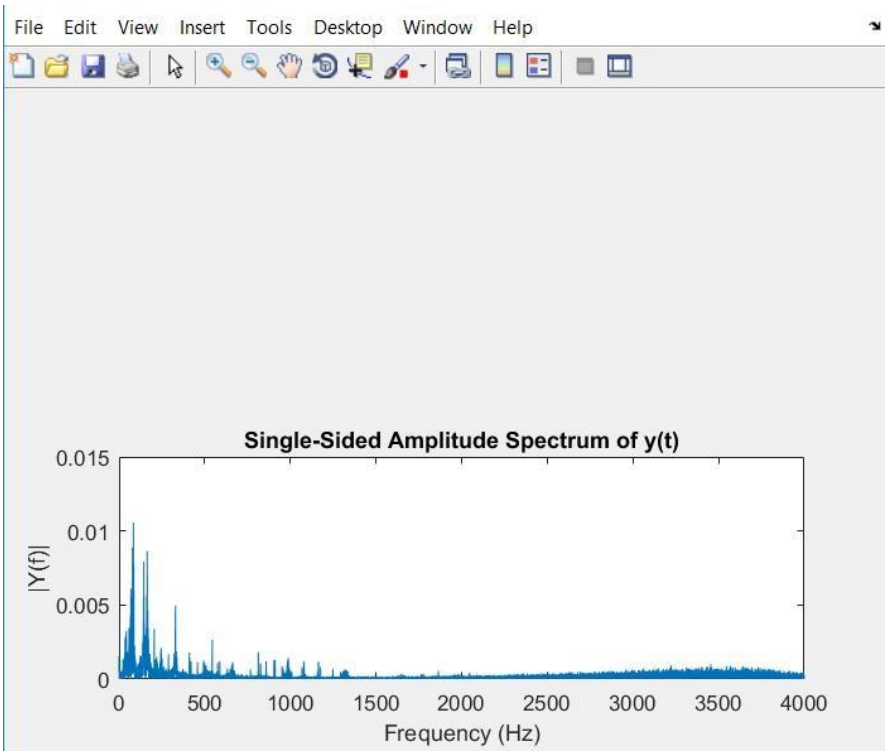

Figure 4.1: Input with Noise

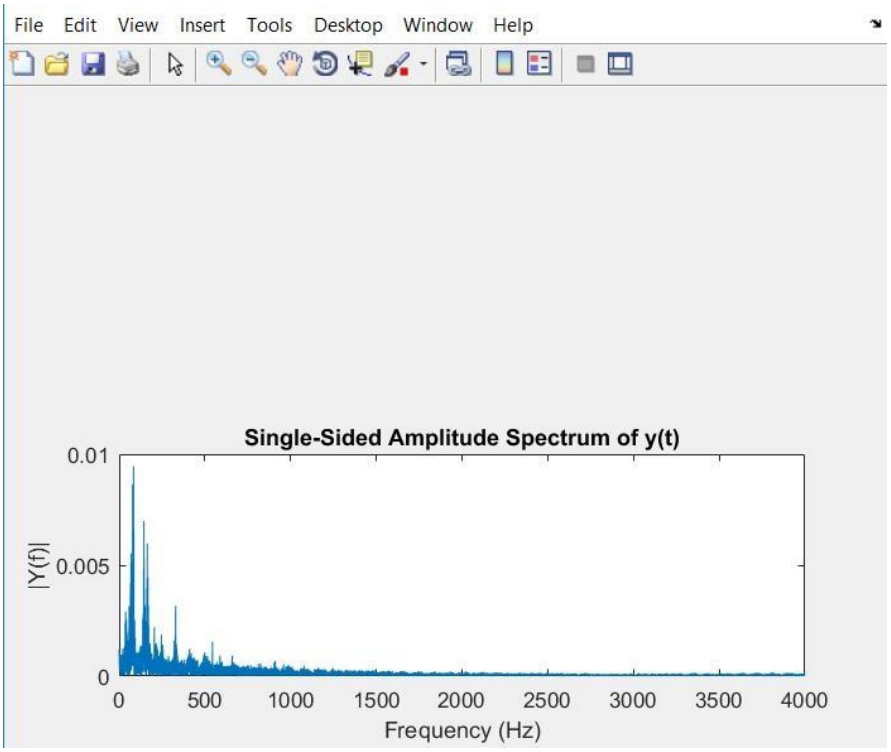

Figure 4.2: Verilog Output

Audio sample rate	8000 Hz (10 sec sample) – 80 000 values			
Clock period (used in simulation)	100 ns			
Coefficients Used	-0.0004	0.0023	-0.0074	-0.0190
	-0.0421	0.0867	-0.1851	0.6271
	0.6271	-0.1851	0.0867	-0.0421
	-0.0190	-0.0074	0.0023	0.0004
Run time (seconds)	174.6 us			

Table 4.1: Verilog Results for Audio signal

4.3 Results from Verilog for images

Gaussian noise (White-noise) with a mean of 0.5 was added to the original image to obtain the sample image shown in Figure. This image was used as the input to the 1D FIR Filter of order 4 which works on the principle of Artificial Bee Colony Algorithm. Since, the input is 2-dimensional in nature, multiple instances of the 1D FIR Filter is called row-wise followed by column-wise. The Artificial Bee Colony algorithm based FIR filter was simulated 30 times for the input image and the resultant output is as shown.

Figure 4.3: Input sample

Figure 4.4: Output from Verilog row wise method

Table 4.2: Verilog Output Specifications

Image Dimension	309 X 309
Clock period (used in simulation)	100 ns
Coefficients Used	-0.3834, 0.8973, 0.8973, -0.3834
Run time (seconds)	190.9 us

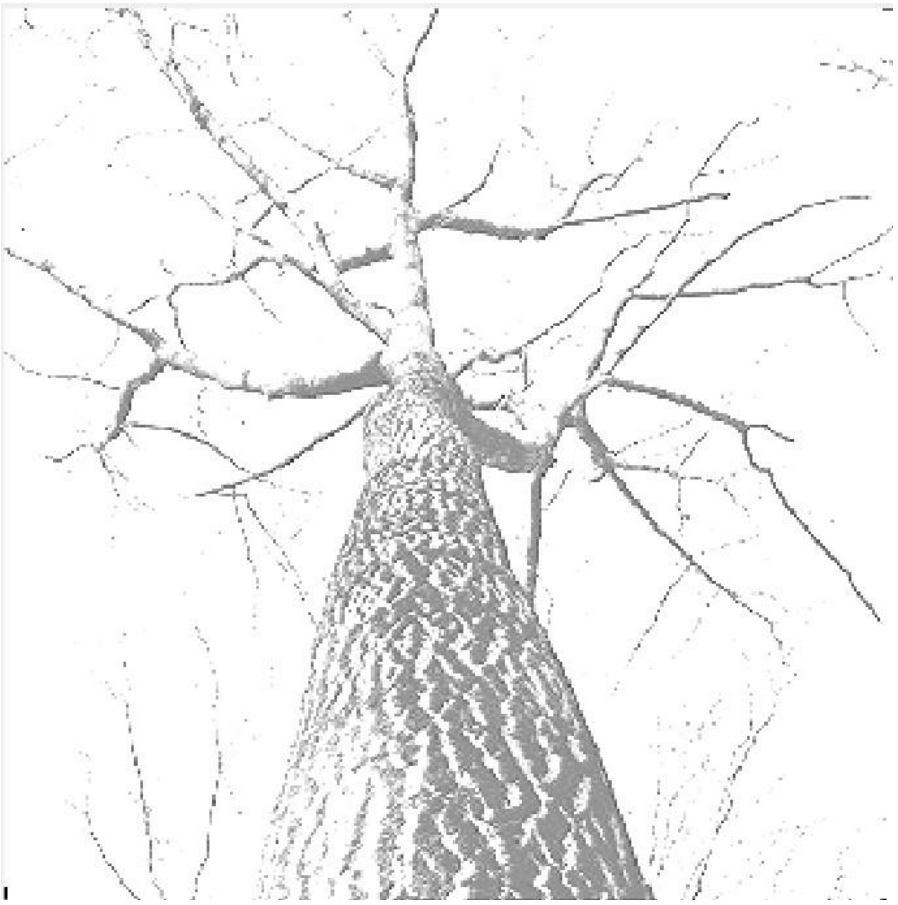

Figure 4.5: Output from Verilog row and column wise method

Table 4.3: Verilog Output Specifications

Image Dimension	309 X 309
Clock period (used in simulation)	100 ns
Coefficients Used	-0.3834, 0.8973, 0.8973, -0.3834
Run time (seconds)	381.8 us

Figure 4.6: Output from MATLAB

Table 4.4: Verilog Output Specifications

Image Dimension	309 X 309
Number of Iterations	20
Coefficients Used	-0.3834, 0.8973, 0.8973, -0.3834
Run time (seconds)	47 s

Figure 4.7: Input sample

Figure 4.8: Output from verilog row wise method

Figure 4.9: Output from Verilog row and column wise method

Figure 4.10: Output from MATLAB

Figure 4.11: Input sample

Figure 4.12: Output from Verilog row wise method

Figure 4.13: Output from Verilog row and column wise method

Figure 4.14: Output from MATLAB

References

1. **Shuangteng Zhang**, Department of Computer Science, Eastern Kentucky University, “Image Denoising Using FIR Filters Designed with Evolution Strategies”, 2011

2. **Serdar Kockanat, Nurhan Karaboga, Turker Koza**, “Image Denoising with 2-D FIR Filter by Using Artificial Bee Colony Algorithm”, Innovations in Intelligent Systems and Applications (INISTA), 2012 International Symposium

3. **Kartik V Hegde, Vadiraj Kulkarni, Harshavardhan R, Sumam David S**, “Adaptive Reconfigurable Architecture for Image Denoising”, IEEE International Parallel and Distributed Processing Symposium Workshops, 2015

4. **Seok-Jae Lee, Ji-Woong Choi, Seon Wook Kim, Jongsun Park**, " A Reconfigurable FIR Filter Architecture to Trade Off Filter Performance for Dynamic Power Consumption”, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, June 2013

5. **Qiang Guo, Caiming Zhang, Yunfeng Zhang, and Hui Liu**, “Image denoising using 2-D FIR filters designed with DEPSO”, Multimed Tools Appl (2014)

6. **Qiang Guo, Caiming Zhang, Yunfeng Zhang, and Hui Liu**, “An Efficient SVD- Based Method for Image Denoising”, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, May 2016

APPENDIX A

CODE ATTACHMENTS

A.1 Verilog and MATLAB Code

A sample code of the Amplitude Detector circuit written in Verilog for implementation of the proposed architecture is shown below. The ABC algorithm used to optimise coefficients of a FIR Filter is written in MATLAB and a code snippet is attached here.

```
assign or1 = data[7] | data[6] | data[5] | data[4] |
data[3];
assign out = and1 ~^ or1;


initial begin
 ff <= 3'b000;
end

amp_detection amp1(
 .data(inp_data),
 .out(ad_output)
);

always @(posedge clk or posedge reset) begin
   if(reset)
          ff <= 3'b000;
   else
          ff <= ff + enable;
end


always @(posedge clk) begin
     inct[15] <= ctrl_xin;
     xin[15] <= x_data;
     for(i = 15; i>0; i = i-1) begin
         if(i > 11)
              inct[i-1] <= #1  inct[i] | ctrl_xin;
         else
              inct[i-1] <=  #1 inct[i];
              xin[i-1] <= #1 xin[i];
         end
     for(j = 15; j>=0; j = j-1) begin
         if(j >0) begin
              if(inct[j] == 0)
                     mul_out[j] <=  #1 xin[j]*cin[j];
              else
                     mul_out[j] <=  #1 0;
              y_inter[j-1] <= #2 y_inter[j] + mul_out[j];
         end
         else begin
```

```
                if(inct[j] == 0)
                      mul_out[j] <=  #1 xin[j]*cin[j];
                else
                      mul_out[j] <=  #1 0;
                      y <=  #2  y_inter[j] + mul_out[j];
                end
          end
     end


%Co-efficient generation for ABC

for i=1:N

      m= randi([3000,8000],[1,Order])/5000-1;
      m(Order)=0;
      summer=0;

      for lo=1:Order-1
           summer=summer+m(lo);
      end
      m(Order)=0.5-summer;
      for j=1:Order
      x(i,j)=m(j);
end

%Evaluation row-wise followed by column-wise

for i=1:N
        for i1=1:Breadth %image
        for i2=1:Length %image
                for k=1:4
                    if(i2 - k >= 0)

I3(i1,i2,i)=I3(i1,i2,i)+x(i,k)*double(I(i1,(i2-k)+1));
                end
            end
        end
        end
    end

    for i=1:N
        for i1=1:Breadth %image
        for i2=1:Length %image
                for k=1:4
                    if(i2 - k >= 0)

I4(i1,i2,i)=I4(i1,i2,i)+v(i,k)*double(I(i1,(i2-k)+1));
                end
            end
        end
        end
    end
```

```
    for i=1:N
        for i2=1:Length %image
            for i1=1:Breadth %image
                for k=1:4
                    if(i1 - k >= 0)
            I1(i1,i2,i)=I1(i1,i2,i)+x(i,k)*I3(i1-
k+1,i2,i);
                end
            end
        end
        end
    end

    for i=1:N
        for i2=1:Length %image
          for i1=1:Breadth %image
                for k=1:4
                    if(i1 - k >= 0)
            I2(i1,i2,i)=I2(i1,i2,i)+v(i,k)*I4(i1-
k+1,i2,i);
                end
            end
        end
        end
    end

    for i=1:N
        for i1=1:Breadth
            for i2=1:Length
    mse1(i)=mse1(i)+(I1(i1,i2,i)-double(I(i1,i2)));
            end
        end
    end

      for i=1:N
        for i1=1:Breadth
            for i2=1:Length
    mse2(i)=mse2(i)+(I2(i1,i2,i)-double(I(i1,i2)));
            end
        end
      end
```