NOVEL IMPLEMENTATION TECHNIQUE FOR A WAVELET-BASED BROADBAND SIGNAL DETECTION SYSTEM

By

Sheng Cheng

Department of Engineering

University of Warwick

A thesis submitted to the University of Warwick for the degree of Doctor of Philosophy

August 2010

Contents

ACKNOWLEDGEMENTS

I would like to express my great appreciations to Dr. Marina Cole for her supervision, guidance and continuous support.

I also would like to give my thankfulness to Chien-Hsun Tseng, who cooperate with me in this project and helped me a lot in realization of the project, and Mr Hugh Dumbrell from Defense Science and Technology Laboratory (DSTL) of Ministry of Defense (MOD) who provides the real sonar data used for simulation in this project.

I want to express my gratitude for all the people helping me in the project implementation. To the staff in the School of Engineering, especially Dr. Richard Staunton, for his support on development tools.

Most of all, I would like to thank my family and Jiayu Yao, my girlfriend for their patience, love and inspiration.

DECLARATION

This thesis is presented according to the regulations for the degree of Doctor of Philosophy.

The work described in this report is original and is the result of my own work and investigations except where otherwise indicated.

The Matlab model presented in chapter IV of this work was development in collaboration with Dr. Chien-Hsun Tseng.

The thesis has not been submitted in any previous application for a degree at another university.

Parts of this work have been presented at international conferences:

1. Sheng Cheng; Chien-Hsun Tseng; Cole, M.;" A Novel FPGA Implementation of a Wideband Sonar System for Target Motion Estimation", Reconfigurable Computing and FPGAs, 2008. Page(s):349 – 354
2. Sheng Cheng; Chien-Hsun Tseng; Cole, M.; ICECS 2007" Efficient and Effective VLSI Architecture for a Wavelet-based Broadband Sonar Signal Detection System". Page(s):593 - 596

ABSTRACT

This thesis reports on the design, simulation and implementation of a novel Implementation for a Wavelet-based Broadband Signal Detection System.

There is a strong interest in methods of increasing the resolution of sonar systems for the detection of targets at sea. A novel implementation of a wideband active sonar signal detection system is proposed in this project. In the system the Continuous Wavelet Transform is used for target motion estimation and an Adaptive-Network-based Fuzzy inference System (ANFIS) is adopted to minimize the noise effect on target detection. A local optimum search algorithm is introduced in this project to reduce the computation load of the Continuous Wavelet Transform and make it suitable for practical applications.

The proposed system is realized on a Xilinx University Program Virtex-II Pro Development System which contains a Virtex II pro XC2VP30 FPGA chip with 2 powerPC 405 cores. Testing for single target detection and multiple target detection shows the proposed system is able to accurately locate targets under reverberation-limited underwater environment with a Signal-Noise-Ratio of up to -30db, with location error less than 10 meters and velocity estimation error less than 1 knot.

In the proposed system the combination of CWT and local optimum search algorithm significantly saves the computation time for CWT and make it more practical to real applications. Also the implementation of ANFIS on the FPGA board indicates in the future a real-time ANFIS operation with VLSI implementation would be possible.

ABBREVIATIONS

ANC	Adaptive Noise Canceller
ANFIS	Adaptive-Network-based Fuzzy inference System
CWT	Continuous Wavelet Transform
DWT	Discrete Wavelet Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
SNR	Signal-Noise-Ratio
SONAR	SOund NAvigation and Ranging
TME	Target Motion Estimator

CHAPTER I

INTROCUTION

This thesis presents the results of a research project into the application of Wavelet-based Broadband Signal Detection techniques for high resolution sonar system. The development of an end-to-end simulation of wideband active sonar is presented, along with the analysis of the effects of unknown target motion and techniques for accurately estimating sonar target movement. The active sonar techniques are discussed. The Wavelet Transform, originally for data analysis of seismic survey in oil and mineral exploration, was proven to be highly suited for sonar imaging. Several enhancements to the wavelet transform algorithm are presented in this thesis. Techniques for adaptive noise cancelling are demonstrated and a new strategy for reducing computation complexity of wavelet transform is presented in the context of this thesis. Real sonar data has been made available to the project, enabling the performance of the developed techniques to be demonstrated both analytically and practically.

This chapter begins with the introduction of the concepts of sonar imaging and the principles of active sonar system. The motivation for this project along with the discussion of the need to use the wavelet transform is then described. The contributions made by this work and the thesis layout are presented.

1.1. Overview of Active Sonar System

SOund NAvigation and Ranging (SONAR) covers a broad range of techniques for detecting and imaging objects using sound. Sonar is one of the most commonly used techniques for forming images of objects underwater; this is mainly due to the much lower attenuation of sound compared to electromagnetic radiation of a comparable wavelength, which makes optical and radar systems unusable over large distances. Sonar systems range in complexity from simple passive listening devices to sophisticated active systems, the subject of this work.

Active side-scan sonar is commonly used for forming high resolution images of underwater scenes. Side-scan systems form an image of a target scene by transmitting a succession of sound pulses (referred to as ‘pings’) and recording the echoes from the target scene using a very narrow beam width receiver. An image of a target scene is built up a line at a time by moving the transmitting / receiving platform past the target area. The azimuth resolution of a side-scan sonar system can be increased by using a higher pulse frequency and / or a larger physical aperture. The attenuation of sound in water increases with frequency, creating a tradeoff between azimuth resolution and maximum range of side-scan sonar systems. The maximum physical length of the sonar array is limited by practical constraints.

The advantage of side-scan sonar is that it is able to provide a through image for underwater scenes in a certain area, however, this image cannot provide any velocity information for a certain target and post-processing of imaging is required to extract

certain target motion information. The wavelet-based active sonar applications, on the other hand, provide another way to examine target motion parameters. Following the principles of active sonar, a succession of sound pulses is transmitted and echoes from the target scene are recorded with a receiver. With the similar representation form between the returned echo in a wideband sonar system and the transformed signal by the Continuous Wavelet Transform (CWT), the seeking of target motion parameters can be achieved by finding parameters of returned echo through CWT. The thesis presents a wavelet-based active sonar system using both real and simulated data.

A requirement for very accurate knowledge of the position of the sonar system, and high computational complexities present a number of problems for a wavelet-based sonar system. This thesis discusses, and proposes solutions to some of these problems.

1.2. Motivation

There is a strong interest in methods of increasing the resolution of sonar systems for the detection of targets at sea. Systems currently used by the Royal Navy to detect targets are based on conventional side-scan and forward looking sonar systems. Such systems require post-processing on imaging of sonar scenes to identify location of mines or fish torpedoes, and are unable to provide the velocity of targets. Wavelet-based active sonar system can provide a solution for extracting target location and velocity information at the same time.

The detection of low- and zero-Doppler signals in real-time is computationally expensive if we are to use the optimum technique using the correlation matrix in a active

sonar system in reverberation-limited enviroment. Previous reseach on narrowband case shows an Adaptive Noise Canceller can be useful and computationlly efficient. The proposed research aims at investigation of the broadband case, in which doppler-shift model, used in the narrowband case, is no longer valid. In particular, since the family of scaled wavelets better models the behaviour of pulses in a reverberation-limited broadband and shallow-water environment, the use of wavelets is proposed. Furthermore, the proposed method allows the possibility of designing a customized wavelet to minimize the noise effect on target detection. Therefore, the proposed active sonar system is able to provide an alternative method of side-scan sonar system to extract target motion parameters more efficiently.

1.3. Layout of this Thesis

The thesis starts with an introduction to wavelet-based wideband active sonar system, this project, the motivation and requirement of this thesis. (This chapter)

Chapter 2: introduces the basic principles of sonar relevant to the proposed sonar system, along with a number of signal processing techniques used to extract target motion parameters in this work. Previous research on related techniques adopted in this project is also introduced.

Chapter 3: The chapter introduces necessary background theory for understanding the project. First of all it introduces the specifications and expectations for this sonar

detection system and then it illustrates respectively why specific techniques such as wavelet transform, ANFIS and Local Optimum Search are adopted in the proposed system and how these techniques work. The chapter also shows the integration of all these techniques into a complete system.

Chapter 4: The theoretical model of the proposed system is presented and the design details of the high-level system are explained. The simulation results in Matlab for the high level system are used to estimate the performance of the proposed system.

Chapter 5: This chapter gives the details of the hardware implementation of the proposed system. At the beginning of the chapter the hardware structure is introduced. The system is composed of two major functions: Adaptive Noise Canceller (ANC) and target Motion Estimation (TME). The chapter introduces the implementation of both functions sequentially as well as the presentation of hardware implementation; simulation results are presented to justify design alternatives.

Chapter 6: The chapter presents the testing environment of the proposed sonar detection system. The hardware platform is introduced and details of the platform are given. Afterwards two different tests, single target detection and multiple target detection, of the proposed system on a hardware platform are described. The test results are used to evaluate the performance of the proposed system.

Chapter 7: The high-level simulation results, hardware application simulation results and the testing results on the hardware platform are examined and compared to characterize the performance of the system developed in this work.

Chapter 8: presents conclusions and suggestions for future work.

1.4 Contributions made by this Thesis

Few studies on actual hardware implementation of active sonar system have been mostly theoretical; this work has not only developed new target detection techniques but also demonstrated their real world potential using generated, as well as real, sonar data.

A principal contribution of this project is a demonstration that the continuous wavelet transform can be used in a sonar system to detect targets with a computation time that is applicable for real time systems. Continuous Wavelet Transform is well known for its high resolution as well as its high data redundancy and computation complexity. It reduces the chances of CWT utilization in real applications. With the proposed system in this thesis, the computation complexity is reduced to a level that makes the real time implementation is possible.

An Adaptive-Network-based Fuzzy inference System (ANFIS) is adopted in the proposed system as the algorithm for Adaptive Noise Canceller. ANFIS is utilized in various theoretical systems for its ability to work with the chaotic nature of the

impulsive noise, but seldom in real work due to its heavy computation load. Implementations of ANFIS on a single FPGA chip in this project demonstrated the possibility to employ this algorithm in real applications, even in real time under certain circumstances. The hardware working frequency is increased so that in a unit time the hardware can complete more computations and therefore reduce the processing time. The FPGA has more hardware resources such as multipliers and adders, making parallel computation more applicable.

The combination of local optimum search algorithm and Continuous Wavelet Transform presented in this thesis is a novel method to reduce the computation complexity of its CWT. The simulation results also show this method is able to reduce the computation load of CWT by 8 to 10 times and enable CWT to be utilized in real time applications.

A final contribution is the publication of a large number of real sonar results for target detection using a number of data sets contributed by the Defense Science and Technology Laboratory (DSTL) of Ministry of Defense (MOD) in the project. Many previous studies into wavelet-based sonar system have been mostly theoretical, and have not concentrated on forming images with real sonar data.

Publications resulting from this work are listed below:

1. Sheng Cheng, Chien-Hsun Tseng and Marina Cole, “An Efficient and Effective VLSI Architecture for a Wavelet-based Broadband Sonar Signal Detection System”, ICECS 07, pp593-596

2. Sheng Cheng, Chien-Hsun Tseng and Marina Cole,” A Novel FPGA Implementation of a Wideband Sonar System for Target Motion Estimation”, ReConfig’08, Mexico

CHAPTER II

LITERATURE REVIEW

This chapter will introduce previous research on active sonar systems and outline the rationale behind techniques adopted in this thesis. First the background of the underwater sonar system is described, and research in this area will be briefly introduced. Then the reason why an adaptive noise canceller is proposed in the system is demonstrated and the chosen algorithm for it is presented. Finally an investigation in the wavelet transforms area is described to demonstrate the suitability of the continuous wavelet transform for application in an active sonar system.

2.1 Underwater Sonar Detection

Sight and hearing are two primary sources we use to get information from surroundings. The two types of radiant energy related to these two senses, sound and light travel well in air. However, in an underwater environment, sound propagates over longer distances than any other source of radiant energy [1]. Therefore, SOund NAvigation and Ranging (SONAR) systems are primarily adopted for gathering information in an underwater environment. Systems that use acoustic echoes for detection are known as

sonar systems [2].

The detection, classification and localization performance of sonar systems depends on the environment and the receiving equipment, as well as the transmitting equipment.

2.1.1 Sound propagation

Sonar operation is affected by variations in sound speed, particularly in the vertical plane. Sound travels slower in fresh water than in sea water, though the difference is small. The speed is determined by the water's bulk modulus and mass density. The bulk modulus is affected by temperature, dissolved impurities (usually salinity), and pressure. The density effect is small. The speed of sound is approximately:

(4388 + (11.25 × temperature (in ℉)) + (0.0182 × depth/0.3048) + salinity (in parts-per-thousand))/0.3048.

This empirically derived approximation equation is reasonably accurate for normal temperatures, concentrations of salinity and the range of most ocean depths.

Water pressure also affects sound propagation: higher pressure increases the sound speed, which causes the sound waves to refract away from the area of higher sound speed. The mathematical model of refraction is called Snell's law [3].

If the sound source is deep and the conditions are right, propagation may occur in the 'deep sound channel'. This provides extremely low propagation loss to a receiver in the channel. This is because of sound trapping in the channel with no losses at the boundaries. Similar propagation can occur in the 'surface duct' under suitable conditions. However in this case there are reflection losses at the surface.

In shallow water propagation is generally by repeated reflection at the surface and bottom, where considerable losses can occur.

The energy of sound will be absorbed as long as it travels through sea water. Sound propagation is affected by absorption in the water itself as well as at the surface and bottom. Absorption of low frequency sound is week. Viscosity is the main cause of sound attenuation at high frequency in sea water. Sound may be absorbed by losses at the fluid boundaries. Near the surface of the sea losses can occur in a bubble layer or in ice, while at the bottom sound can penetrate into the sediment and be absorbed.

2.1.2 Scattering

The major difficulty in detecting a specific target underwater is the presence of other objects. When the pulse is emitted into the water, it not only strikes the target but also other objects underwater. The pulse is reflected off these objects and is retrieved by the receiver. The signal from these objects underwater is known as reverberation and the objects are called scatters [4, 5].

The scattering of sound underwater can be described either by Rayleigh's law or by geometrical acoustic scattering [6]. Rayleigh scattering (named after the British physicist Lord Rayleigh) is the elastic scattering of electromagnetic radiation by particles much smaller than the wavelength of the pulse, which may be individual atoms or molecules. Rayleigh scattering is a function of the electric polarizability of the particles. Therefore, Rayleigh's law only applies when the size (d) of particles is much smaller than the wavelength (λ) of the incident sound. Geometric acoustic scattering, on the other hand, is

valid when d is much larger than λ. Therefore, when d<<λ, the pressure of scattered sound is proportional to the square of acoustic frequency and to the volume of scatterer, regardless of its shape (Rayleigh's law). If d is similar to λ, the pressure of scattered sound is combination of functions of sound frequency, acoustic properties of scatterer and characteristics of the underwater medium or boundaries. However, if d>>λ, the scattering of sound is only dependant on acoustic properties of scatterer and its cross-section, not on sound frequency anymore (Geometrical acoustic scattering).

This situation leads to the development of two different approaches for reverberation modeling underwater: cell-scattering and point-scattering models.

In cell-scattering models, it is assumed that the scatterers are uniformly distributed through the ocean. The ocean then can be divided into cells, each containing a large number of scatterers. Summing up the contribution each cell makes together leads to the reverberation form.

In point-scattering models, the scatterers are assumed to be randomly distributed throughout the ocean. The reverberation waveform is calculated by summing the echoes from each individual scatterer. It is a statistical based approach.

Simplifying assumptions are often necessary to make reverberation models feasible. "These assumptions may seem to set the results to pure idealized situations, however, the resulting expressions for reverberation have been found to be practical for many sonar design and predictions"[6]. The assumptions are [7]:

1. Straight-line propagation paths. All sources of attenuation other than spherical

spreading are neglected.

2. A pulse length short enough to ensure propagation effects over the range extension of the elemental volume or area can be neglected.
3. An absence of multiple scattering. (I.e. the reverberation produced by reverberation is negligible)

In cell-scattering models, the reverberation usually is combined with three scatterings: surface scattering, volume scattering and bottom scattering, which are reverberation from ocean surface, scattering in ocean and ocean bottoms. Detail equations can be found in [6]. Cell-scattering is widely used in sonar modelling, and lots of experimental models are built based on it, such as DOP (Doppler Content), BAM (bidirectional associative memory), MAM (monostatic associative memory), PERVE (Tappert's PE Reverberation model), etc [6].

Point-scattering models are based on a statistically approach which assumes the scatterers are randomly distributed in the ocean. The scatters from each individual object are then summed up to compute the reverberation level. It is not widely used as cell-scattering models; however, when processing scatterers with dimensions comparable to the acoustic wavelength, the point-scattering models are probably preferred alternative. Due to the use of modem, high resolution, wideband sonar systems, the characteristics of individual scatterers become increasingly important. Therefore, non-Rayleigh point-scatter models attract more and more researchers and lots of researches have been done in this area such as [8-11], etc.

Reverberation models are typically combined with environmental, propagation, noise and signal processing models to form a new class of models referred as active sonar models [6]. Reverberation and ambient noise are usually jointly considered as background noise against target pulse which must be detected in sonar model evaluation. It is important to understand which of them, ambient noise or reverberation, is most responsible for creating the interference background.

2.1.3 Sonar Detection System

Sonar systems fall into two categories known as passive and active sonar systems [3, 4]. Passive sonar systems detect objects by detecting the acoustic echoes emitted from the object being looked for. In the case of active sonar systems, a pulse whose characteristics are known is transmitted. The pulse is then reflected back to the transmitter from the target to be detected. From the returned pulse it is possible to compute the range and speed of the target.

Sonar systems have been used for more than a century to detect underwater objects using sound. Probably the first recorded use of the sonar principle (although not referred to as sonar at the time) was by Leonardo da Vinci in 1490:

If you cause your ship to stop, and place the head of a long tube in the water and place the outer extremity to your ear you will hear ships at a great distance.

The system described can be considered to be a very basic passive sonar system. Although simple, the system has a number of important features such as covert operation (the system is passive) and, with user training, the ability to classify target

type (a large ship will sound different to a small boat). The design of this system is based purely on empirical observation. Given a modern knowledge of the physics of the propagation of sound through water, Leonardo da Vinci may have been able to predict the likely sounds of different vessels under different ocean conditions.

Issac Newton made the first theoretical prediction of the speed of sound in water in Book II of the *Principia*, showing that the speed of sound is related to the density of the medium through which it is travelling.

Due to the development of transducter technology after World War II, which converts underwater acoustic energy into electrical signals (and vice versa), and the ability to implement signal processing algorithms with digital computers, the use of underwater sound for target detection has also been progressed [1].

The major problem for sonar systems is to identify pulse bounced back from target within all received pluses containing background noises, which includes reverberation from scatterers in ocean, ocean surface and bottoms, and ambient noises [6]. Usually the amplitude of background noises is much larger than pulse reflected back from the target, which is the signal the system must detect. Therefore it is hard for sonar system to abstract desired signal from received pulses directly. Signal processing algorithms such as convolution, wavelet transform, adaptive noise canceller, etc, is then required to extract desired signal out of received signals.

Digital signal processing algorithms developed fast for active and passive sonar systems since 1960’s. Much of the work in 1960’s was based on concepts brought by

Vic Anderson such as DELTIC correlator, Digital multibeam steering beamformer [12], and the adaptive array processor[13]. In the 1970's and early 1980's, adaptive and high resolution underwater acoustic signal processing became the focus of research. Works from Howells [14], Anderson [15], Widrow [16], Griffiths [17] etc contributed a lot for the research in this area.

Maximum likehood forms the basis for high-resolution signal processing. Results of work by many authors including Slepian [18], Youla [19] and Schweppe [20] point it out. Tuefs and Kumaresan [21-24] reduce computation load for extract exact maximum likehood by implicit substitution of linearly entering parameters.

Driven by tremendous technical development on computation capacity and speed, which allows computationally intensive techniques to be implemented, research on adaptive signal processing algorithms to sea test data, wideband processing on both active and passive sonar, detection on transient signals with unknown parameters, reverberation processing and etc developed very fast since 1980's to present.

The use of continuous wavelet transforms to detect transient signals, proposed in this project, in the application of active sonar system has been investigated by various researchers such as A.C. Dubey, C.Yan and M.R.Azimi-Sadjadi[27] and Lora.G.Weiss [28,29]. The work of A.C, Dubey, C.Yan and M.R.Azimi-Sadjadi investigated the application of the Discrete Wavelet Transform (DWT) [27]. They used an application known as wavelet shrinkage which proposes that if a signal f(t) contains the echo of a target (i.e. target to be detected) and environment noise, the noise will contribute to

most of the wavelet coefficients while the target echo contributes to only a few coefficients. Therefore by making the smaller coefficients zero in an appropriate method, the noise can almost be eliminated whist preserving the target echo. The results from their work showed that this method worked well but as they stated and as seen in their work the method only works well in environments where the effects of reverberation are not severe.

Their subsequent work using DWT in underwater acoustic signal recognition is reported in [30, 31, 32]. Their work shows the wavelet transform is a powerful method for underwater sonar recognition.

The work carried out by Lora.G, Weiss [29] is very beneficial since it looks at the application of wavelets to wideband processing. Although her work does not focus directly on their applications in active sonar it provides a good introduction into wideband signals and their properties and at the same time shows how wavelets can be easily fitted into this model. In her work a wideband model is used to model the target echo and good insight to the effects of the target echo shift when it is modelled on the wideband equation is given. The paper investigates both active and passive methods of sonar and describes both deconvolution and convolution in order to determine the motion parameters of several targets. The method presumed that there is more than one target to be detected and a cross wavelet transform is performed in order to gain and estimate the parameters of the targets. Moreover, in this paper the Continuous Wavelet Transform is preferred to the Discrete Wavelet Transform due to its ability to provide

high resolution in both the frequency and time domains.

Though the work on the application of wavelets to active sonar is still not well established there has been a number of works carried out on target detection of underwater objects. One of the well established processes of detecting targets can be accredited to Harry L.Van Trees [33]. He proposes the idea of a correlation matrix also referred to as a correlation process. However, the methods seem complicated and cumbersome, the heavy computation load makes it is hard for real-time application.

2.2 Wavelet Transform

Wavelet transform is a mathematical method to decompose signals into different frequency components, and then study each component with a resolution matched to its scale. The word wavelet means “small wave”, referring to the fact that the window function is of finite length [32]. Wavelet theory is seen as an extension of Fourier analysis.

The basic idea of wavelet transform is to analyze target signals according to scale. It uses functions which satisfy certain mathematical requirements to represent target signals. However, the representation methods using superposition of certain functions which is used in wavelet transform is not original. In the early 1800’s, Joseph Fourier had already discovered that he can superpose sines and cosines to represent other functions and this representation method is now well known as the Fourier Transform. The wavelet Transform is similar to the Fourier Transform in the sense that both of them decompose

the target signals into their constituent parts for analysis. Whereas the Fourier Transform breaks signals into sine waves and cosine waves, the Wavelet Transforms breaks signals into their wavelets, the dilated and translated versions of mother wavelet. However, when compared with Fourier Transform, wavelet transform can provide the possibility to acquire a 2D plot of the time and scale (1/Frequency) information in a signal via translations and dilations of the mother wavelet. Thus it has advantages over traditional Fourier methods in analyzing physical situations where the signal contains discontinuities and sharp spikes that need to be localized in time. Wavelet transform is widely used in the fields of data compression, signal and image processing, acoustics, radar, human vision, etc. O. Rioul et .al give more details about wavelet transform features in [35, 36].

A wavelet w is a function with zero average:

$$\int_{-\infty}^{+\infty} w(t)dt = 0\,, \qquad (2.1)$$

which is dilated with a scale parameter s, and translated by u:

$$w_{u,s}(t) = \frac{1}{\sqrt{s}} w(\frac{t-u}{s})\,. \qquad (2.2)$$

The wavelet transform of f (t) at the scale s and position u is computed by correlating f (t) with a wavelet atom [14]

$$Wf(u,s) = \int_{-\infty}^{+\infty} f(t)\frac{1}{\sqrt{s}} w^{*}(\frac{t-u}{s})dt\,. \qquad (2.3)$$

O. Rioul and M. Vetterli introduce the DWT (Discrete Wavelet Transform) and CWT (Continuous Wavelet Transform) in [35], which are two different kinds of wavelet transform. In DWT, the signal is broken into dyadic blocks where only specific scale

values can be used [35]. In the CWT implementation, the discrete sample data is also used, however the scaling range can be defined as any value by the user, thus giving much better resolution for target analysis. Also the CWT is shifted along the analyzed signal smoothly and therefore the CWT is highly redundant in both time and scale. The trade off is between improved resolution and increase in computational time and required memory to calculate the redundant wavelet coefficients.

The main idea of the CWT is to obtain time-scale digital signals using digital filter techniques by correlating wavelets at different scales with target signals at the scale to measure the similarity. The Continuous Wavelet Transform is computed by changing the scale of the analysis window, shifting the window in time, multiplying by the signal, and integrating over all times as shown in Eq. 2.3 [37]. In order to implement the CWT by computer methods, discretely sampled data are used for CWT. In this case, filters of different cutoff frequencies are used to analyze the signal at different scales. Theoretically researchers can obtain any scale they want by using the CWT; however this will also bring heavy computation loads. CWT provides very fine details of target signals but in some applications not all of them are needed. The discrete wavelet transform (DWT), on the other hand, provides sufficient information both for analysis and synthesis of the original signal, with a significant reduction in the computation time [38]. Generally DWT coefficients are samples from the CWT in dyadic grids. i.e., $s_0 = 2$ and $u_0 = 1$, yielding $s=2^j$ and $u =k*2^j$, where j is usually called the number of octave. Also, DWT is considered easier to implement when compared to CWT.

Fast wavelet Transform algorithms are widely investigated to reduce the computation complexity of WTs (Wavelet Transforms) and to make the WT implementations realistic. The Mallat Algorithm [40], also known as PA (Pyramid Algorithm), is one of the best fast DWT algorithms and is widely adopted for orthogonal wavelets. In this algorithm, the DWT coefficients at any stage can be calculated from the DWT coefficients of previous stages as follows:

$$W_L(n, j) = \sum_m W_L(m, j-1)h(m-2n) \qquad (2.4)$$

$$W_h(n, j) = \sum_m W_L(m, j-1)g(m-2n) \qquad (2.5)$$

where $W_L(p,q)$ and $W_h(p,q)$ is the pth scaling coefficient and wavelet coefficient at qth stage, respectively, and h(n), g(n) are the dilation coefficients corresponding to the scaling and wavelet functions, respectively [39]. In practice, h (n) and g (n) represent a low pass and high pass filter, respectively. The computation complexity is O (N) each octave instead of $O(N \log N)$ each octave. Recursive Pyramid Algorithm (RPA) is introduced in [40]. This algorithm reduces the words of storage from O (N) to $L(\log N - 1)$ when compared with traditional PA, where L is the length of wavelet filter. This is achieved by reformulating the classical Pyramid Algorithm. The order of N outputs is rearranged so that an output can be scheduled at the earliest time slot. Several VLSI architectures for 1-D DWT based on RPA are proposed in [41], which include systolic, semisystolic and RAM-based architectures. The difference between these architectures is only in their routing network architectures. Also M-band DWT and 2-D DWT architecture are also discussed in the paper. VLSI architecture for 2-D DWT which

is also based on RPA is proposed in [42]. However, Several VLSI architectures which are lifting-based are discussed in [43] [44]. The lifting-based architecture uses less hardware resources than RPA but have relatively long critical path. A detailed analysis of different VLSI architectures for 1-D and 2-D DWT is given in [45].

Though the Mallat Algorithm is very successful in DWT, it is not perfect for CWT. First of all, it can only be used for WT on a dyadic grid, but the scaling and shifting of CWT could be any value. Secondly, the translation invariance is lost due to the decimation in the Mallat Algorithm. Thirdly, the decimation (reduction in the number of samples) reduces the redundancy provided by CWT, which is also used as finer details of the target signal. In [46] a fast WT algorithm, the "A Trous" algorithm is proposed. This algorithm is proposed for DWT at first, but is widely used as a basis for fast WT transforms of CWT. It is similar to the Mallat Algorithm but with no decimation after filters at each octave and the lowpass filter should be an a trous filter by satisfying $f_{2k} = \partial(k)/\sqrt{2}$. Thus the product of this algorithm is 2N, which is the double the size of the input signal. Reorganization of the "A Trous" computational structure is proposed in [47] and is used for CWT. The filters which are used in "A Trous" are usually Lagrange interpolators [48]. However, there is no evidence that this filter is most suitable for all applications. Thus the filter design for "A Trous" is also an interesting topic to investigate. K.C.Ho and Y.T.Chan proposed several filters for "A Trous" in [49] and give a detailed comparison in [50]. However, generally the octave-by octave computation of the DWT is not enough in signal analysis. A method which can provide better resolution

is proposed in [51] [52]. It over samples the discretization to obtain "M voice per Octave" [51] [52], that is $s = 2^{j+m/M}$ instead of 2^j, m=0, 1…M-1, where m is called "voice", and this is achieved by applying octave-by-octave computation M times with different dilated wavelets

$$2^{-(j+m/M)/2}\partial(2^{-(j+m/M)}(t-k2^j)), \quad j,k \in \mathbb{Z}, m = 0,\ldots,M-1 \qquad (2.6)$$

Obviously this method requires about M times the computation load of the octave-by-octave algorithm. In fact precision and computation workload is always a trade-off.

Another CWT algorithm called the *Bertrands and Ovarlez* algorithm is proposed in [53]. It uses the scaling property of the wavelet instead of the convolution form. It needs pre-computation of the whole Fourier Transform of the input signal, and it is difficult to estimate the approximation error. To overcome the disadvantage of this algorithm that it is not suitable to handle continuous data flow, a variation of the *Bertrands and Ovarlez* algorithm is also introduced in [53]. Assume the signal and wavelet are casual, (i.e., supported by t>=0), and assume $T = \ln t$, One can obtain a convolution in $a = \ln a$:

$$CWT\{x(t); s, u\} = \int e^{T/2} x(e^T + b) e^{(T-a)/2} \partial^*(e^{T-a}) dT \qquad (2.7)$$

In this algorithm the computation load will be higher than in the algorithms described previously. But on the other hand, it gives a required resolution of ln a. This property is useful if high resolution is needed in applications.

M. Unser et al introduce another CWT algorithm which provides fast CWT in integer

scales [48] [54]. In this method the input signal and the wavelet are represented by polynomial splines. A combination of moving sum and zero-padding filters is adopted in this method and a computation complexity of O (N) per scale is provided by this algorithm. However, scales in integer may not be sufficient for some applications which require very good resolutions.

In [55] two architectures for 1-D CWT are introduced, one is a parallel filter architecture, and the other is a Single Instruction Multiple Data (SIMD) linear array. These two architectures are also discussed in [56], which also provides a detailed comparison for several other architectures of 1-D CWT and 2-D CWT implementations. From these comparisons it is hard to tell which architecture has the best performance. The decision of which implementation architecture will be used bases on the specific application and requirements.

Most of the architectures mentioned above can be categorized into convolution-based, lifting-based, and B-spline based. RPA and “A Trous” are convolution –based; VLSI architectures mentioned in [43] [44] are based on the lifting-scheme, and M. Unser proposes a B-spline based architecture in [48] [54]. The Table 1 below shows the performance of different algorithms and VLSI architectures mentioned above. The contents of the tables are based on 1-D Wavelet transforms on dyadic grids. The table shows that B-spline based architectures can provide the smallest area, and lifting-based algorithms can use fewest registers with a relatively short critical path. However, the hardware complexity of convolution based architecture is better than those two. It shows

the tradeoff between speed, hardware complexity and area. For different applications and requirements the most suitable architecture will be different. Using different methods such as introducing M-voice in each octave or represent the input signal and wavelets in different splines, a better resolution and finer details can be achieved. The exact expressions of critical path or area will differ when these modifications are made, but the relations of these architectures will not change.

	Convolution Based		Lifting-based		B-Spline based	
Implementation Method	Parallel Filter	Serial Filter	Conventional	Flipping	Parallel Filter	Serial Filter
Critical Path	Tm+(log2F)Ta	Tm+2Ta	F/2(Tm+2Ta)	Tm+(F/2)Ta	Tm+(F/2+ log2F)Ta	Tm+(F/2)Ta
Registers	F	FH+FG	F/2	F/2	FH+FG	FH+FG
Multipliers	FH+FG		(FH+1)/2 +(FG+1)/2 +3		(FH+FG)/2	
Adders	FH+FG-2		(FH+1)/2 +(FG+1)/2 +1		3(FH+FG)/2-2	

Table 1 Approximation of performances of 1-D WT algorithms [50]

- F_H and F_G are filter lengths of high-pass filter and low-pass filter, respectively.
- T_m and T_a are logic delays of multipliers and adders, respectively
- F= max (F_H F_G)

There are various forms of non-orthogonal wavelets from the well known Morlet wavelet, Car wavelet, Difference of Gaussian wavelet, and Mexican hat wavelet [57].The choice of wavelets is vast.

Many functions can be used as wavelets as long as they satisfy two conditions, which are:

1) The function is continuous and has an exponential decay;

2) The integral of the function is zero, i.e., shown in Equation 2.1.

These conditions can be easily satisfied and therefore the choice of mother wavelet can be very flexible and allow users to define their own wavelets to suit their application environment best. Research on wavelets applications on active sonar signal processing has been carried out, but most of them investigate the Discrete Wavelet Transform, which has very limited success in reverberation limited environments [27]. Therefore, the Continuous Wavelet Transform is chosen for this project, enabling motion parameters extraction of the target including range and velocity to be assessed in this research.

2.3 Adaptive Noise Canceller

As mentioned previously in Chapter I, the motivation of this project is to build an active sonar system which is able to identify accurate motion information of targets from received echoes and in a reverberation-limited environment. The received echoes contain the target echo as well as noise, which normally are composed of reverberation noise and white Gaussian noise. Convolution based digital signal processing algorithms can give satisfactory results for cancelling noise effects from white Gaussian noise, but the performance is limited regarding to reverberation noises due to the similarity between target echo and reverberation noises. Therefore, algorithms which are efficient on removing reverberation noises are required in order to minimize the effect of noise during target extraction. Therefore, Adaptive Noise Cancelling algorithm is introduced

in the proposed project.

Although various work have been done on ANC, very few researches has been done on ANC implementation on underwater sonar detection. Alexandrou and De Moustier [82] used Least-Square Lattice filter to reject sea beam sidelobe interference in building reference models. Lawson [83] investigated the performance of a simple ANC in narrowband signal detection in reverberation-limited environment by using Least-Squares Lattice filter again.

The conventional way to detect a target buried in additive noise is to feed whole signals into a filter which is designed to suppress noise while leaving the target relatively unchanged [58-62]. The filter could be fixed or adaptive. The fixed filter is build based on prior knowledge of the target and noise; the adaptive filter on the other hand, can adjust its parameters automatically, and with little or no prior knowledge of target or signals [63]. Adaptive noise cancelling has been successfully applied to a number of additional problems, including some aspects of electrocardiography, to the elimination of periodic interference in general [64], and to the elimination of echoes on long-distance telephone transmission lines [65, 66].

For white Gaussian Noise, Adaptive Noise Cancelling with linear filtering can give a satisfactory performance on removing it [67-69]. However, for non-linear noises like reverberation noise, an Adaptive Noise Canceller with linear filtering is no longer effective to suppress noise and keep the target echo unchanged at the same time. Therefore, an Adaptive-Network-based Fuzzy Inference Systems (ANFIS) is introduced

in the system described in this thesis due to its ability of working with the chaotic nature of the impulsive noise. It is the combination for Fuzzy logic and neural network and it has advantages of both techniques. Therefore, more accurate results are expected from ANFIS than least-square Lattice filters used in [82] [83].

An ANFIS is proposed by J.S. Roger Jang in [70, 71]. It is a combination of Fuzzy inference system and adaptive networks. It proposes a class of adaptive networks which are functionally equivalent to fuzzy inference systems. It serves as a basis for constructing a set of fuzzy if-then rules with proper membership rules to generate the stipulate input-output pairs [70].

Fuzzy if-then rules or fuzzy conditional statements are expressions of the form IF A THEN B, where A and B are labels of fuzzy sets [72] characterized by appropriate membership functions. Due to their concise form, fuzzy if-then rules are often employed to capture the imprecise modes of reasoning that play an essential role in the human ability to make decisions in an environment of uncertainty and imprecision.

In 1985 Takagi and Sugeno propose a non-linear fuzzy inference system, which is known as Takagi and Sugeno's model [73]. It combines the advantage of both a fuzzy inference system and a neural network to improve the traditional fuzzy inference system's design. Traditionally in fuzzy inference system the parameters are required to be updated manually to minimize error rate. In Takagi and Sugeno's model, the parameters are undated automatically through a hybrid learning algorithm [73].

ANFIS represents Sugeno & Tsukamoto fuzzy models based on neural network, which is a multi-layer forward pass network, as shown in Fig 2.1:

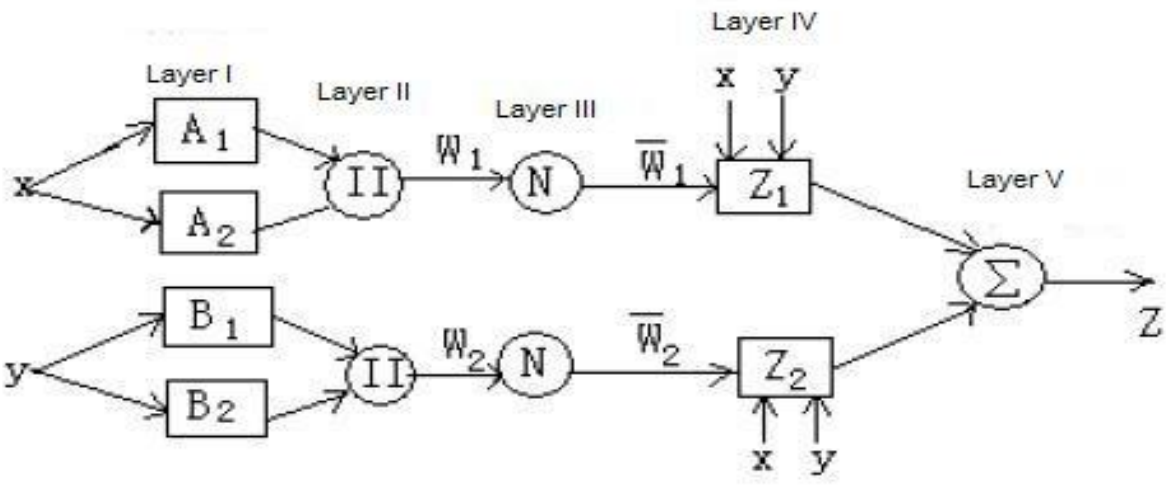


Fig 2.1 ANFIS network Structure

Fig 2.1 shows a two-input and 2 membership for each input ANFIS structure. There are five layers in the system. X and Y represent input data for the system and Z is the estimate output of ANFIS. Ai and Bi represents membership functions under different parameters. The details of ANFIS will be described in Chapter III.

ANFIS is capable to provide satisfactory noise reduction performance under a severe non-linear noisy environment and is adopted in many areas such as noise cancelling, signal analysis, behaviour prediction or modelling, etc [74-81]. However, most of them only investigate the implementation of ANFIS theoretically. Also little research has been done on utilizing ANFIS in an active sonar system. Therefore the system proposed in this thesis combing ANFIS along with the Continuous Wavelet

Transform to realize an active sonar system on a hardware platform is a new challenge and an exploration in a new direction.

2.4 Conclusion

The history of underwater sonar system is introduced in this chapter, and the development of researches on wavelet theory is given. In literature the continuous wavelet transform is shown to have an advantage in sonar target detection for its capability of providing high resolution analysis in both frequency domain and time domain. Then research relative to implementation of wavelet transform is also reviewed. It shows that due to the high computation redundancy, Continuous Wavelet Transform is seldom adopted in practical applications. Therefore, the simplification of the computation process for Continuous Wavelet Transform is the first and new challenge in the proposed system. The Adaptive Noise Canceller is also introduced into the proposed system to minimize the noise effect for target detection. The development of Adaptive-Network-based Fuzzy Inference Systems used as Adaptive Noise Canceller is given afterwards. Literatures shows ANFIS is a powerful way to reduce noises in sonar signals, however, the efficiency of implementation of ANFIS is also a major concern in practical applications. All the relative research works indicates the proposed active sonar system is a new challenge to solve.

References

1. Underwater Acoustic Signal Processing Technical Communite,"The Past, Present and Future of Underwater Acoustic Signal Processing", IEEE Signal Processing Magazine, pp21-51, July, 1998.
2. H.Naparst,"Dense Target Signal Processing", IEEE Trans. Inform Theory, Vol 37(2), pp 317-327, 1991.
3. Andrew S. Glassner, "An Introduction to Ray Tracing". Morgan Kaufmann, 1989
4. William S. Burdic, "Underwater Acoustic System Analysis", Prentice Hall,Second Edition, 1991
5. Pierre Faure,"Theoretical Model of Reverberation Noise", The Journal of the Acoustical Society of America, Vol 36(2), pg 259-266, 1964
6. Paul C. Etter, " Underwater acoustic modeling and simulation", Spons Architecture Price Book,2003
7. Robert J. Urick," Principles of Underwater Sound", McGraw-Hill, 1983
8. Hodgkiss, W., Jr.; Alexandrou, D," Under-ice reverberation rejection", IEEE Journal of Oceanic Engineering, Volume: 10, Issue 3 , 1985
9. Alexandrou, D, C. de Moustier, G. Haralabus, "Evaluation and verification of bottom acoustic reverberation statistics predicted by the point scattering model", The Journal of the Acoustical Society of America Vol.91, No.3 , March 1992
10. Abraham, D.A, "Choosing a non-Rayleigh reverberation model", OCEANS '99 MTS/IEEE, Vol 1, Pg 284 – 288, 1999

11. Lew, H,Drumheller, D.M,” Estimation of non-Rayleigh clutter and fluctuating-target models”, Radar, Sonar and Navigation, IEE Proceedings, Vol 149, Issue 5, 2002, Pg 231 – 241
12. V.C.Anderson, “Digital array phasing”, The Journal of the Acoustical Society of America, Vol. 32, pp 867-870, 1960
13. V.C.Anderson, “DICANNE, a realizable adaptive process”, The Journal of the Acoustical Society of America, vol 45, pp 398-405, 1969
14. P. Howells, “Intermediate frequency side-lobe canceller”, U.S. Patent NO 3202990, 1959
15. V.C. Anderson and P. Rudnick, “Rejection of a coherent arrival at an array”, The Journal of the Acoustical Society of America, vol 45, pp 406-410, 1969
16. B. Widrow, P.E. mantey, L.J.Griffiths, and B.B. Goode, “Adaptive antenna systems”, Proc. IEEE, vol 55, pp 2143-2159, 1967
17. L.S. Griffiths, “A simple adaptive algorithm for real-time processing in antenna arrays”, Proc. IEEE vol 57, pp, 1696-1704, 1969
18. D. Slepian, “Estimation of signal parameters in the presence of noise”, IRE Tras. Inform, Theory, p.68, 1954
19. D.C. Youla,”The use of maximum likehood in estimating continuous-modulated intelligence which has been corrupted by noise”. IRE Trans. Inform Theory, pp 90-106, 1954
20. F.C. Schweppe, “Sensor-array data processing for multiple signal sources”, IEEE

Trans. Inform, Theory, pp 294-305, 1968

21. D.W. Tufts and R. Kumerasan, “Improved spectral estimation”, Proc. IEEE, vol. 68, pp. 419-420, March 1980
22. D.W. Tufts and R. Kumaresan, “Estimation of frequencies of multiple sinusoids: Making linear prediction perform like maximum likehood”, Proc. IEEE, vol. 70, pp. 975-989, Sept. 1982
23. D.W. Tufts and R. Kumaresan, “Improved spectral resolution II,” Proc. ICASSP’80, pp 592-597, 1980
24. R. Kumaresan, “Optimal frequency estimation of sinusoidal signals in noise and applications,” Master’s thesis, Univ. Rhode Island, 1979
25. Coates Rodney F,W., “Underwater Acoustic Systems”, Macmillan,1990
26. Richard O. Nielsen, “Sonar Signal Processing”, Norwood: Artech House, 1991.
27. A.C. Dubey, C.Yuan and M.R.Azimi-Sadjadi, “Comparison of two different wavelet-based approaches for target detection in activesonar data”, Proceedings of SPIE, vol 3079, pp225-230,1997
28. Lora.G,Weiss,”Wavelets and Wideband Correlation Processing”, IEEE Signal Processing Magazine, pp 1-32, Jan 1994
29. Lora.G.Weiss,et,al.,”Wideband Processing of Acoustic Signals using wavelet Transform”, The Journal of the Acoustical Society of America, vol 96(2), pp 857-866, 1994
30. Quinquis, A.; Kalkoff, I.; Rossignol, S.;” Underwater detection of transient magnetic

signals: an application of wavelet packets", OCEANS '96. MTS/IEEE. 'Prospects for the 21st Century'. Conference Proceedings, Volume 2, pp 919 - 924 vol.2, 23-26 Sept. 1996

31. Chu-Kuei Tu; Yi-Chiu Lin;" Using wavelet packet decomposition technique on fuzzy classify model for underwater acoustic signal recognition", Underwater Technology, Proceedings of the 2002 International Symposium, Page(s):302 – 306, April 2002
32. Fargues, M.P.; Bennett, R.;" Comparing wavelet transforms and AR modeling as feature extraction tools for underwater signal classification", Conference Record of the Twenty-Ninth Asilomar Conference, Volume 2, Page(s):915 – 919, 1995
33. Harry L, Van Trees,"Detection, Estimation and Modulation Theory Part 3", John Wiley and Sons, 1971.
34. Albert Boggess, Francis J. Narcowich, "A First Course in Wavelets with Fourier Analysis", Prentice Hall, 2001
35. O. Rioul and M. Vetterli, "Wavelets and Signal Processing", Signal Processing Magazine, IEEE, Volume 8, Issue 4, Page(s):14 – 38, Oct. 1991
36. Stephane Mallat, "A Wavelet Tour of Signal Processing", Academic Press, 1997
37. Robi Polikar, "The Wavelet Tutorial", http://users.rowan.edu/~polikar/WAVELETS/
38. Ho, K.C.; Chan, Y.T, "Filter design and comparison for two fast CWT algorithms", IEEE Transactions on Signal Processing, , Volume 47, Issue 11, Page(s):3013 – 3026, 1999
39. Aleksander Grzeszczak, Mrinal K. Mandal, Sethuraman Panchanathan and Tet Yeap,

"VLSI Implementation of Discrete Wavelet Transform", IEEE Transactions on Very Large Scale Integration (VLSI) SYSTEMS, VOL. 4, NO. 4, 1996

40. M. Vishwanath, "The recursive pyramid algorithm for the discrete wavelet transform", IEEE Transactions on Signal Process, Vol. 42, no.3, pp. 673-677, Mar. 1994
41. M. Vishwanath, R.M. Owens, M.J.Irwin, "VLSI Architectures for the Discrete Wavelet Transform", IEEE trans. Circuits and Systems-II: Analog and Digital Signal Processing, Vol. 42, No.5, May 1995
42. Chien-Yu Chen, Zhong-Lan Yang, Tu-Chih Wang, Liang-Gee Chen, "A Programmable VLSI Architecture for 2-D Discrete Wavelet Transform", IEEE International Symposium on Circuits and Systems, May 28-31, 2000
43. Jer Min Joii, Yeu-Horng Shiati, Chin-Chi Liu," Efficient VLSI Architectures For The Biorthogonal Wavelet Transform By Filter Bank and Lifting Scheme", Circuits and Systems, Page(s):529 - 532 vol. 2, ISCAS 2001
44. Chao-Tsung Huang; Po-Chih Tseng; Liang-Gee Chen, "Flipping structure: an efficient VLSI architecture for lifting-based discrete wavelet transform", IEEE Transactions on Signal Processing, Volume 52, Issue 4, Page(s):1080 – 1089, 2004
45. Chao-Tsung Huang, Po-Chih Tseng, and Liang-Gee Chen," Analysis and VLSI Architecture for 1-D and 2-D Discrete Wavelet Transform", IEEE Transactions on Signal Processing, Vol. 53, No. 4, April 2005
46. M. J. Shensa, "The discrete wavelet transform: wedding the A trous and Mallat algorithms", IEEE Trans. Signal Processing, vol. 40, pp. 2464-2482, 1992.

47. O. Rioul and P. Duhamel, "Fast algorithms for discrete and continuous wavelet transforms", IEEE Trans Information Theory, vol. IT-38, pp. 569-586, 1992.

48. M. Unser, “Fast Gabor-like windowed Fourier continuous wavelet transforms,” IEEE Signal Processing Lett., vol. 1, pp. 76–79, May 1994.

49. Ho, K.C.; Chan, Y.T, “Optimum filter design for the A trous algorithm”, Time-Frequency and Time-Scale Analysis, 1998. Proceedings of the IEEE-SP International Symposium, Page(s):125 – 128, 1998

50. Ho, K.C.; Chan, Y.T, “Filter design and comparison for two fast CWT algorithms”, IEEE Transactions on Signal Processing, Volume 47, Issue 11, Page(s):3013 – 3026 , Nov. 1999

51. I. Daubechies, “The wavelet transform, time-frequency localization and signal analysis,” IEEE Transactions on Information Theory, vol. 36, pp. 961-1005, Sept. 1990.

52. J. M. Combes, A. Grossmann, Ph. Tchamitchian, Eds., Wavelets, Time-Frequency Methods and Phase Space, Berlin: Springer, IPTI, 1989.

53. J. Bertrand, P. Bertrand, and J. P. Ovarlez, “Discrete Mellin transform for signal analysis,” in IEEE Int. Conf. Acoustic., Speech, Signal Processing, Albuquerque, NM, pp. 1603-1606, 1990.

54. M. Unser, A. Aldroubi, and S. J. Schiff, “Fast implementation of the continuous wavelet transform with integer scales,” IEEE Trans. Signal Processing, vol. 42, pp. 3519–3523, Dec 1994.

55. Chaitali Chakrabarti, Mohan Vishwanath and Robert M. Owens," Architectures for Wavelet Transforms", VLSI Signal Processing, VI, Page(s):507 – 515, Oct. 1993

56. Chaitali Chakrabarti,.and Mohan Vishwanath, "Efficient Realizations of the Discrete and Continuous Wavelet Transforms: From Single Chip Implementations to Mappings on SIMD Array Computers", IEEE Transactions on Signal Processing, Vol. 43, No. 3, March 1995

57. Bruce W.Suter,"Wavelets: what kind of signal processing is that?" Proceeding of SPIE, vol. 2750, Pp118-128, 1996.

58. N. WIENER, "EXTRAPOLATION, INTERPOLATION AND SMOOTHING OF STATIONARY TIME SERIES, WITH ENGINEERING APPLICATIONS." NEW YORK, WILEY, 1949.

59. H. BODE AND C. SHANNON, "A simplified derivation of linear least squares smoothing and prediction theory," Proceedings of IRE, vol. 38, pp. 417-425, Apr. 1950.

60. R Kalman, "On the general theory of control," in Proc. 1st IFAC Congress. London: Buttenvorth, 1960.

61. R. Kalman and R. Bucy, "New results in linear filtering and prediction theory", Trans. ASME, Eng., vol. 83, pp95-107, Dec 1961.

62. T. Kailath, "A view of three decades of linear filtering theory," IEEE Transactions of Information Theory, vol. IT-20, pp. 145-181, Mar. 1974

63. Widrow, B.; Glover, J.R., Jr.; McCool, J.M.; Kaunitz, J.; Williams, C.S.; Hearn, R.H.; Zeidler, J.R.; Eugene Dong, Jr.; Goodlin, R.C.;" Adaptive noise cancelling: Principles

and applications", Proceedings of the IEEE Volume 63, Issue 12, Page(s):1692 – 1716, 1975

64. J. Kaunitz, "Adaptive filtering of broadband signals as applied to noise cancelling," Stanford Electronics Lab., Stanford Univ., Stanford, Calif., Rep. SUSEL-72-038, Aug. 1972 (Ph.D dissertation).
65. M. Sondhi," adaptive echo canceller," Bell System Technical Journal, vol.46, pp497-511 , Mar 1967
66. J. Rosenberger and E. Thomas, "Performance of an adaptive echo canceller operating in a noisy, linear, time-invariant environment," Bell System Technical Journal, vol. 50, pp. 785-813, 1971.
67. Markovich, S.; Gannot, S.; Cohen, I.; "Multichannel Eigenspace Beamforming in a Reverberant Noisy Environment With Multiple Interfering Speech Signals", IEEE Transactions on Audio, Speech, and Language Processing, Volume 17, Issue 6, pp 1071 – 1086, 2009
68. Khorram, S.; Sameti, H.; Veisi, H.; "LP-based over-sampled subband Adaptive Noise Canceller for speech enhancement in diffuse noise fields", 9th International Conference on Signal Processing, Page(s):157 – 161, 2008
69. Himel, T.; Allison, S.; Grossberg, P.; Hendrickson, L.; Sass, R.; Shoaee, H.; "An adaptive noise cancelling system used for beam control at the Stanford Linear Accelerator Center"Proceedings of IEEE workshop on Real time Applications, Page(s):212 – 215, 1993

70. J.-S. Roger Jang, `` ANFIS: Adaptive-Network-Based Fuzzy Inference Systems," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 03, pp 665-685, May 1993.

71. J.-S. Roger Jang and C.-T. Sun, `` Neuro-Fuzzy Modeling and Control", The Proceedings of the IEEE, Vol. 83, No. 3, pp 378-406, March 1995.

72. L. A. Zadeh. “Fuzzy sets”, Information and Control, 8:338-353, 1965.

73. T. Takagi and M. Sugeno, “Derivation of fuzzy control rules from human operator’s control actions,” in Proc. IFAC Symp. Fuzzy Inform. Knowledge Representation and Decision Analysis, pp. 55-60, July 1983

74. Wen Ding; Deliang Liang;” Modeling of a 6/4 Switched Reluctance Motor Using Adaptive Neural Fuzzy Inference System”, IEEE Transactions on Magnetics, Volume 44, Issue 7, Part 1, Page(s):1796 – 1804, 2008

75. Limpatthamapanee, S.; Phichaisawat, S.; “Determination of transfer capability using ANFIS with system condition separability”, 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2009, Volume 01, Page(s):14 – 17, 2009

76. Mar, J.; Feng-Jie Lin; “An ANFIS controller for the car-following collision prevention system”, Vehicular Technology, IEEE Transactions on Volume 50, Issue 4 , Page(s):1106 – 1113, July 2001

77. Ubeyli, E.D.; “Adaptive Neuro-Fuzzy Inference System for Analysis of Doppler Signals”, Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th

Annual International Conference of the IEEE, Page(s):2167 – 2170, 2006

78. Chen Bao-ping; Ma Zeng-qiang; “Short-Term Traffic Flow Prediction Based on ANFIS”, International Confernce on Communicatin Software and Networks, Page(s):791 – 793, 2006

79. Paramasivam, S.; Vijayan, S.; Vasudevan, M.; Arumugam, R.; Krishnan, R.; “Real-Time Verification of AI Based Rotor Position Estimation Techniques for a 6/4 Pole Switched Reluctance Motor Drive”, IEEE Transactions on Magnetics, Volume 43, Issue 7, Page(s):3209 – 3222 , July 2007

80. Yu-Shan Chen; Ke-Chiun Chang; “The Application of the Neuro-Fuzzy Computing Technique for the Forecasting of the R&D Project Performance”,Technology Management for the Global Future, 2006. PICMET 2006 Volume 5, Page(s):2228 – 2235, 8-13 July 2006

81. Drake, J.T.; Prasad, N.R.; “ANFIS for parameter estimation in coherent communications phase synchronization”, Neural Networks for Signal Processing XI, Proceedings of the 2001 IEEE Signal Processing Society Workshop, Page(s):433 – 442, 2001

82. Alexandrou, D.; de Moustier, C, “Adaptive noise canceling applied to Sea Beam sidelobe interference rejection”, IEEE Journal of Oceanic Engineering, Volume: 13 , Issue 2, pp 70 – 76, 1988

83. Aineto, M.; Lawson, S,” Narrowband signal detection in a reverberation-limited environment”, OCEANS '97. MTS/IEEE Conference Proceedings, Volume: 1, pp 27

– 32, 1997

CHAPTER III

THEORETICAL BACKGROUND

3.1 Introduction

The chapter introduces necessary background theories for understanding the project. The specifications and expectations for this sonar detection system will be introduced and together with specific techniques relevant to the project such as the wavelet transform, ANFIS and the Local Optimum Search adopted in the proposed system. Finally this chapter shows an integration of all these techniques into a complete system.

3.2 Introduction & Specification of the system

The purpose of this project is to seek an optimum solution for sonar signal detection in a reverberation-limited broadband and shallow-water environment. Given a specific enviroment, the Signal-Noise-Ratio(SNR) is expected to be less than -20dB. Therefore, the proposed system should be able to deal with signals buried in severe noise. Also in real life situations such as the requirement of the MOD, the system should be able to determine the speed of a moving target up to 50 knots with the resolution of 1 knot .

Since the family of scaled wavelets model the behavior of sonar pulses in a reverberation-limited broadband enviroment better than other signal detection techniques as explored in Chapter II[1,2], the use of a wavelet is proposed. However, reverberation noises have similar forms to the target pulses of the system, and previous research shows reverberation poses a serious problem in many active sonar applications. This is especially true in shallow water environments [3, 4, 5]. Therefore, an Adaptive-Noise-Canceller is also proposed in this system to filter unwanted noise and make the enviorment suitable for a wavelet transform to process target motion parameters.

To address the above considerations, the system shown in Fig 3.1 is proposed. The system sends out a sonar pulse at certain time inteveals, and the pulse will be bounced back from a moving target. The returned signal, which is shown in Fig 3.1 as primary channel, composed of target return pulses, reverbertion noises and white noise will be recorded by the receiver. The returned signal will be used as the input of the ANC first

to remove most of the noise by ANFIS operation as shown in Fig 3.1, then the wavelet transform will examine the remainder of the signals with the family of scaled wavelets, which is the sonar pulse the sytem sent out, to determine which signal is the right target as shown in Fig 3.1 as TME(Target Motion Estimator). When the reflected pulse is extracted from background noise, the location of the target relative to the receiver and the speed of the target can be calculated based the travel time to the target and the scale information from the wavelet transform result. The following paragraphs will introduce the theoritical background of the system.

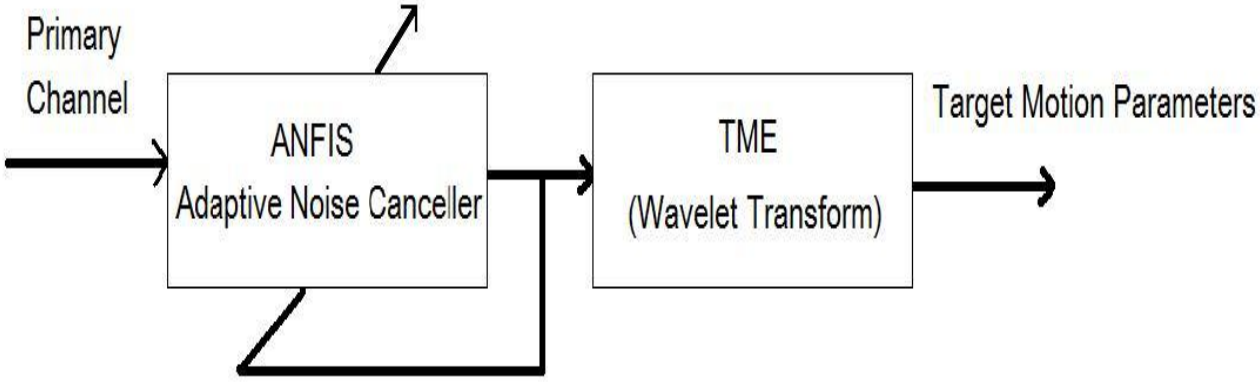


Fig 3.1 System Overview

3.3 Wideband Sonar Detection

The basic concept of transmitting and receiving signals and processing them to identify and localize objects is known as active *echo* location [1]. The processing is called sonar when the sound waves are used, which stands for Sound Navigation and Ranging. Systems used sound waves for detection are known as sonar systems [1].

There are two categories of sonar systems: passive and active [6, 7]. Passive sonar systems estimate objects by detecting the acoustic waves emitted from the object which is being looking for. In active sonar systems, a pulse whose characteristics are already known is transmitted and then this pulse is reflected back from the target to be detected and recorded by a receiver. Active sonar systems will then compute the velocity and the location of the target from the returned pulse as shown in Fig 3.2. In the Figure it is clear that on the receiver side the received signals contains reverberation from all scatters, including target which must be detect, in the ocean and ambient noise.

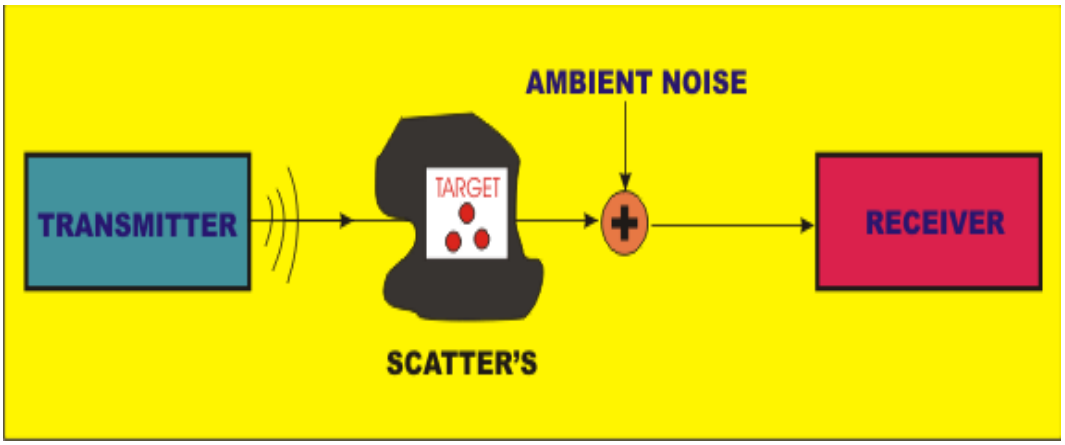


Fig 3.2 Active Sonar System

In this project an active sonar system is investigated and implemented. The active sonar system is more flexible since it can choose a pulse whose characteristics are most

suitable to describe the characteristics of target.

Methods used to identify and localize objects are commonly known as correlation processing or matched filter processing [8, 9, 10, 11], which measures the range and Doppler (frequency) components of the target echo by the cross-correlation operation of overlapping segments of the returned signal with a set of stored references (hypothesized signals). Narrowband model and the wideband model are two commonly used models for representing returned signal in cross-correlation. Which model should be selected for processing the signal can be decided based on applications for which characteristics of the received signal are assumed. Assuming that the object is moving with constant velocity v relative to the speed of sound c and for the duration T, the signal transmitted has bandwidth B. For the narrowband model, the main constraint is then subject to the assumption of

$$\frac{2v}{c} << \frac{1}{TB} \qquad (3.1)$$

which is called the narrowband condition [12, 13, 14]. A typical TB product in active sonar would be around 800 and c is of the order of 1500m/s, therefore Eq. 3.1 would give v << 0.9m/s (≃ 1.8knots) as the narrowband condition [15]. Even for slow moving objects, the narrowband condition would be violated. Removing the narrowband restrictions thus allows for wideband processing, which possesses several advantages including greater gain, better noise immunity and increased range resolution. Therefore the wideband model of underwater signal detection is adopted in the system described in the thesis.

The returned emitted signal after it strikes the target is referred to as the target echo, which can be defined mathematically in wideband model as [7]:

$$g(t) = \sqrt{S}\, f(S(t-\tau)) + n(t) \qquad (3.2)$$

Where

g (t) = Target Echo

$\sqrt{S}$ = Normalization term keep the transmitted signal and received Signals at the same energy.

$f(S(t-\tau))$ = Scaled and delayed version of transmitted pulse

n (t) = noises

The target echo is scaled and delayed in Eq 3.2. This is due to the fact that when a pulse is transmitted and reflects off an object (target) the received signal is created. It differs from the transmitted signal because of the speed and position of the object. In Equation 3.2 g (t) presents the returned signal from one single object in the ocean. In practical there are much more scatterers in the environment and therefore the signals received at the receiver will contain all g (t)s from different scatters with different scales and delays. Apart from the target echo from real target, all other g (t)s are categorized as reverberation noises. As introduced in Chapter II, reverberation and ambient noises are jointly considered as the background noises in wideband active sonar system. In most situations the background noises is dominating and therefore the major concern of a wideband sonar detection system is to extract the desired target echo out of the received signals. It is obvious that the form of reverberation noises is very similar to the target

echo. Understandings on similarity and difference between desired signal and noises would be huge benefit for sonar system development. As introduced in Eq 3.2, the target echo which is reflected off the target is what a sonar system wants to extract. Amplitude, nature of function of the signal, frequency and time are parameters which can affect a signal. In a sonar model, there are many factors which can affect amplitude of target echo, such as characteristics of scatters, ocean bottom or surface, nature of water, pressure or temperature, etc. However, there is little relation between amplitude and target's speed and location which are the two desired parameters the sonar system wants. Therefore, even we retrieve the amplitude of reflected signal, there are too many variables on this parameter and it is hard to identify any target motion parameters from it. In the mean time, the nature of the signal would not be changed during reverberation process; the frequency of target echo only relates to the velocity of the target scatter, and the arrival time only relates to the underwater acoustic propagation speed and the location of the target. In shallow water environment, the speed of acoustic propagation can be treated as a constant value. Therefore, the arrival time in equation 3.2 is only related to the location of the target. By extracting these two parameters-speed and location the target's speed and velocity can be easily obtained. Details of reverberation noise models can be found in Chapter II, and in the following text the characteristics of signal reflected from target are presented.

When a transmitted signal, f (t), reflects off a single object (a point scatterer), the received signal, g (t), is created. These signals differ due to the position and motion

between the receiver and the object [16]. If the transmitter/receiver and object are motionless, and if the distance between them is R, then the time for the signal to travel the distance R is R/c, and the roundtrip travel time, τ, is

$\tau=2R/c$ (3.3)

In this case, the received signal is $g(t) \approx f(t-\tau)$. When the object and/or the transmitter/receiver are moving, the roundtrip travel time is expressed by the time-varying time-delay, $\tau(t)$. The received signal is then $g(t) \approx f(t-\tau(t))$. If R (t) is the time-dependent range between the receiver and the object at time *t*, then the signal received at time *t* is reflected from the object at time $t-\tau(t)/2$. At that time, the object was at range $R(t-\tau(t)/2)$. Therefore, the roundtrip time delay is [2]:

$$\tau(t)=\frac{2}{c}R[t-\frac{\tau(t)}{2}] \quad (3.4)$$

When the target is moving with a velocity of v, the round trip time in Eq. 3.4 can be expressed by the time-varying delay,

$\tau(t) = \tau_0 + (2v/c+v)(t-\tau_0)$, $\tau_0=2R/c$ (3.5)

Where:

τ = Time taken for pulse to travel to target and back

R = Range from transmitter to target

c = Speed of sound in water (m/s)

If the target is moving relative to the transmitter/receiver then the transmitted pulse will experience a scaling factor, which will either compress or dilate the transmitted pulse. The scaling factor is calculated with the following formulae [17]:

$$s = \frac{c - v}{c + v} \tag{3.6}$$

Where:

s = scaling factor

v = speed of target (m/s)

If v is positive, it indicates the target is moving apart from the transmitter/receiver and vice versa for a negative target speed. If the scaling factor is less than 1 the pulse will be dilated and if it is greater than 1 the pulse will be compressed.

From Eq 3.5 and Eq 3.6 it is possible to calculate the range and the velocity of the target with the knowledge of the time taken for the pulse to travel from the receiver to the target and back to the receiver as well as the scaling factor.

Active echo localization techniques seek to identify and localize targets from received signals. This can be accomplished by cross correlation processing, which correlates the received signal, g (t), with a collection of hypothesized replicas of transmitted signal f (t). The hypothesized replicas serve as templates to which the received signal is matched. The peak value of correlated results arises when the received signal is correlated with the matched template. From the template which provides highest correlating results the estimated values of the parameters of the received signals can be obtained. In the wideband model, the outputs of the correlation process are [2]:

$$WC(\frac{1}{s}, \tau) = \sqrt{|s|} \int_{-\infty}^{\infty} g(t) f^{*}(s(t-\tau)) dt \tag{3.7}$$

Where $*$ denotes complex conjugation. Detection is accomplished if $WC(\frac{1}{s}, \tau)$ exceeds

a preset threshold, and target parameters $(\frac{1}{s},\tau)$ are estimated in terms of maximizing $WC(\frac{1}{s},\tau)$. In general, the correlator output gives a measure of how well the hypothesized signal matches the received signal as a set of parameters usually representing the target range and velocity. This processing has been shown to be an optimal technique especially for signals corrupted by an additive, Gaussian white noise where the output SNR is maximized [18, 19].

3.4 Wavelet Transform

The wavelet transform is a mathematical method to decompose signals into different frequency components at different times, and then study each component with a resolution matched to its scale. The basic idea of the wavelet transform is to analyze the target signals according to time and scale. It uses functions which satisfy certain mathematical requirements to represent target signals. It transforms a function by integrating it with modified versions of some kernel function [2]. The kernel function generally is called the mother wavelet, and the modified versions are time-shifted and dilated replicas of mother wavelets. For a given mother wavelet function f(t), the wavelet transform of a function g(t)$\in L^2(R)$ with respect to f(t) is:

$$WT(s,\tau)=\frac{1}{\sqrt{|s|}}\int_{-\infty}^{\infty} g(t)\,f^{*}(\frac{t-\tau}{s})dt \qquad (3.8)$$

Where $L^2(R)$ is the set of all square integrable or finite energy signals, and R denotes real numbers.

By comparing Eq. 3.7 with Eq. 3.8, it is obvious that if denote s→1/s, the output of the

wideband correlation can be represented as wavelet transform of the received signal g(t) with respect to the transmitted signal, f(t). Therefore, the seeking of optimum values of parameters of received signal can be solved by seeking the maximum values of the Continuous Wavelet Transform (CWT) coefficients.

The main idea of the CWT is to obtain time-scale digital signals using digital filter techniques by correlating wavelets at different scales with target signals at the scale to measure the similarity. The Continuous Wavelet Transform is computed by changing the scale of the analysis window, shifting the window in time, multiplying by the signal, and integrating over time. In order to implement the CWT by computer methods, discretely sampled data are used for CWT. In this case, filters with different cutoff frequencies are used to analyze the signal at different scales. The advantage is that the CWT can provide very fine details of target signals. Theoretically any scale can be obtained by using CWT; however this will also bring a heavy computation load.

Taking the equations above as an example, for each desired (s, τ), the output of wavelet transform can be considered as the output of a matched filter, which has coefficients corresponding to particular f(s, τ). Outputs from such matched filters then constitute a bank of filter outputs. Applying filter concepts, the maximum CWT coefficient at certain scale s can be obtained from a Finite impulse response (FIR) filtering of the received signal with the corresponding FIR coefficients[2,20].

Finite Impulse Response (FIR) filters are widely used in digital signal processing for cross-convolution operation. An N-tap FIR filter is defined by the following input-output

equation:

$$out(n) = \sum_{i=0}^{N-1} X(N-i)h(i) \qquad (3.9)$$

where {h(i): i = 0,…, N-1} are the filter coefficients. Making h(i) in Eq 3.9 as a wavelet function at certain (s, τ), and X(n) as the returned echo samples, the output of this FIR filtering gives the continuous wavelet transform results for the returned echo at certain (s, τ). As mentioned before, the Continuous Wavelet Transform is computed by changing the scale of the analysis window, shifting the window in time, multiplying by the signal, and integrating over all times. The N-tap FIR filter, which represents a wavelet function at certain scale s, is the analysis window used for Continuous Wavelet Transform. By feeding returned signals into this FIR filter, the Continuous Wavelet Transform at certain scale s on returned signal is performed. By extracting maximum value from CWT coefficients of different scales in filter output bank the optimum value of scale s is determined, and used to estimate another target parameter time-delay, τ,.

As mentioned above, CWT provides very good resolution of target signals; however it requires a heavy computation time and resources. The discrete wavelet transform (DWT), which O. Rioul and M. Vetterli introduced in [21], on the other hand, provides sufficient information both for analysis and synthesis of the original signal, with significant reduction in the computation time.

The basic theory behind DWT and CWT is the same - a time-scale representation of a digital signal is obtained using digital filtering techniques. However, The DWT analyzes the signal at different frequency bands with different resolutions by decomposing the

signal into a coarse approximation and detail information. In the discrete case, filters of different cutoff frequencies are used to analyze the signal at different scales. The signal is passed through a series of high pass filters to analyze the high frequencies, and it is passed through a series of low pass filters to analyze the low frequencies. Scaling functions and wavelet functions, which correspond to low-pass filter and high-pass filter, respectively, are two major functions employed in DWT. By successive high-pass and low-pass filtering of the time domain signal the decomposition of the input signal into different frequency bands is achieved. After filtering, the data samples can be sub-sampled by 2 according to the Nyquist's rule since the frequency range is halved after high-pass filtering and low-pass filtering. One level of decomposition can be expressed as:

$$y_h(k) = \sum_n X[n] * g[2k - n]$$

$$y_l(k) = \sum_n X[n] * h[2k - n] \qquad (3.10)$$

where $y_h[k]$ and $y_l[k]$ are the outputs of the high-pass and low-pass filters, respectively, and g[n] and h[n] presents half band high-pass filter and low-pass filter.

The time resolution is also halved by decomposition since only half of each filter output represents the signal. However, frequency resolution is doubled because each filter halves the frequency band of its input. For example, the frequency domain representation of a signal with a frequency range from 0 to f_n, and 3 levels of decomposition are shown in the Fig 3.3 and Fig 3.4.

The decomposition is repeated by decomposing approximation values from previous decomposition level with high-pass filter and low-pass filter and down-sampled the

outputs. Fig 3.4 shows the signal process of 3-level decomposition.

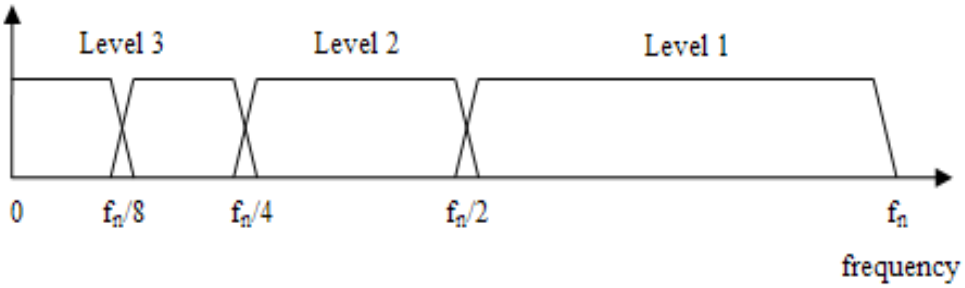


Fig 3.3 Frequency domain representation of the DWT

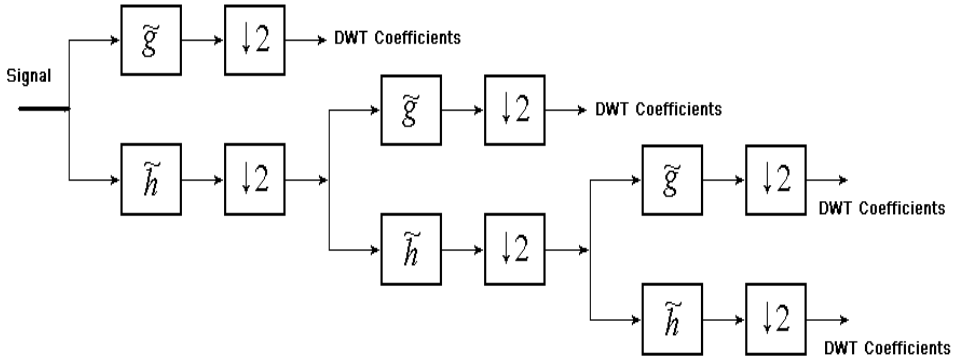


Fig.3.4 3-level DWT Decomposition

The Inverse Discrete Wavelet Transform (IDWT) reconstructs the DWT coefficients into the original signal format by inverting the steps of the DWT decomposition. It up-samples the DWT wavelet coefficients and convolves the results with reconstruction high-pass filter and low-pass filter. Fig 3.5 gives an example of the signal process of a 3-level reconstruction. The IDWT is usually used along with the DWT in real applications such as image compression, digital watermarking, data de-noising, etc. In these applications the DWT is used to decompose original data into different scales, and then these DWT coefficients will be processed according to the application requirements. At last these processed DWT coefficients will be reconstructed into the original signal's

format by the IDWT.

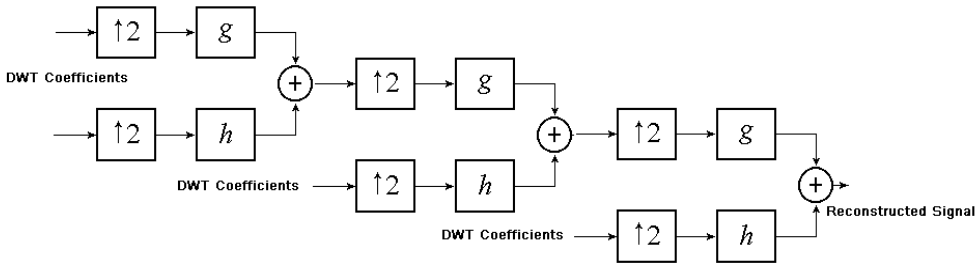


Fig.3.5 3-level IDWT Reconstruction

The discrete wavelet transform has a huge number of applications in science, engineering, mathematics and computer science for its ability to capture frequency and location information, its relatively good resolution and affordable computation time and resources. However, in situations which requires high resolution on both frequency and location information, for example, sonar detection, CWT is still a better choice than DWT. In order to achieve high resolution of detection the redundancy introduced by the CWT is necessary.

3.5 Local Optimum Search

The CWT operation obtains high resolution by producing highly redundant information when it convolves with so many scaled signals. Therefore a heavy computation time and resources are required for CWT implementation. This heavy resource consumption limits the uses of CWT operation in real applications. In order to reduce the computation load of the CWT operation, and to keep its resolution advantage, a Local Optimum Search algorithm is introduced in the system described in this thesis.

Traditionally in order to find an optimum result from numerous possible solutions, all possibilities will be examined using a certain algorithm. Since this result is the optimum result among all possible solutions, it is called a global optimum result and this method is called global optimum search. The advantage of this algorithm is its precision. It can guarantee the final result is the best among all possibilities. However, if the number of all possible solutions increases, the computation load of this algorithm increases linearly as well. When the number of possibilities reaches at certain level, the implementation of a global optimum search will be very costly either on hardware resources or on time. Therefore a local optimum search is introduced into our implementation.

A local optimum of a combinatorial optimization problem is an optimal solution within a neighboring set of solutions instead of all solutions. The result may not be the best result among all solutions, but this algorithm can give a relatively good result within an acceptable time period. Instead of seeking a global optimum result from all possible solutions in a predefined range of target motion parameters, the local optimization search will start from a candidate solution and iteratively move to a neighboring solution until the time bond is reached or the best solution found has not been improved for a certain number of steps. Without going through all possible solutions in the range the local search will generally give a reasonable solution and therefore the computation load is significantly reduced. Brent's search method [22] is adopted in this thesis to produce local optimum results.

Brent's search is a linear search that is a hybrid of the golden section search and a quadratic interpolation [22]. The detailed steps of Brent's search are shown in figure 3.6:

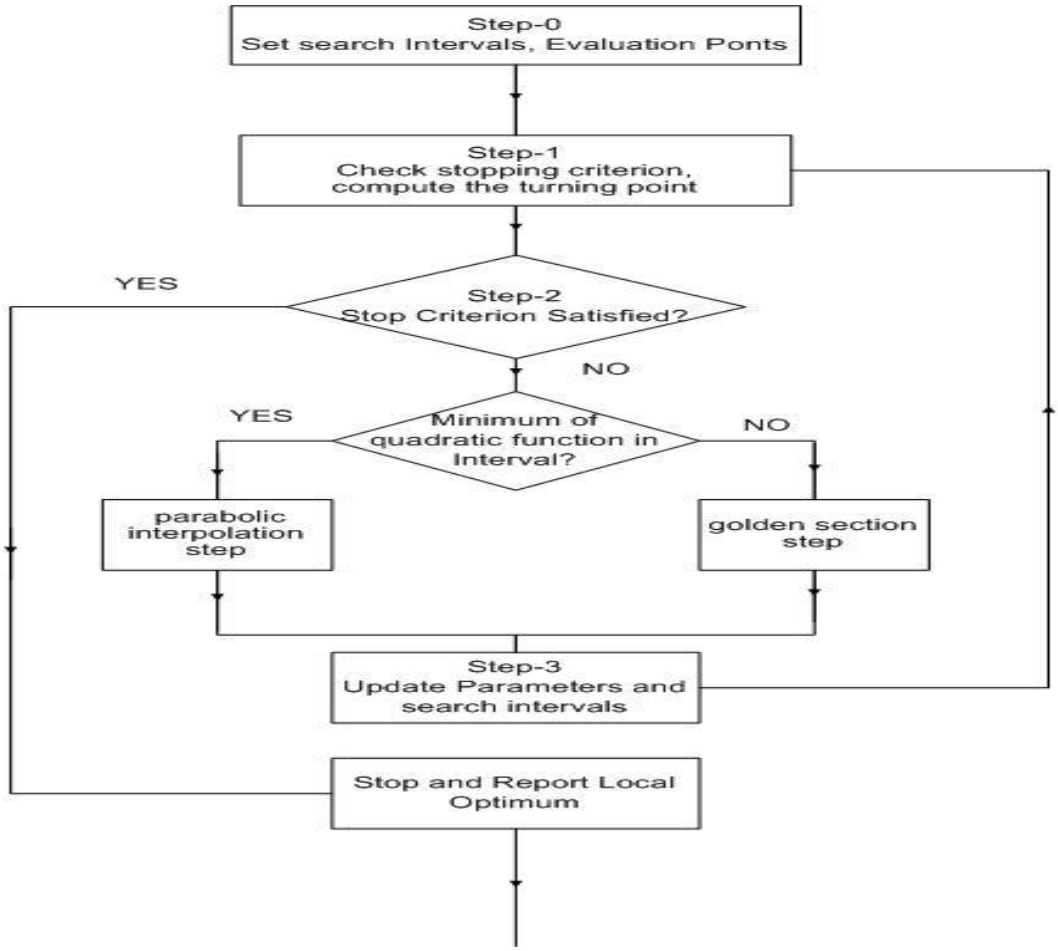


Fig 3.6 Brent's Search

Step-0.: Set effective search interval [a, b], the final evaluated point u. Use golden section search to set internal points x (recent evaluation point), w (previous evaluation point of x), and v (previous evaluation point of w);

Step-1: Check stopping criterion and compute the turning point;

Step-2: Choose approximation step by examining whether this minimum of the quadratic function is within the appropriate interval of uncertainty. If it is, it is used in

the next stage of the search and a new quadratic approximation is performed. If the minimum falls outside the known interval of uncertainty, then a step of the golden section search is performed.

Step-3: Update parameters based on bisection and go to Step-1 until the termination criterion is satisfied.

3.6 Adaptive Noise Canceller

Cross-correlation is optimal for signals buried in additive, Gaussian, white noise [18]. However, the specification for the proposed system is target detection in reverberation-limited broadband and shallow-water environment.

Therefore difficulty in detecting a specific target in the ocean due to the presence of other objects [23]. When the pulse is emitted by the transmitter to the ocean, it not only strikes the target but also other bodies in the ocean such as the surface and the bottom of the ocean, fish, and plankton, etc. The pulse is reflected off these bodies and is picked up by the receiver as well. The signal from these bodies in the ocean is known as Reverberation and the bodies are called scatters [24, 25].The Reverberation waveform causes a lot of problems in detecting the target echo simply because the pulse that strikes the target is the same pulse that strikes the scatters; therefore the target echo and the reverberation waveform have a very high correlation. This is undesirable and makes it hard to detect the target echo from the received signal at the transmitter. Therefore, an ANC is introduced to eliminate reverberation noise as described at the beginning of this chapter as a solution

for the proposed system. Though various noise cancellers are available, the nature of reverberation noise, unpredictable frequency and amplitude, makes most fixed amplitude threshold or fixed cutting frequency noise cancellers not suitable for this project. Therefore, Adaptive Noise Canceller was proposed for the system. In addition Adaptive Network-based Fuzzy Inference System (ANFIS) is adopted in the system as ANC due to its ability to work with the chaotic nature of impulse noises.

Adaptive-Network-based Fuzzy inference System, which is proposed by Jyh-Shing Roger Jang in [26], is a combination of a Fuzzy inference system and an adaptive network. It serves as a basis for constructing a set of fuzzy if-then rules with proper membership rules to generate the stipulate input-output pairs [26].

3.6.1 Fuzzy Inference System

Fuzzy if-then rules or fuzzy conditional statements are expressions of the form IF A THEN B, where A and B are labels of fuzzy sets [27] characterized by appropriate membership functions. Due to their concise form, fuzzy if-then rules are often employed to capture the imprecise modes of reasoning that play an essential role in the human ability to make decisions in an environment of uncertainty and imprecision. For example, the following expression describes a simple if-then rule:

If *temperature* is *high*, then *ice* is *thin;*

Where *temperature* and *ice* are linguistic variables and *high* and *thin* are linguistic values which are known as membership functions.

A Fuzzy inference system is a system to process a corresponding mapping from given

input to an output using fuzzy logic. This mapping then provides a basis for decision making or pattern reorganization. The use of membership functions, fuzzy if-then rules and logic operations are essential in the process.

3.6.2Adaptive Network

An adaptive Network is a network that consists of nodes and directional links which connect all the nodes. Among the nodes in the network part or all of the nodes are adaptive, which means their outputs depend on the parameter(s) pertaining to these nodes, and the learning rule specifies how these parameters should be changed to minimize a prescribed error measure[28].

Fig 3.7 shows an example of an adaptive network structure. It is a multi-layer feed forward network consisting of adaptive nodes which perform different functions on input signals and parameters pertaining to each node. The function of each node depends on the specific task this network is supposed to perform. To reflect different adaptive capabilities, both circle and square nodes are used in an adaptive network. A square node (adaptive node) has parameters while a circle node (fixed node) has none. The parameter set of an adaptive network is the union of the parameter sets of each adaptive node. In order to achieve a desired input-output mapping, these parameters are updated according to given training data and a gradient-based learning procedure. It should be noticed that though it is not shown in Fig 3.7, ANFIS will need a set of training data to evaluate the performance of the system by comparing outputs with the training data. Training data is not used in forward-pass shown in Fig 3.7(b), but it is

essential in learning procedure afterwards to modify premise parameters by back-propagation.

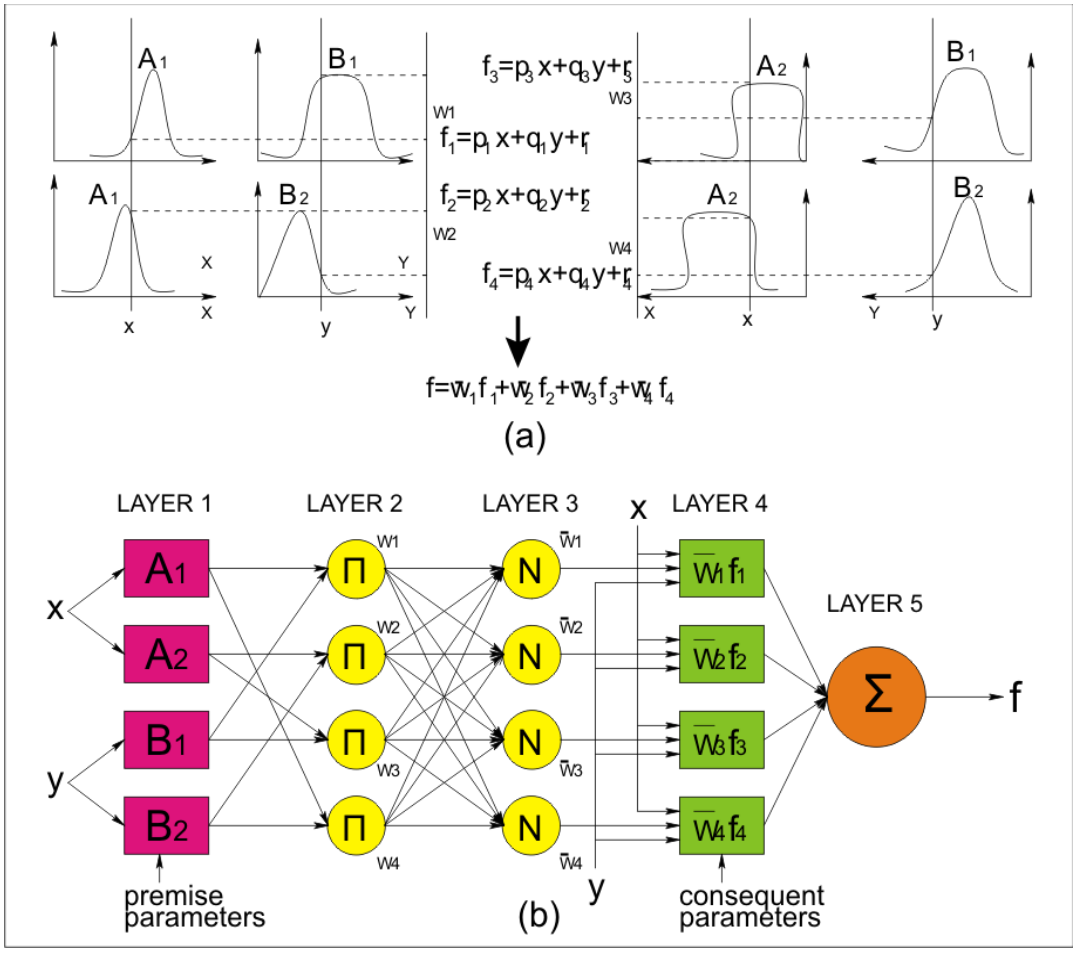


Fig 3.7 Fuzzy Inference System and ANFIS Structure

There are two types of learning procedure for adaptive networks. With the batch learning (offline) procedure, the parameters for adaptive nodes will be updated only after the whole training data is presented. On the other hand, if we want the parameters to be updated right after each input-output pair, the pattern learning procedure should be introduced.

3.6.3 Adaptive-Network-based Fuzzy Inference System

The architecture of the Adaptive-Network-based Fuzzy Inference System is designed to combine a hybrid learning procedure of a neural network with reasoning capacities of fuzzy logic to find a model or mapping that correctly associate with the stipulated input-output pairs. The ANFIS operation implements the first or zero order Takagi and Sugeno's fuzzy if-then rules [30], which are designed to formalize a systematic approach for generating fuzzy rules from a given input-output data set. In the following, the ANFIS architecture is described in terms of the Sugeno-type model of fuzzy inference systems followed by its hybrid learning algorithm.

A typical fuzzy rule in a first-order Sugeno-type model has a format

If x is in A and y is in B then $z = px + qy + r$

where $A \subseteq X$ and $B \subseteq Y$ are fuzzy sets in the premise with input variables (or objects collected) $x \in X$,and $y \in Y$, respectively, and X, Y are space of objects; z is a crisp output function linearly combining input variables with constant weights p and q plus a constant term r in the consequent. For an adaptive network, which is a multi-layer feed-forward network, the final crisp output is then the weighted average of each rule's output. For simplicity, a first-order Sugeno fuzzy inference system is described as a example, which has two inputs x and y and one output f. Suppose that two bell-shaped membership functions are provided on each of the two inputs, four together, so the input space of the generated FIS structure is partitioned into 4 fuzzy subspaces, each of them

is governed by fuzzy if-then rules:

Rule 1: If x is in A1 and y is in B1, then f1 = p1x + q1y + r1

Rule 2: If x is in A1 and y is in B2, then f2 = p2x + q2y + r2

Rule 3: If x is in A2 and y is in B1, then f3 = p3x + q3y + r3

Rule 4: If x is in A2 and y is in B2, then f4 = p4x + q4y + r4.

Here Ai (and Bi) is fuzzy sets describing the degree of chosen membership functions (MFs) μAi (x) and μBi (y) to which the given input x (and y) satisfying Ai (and Bi). Thus, the fuzzy sets are defined as

Ai = {(x, μAi (x))|x ∈ X},Bi = {(y, μBi (y))|y ∈ Y }.

The chosen membership function, in a statistical sense, must form a bell-shape in the range [0, 1] and usually can be the generalized bell function, such as:

$$\mu_{A_i}(x) = \frac{1}{1+[x-c_i / a_i]^{2b_i}} \qquad (3.11)$$

Or the Gaussian function

$$\mu(x) = \exp[-(\frac{x-c}{a})^2] \qquad (3.12)$$

where {a, b, c} (or {a, c} in the latter case) is the parameter set controlling various forms of the membership functions on fuzzy sets Ai or Bi.

The topology of the proposed ANFIS operation is depicted in Fig. 3.7 where Fig. 3.7(a) shows graphically the fuzzy reasoning mechanism to derive an output f from a given input training data set {x, y}, while the adaptive network represented by the ANFIS architecture in Fig. 3.7(b) is functionally equivalent to a fuzzy inference systems in Fig. 3.7(a). The ANFIS architecture contains the following five layers:

Layer–I. Every node i in this layer is an adaptive node containing node function (or membership function) $\mu_{A_i}(x)$ defined above whose parameters {ai, bi, ci} or {ai, ci} are called premise parameters. During the learning phase, these parameters change continuously resulting in various forms of membership functions on each fuzzy set. These exhibited membership functions are then compared with input variables to obtain the membership values.

Layer–II. Every node in this layer is a fixed node labeled II whose output is the product of all incoming signals. The output of the node represents the firing strength of a rule, for example, given below:

$$w_j = \mu_{A_i}(x) \times \mu_{B_i}(y), i = 1,2; j = 1,\cdots,4 \qquad (3.13)$$

Layer–III. Every node i in this layer is a fixed node labeled N. The node i gain the ratio of the i – th rule's firing strength to the sum of all rules' firing strengths:

$$\bar{w}_i = \frac{1}{\sum_{j=1}^{4} w_j}, i = 1,\cdots,4 \qquad (3.14)$$

Layer–IV. Every node i in this layer is an adaptive node with a set of consequent parameters {pi, qi, ri} pertaining to it to result in a weighted node function

$$\bar{w}_i f_i = \bar{w}_i(p_i x + q_i y + r_i), i = 1,\cdots,4 \qquad (3.15)$$

The set of consequent parameters is awaited to be determined.

Layer–V. The single node in this layer is a fixed node labeled V, which computes the overall outputs as the summation of all incoming signals.

$$O = \sum_{i=1}^{4} \bar{w}_i f_i = \sum_{i=1}^{4} \frac{w_i}{\sum_{j=1}^{4} w_j} f_i \qquad (3.16)$$

An overview of the adaptive network suggests that ANFIS is a hybrid learning algorithm, which combines the gradient decent method to upgrade the premise parameters and least-squares method to identify the consequent parameters. More specifically, in each epoch of this hybrid learning procedure, given input data with premise parameters fixed, functional signals go forward to gain each node output until the layer-IV and the consequent parameters are identified by the sequential least squares estimator. This procedure has formed a forward pass.

After the parameters have been identified, the functional signals keep going forward to calculate the error measure E of the training set:

$$E = \sum_{m=1}^{n} (R_m - O_m)^2 \qquad (3.17)$$

Where R_m is the m-th target output and O_m is the m-th estimated output, and n is the total number of input-output pairs of training set.

In [26] details of using error rate to update parameters in adaptive networks are described. Suppose that a given adaptive network has L layers and the kth layer has #(k) nodes, the node in the i-th position of the k-th layer can be denoted by (k,i) and its node function (or node output) by O_i^k .Defining error rate for p-th training data and for each node output O is $\partial E_p / \partial O$, The error rate for the output node at (L,i) can be calculated as:

$$\frac{\partial E_p}{\partial O_{i,p}^L} = -2(R_{i,p} - O_{i,p}^L) \qquad (3.18)$$

Where L is the number of layers adaptive networks has.

For the internal node at (k,i)， the error rate can be derived by the chain rule:

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \frac{\partial E_{m,p}^{k+1}}{\partial O_{i,p}^k} \qquad (3.19)$$

Where $1 \le k \le L-1$.That is, the error rate of an internal node can be expressed as a linear combination **of** the error rates of the nodes in the next layer.

If α is a parameter of the given adaptive network, the error rate can be represented as:

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha} \qquad (3.20)$$

where S is the set of nodes whose outputs depend on α. Then the derivative of the overall error measure E with respect to α is:

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial \alpha} \qquad (3.21)$$

Accordingly, the update formula for the generic parameter α is:

$$\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha} \qquad (3.22)$$

in which η is a learning rate which can be further expressed as:

$$\eta = \frac{k}{\sqrt{\sum_{\alpha} (\frac{\partial E}{\partial \alpha})^2}} \qquad (3.23)$$

Where k is the step size, the length of each gradient transition in the parameter space.

Since the error rates apparently propagate backward from the output end toward the input end, for a given set of fixed consequent parameters, the premise parameters can be updated by the gradient decent method as a result of the minimization of the error measure for each input-output pair. This forms a backward pass which is exactly the same as the back-propagation learning rule commonly used in the feed-forward neural

networks [29].

3.7 Conclusion

This chapter introduces the specifications and working environments for the proposed system. Then the general introduction of underwater sonar detection is presented afterwards. Due to the ability to capture both frequency and location information of target signals and providing high resolution in both the frequency domain and time domain, the continuous wavelet transform is chosen as the target detection technique. In order to reduce the computation load of CWT and keep the resolution advantage, a local optimum search is introduced into the system. At last the ANC is proposed to reduce reverberation noises and make the received signals more suitable for CWT to operate on.

3.8 References

1. R.A.Altes, “Detection, Estimation, and Classification with Spectrograms.” JASA, (4) pp. 1232-1246, Apr. 1980.

2. Lora G. Weiss, “Wavelet and Wideband Correlation”, IEEE Signal Processing Magazine, pp 13-32, Jan 1994.

3. R. B. Mitson, Fisheries Sonar. Farnham, Surrey, England: Fishing News Books, Ltd., 1983.

4. D. V. Holliday, “Doppler structure in echoes from schools of pelagic fish,” Journal of Acoustical Society of America., vol. 55, pp. 1313-1322, 1974.

5. D. G. Pincock and N. W. Easton, “The feasibility of Doppler sonar fishing counting,” IEEE J. Ocean. Engineering, vol.3, pp 37-40, 1978

6. Coates Rodney F.W., “Underwater Acoustic Systems”, Macmillan, 1990

7. Richard O. Nielsen, “Sonar Signal Processing”, Norwood: Artech House, 1990

8. M. J. Jacobson, “Space-time correlation in spherical and circular noise fields”, JASA. vol. 34, no. 7, pp 971-978, 1962.

9. T. H. Glisson, C. I. Black, and A. P. Sage, “On digital replica correlation algorithms with applications to active sonar,” IEEE Trans. Audio and Elect, vol. 17, no. 3, pp. 190-197, 1969.

10. P. M. Schulthesis and E. Weinstein, “Estimation of differential doppler shifts”, JASA,vol. 66, no. 5, pp. 1412-1419, 1979.

11. E. J. Kelly and R. P. Wishner, “Matched-filter theory for high-velocity targets,” IEEE Trans. Military Elect., vol. 9, pp. 56-69, 1965.

12. J. L. Stewart and E. C. Westerfield, “A theory of active sonar detection,” Proc. IRE, pp. 872-881, 1959.

13. C. E. Cook and M. Bernfeld, “Radar Signals, An Introduction to Theory and Applications”, Academic Press, 1967.

14. D. A. Swick, “A review of wideband ambiguity functions,” NRL Report 6994, 1969.

15. Jason Tseng,” Application of Adaptive Neuro-Fuzzy Inference Systems to Active Wideband Signal Detection in a Reverberation-Limited Environment-Part I: Single target environment”, Research Report, University of Warwick, 2006

16. E. J. Kelly and R. P. Wishner, “Matched-Filter Theory for High-velocity Targets,” IEEE Trans. Militmy Elect**,** pp. 56-69, 1965.

17. William C. Knight et al., “Digital Signal Processing for Sonar”, Proceedings of the IEEE, vol. 69, no. 11, November 1981

18. H. Van Trees, Detection, Estimation, and Modulation Theory, Parts I, 11, Ill, Wiley, 1968.

19. M. I. Skolnik, Introduction to Radar Systems, McGraw-Hill, New York, 1980.

20. Rabiner, Lawrence R., and Gold, Bernard, “Theory and Application of Digital Signal Processing”, Prentice-Hall, 1975

21. O. Rioul and M. Vetterli, “Wavelets and Signal Processing”, Signal Processing

Magazine, IEEE, Volume 8, Issue 4, pp14 – 38, Oct. 1991

22. R. P. Brent, Algorithms for Minimization without Derivatives, Prentice-Hall, Englewood Cliffs, New Jersey, 1973

23. Paul C, Etter, “Underwater Acoustic Modeling”. E&FN SPON, 1996

24. William S, Burdic, “Underwater Acoustic System Analysis”. Prentice Hall, Second Edition, 1991

25. Pierre Faure, “Theoretical Model of Reverberation Noise”. The Journal of the Acoustical Society of America, vol 36(2), pg 256-266, 1964

26. Jyh-Shing Roger Jang, “ANFIS: Adaptive-Network-Based Fuzzy Inference System”, IEEE Transctions on Systems, man, and Cybernetics, Vol. 23, No. 3, June 1993

27. L. A. Zadeh. “Fuzzy sets”, Information and Control, pp 338-353, 1965.

28. S. Cben, C. F. N. Cowan, and P. M. Grant, “Orthogonal least squares learning algorithm for radial basis function networks,” IEEE Trans. Neural Networks, vol. 2, no. 2, pp. 302-309, Mar. 1991.

29. D. E. Rumelhart, G. E. Hinton, and R. J.Williams, “Learning internal representations by error propagation.” In D. E. Rumelhart and J. L. McClelland, editors, Parallel Distributed Processing: explorations in the microstructure of cognition, vol. 1, chapter 8, pp. 318-362, MIT Press, Cambridge, MA, 1986.

30. M. Sugeno, “Industrial applications of fuzzy control,” Elsevier Science Pub. Co,1985.

CHAPTER IV

HIGH-LEVEL SYSTEM ARCHITECTURE AND SIMULATION RESULTS

4.1 Introduction

As described in Chapter III, the proposed system is based on Adaptive Network-based Fuzzy Inference System (ANFIS) for Adaptive Noise Canceller and Continuous Wavelet Transform (CWT) to determine the target echo's motion parameters. Local Optimum Search is adopted to reduce the computation load for wavelet transform.

In this chapter the theoretical model of the proposed system is presented as well as the design details of the high-level system. The Matlab simulation results for the high level system are illustrated to demonstrate the performance of the proposed system.

4.2 High level System Architecture

The High level system architecture of the proposed active sonar system is presented in Fig 4.1. The system consists of two function blocks: Adaptive Noise Canceller (ANC) and Target Motion Estimator (TME). The ANC part adopts an Adaptive Network-based Fuzzy Inference System to minimize noise effects and then TME will extract target motion parameters using the Continuous Wavelet Transform (CWT).

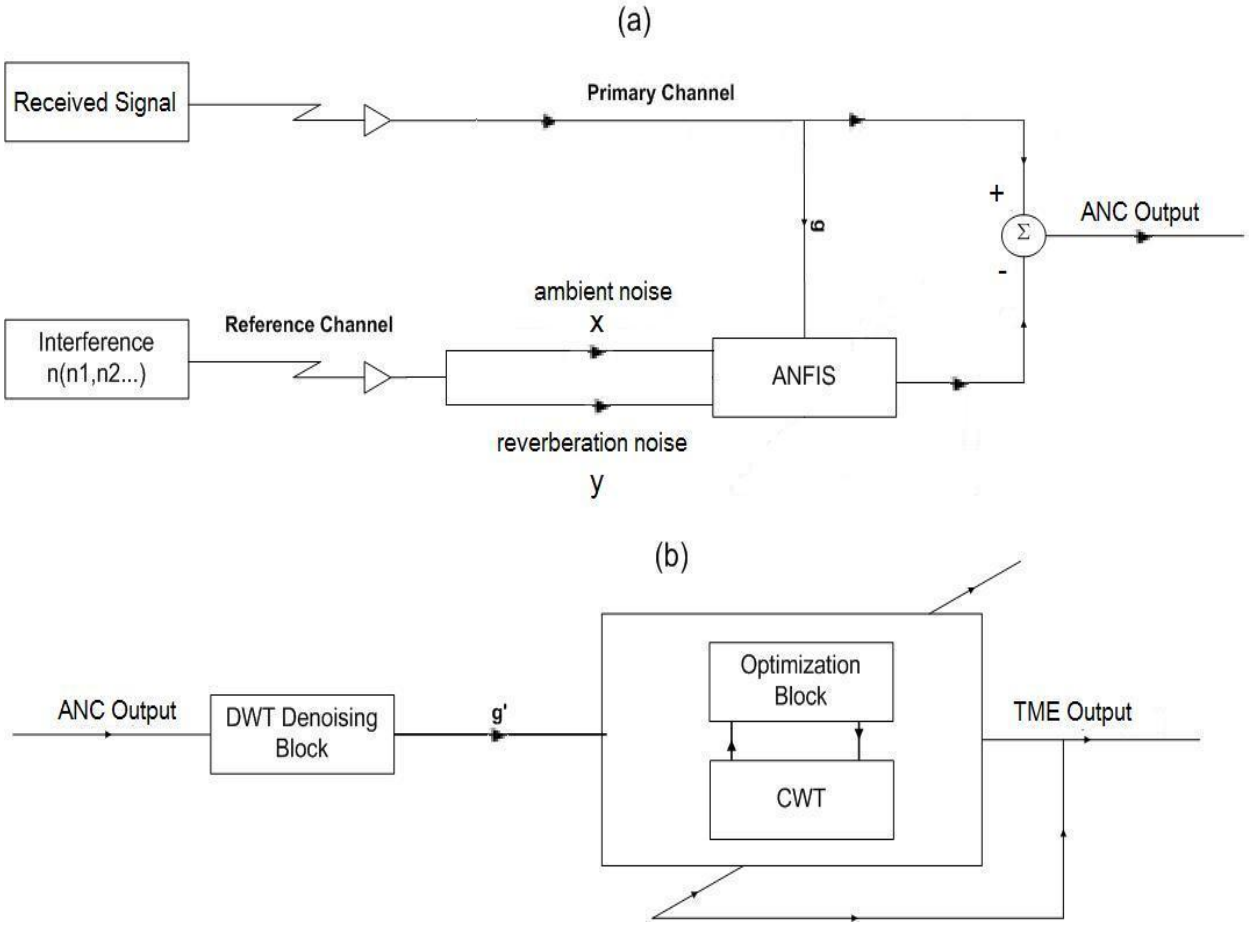


Fig.4.1 Hybrid System (a) Adaptive Noise Canceller (b) TME

The primary channel for Adaptive Network-based Fuzzy Inference System (ANFIS), shown in Fig 4.1(a), is the combination of target signal and interference noise, which is composed of reverberation noise and white-Gaussian noise, in practice is the received

signal at the receiver. The interference noise is used as the reference channel for ANFIS so that the ANFIS will adjust its parameters to identify noise in the primary channel. Two sets of interference noises, shown in Fig 4.1 (a) as ambient noise and reverberation noise, are used as two inputs for ANFIS model, corresponding to input X and input Y in Fig 4.2. The primary channel, which in practice is the received signal at receiver, is used as training data for ANFIS model. As described in Chapter III, ANFIS model generates estimated outputs by feeding inputs into its non-linear filters and adjusting the filters' parameter through its five-layer structure to achieve better results. This process is called forward-pass. Once input data are passed trough five layers and the output data of forward-pass is obtained, the ANFIS model compares the output of forward-pass with training data, which in this case is the received signal. Then the ANFIS back-propagates the parameters from layer 5 to layer 1 to modify non-linear filters according to evaluation results based on comparison, as described in chapter III through Eq3.18 to Eq 3.23. In next epoch, the input data is passed through ANFIS' five layers again with parameters modified in previous epoch and back-propagate parameters again when forward-pass is finished. As the number of iterations increases, the estimated output from ANFIS will become more and more similar to the training data. As the end result of these iterations the output of ANFIS will produce an estimated background noise similar to the one present in the primary channel. By subtracting the estimated noise signals from the primary channel the reflected target signal will be extracted from the primary channel, which is shown in Fig 4.1 (a) as ANC output.

Output from the ANC is then fed into the Target Motion Estimator (TME) block to identify the motion parameters of the target. As shown in Fig 4.1 (b), the output of ANC is fed into a DWT denoising block to further reduce noise in the low frequency range. The the optimization block and the CWT function block are then used to obtain a set of most appropriate coefficients to extract target motion parameters from the received signal as described in Chapter III.

4.3 ANFIS modelling

As described in Chapter III, ANFIS operation is adopted for the ANC architecture for its ability of offering a best solution to fast track linearity and nonlinearity between signals in the multidimensional input space. The architecture of ANFIS is to combine a hybrid learning procedure of a neural network with the reasoning capacities of fuzzy logic to find a model or mapping that correctly associates with the stipulated input-output pairs [2, 3]. The ANFIS operation implements the first or zero order Takagi and Sugeno's fuzzy if-then rules [2], which were originally developed to formalize a systematic approach for generating fuzzy rules from a given input-output data set [3].

The topology of the ANFIS operation is depicted in Fig. 4.2 where Fig. 4.2(a) shows graphically the fuzzy reasoning mechanism to derive an output from a given input training data set $\{x, y\}$, while the adaptive network represented by the ANFIS architecture in Fig. 4.2(b) is functionally equivalent of a fuzzy inference systems in Fig. 4.2(a). However, Fig 4.2(b) only shows the forward pass described in Chapter III. Once

forward pass is finished, the premise parameters are modified by back propagation through layer 5 to layer 1 with the same topology according to performance evaluation.

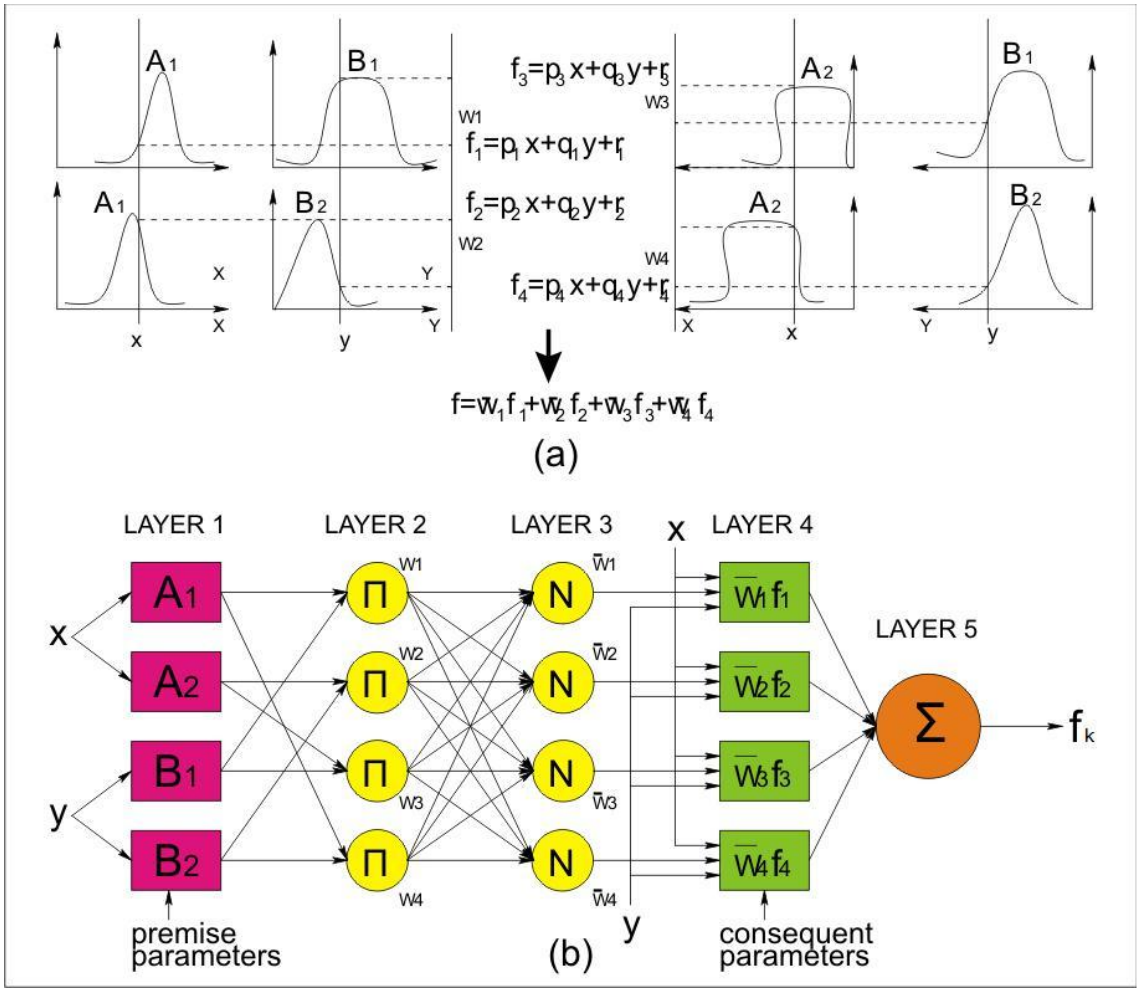


Fig 4.2 Fuzzy Inference System and ANFIS Structure

Assume N to be the number of inputs for ANFIS operation and M to be the number of membership functions for each input, the number of fuzzy rules in the architecture can be easily calculated as M^N, since each rule represents a combination of membership functions, one from each input[3]. Due to this exponential dependency on the number of inputs, the value of N should be kept as low as possible, or the complexity of the receiver could be significantly increased.

Fig 4.2 shows the architecture of ANFIS operation with 2 inputs and 2 membership

functions for each input. As mentioned previously in this chapter, interference noise is used as input for ANFIS. As described in chapter II, reverberation noise and ambient noise constitute background noises in active sonar system. Therefore one of these two inputs will be ambient noises and the other will be reverberation noises, which can be considered as x and y in Fig 4.2.

In Fig 4.2, there are 4 fuzzy rules in the network since M=2 and N=2. In order to reduce complexity, the final solution for this project uses 2 inputs and 4 membership functions for each input. Therefore, an ANFIS network of 16 fuzzy rules is generated. As described in Chapter III, there will be 3 consequent parameters for each fuzzy rule under 2- input network. In total 48 consequent parameters exist in the proposed network.

The chosen membership function, in a statistical sense, forms a bell-shape in the range [0, 1] generalized by the bell function

$$\mu_{A_i(x)} = \frac{1}{1+[x-c_i/a_i]^{2b_i}} \qquad (4.1)$$

or the Gaussian function

$$\mu(x) = \exp[-(\frac{x-c}{a})^2] \qquad (4.2)$$

Where {a,b,c} (or {a, c} in the latter case) is the parameter set controlling various forms of the membership functions on fuzzy sets[4]. In the current project, the Gaussian function is chosen since it only needs 2 instead of 3 parameters to determine the form of the membership function. It is easy to conclude there are 2*2*4=16 premise parameters in the network. The shape of initial membership functions is shown in Fig 4.3. The

connections of the ANFIS network are shown in Fig 4.4.

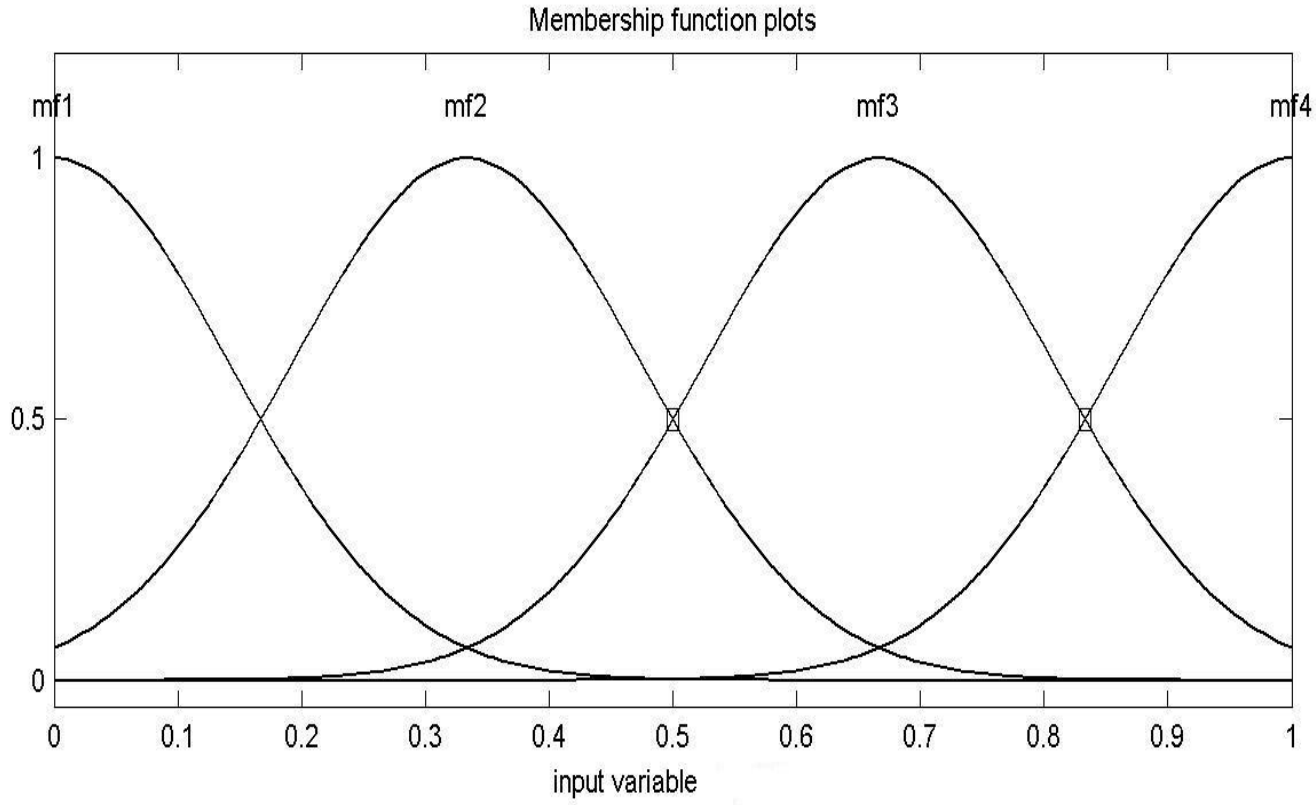


Fig 4.3 Initial Membership functions

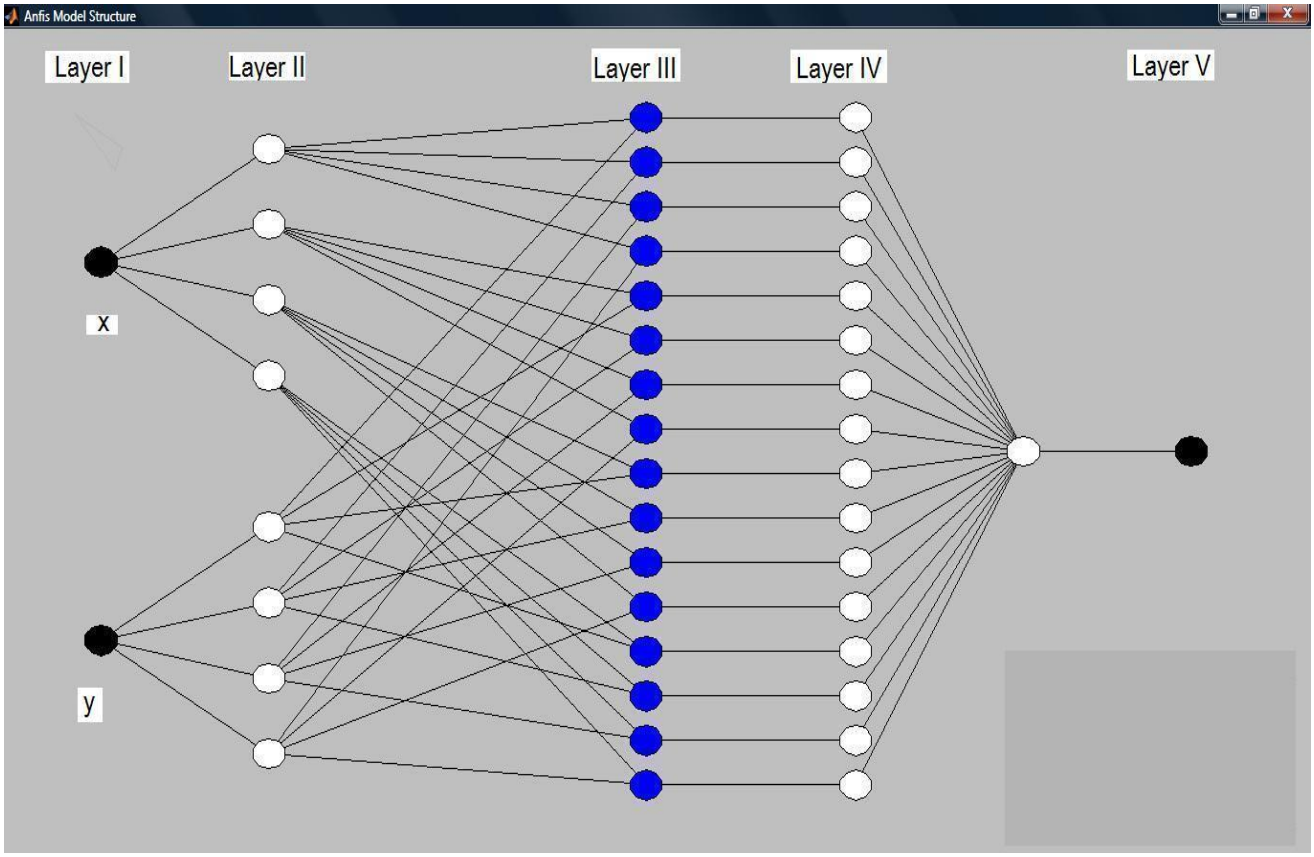


Fig 4.4 ANFIS model Structure

In order to examine the performance of the ANFIS network, extensive simulations were carried out to verify the functionality of proposed ANC system.

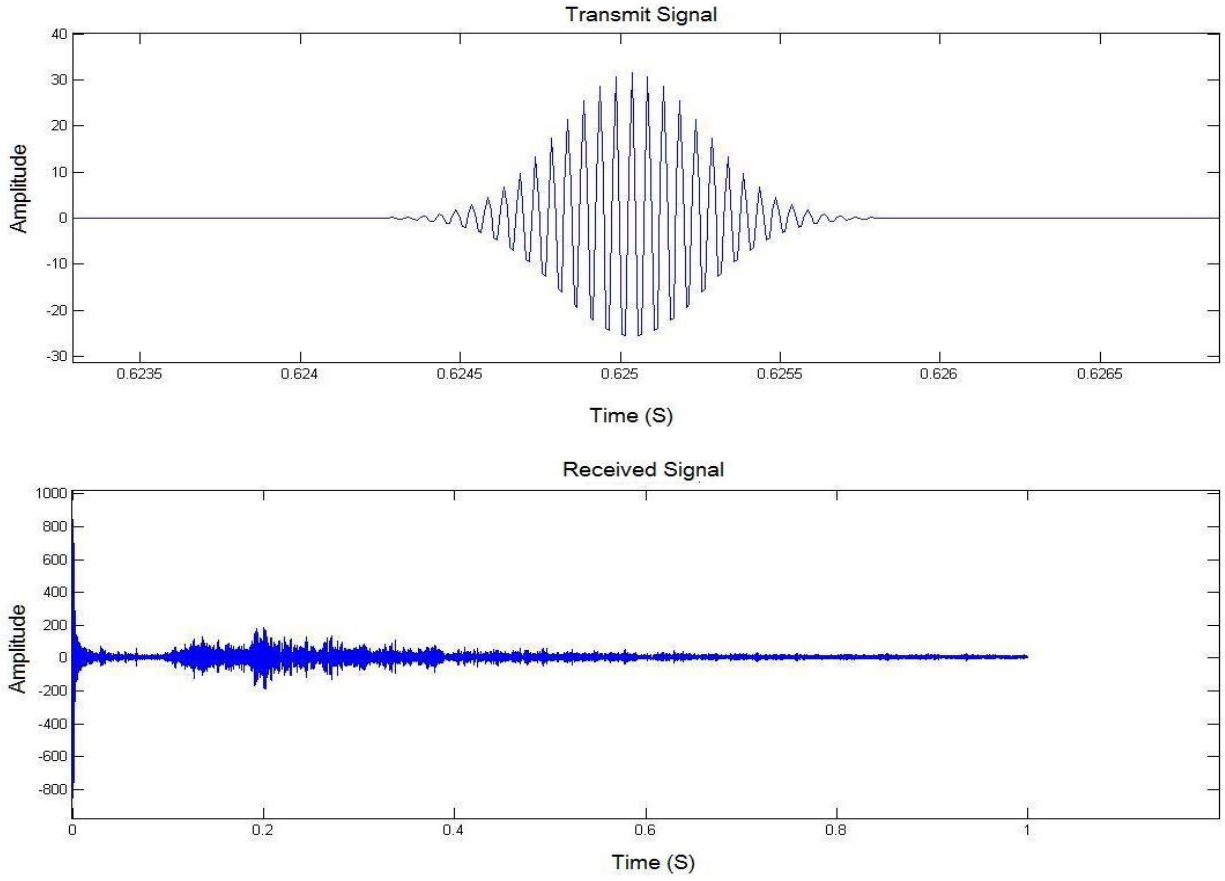


Figure 4.5. (a). Transmitted Signal (b) Received Signal

A 2ms long Morlet wavelet signal depicted in Fig. 4.5(a) is adopted as the transmitted signal. Fig. 4.5(b) illustrates the returned composite signal where the contact signal is buried in the noise with the target strength of approximately -24dB at the position of 0.8359s with velocity of 20 knots. Since the noise amplitude is much higher than contact signal, it is not possible to see contact signal in Fig 4.5 (b). The noise, which is provided by Defense Science and Technology Laboratory (DSTL) of Ministry of Defense(MOD)

to simulate the real underwater environment, is constituted by white Gaussian noise and reverberation waveform with the following environment settings: sea depth (=100m), sonar depth (=50m) wind speed (=6m/s ~ sea state 3), seabed type (=medium sand). DSTL provides two sets of data for simulation. One set is the received signals with different SNR (Signal Noise Ratio) at receiver end, which contain target echo, reverberation noises and ambient noises; and the other set is signals received at the receiver under the same environment with different SNR, but no target in the environment. The second set of data is used as input y for ANFIS model as reverberation noises in the proposed project. Since the reverberation noise is sampled without target in it, it would not have information of signals reflected from target, and it also would not have information of secondary reverberations such as reverberations from target to seabed to receiver or from seabed to target to receiver, etc. Therefore, besides the reflected target signal, it is unlikely for ANFIS to eliminate these secondary reverberation signals from received signals based on the reverberation noise given. However, as mentioned previously in introduction of reverberation modelling in Chapter II, simplifying assumptions are often necessary for reverberation modelling. These assumptions may seem ideal but it has been found practical in many sonar modelling and designs [11]. One of the assumptions is that reverberation produced by reverberation can be negligible. Energy absorption underwater may be the cause of this assumption. Once sound is reflected by target, seabed, water surface or even just travels in sea water, the energy will be absorbed. Therefore, the energy of multiple scattering (reverberation of reverberation)

will be much less than reverberation travels direct from target back to receiver. Therefore, even it is unlikely for ANFIS to remove secondary reverberations in the received signal; it would not undermine the performance of TME.

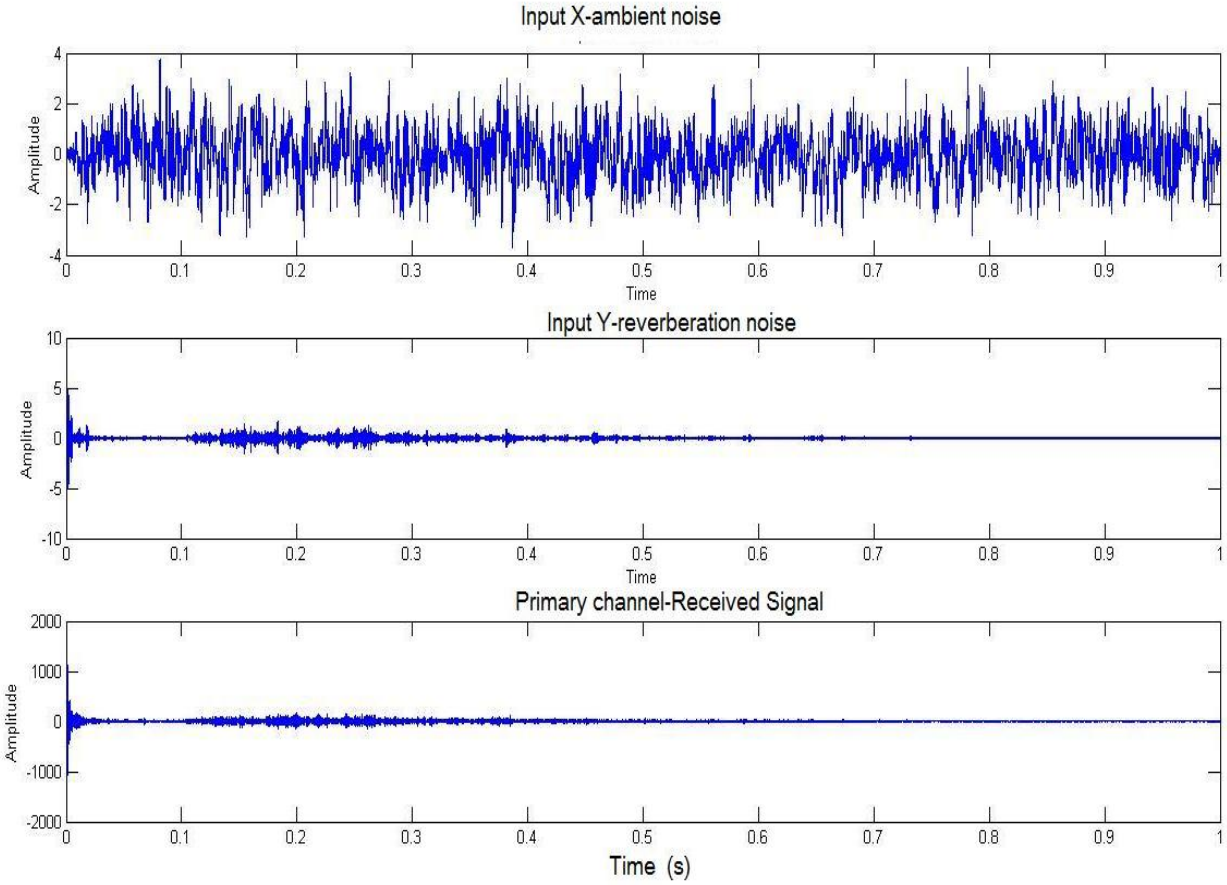


Fig 4.6. Input data for ANFIS

All these signals are sampled at 100 kHz in the maximum time range [0, 1] sec. Fig 4. 6 shows two input data sets, ambient noise and reverberation noise for ANFIS network, which are corresponding to two inputs x and y of ANFIS model shown in Fig 4.4. It also presents the primary channel shown in Fig 4.2 (a), which is the received signal. The received signal is used in ANFIS as training data. While Fig 4.7 illustrated what the signals like after ANFIS operation. Fig 4.7 (a) is the output of ANFIS block in Fig 4.1(a).

This signal is the estimated noise signal in the received echo as shown in Fig 4.5(b). By subtracting these estimated noises from the received echo the signal in Fig 4.7(b) shows the estimated target signals after the ANFIS operation.

The SNR of the signals in fig 4.7 (b) is -1.83 dB. Comparing the SNR before ANFIS –24dB and after ANFIS- -1.83dB, it is clear that the majority of noise is removed by ANFIS. However, the amplitude of noise is still much larger than target signal and in this case further processing such as wavelet transform is necessary. In the simulation the ANFIS runs 14 epochs to get the result.

Fig 4.8 shows the final version of membership functions for input 1shown as x in Fig 4.4 and input 2 shown as y in Fig 4.4. Comparing Fig 4.8 with the initial membership functions shown in Fig 4.3, it is obvious the parameters for the membership function formula in Eq. 4.2 have been modified during ANFIS operation. Especially in the membership functions for input 1, the changes of shape of the membership functions can be easily observed from the figures. The changes of shape of membership functions means the parameters of membership function, a and c in Eq 4.2 are modified during the ANFIS process by back-propagation and it indicates the ANFIS adjusts its filters to simulate the background noise according to the two input data sets.

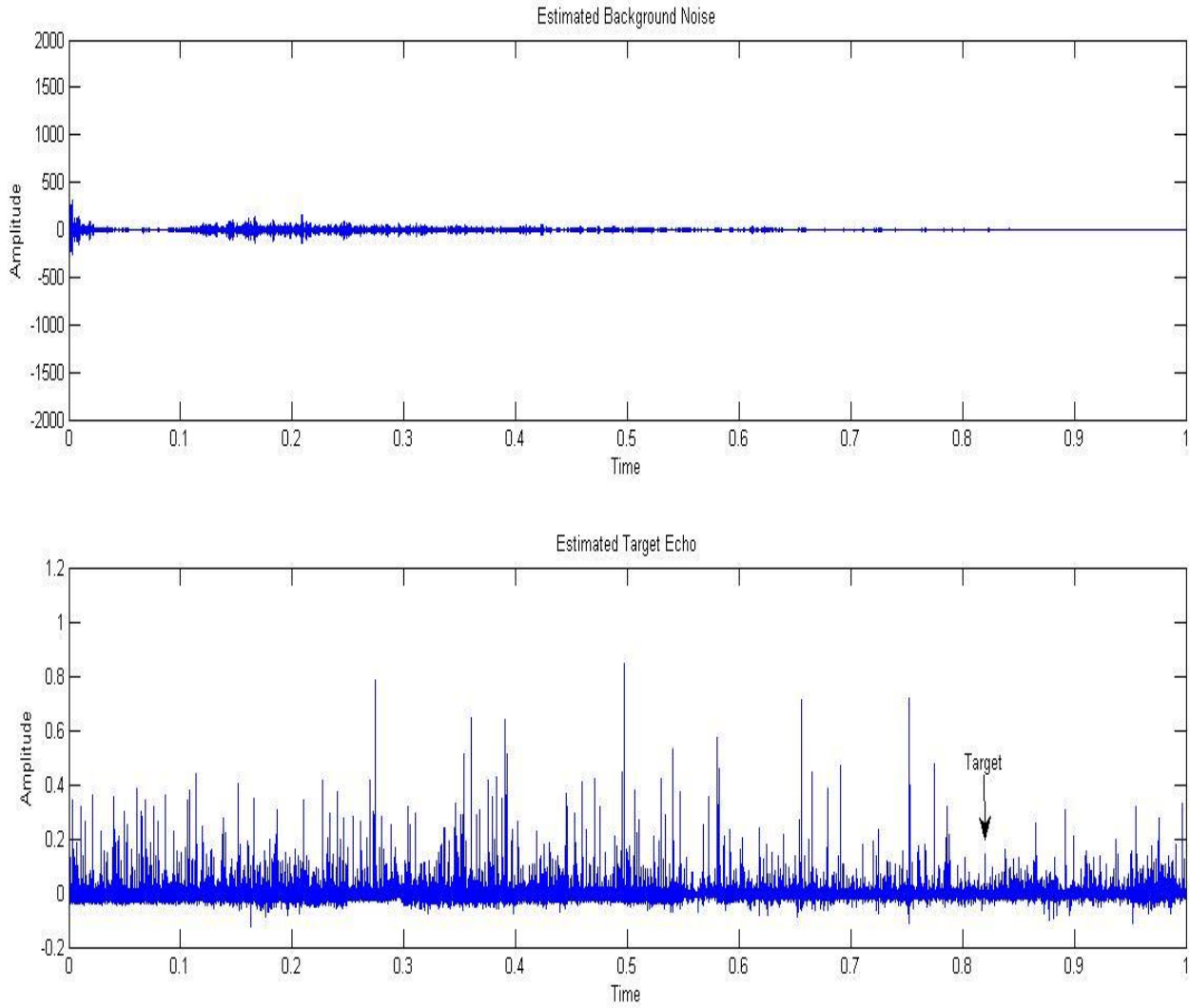


Fig 4.7 Output of ANFIS

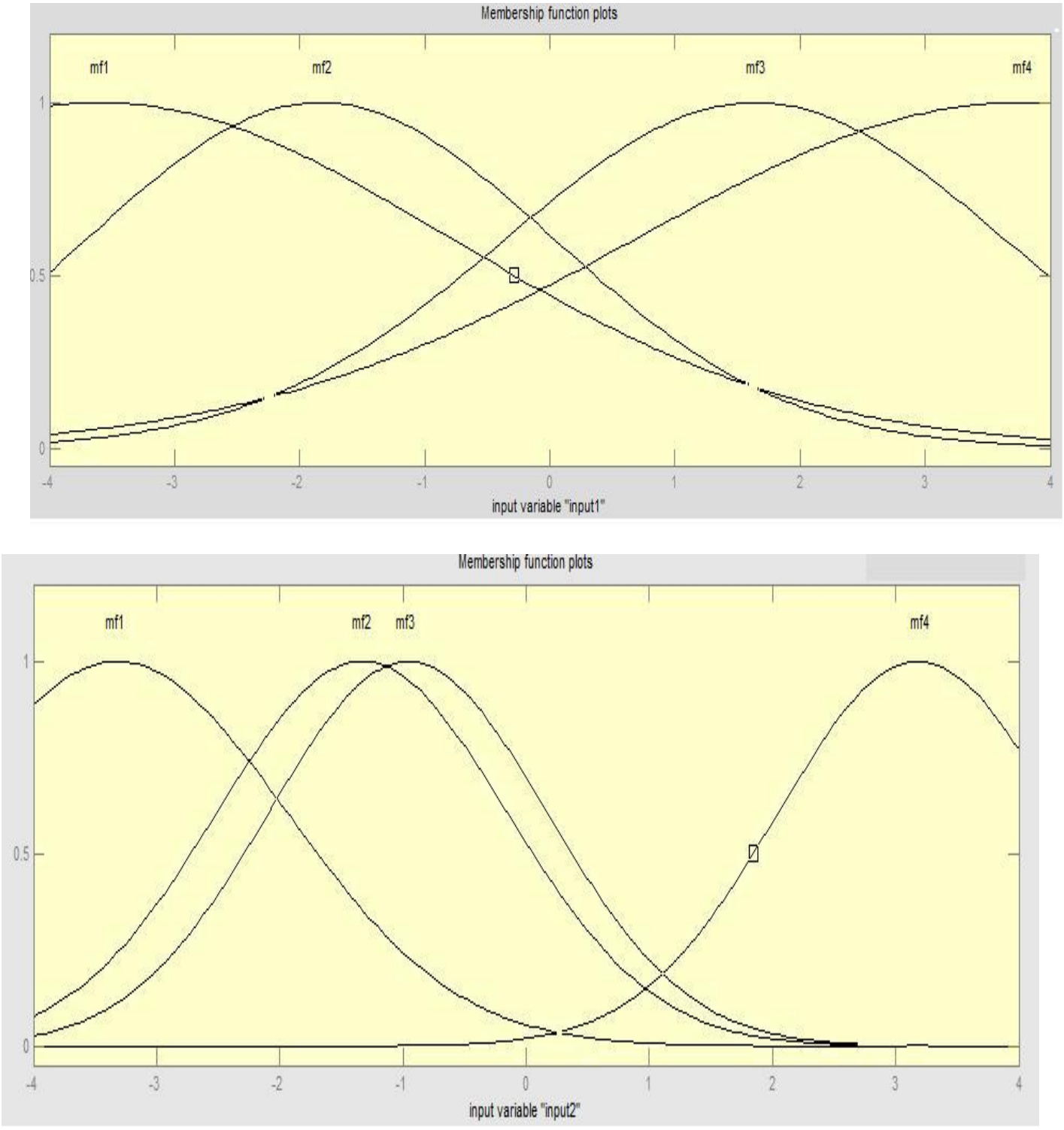


Fig 4.8 Membership Functions after ANFIS

As shown in Fig 4.1, interference noises are used as reference channels for ANFIS block and received signal is used as primary channel for ANFIS block. When looking into the ANFIS block diagram as shown in Fig 4.2(b) and Fig 4.4, the interference

noises are input signals for ANFIS and the received signal is used as training data. As mentioned in introduction of reverberation models in Chapter II, ambient noise and reverberation noises are jointly considered as background noises in active sonar system. Therefore, a set of randomly generated ambient noise is used as one input data set for ANFIS and one set of reverberation noises is used as the other input data for ANFIS, as shown in Fig 4.6. This set of reverberation noise has been given by DSTL simulating reverberations under the same environment as the received signal but with no target in it. As described in Chapter III, ANFIS adjust the weights of each input data through five layers and then combined the ambient noise and reverberation noise together through non-linear filters to try to simulate the background noise in the received signal, which is used as training data in ANFIS. The ANFIS model will compare the simulated output, f_k of forward pass of ANFIS as shown in Fig 4.2 (b), with the training data- received signal at receiver end in practice and shown as primary channel in Fig 4.1. According to the error rate from comparison results, parameters of non-linear filters are modified through back-propagation described in Chapter III to adjust non-linear filters. When back-propagation is over, the ANFIS tries again to simulate the background noise with two input reference noises in order to achieve better results. The process will be carried on repeatly until the error rate reaches satisfactory limit, which means ANFIS successfully simulates the background noise. By subtracting the simulated background noise from received signal, theoretically only target echo is left. Though reverberation noise used as input for ANFIS is sampled under the same environment as the received

signal, the background noise will not be the same due to the presence of the target in the received signal and possible different positions of the scatterers. Therefore, non-linear filters are needed to simulate background noises in received signal using reverberation noises. In real detection scenario the receiver samples signals all the time. Returned signals sampled before desired target appears can be used as the reference reverberation noise. Alternatively, simulated noise models defined by environment parameters such as seabed type, sea depth and wind speed, etc can also be used as one of the input for ANFIS. Fig 4.9 shows how ANFIS gradually improves SNR. Fig 4.9 (a) shows the estimated target echo after the first epoch, Fig 4.9 (b) shows the estimated target echo after 5 epochs and Fig 4.9 (c) shows the estimated target echo after 10 epochs. It is obvious the SNR performance of estimated echo improves gradually.

Besides this set of input data with an SNR of -24 dB, other versions of returned signals with different SNR ranging from 0dB to -30dB have also been simulated through the proposed ANFIS. Fig 4.10 presents the results for simulations under different SNR. The results indicate the proposed system performs efficiently to minimize the noise effect in the returned signals by reducing SNR from -30dB to -2dB. However, since the input data X and Y used in ANFIS model are theoretical ambient noise and experimental reverberation noise, both of which would not be exactly the same as the composition of background noise in received signal. Therefore, it is hard for ANFIS to proceed further. Experiments have been done on letting ANFIS run for 1000 epochs where the SNR was still -1.8dB. In this case other digital signal processing

algorithms in TME block are introduced to further minimize the noise effect and try to estimate the target parameters.

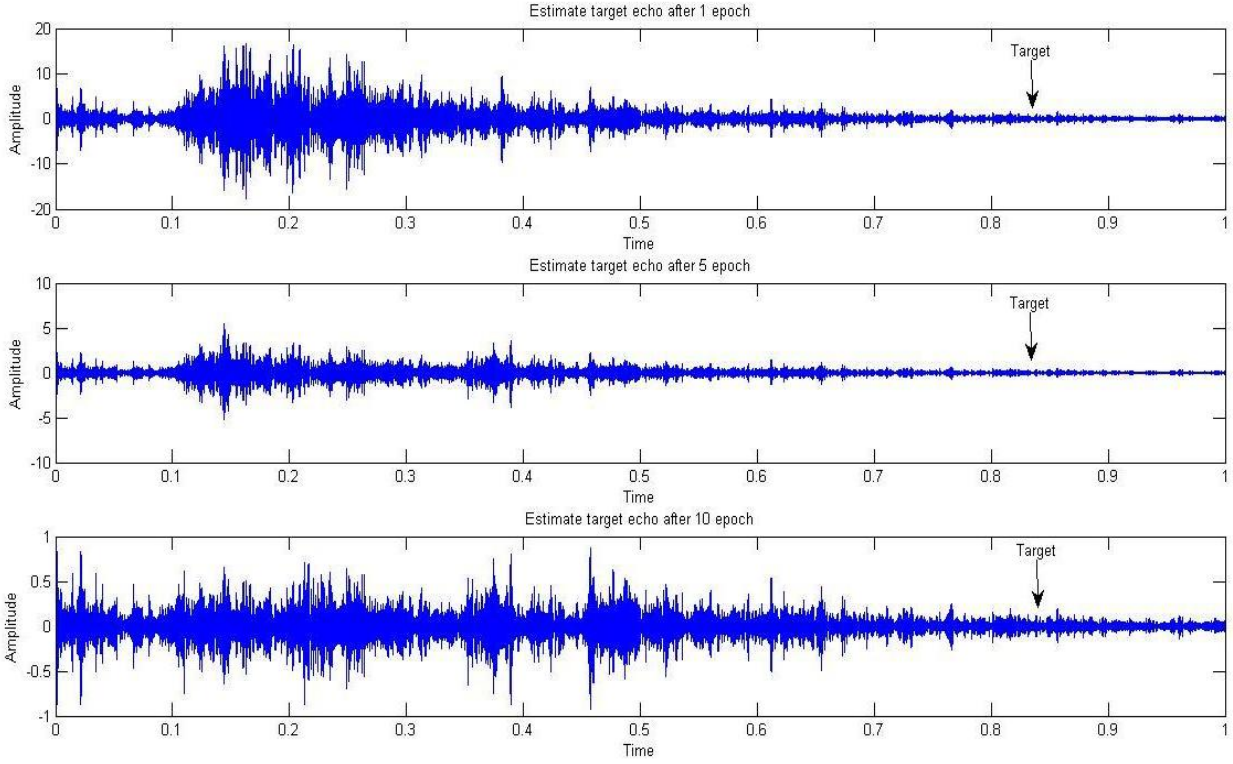


Fig 4.9 Different Stages during ANFIS operation

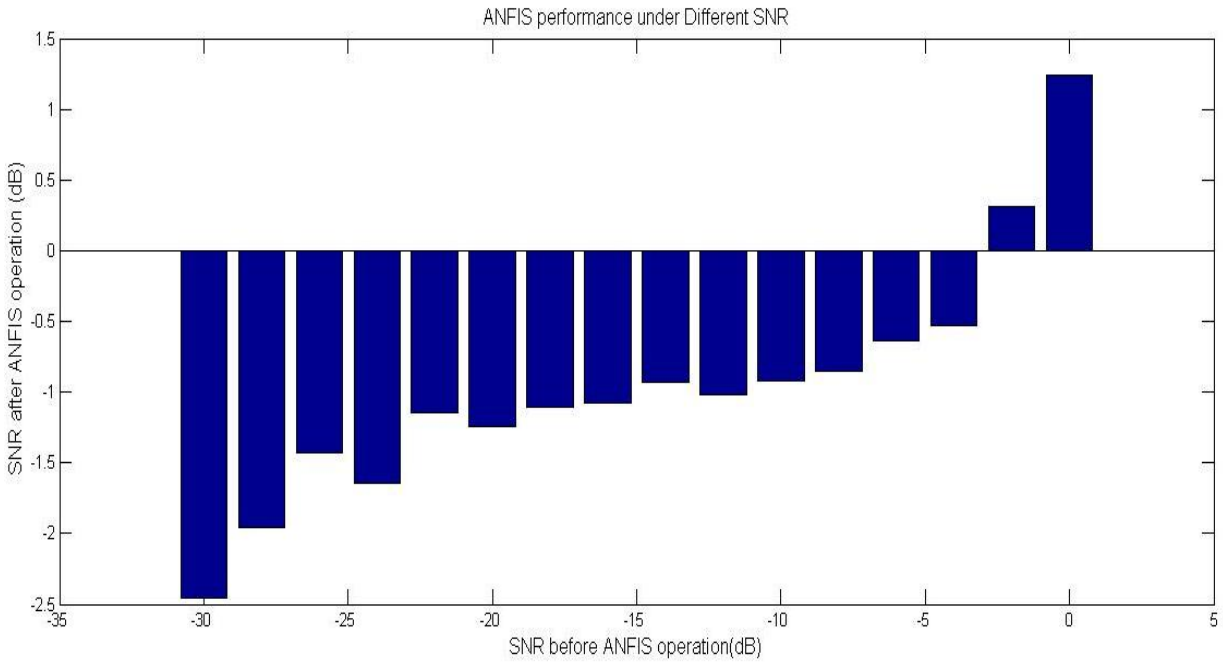


Fig4.10 ANFIS performance

4.4 TME algorithm

As shown in Fig 4.1, the TME block contains three major functional parts: DWT denoising block; Optimization block and CWT block. The DWT denoising block is introduced to further reduce noise in the estimated signals. It uses the discrete wavelet transform to decompose signals into different octaves and remove those coefficients which are lower than a set threshold. In this step, the wavelet decomposition is implemented as a lowpass-highpass filter pair (band pass filtering structure) [4, 5]. The low-pass filter yields approximation (or scaling) coefficients, while the high-pass filter produces detail (or wavelet) coefficients [4]. More specifically, an orthogonal mother wavelet is chosen to decompose the noisy data set at the scale N. This has resulted in N + 1 sets of coefficients including N sets of detail coefficients and one set of approximation coefficients. The approximation coefficients contain the basic structure (or information) about the signal, while the detail coefficients contain sharp transition details and the noise components [3]. If the sampled signal is of length n, the DWT decomposition will consist of $\log_2$ (n) scales at the most. Increasing scale not only makes the signals more correlated but also can suppress the noise to a greater extent in the frequency domain. The drawback is that as scale increases, the discrete time resolution is halved at each scale due to subsampling. The resolution in the time domain will be less and less accurate. Also increasing scaling will lead to increasing computation load and extra storage requirements for partial results. The Daubechies wavelet [5] of order 20 was used to perform the DWT decomposition to N = 10 levels.

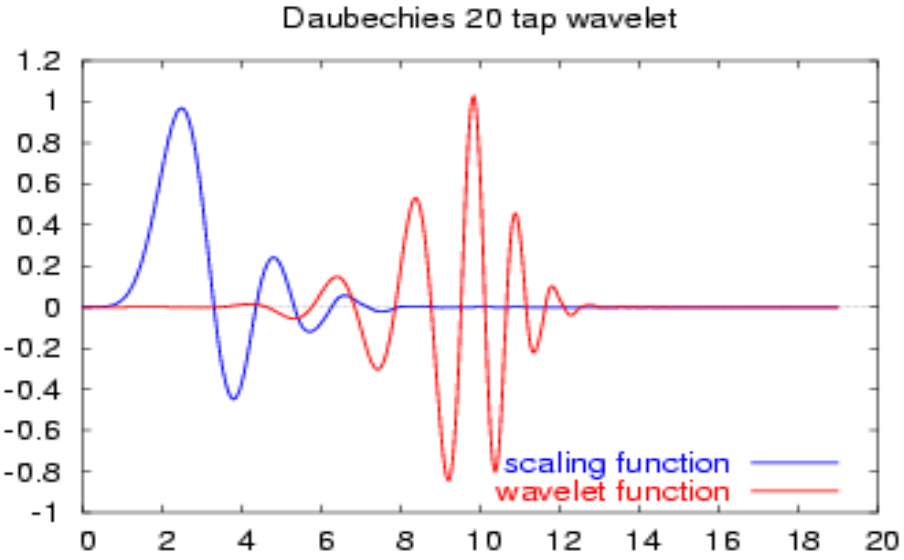


Fig 4.11 Daubechies wavelet DB-20

Since the detail coefficients contain noise components, one way to remove these unwanted details is to omit them if they are small enough. For example, to set for all coefficients those are less than a particular threshold to zero. Thresholding wavelet coefficients can use hard or soft thresholding rules [6]. The hard threshold rule can be described as the usual process of setting to zero the elements whose absolute values are lower than the threshold. On the other hand, the soft threshold rule is an extension of the hard thresholding by shrinking all the coefficients towards the origin. In the proposed work, the soft threshold rule is adopted with a universal fixed form threshold $\lambda = \sqrt{2\log(n)}$ according to [6] where n is the sample size of the signal. When all the approximation coefficients are processed by the thresholding algorithm, these post-processed coefficients are reconstructed through the IDWT as described in Chapter III.

As described in the previous chapters, the idea provided for the target localization and

identification is straightforward and commonly known as correlation processing [1, 7], which cross correlates the received signal with a set of references that are hypothesized replicas of the incident signal. The hypothesized signals technically serve as template functions or basis functions to which the received signal is matched and results in high correlation. As the template function is shifted in time over the whole signal and consecutive correlations are performed, the highest correlation is achieved, which provides an estimate of the desired parameters of the received signal. This type of similarity measurement between the known signal and the basic functions is a CWT when wavelets are defined as basis function [7, 8, 9]. Due to the similarity of the wideband sonar model and continuous wavelet transform representation, the seeking of the highest correlation coefficients leads to the seeking of the scaling functions of the CWT. The optimization block will choose the scaling factor for CWT block and then the CWT block will get the highest correlation coefficient at the current scale and compare it with the previous ones. The optimization block will decide which scale suits the returned target echo signal most and theoretically the signal which the CWT coefficients represent at this scale has the same frequency information as the returned target and thus the velocity parameter can be determined by Eq 3.3 in Chapter III. Also by determining the time-domain information of the highest coefficient the location of the target can be extracted by Eq 3.2 in Chapter III. Since this system is proposed to detect a target underwater, and the speed of objects underwater do not exceed 15 m/s [10], which is 30 knots, it is safe to arrange the searching area for local optimum algorithm search from

scales corresponding from -60 knots to 60 knots, where -60 knots means the target is moving toward the observer at 60 knots and 60 knots means the target moves away from the observer at 60 knots.

With the output signals of the ANFIS operation as the stimulus for TME block, the performance and the functionality of TME is examined in Matlab.

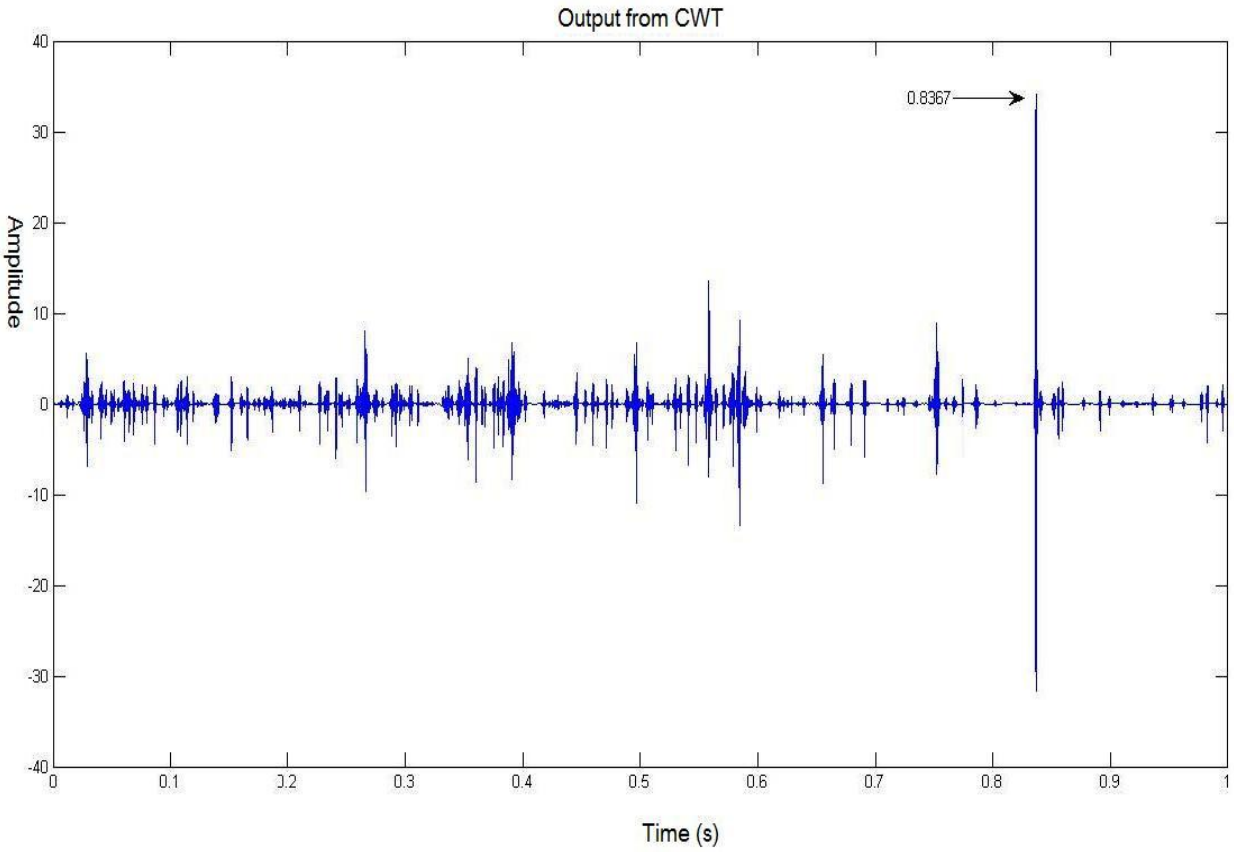


Fig 4.12 Output from CWT

Input signal into the TME block is shown in Fig 4.7(b), while the output signal is as shown in Fig 4.12. From the Figure 4.12 it is clear that the highest correlation coefficient appears at 0.8367s,and this represents the target, which only has 0.0008s error in time domain. Since the sonar travels at around 1500m/s underwater, the error of round-travel time can only cause a distance error of around 0.6m. Also according to the

scaling information, the speed is 19.374 knots, which is only less than 1 knot away than the actual velocity.

To characterize the performance of the proposed system, simulations are performed using 16 different sets of input data with different SNR ranging from 0dB to -30dB. Fig 4.13 shows the location error rate and the speed error rate under different SNR conditions. The error rate is calculated by comparing the measurement error against the total range and velocity. It is clear that the location error rate is less than 0.6% and the speed error is less than 4%. In Fig 4.13 it can be observed that at some point, when SNR increases, the performance of TME decreases. Theoretically when SNR increases, it means the noise is getting weaker, and the performance should be increased. However, since the chaotic nature of reverberation noise, it is possible that even the reverberation noise is weaker; it may cause more interference on target detecting because the frequency is closer to the target signal. By observing the trend of performance in Fig 4.13, in general when the SNR decreases, it is more possible for the error to increase and the trend fits the theory. Therefore, it is safe to conclude, based on Matlab simulations, the proposed system can fulfil the specifications for this project for single target detection.

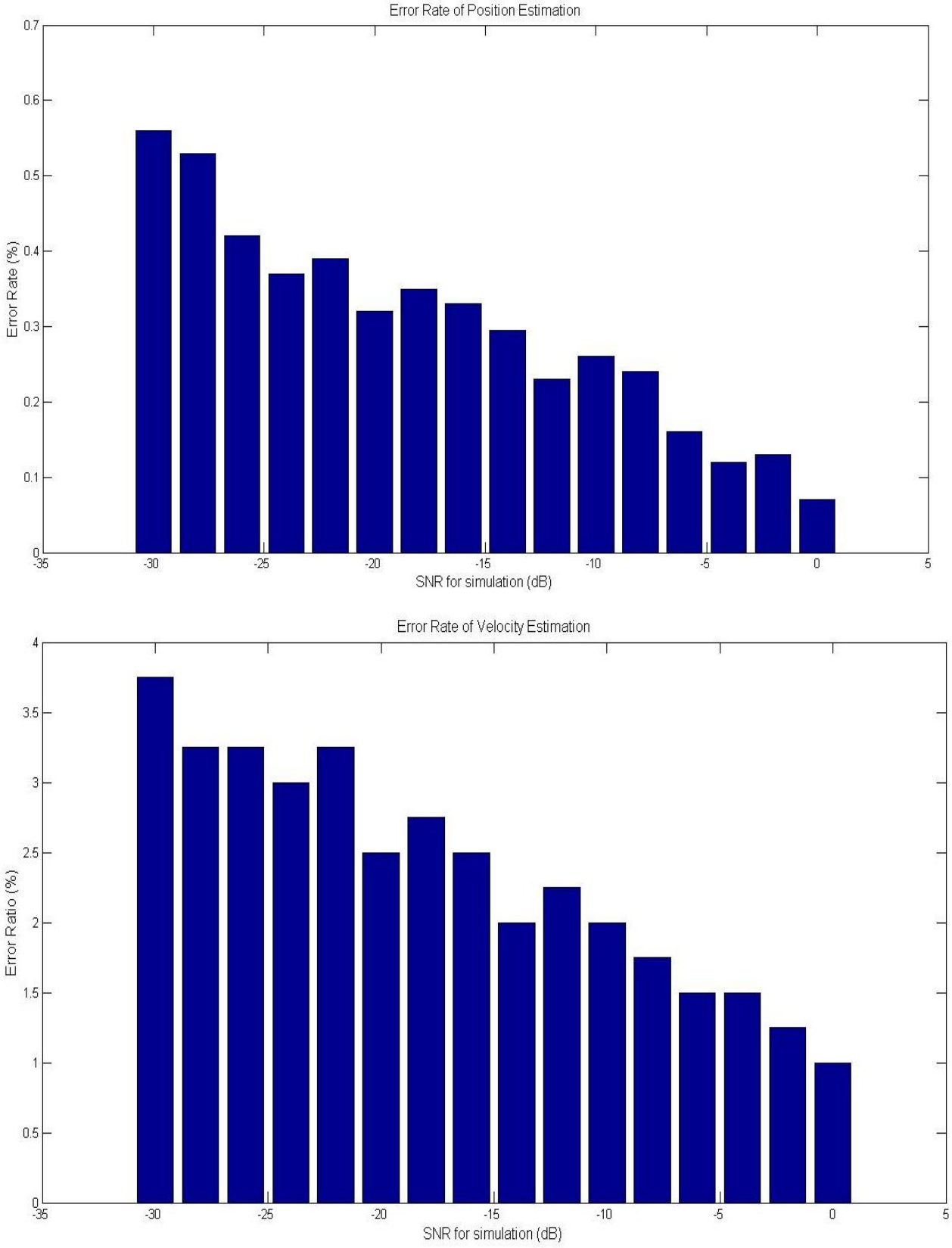


Fig 4.13 Location & Speed Error Rate for Single Target

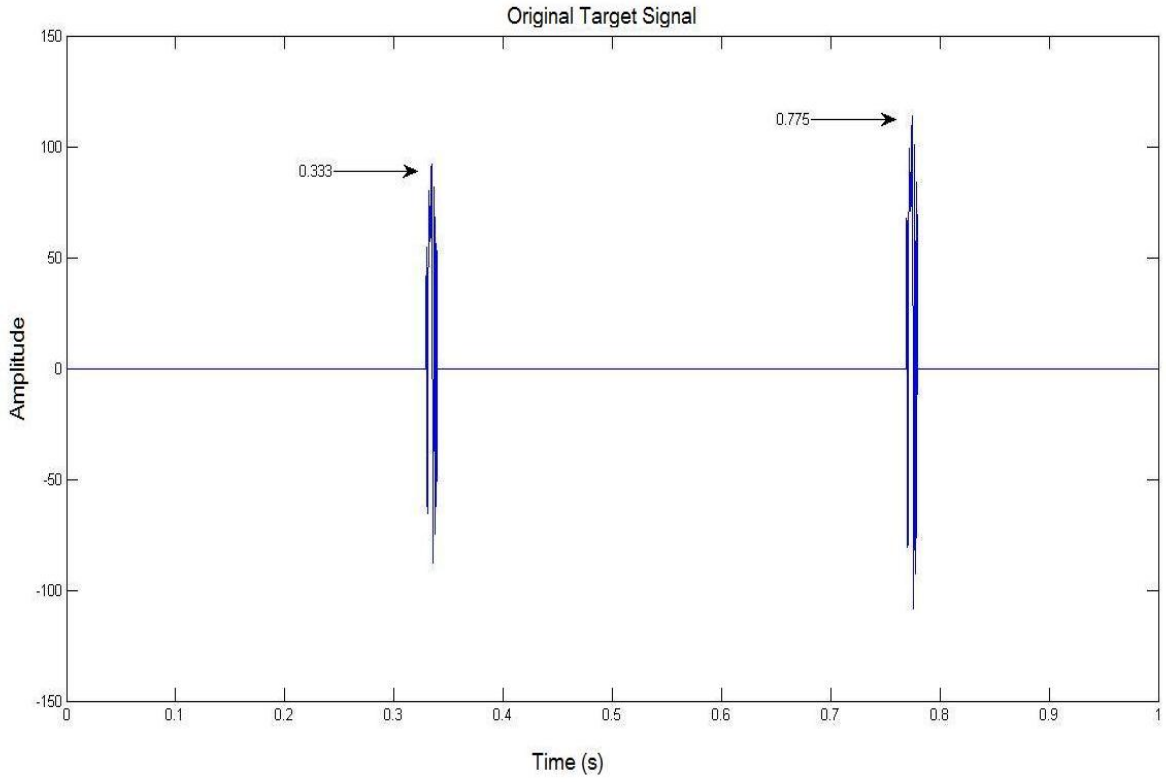


Fig 4.14 Target Signals

Simulation for multiple target detection was also carried out. As shown in Fig 4.14, there are two target signals at 0.333s and 0.775s. The target signals are buried in a combination of reverberation and white Gaussian noises with a SNR of -20dB. Fig 4.15 shows the 2 input data sets, ambient noise and reverberation noise, as input X and input Y for ANFIS block to simulate the background noise, and one training data, which is the received signal at receiver shown in Fig 4.1 (a) as primary channel, for ANFIS block to train and validate results. Fig 4.16 gives the estimated noises from the ANFIS block and the processed returned echo after processing by the ANC part.

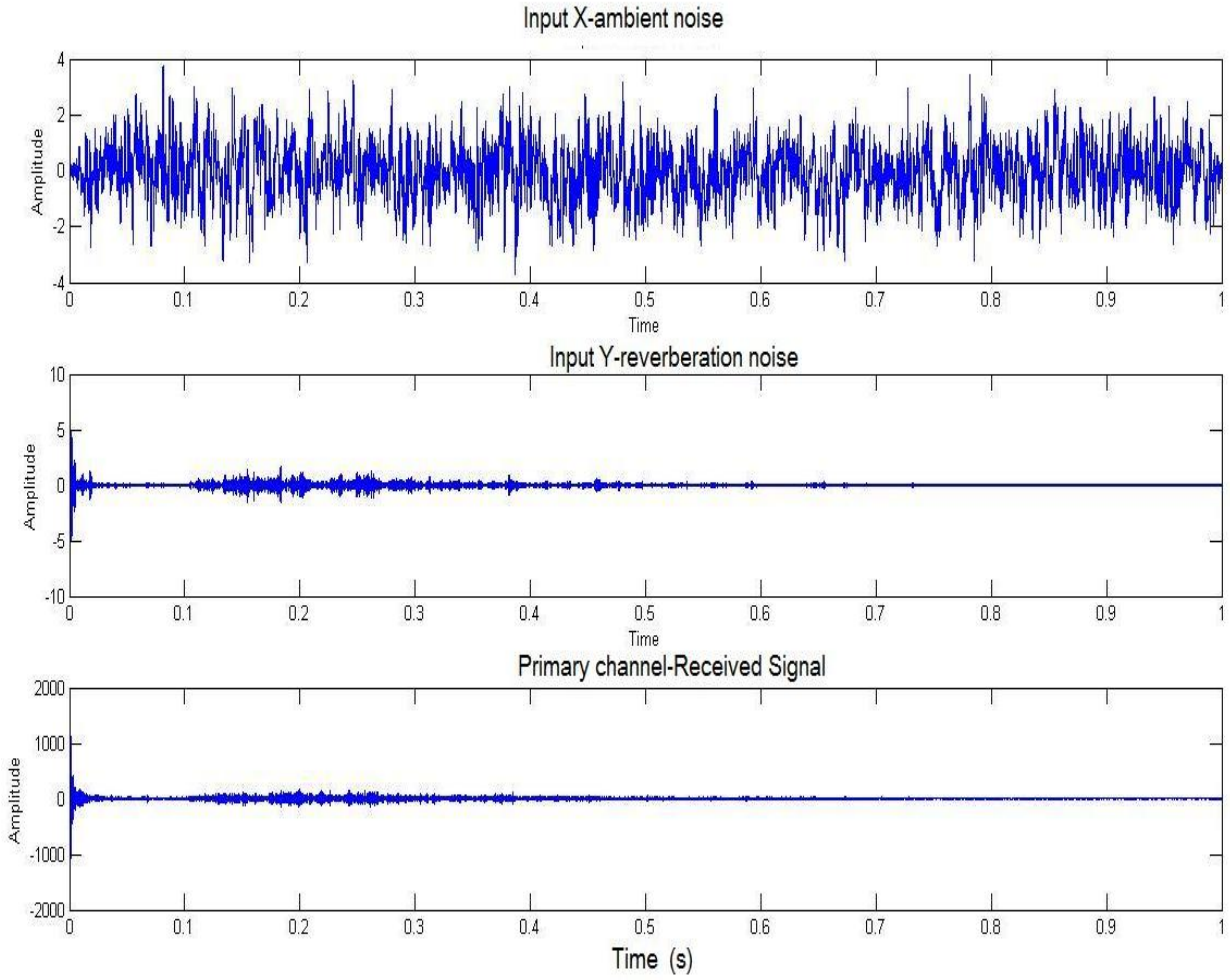


Fig 4.15 Training Data for ANFIS

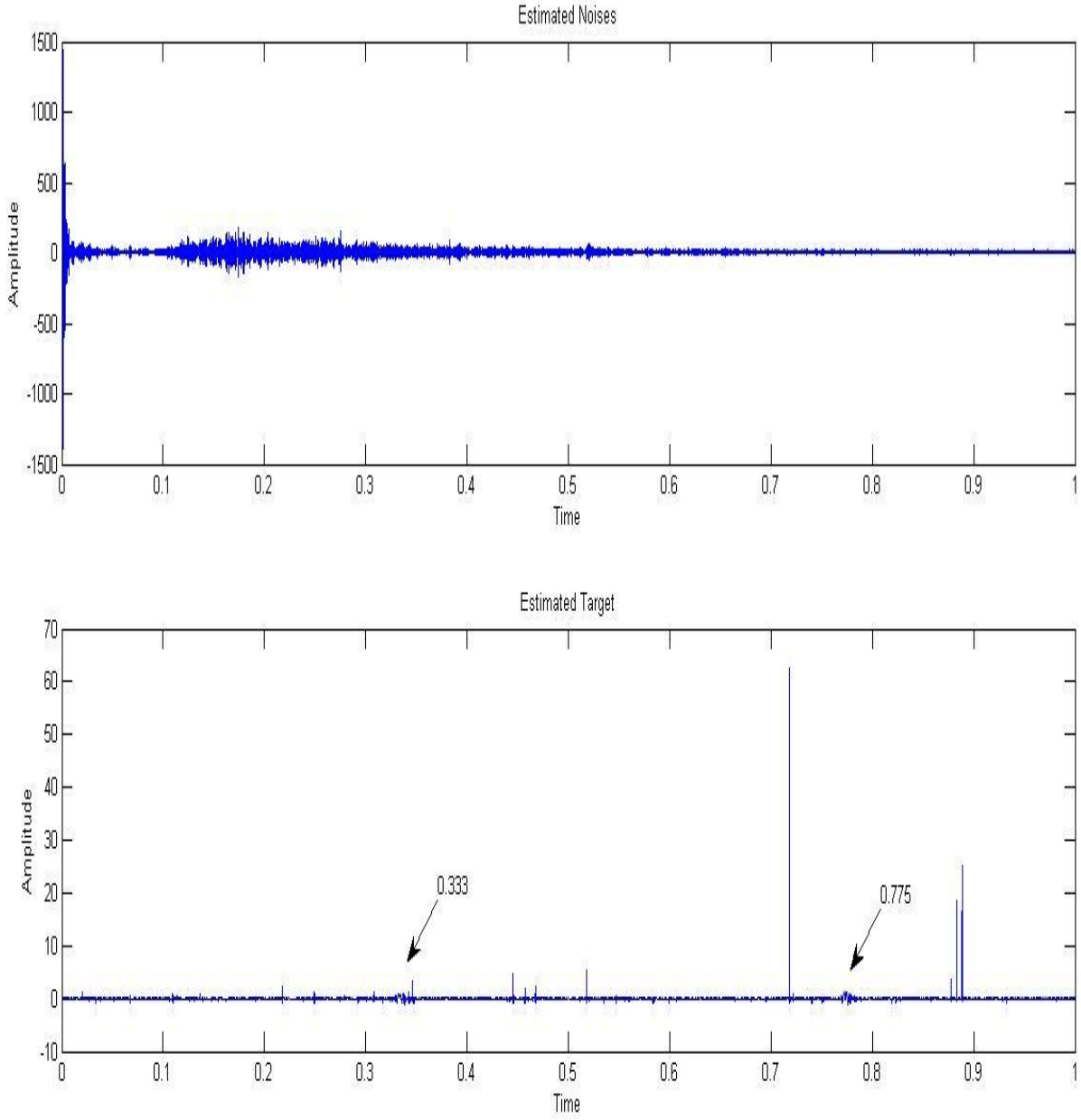


Fig 16 Output from ANFIS

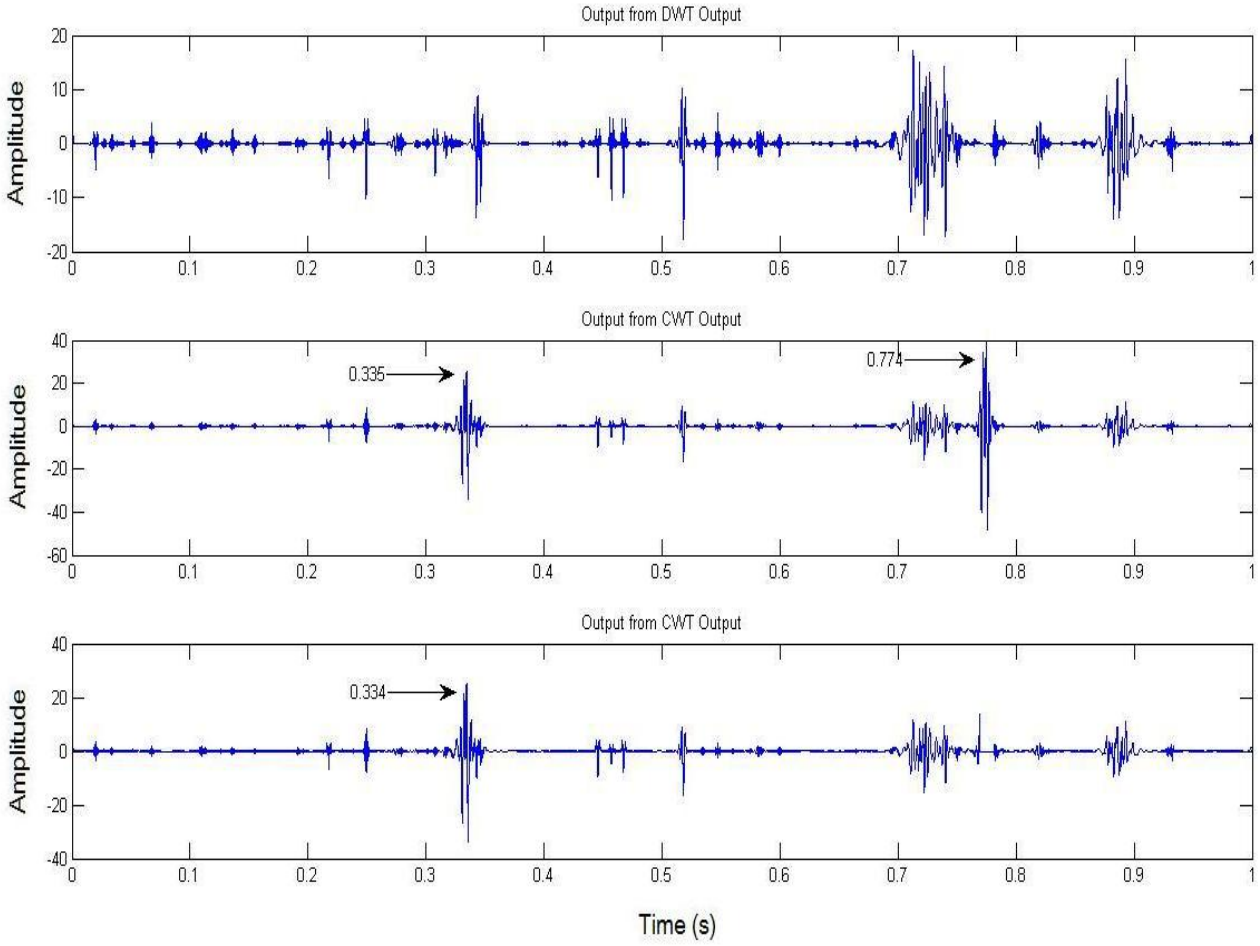
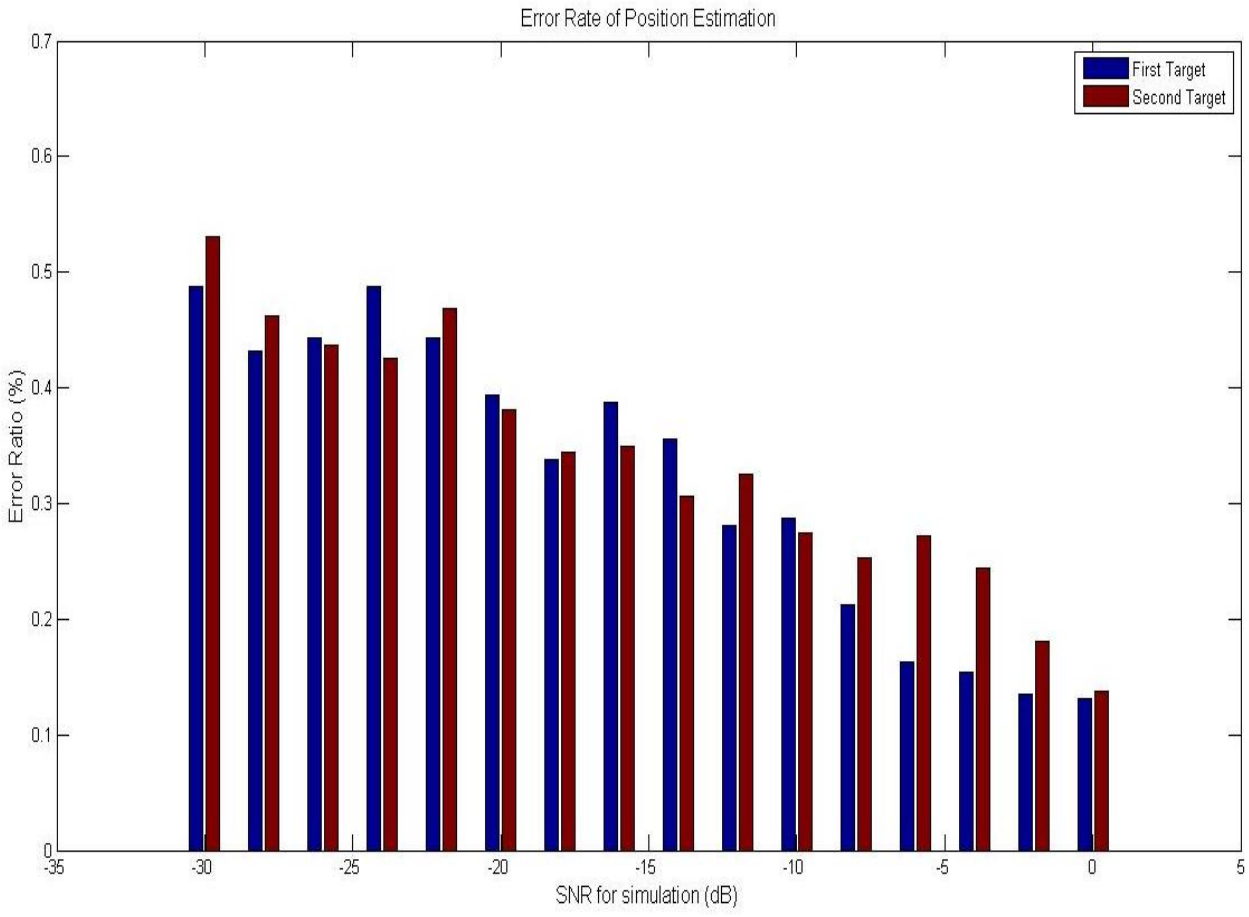


Fig 4.17 Output from TME

Fig 4.17 shows the output after the DWT block and the final coefficients from CWT block. After the first TME operation, two target signals are extracted at 0.336s and 0.774s as shown in the middle figure of Fig 4.17. It is obvious the signal at 0.774s has a stronger response with the TME operation and according to the process of the proposed system signal at 0.774s is determined as the first target. Then this signal is removed so that the system can examine other targets without influence from the first one. As shown in the bottom figure in Fig 4.17, after removing the first signal at 0.774s, the signal at 0.334s gives the strongest response for the TME operation. Therefore the

signal at 0.334s is recorded as the second target. Based on calculation of Eq 3.2 and 3.3 in Chapter III, the location error is less than 1 meter and the speed error is less than 1 knot, in terms of error ratio against total range, the velocity error ratio is less than 4% and location error ratio is less than 1%. Fig 4.18 gives all the results from the different SNR ranges from 0 dB to -30dB. From the figure it is clear that under different SNR situation range from 0dB to -30dB the speed error rate is less than 4.5% and the location error is less than 0.6%. Therefore based on these simulations the proposed system is able to work in a severe noisy underwater environment and give a satisfactory performance.



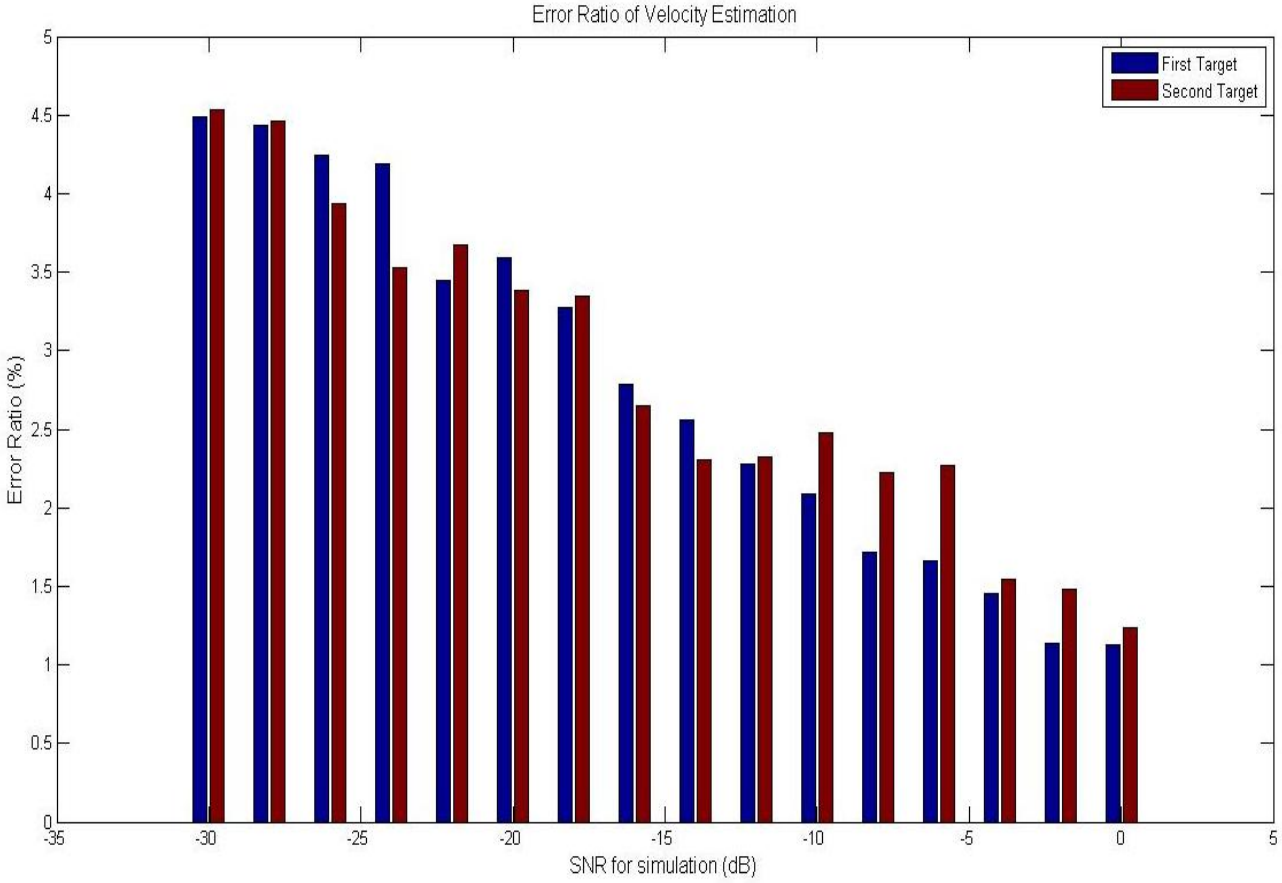


Fig 4.18 Speed Error & Location Error from 0dB to -30dB for multiple target detection

4.5 Conclusion

In this chapter the high level system architecture is described first. Following the implementation details of the ANFIS network, the design considerations of the TME block are also explained. In order to verify the functionality and the performance for each block and the whole proposed system, Matlab simulations for single and multiple target detection under different SNR environments are carried out and the results are presented. The noise which simulates the real underwater environment was provided by MOD amid the following environment settings: sea depth (=100m), sonar depth (=50m) wind speed (=6m/s ~ sea state 3), seabed type (=medium sand). The system is tested under this environment with SNR ranging from 0 to -30dB. The results for single target detection and multiple target detection are satisfactory are shown above and the proposed system gives a satisfactory performance by locating the target accurately with speed error within 1 knots and location error within 2 meter in all simulations, and the speed error rate against total range is less than 5% and location error rate against total range is less than 0.6%. The simulation results explain the system's ability to fulfil the specifications for the project as mentioned in Chapter III.

References

1. L. G. Weiss, “Wideband processing of acoustic signals using wavelet transforms. Part I. Theory”, The Journal of the Acoustical Society of America, vol. 96, no. 2, pp. 850-856, 1994.

2. M. Sugeno, “Industrial applications of fuzzy control”, Elsevier Science Pub. Co, 1985.

3. J-S R. Jang, “ANFIS: adaptive-network-based fuzzy inference systems,” IEEE Tran. Sys. Man, Cybernetics, vol. 23, pp. 665-685, 1993.

4. C. H. Tseng, “Application of adaptive neuro-fuzzy inference systems to active wideband signal detection in a reverberation-limited environment-Part II: Dense target environment”, Research Report, School of Engineering, University of Warwick

5. I. Daubechies, “The wavelet transform, time-frequency localization and signal analysis,” IEEE Trans. Inform.Theory, vol. 36, pp. 961-1005, 1990.

6. D. L. Donoho, “De-Noising by soft-thresholding,” IEEE Trans. on Inf. Theory, vol. 41, 3, pp. 613-627, 1995.

7. O. Rioul and M. Vetterli, “Wavelets and signal processing”, IEEE Signal Processing Magazine, pp. 14-38, October 1991

8. A. Grossmann, R. Kronland-Martinet, and J. Morlet, “Reading and understanding continuous wavelet transforms,” Int. Conf. Marseille, France, pp. 2-20, 1989.

9. P. M. Bentley and J. McDonnell, “Wavelet transforms: an introduction,” Electronics & Communications Engineering Journal, vol. 6, pp. 175-186, 1994.

10. M Arumemi-Ikhide, “Investigation into the Use of Wavelets in Broadband Active Sonar Signal Processing”, Master Thesis, University of Warwick, UK, 2001

11. Paul C. Etter, “Underwater acoustic modeling and simulation”, Spons Architecture Price Book, 2003

CHAPTER V

HARDWARE IMPLEMENTATION

5.1 Introduction

This chapter gives the details of the hardware implementation of the proposed broadband sonar system. As described in the previous chapter, the system is composed of two major components: Adaptive Noise Canceller (ANC) and Target Motion Estimation (TME) component. This chapter will introduce the implementation of both functions respectively. The simulation results of the proposed system will be presented to justify the design choices.

5.2 Hardware Structure

The system is composed of two major parts: Adaptive Noise Canceller (ANC) with Adaptive Neuro-Fuzzy Inference System (ANFIS) and Target Motion Estimation (TME) block using continuous wavelet transform as described in Chapter IV. The system is illustrated in Fig 5.1. The hardware platform used in this project is the Xilinx XUP Virtex II Pro Development System. It contains a Virtex II pro XC2VP30 FPGA chip with 2 powerPC 405 cores. The board provides a comprehensive collection of peripheral components that can be used to communicate with PCs. The overview of the hardware platform is shown in Fig 5.2.

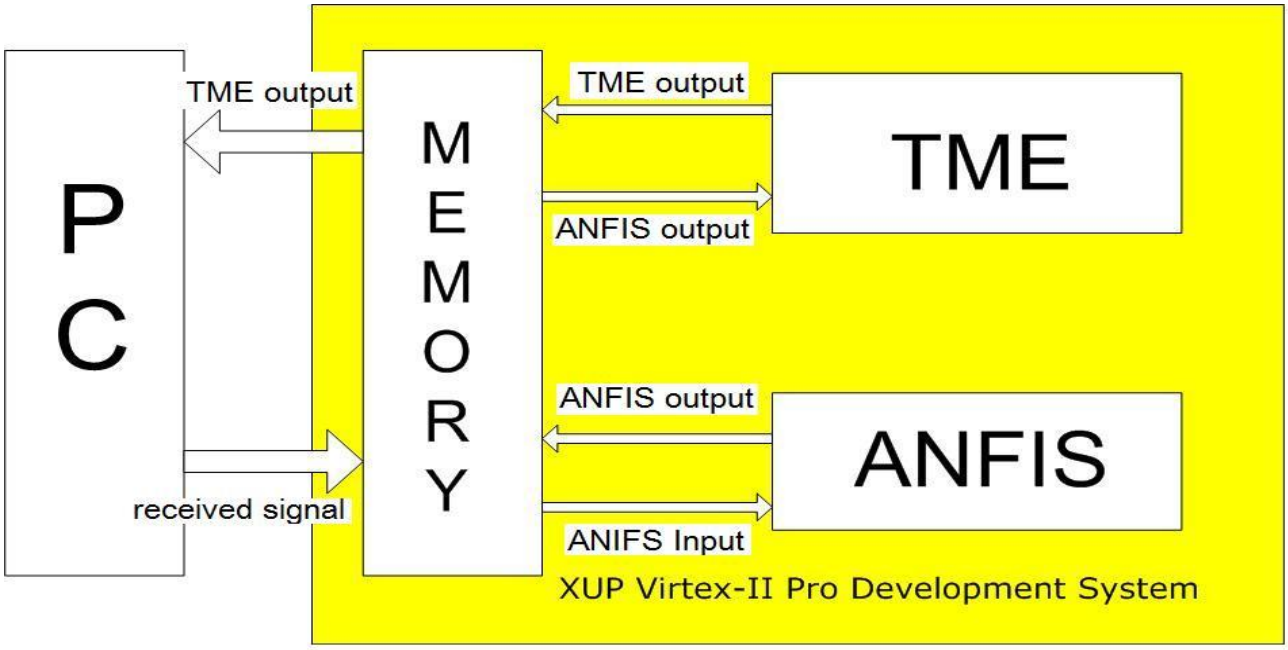


Fig 5.1 Hardware Implementation Block Diagram

As shown in Fig 5.1, the received signals are fed into the system from the PC through a serial port RS-232 to the 256MB DDR RAM on the test board. The ANFIS will perform a noise reduction process as described in Chapter IV on the received signal and then

feed it back to the memory, which in turn will feed it to the TME block for further processing and final detection of the target motion parameters using Continuous Wavelet Transform is performed. After all the processes are finished, the PC will fetch the processed data from the memory and stored it.

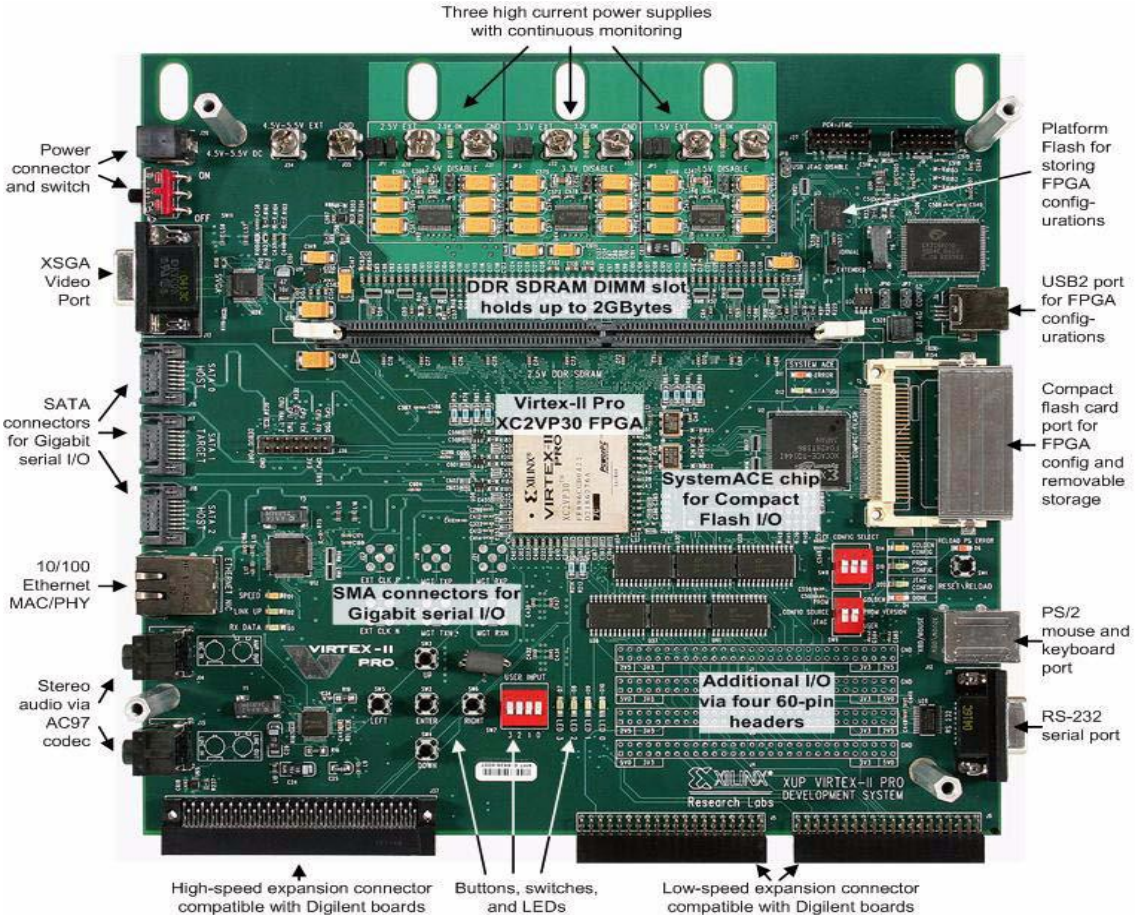


Figure 5.2: XUP Virtex-II Pro Development System Board Photo [6]

5.3 ANFIS Implementation

The purpose of introducing the ANC into the system is to reduce the noise in the received signals and provide a better environment for TME to extract target parameters. Adaptive Neuro-Fuzzy Inference System (ANFIS) is a combination of neural network and fuzzy logic techniques. As described in chapter IV, the ANFIS operation is designed to find a

model or mapping that correctly associates the stipulated input-output pairs by combining a hybrid learning procedure of a neural network with the reasoning capacity of fuzzy logic. Based on Sugeno's fuzzy if-then rule [1], an output of the ANFIS can be obtained from a given input training data set being weighted by the firing strength of the rule. Taking advantage of the ANFIS operation, the fuzzy detector is able to track both the nonlinear and linear relations among signals without any prior knowledge of the noise waveform. In the proposed system the ANFIS is implemented in C code in XILINX EDK 10.1 and the embedded DSP chip-POWERPC 405 is used to calculate all parameters.

As shown in Chapter III, there are five layers for the input signals to go through in ANFIS. Fig 5.3 is the structure of the proposed ANFIS as described in Chapter IV.

In order to simplify the design complexity and reduce ANFIS iteration time, a typical Takagi-Sugeno type network [1] ANFIS with 2 inputs with 4 membership functions for each input are adopted in this proposed system, as shown in Fig 5.3. In Fig 5.3 a square node (adaptive node) has parameters while a circle node (fixed node) has none.

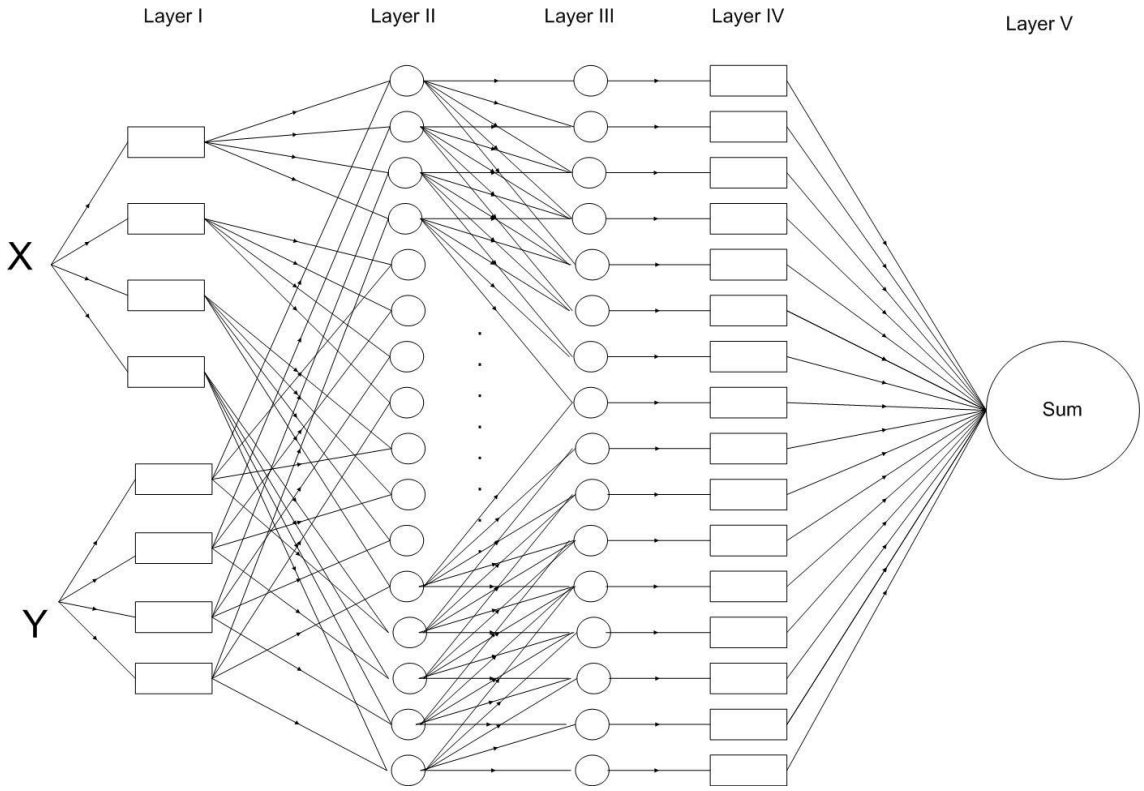


Fig 5.3 Fuzzy Inference System and ANFIS Structure

As described in Chapter III, the two inputs are two sets of noise data and membership functions are Gaussian functions with the formula as shown below [1]:

$$\mu(\mathrm{x}) = \exp[-(\frac{x-c}{a})^2] \qquad (5.1)$$

To implement this formula, several divide and several square operations are required to get a result for one input data, which is more expensive in terms of computation time than add and multiply operation. For example, for PowerPC 405, it only takes 4 cycles to multiply but 35 cycles to divide operation. For the proposed system, there will be 100k input data for a training set, and this computation time for only membership functions is not affordable. Therefore, an alternative method is adopted in our system. Each membership function, with different parameters of a and c will be discretely sampled and the values of the membership functions will be stored as a lookup table in

the on-board DDR RAM. Assuming the resolution of the input data is 0.001 and the range of input data is from -4 to 4, it only needs an 8k storage unit for one membership function. In order to simplify the addressing problem for a PC to fetch coefficients from RAM, each value of the membership function is represented with 8 bit 2's complement format in the hardware implementation. Therefore, only 8k bytes is needed for implementing one membership function with a certain a and c. Since the RAM is 256MB, there is enough room for storage of the membership functions. When the input data is fed in, the system will look up the corresponding value according to input data's value in the look-up table and will automatically give the output. In this way the computation time for membership values is significantly reduced to only several cycles of data communication from the RAM and the chip.

When all membership function values are given, the second layer multiplies the output of the selected membership function and the third layer will normalize those weights. As stated in Chapter III, it is easy to calculate the number of rules from the network scheme, as each rule represents a combination of membership functions, one from each input. There will be F^N rules, where F is the number of membership functions of each input and N is the number of inputs. Since there are 2 inputs for the ANFIS system and 4 membership functions for each of the inputs, there will be 16 fuzzy rules and therefore there will be 16 nodes in layer 2 and layer 3. All weights in Layer 2 and 3 are stored as an 8-bit signed integer and 16 byte buffer for storage of weights in Layer 2 and 3 are created.

After all weights for each fuzzy rule are prepared through layer 2 and layer 3 according to Eq.3.13 and Eq.3.14 in Chapter III, layer 4 will give a linear combination of the inputs for each fuzzy rule as shown in the equations below [1]:

$$f_r = x * p_{r,1} + y * p_{r,2} + \cdots + s_r \qquad (5.2)$$

where there are N + 1 parameters to adjust, N being the number of inputs and r indicates the index of the fuzzy rule. In this case, N=2, therefore, there are 3 parameters left for adjustment for each fuzzy rule. Therefore, 48 parameters are required for data process at this layer. Each parameter is stored as 8 bit integer. These parameters are determined through Least Square Method. As the last step, the final layer sums up all the contributions of the rules to calculate the output.

The process described above is the forward pass for one input data to go through all five layers in the ANFIS system. The consequent parameters (from the fifth layer) are updated with a sequential algorithm in a first forward pass, once this is accomplished and the output calculated, the premise parameters (from the first layer) should be updated with the Back-Propagation algorithm.

After the parameters have been identified, the functional signals keep going forward to calculate the error measure. Each epoch of this hybrid learning procedure is composed of a forward pass and a backward pass. In the forward pass, we supply input data and functional signals go forward to calculate each node output until every parameter in each node is identified. After identifying parameters, the functional signals keep going forward till the error measure is calculated by comparing the estimated output with the

output data contained in the training data. In the backward pass, the error rates propagate from the output end toward the input end, and the parameters are updated by the gradient method. Then according to the updated parameters the lookup table for membership functions is updated for next iteration.

All these procedures are realized on the FPGA board using XILINX EDK 10.1. It allows designers to use an embedded DSP core PowerPC 405 to run complex algorithms without translating to a hardware description language. All these five layers are described in the C language in the EDK environment. All input data are stored in the on-board RAM and the DSP core communicates with the RAM through a local bus. 50% of input data is used as training data and 50% is used for checking data.

5.4 TME Implementation

The output of the ANC is used as an input in the TME block for final target parameter estimation. In this section the hardware implementation of the TME operation used in the hybrid algorithm is presented [2]. The system is composed of three parts: Noise Reduction block based on the Discrete Wavelet Transform (DWT) used for further noise reduction, Optimization used to determine an appropriate set of filter coefficients for

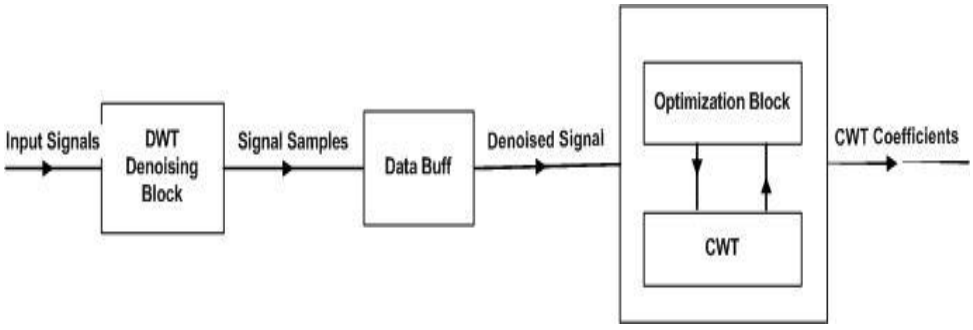


Fig 5.4 Block Diagram of TME Hardware Implementation

Continuous Wavelet Transform (CWT) computation, which is in turn used for the final target parameter (distance and velocity) estimation. Fig.5.4 shows the block diagram of the TME hardware implementation. In view of the diagram, the input signal is the output from the ANC operation in which the total signal reflected from the target was pre-filtered by the ANFIS operation, in order to minimize spurious returns such as background noise and reverberation from the ocean surface or bottom.

The output from the ANC is fed as an input into the DWT noise-reduction block to reduce the coefficients related to noise. Following the process of noise reduction, the Optimization block will choose the best set of filter coefficients from a pre-defined collection of filter coefficients which characterize different replicas of the transmitted signal and send it to the CWT computation unit for target motion parameters extraction. The target parameters will be extracted from CWT part by examining similarities between filter coefficients and input data samples. According to the peak value extracted by the CWT, the Optimization block will continue to choose different sets of filter coefficients and run the CWT again to obtain a local optimum solution in a pre-defined area so that the most similar replica of the transmitted signal compared to the received data can be found and the motion parameters of the target can be calculated according to the frequency of the replica.

5.4.1 DWT Denoising filter bank

The functionality of this block is to decompose input data using the DWT and hence remove unwanted wavelet coefficients according to threshold value. The wavelet

coefficients are then reconstructed into signal data again using the IDWT.

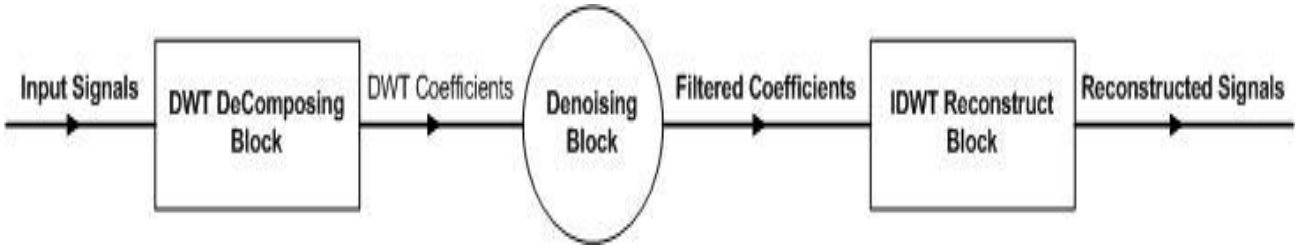


Figure 5.5 DWT Denoising Block

For DWT implementation a pair of FIR filters is used based on the pyramid algorithm developed by Mallat [3]: one low-pass filter for the scaling function, g (n) and one high-pass filter for wavelet function, h (n) [3]. Fig. 5.5 gives the block diagram of the DWT implementation, where the DWT Decomposing Block decomposes input signals; the denoising Block is employed to remove low DWT coefficients according to pre-defined thresholds; and the IDWT Reconstruct Block is used to rebuild denoised signals.

For the DWT filter design the first task is to choose an appropriate mother wavelet for the specific situation. The most commonly used set of discrete wavelet transforms was formulated by the Belgian mathematician Ingrid Daubechies in 1988. This formulation is based on the use of recurrence relations to generate progressively finer discrete samplings of an implicit mother wavelet function. Named after Ingrid Daubechies, the Daubechies wavelets are a family of orthogonal wavelets defining a discrete wavelet transform and characterized by a maximum number of vanishing moments for some given support. With each wavelet type of this class, there is a scaling function (also called father wavelet) which generates an orthogonal multiresolution analysis. Daubechies

orthogonal wavelets D2-D20 are a widely used mother wavelets in many DWT applications. The index number refers to the number of coefficients *N*. The halving of the coefficients number of each wavelet gives the number of vanishing moments that each wavelet has, which limits the wavelet's ability to represent the polynomial behavior or the information in a signal. D20 has 10 vanishing moments and has the ability to represent complex signals containing constants, linear, quadratic or even higher order components. Therefore, D20 is chosen as the mother wavelet for DWT implementation in this project due to the complexity of the received sonar signals. The coefficients for the D20 wavelet are shown in Fig 5.6, in which scaling function refers to low-pass filtering and wavelet function refers to high-pass filtering.

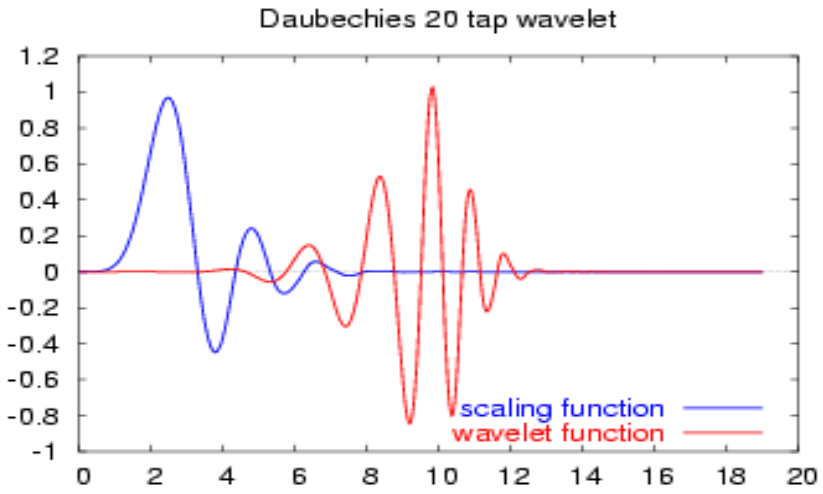


Fig 5.6 Daubechies 20 tap wavelet Coefficeints

From Fig 5.6 it can be observed that the original coefficients for the high-pass and low-pass filters are all between -1 to 1. In a hardware implementation fractional number computation is more complex than integer number computation. Therefore, the original filter coefficients are all multiplied by 1024, which corresponds to left-shifting by 10 bits

in the hardware representation. The modified filter coefficients are then represented by signed 2's complement format in 11 bits. When all the computation is completed, the result is then right-shifted back by 10 bits to compensate for the amplitude change of filter coefficients.

The coefficients for the high-pass and low-pass filters are determined by the chosen mother wavelet; however, the architecture for filters is still a major factor in performance characterization.

As mentioned in Chapter III, the maximum CWT coefficient at a certain scale s can be obtained from the Finite impulse response (FIR) filtering of the received signal with the corresponding FIR coefficients. Also DWT coefficients at a certain scale can also be obtained from an FIR filter with the correct coefficients.

Finite Impulse Response (FIR) filters are widely used in digital signal processing. An N-tap FIR filter is defined by the following input-output equation:

$$out(n) = \sum_{i=0}^{N-1} X(N-i)h(i) \qquad (5.3)$$

where {h(i): i = 0,…, N-1} are the filter coefficients.

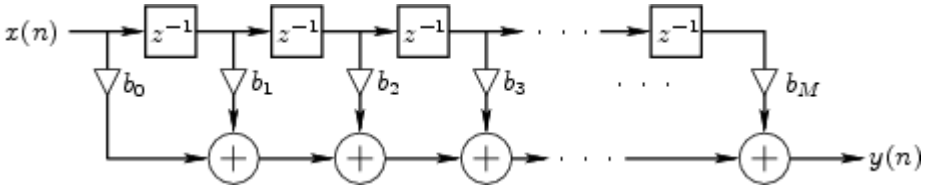


Fig 5.7 Data Flow Diagram of N-tap FIR

The function of an FIR filter is to implement a convolution operation. The basic hardware architecture of an FIR filter is shown above in Fig 5.7. Since the VLSI architecture we are

currently investigating is convolution-based, FIR filters are the primary elements for wavelet transform implementations.

There are numerous types of FIR filters and each has their own advantages. Serial architecture is an area-saving method by reusing adders and multipliers sequentially; meanwhile parallel architecture performs better at speed. Most improvements of FIR architectures are based on the architectures of multipliers or adders and rescheduling of outputs. The purpose of FIR design investigation in this work is to find a good trade-off between speed and area and most importantly to fit the requirements of this project.

Three types of FIR filter architectures have been built and functionality test have been performed. The first one is an FIR filter with parallel multipliers. The basic architecture is shown below in Fig 5.8

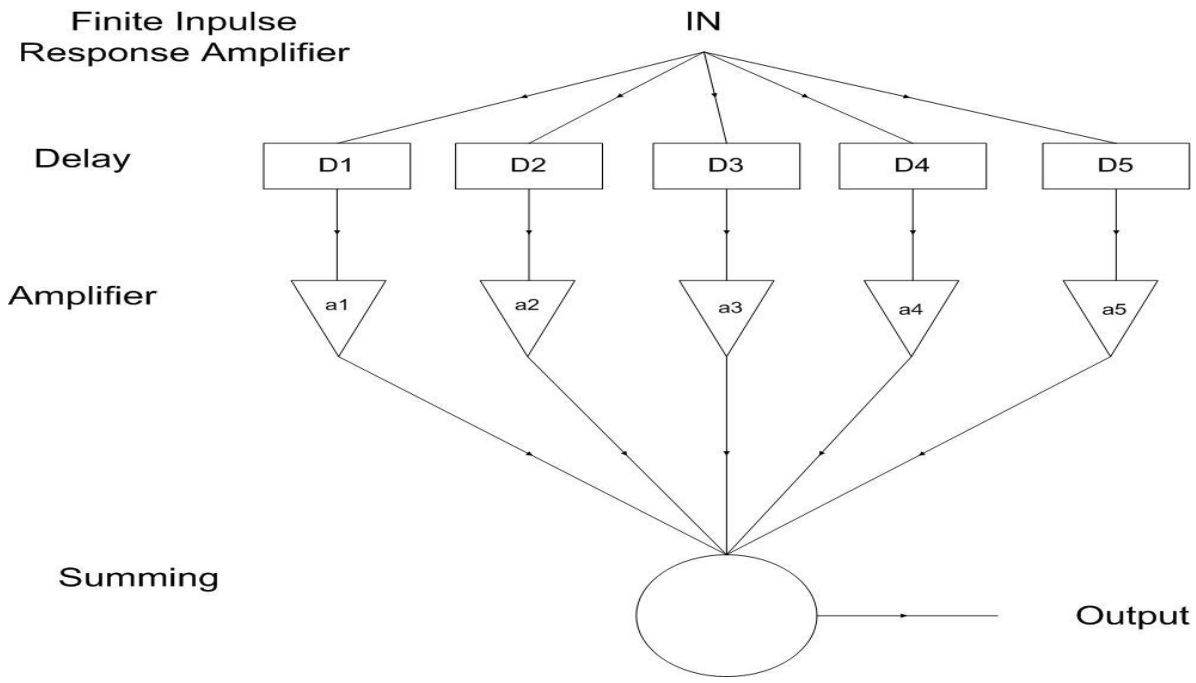


Fig 5.8 FIR filter with parallel multipliers

It uses several multipliers working in parallel to calculate the results. This

architecture has good working speed, but requires many more multipliers and the area cost is also great.

The second architecture investigated uses a Canonic Signed Digit (CSD) code multiplier instead of normal multipliers. Canonic Signed Digit code is used to represent the coefficients of the FIR filters and can reduce the number of none zero digits (1 or -1) significantly. With this code CSD multiplier can reduce the hardware cost of the FIR filter significantly. At its maximum a CSD multiplier uses only half of the hardware resources of a normal multiplier. However, this approach is costly in terms of design timing and is less flexible since each CSD multiplier should be designed individually for each coefficient.

In the third FIR architecture pipelining is introduced in the design and adopted for the current project. At its minimum only 1 multiplier and 1 adder are needed for a FIR filter design. The basic idea is to feed appropriate filter coefficients to multipliers and adders according to the time slot and reuse the multiplier and the adder several times during the whole computation process. This method heavily reduces the hardware cost of the FIR filter but the computation delay will be much longer than for the other two architectures. Since the two filters each have 20-taps, using a traditional parallel multiplier-tree would require 40 multipliers working in parallel. Taking a second set of high-pass and low-pass filter for IDWT into consideration, these four filters will consume 80 multipliers. In implementation of the CWT operation which is also realized by FIR filter more multipliers will be needed. Considering the limited multiplier resources in the FPGA

environment this imposes a huge consumption of hardware. Therefore, reusing multipliers is taken into consideration and adopted here.

An extreme area-saving design is proposed for filters in the DWT implementation as shown in Fig 5.9. Only one multiplier is used for the high-pass filter and only one is assigned to the low-pass filter. A FIFO (First In First Out) register of size 20 samples is build to store the data samples needed for the filter computation. At each clock cycle of the sample frequency, a new data sample is fed into the FIFO and the sample fed into the FIFO 20 cycles before is shifted out. Then both filters get samples from the FIFO in turn. These are multiplied by the corresponding filter coefficients at a frequency, namely the filter frequency that is 20 times faster than the sample frequency. Working at the filter frequency, the high-pass and low-pass filter are able to provide computation results during one clock period of sample frequency.

As described previously, the output from the low-pass filter is down-sampled and used for next-level decomposition. A register file is build to store those outputs from the low-pass filter.

The longer the input sample sequence is, the more area for the register file is needed.

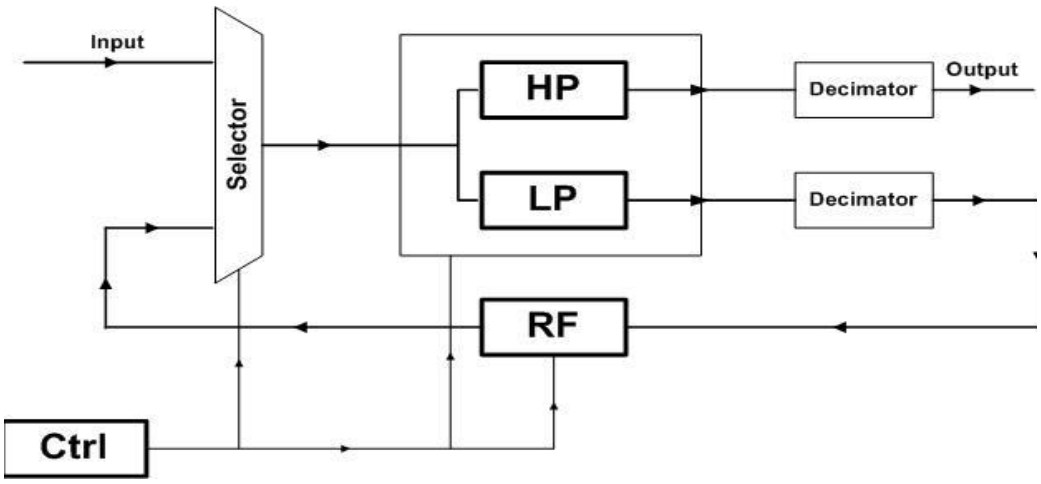


Figure 5.9. DWT Decomposition architecture

For example, if the sample frequency is 100 kHz and a sample sequence of 1s samples is fed into the denoising filter bank, the register file will be 50k samples long to store the output of the low-pass filter for first decomposition level if no action is taken.

In order to save the area consumption for the register file, a RPA (Recursive Pyramid Algorithm) [4] is adopted in this project. According to the characteristics of FIR computation, when computing the second level coefficients, only the first K outputs from the first level low-pass filter outputs are needed, where K is the length of the wavelet filter taps. Therefore, by scheduling the feeding sequences of two filters, the first output of the second level decomposition can be obtained right after the Kth output of the first level decomposition. For the following level decomposition, coefficients also can be obtained before the previous level decomposition is finished. Therefore, the register file is no longer required to store the whole output sequences from the previous decomposition, instead only a K*J sized space is needed for register file, where J is the decomposition level.

The decomposition level is also an issue for performance measurement. Increasing the decomposition level will improve the performance; however the hardware cost and the computation time are also increased. In the first place, for the simplicity of hardware implementation, the level of decomposition is set to 5. However, during the simulation process, it was found that at this decomposition level the TME block cannot meet the accuracy requirement due to the noise affect. After several design iterations and simulations, the decomposition level was set to 7 and the final output of the TME is then

acceptable for target motion parameter extraction. In the simulation explained in Chapter VI several simulation results from the design iterations for decomposition will be presented.

As shown in Fig 5.10, After the DWT processing, all DWT coefficients are fed into the denoising block to remove the coefficients representing noises according to the denoising threshold value. Then the DWT coefficients are reconstructed through the IDWT.

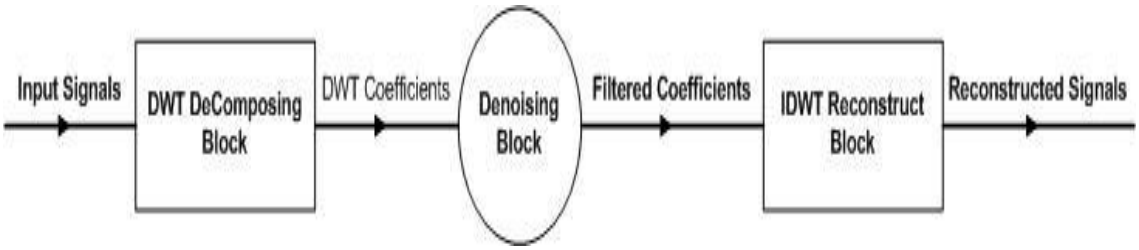


Fig 5.10 DWT Denoising Block Diagram

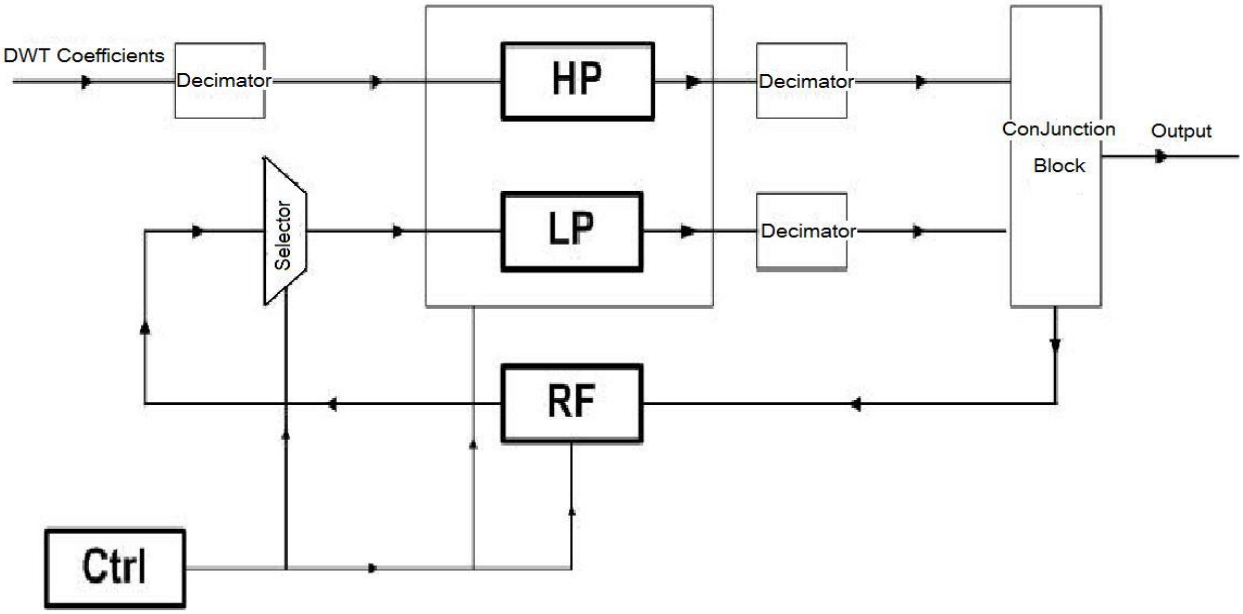


Figure .5.11 IDWT Reconstruction architecture

As described in chapter II, high-pass and low-pass filter are still two main functions

in IDWT implementation. The architecture of the IDWT is almost the same as the DWT implementation, sharing the same filter pair, except these two filters are fed with different wavelet coefficients. The Details are shown in Fig 5.11.

5.4.2 Optimization Block

The purpose of this function block is to reduce the computation time of the TME without sacrificing too much resolution accuracy for target detection. As described in Chapter II, in order to extract the accurate target parameters, the FIR filter coefficients, which have the characteristics of the dilated mother wavelets that are used for similarity detection between received signals and dilated mother wavelets, should be as similar to the received signal as possible. Since the target information cannot be predicted before the computation, different filter coefficients corresponding to different dilated mother wavelets should be prepared before the target detection takes place. Theoretically the received signal should be examined by every set of prepared dilated wavelets to determine which one is the most similar one and then the corresponding target parameters will be extracted from corresponding dilated wavelet parameters. However, there will be 121 sets of dilated mother wavelet if the resolution of the velocity estimation is 1 knot in the range of -60 knots to 60 knots, where a negative number indicates the target is moving toward the receiver and a positive number indicates the target is moving away from the receiver. The system will not be capable of working in real time if every set of dilated wavelets is examined due to the long computation time of the convolution operation. Therefore this optimization block is introduced into the

system for saving computation time, while keeping the accuracy of the target motion parameters extraction.

As described in Chapter III, The Brent's optimum search algorithm is employed in this optimization block. The main purpose of this algorithm is to choose the most suitable set of filter coefficients from all 121 sets to extract the target motion parameters. Therefore, the search intervals in the algorithm are [1,121] if all the sets of coefficients are sorted by frequencies. The evaluation point is then easily determined by the given intervals range. The golden ratio is pre-computed and defined as a constant for the ease of computation. Once an evaluation point is chosen, the control unit will prompt the CWT convolver to convolute the input data samples with the corresponding filter coefficients, and the values obtained through the convolution represents the values of the CWT operation for the current evaluation point. When those values are fed back by the CWT convolver, step-1 and step-2 described in Chapter III will be carried out to determine the position of next evaluation point. All the parameters are then updated and the next computation iteration begins unless the termination criterion is met.

The algorithm is implemented as a Finite State Machine (FSM) depicted in Fig 5.12.

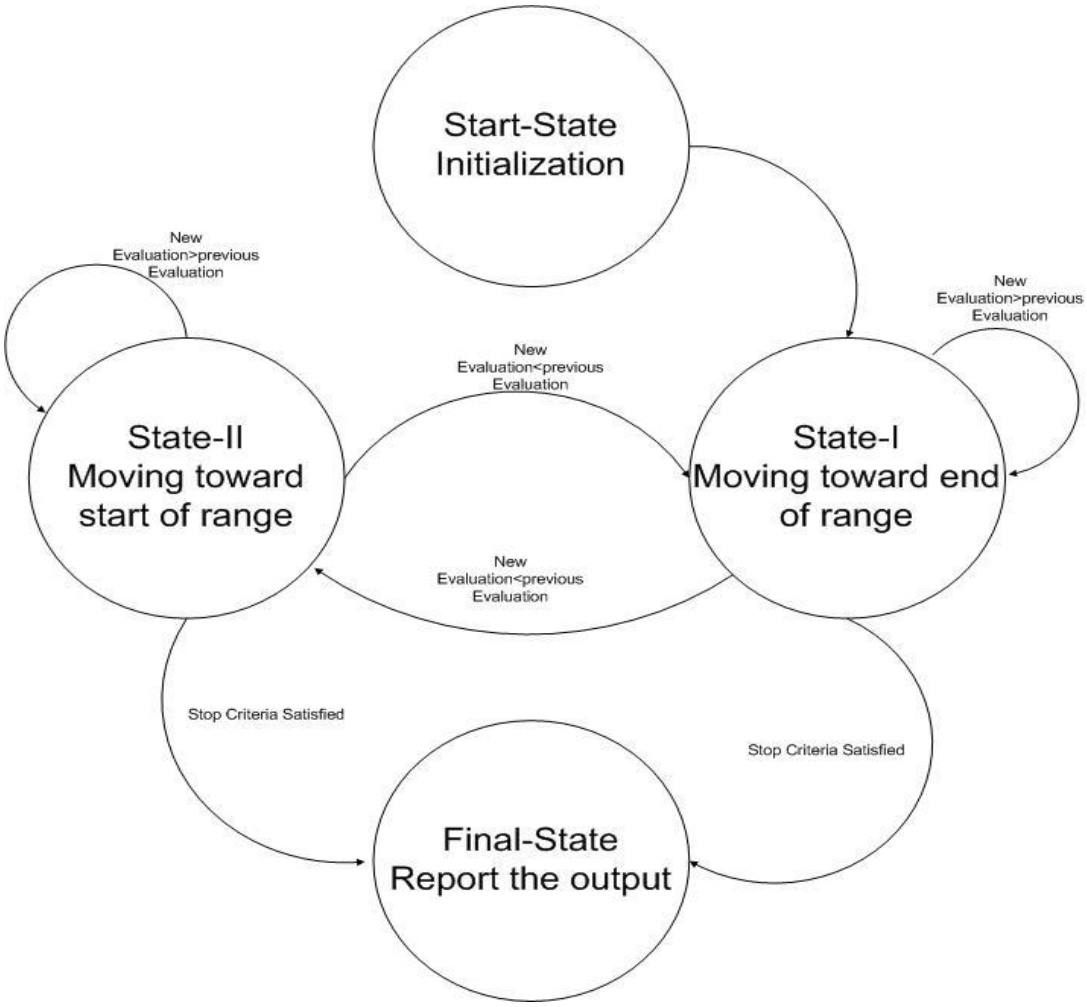


Fig 5.12 Finite-State-Machine of Optimization

From the beginning, the start-state will determine the initial evaluation point and move to state-I, which will move the evaluation point toward to the end of the interval range. If the result from the computation matches the rule of state-I, the process will stay in state-I, otherwise, the process will go to State-II to move the evaluation point towards to the start of the search interval. The state in which the process stays will determine whether the computation process will go on or not after every computation. If the stop criteria are matched, the process will go to the final state to report the final result.

This process will go on until the local optimal filter coefficients are found or the time limit is met. The time limit is essential in this function block because it not only affects the precision of the optimum search algorithm, but also determines the system's real-time working ability. If the time limit is too long, the local optimum search will not able to find a good result in most cases before the time limit expires and therefore the precision of the algorithm is undermined. However, if the time limit is set too wide, the algorithm might spend too much time on searching for the optimal result and cannot finish the process. In this project, the computation iteration number is limited to 20 to make sure the system can work in real time. The simulation results presented later also illustrate that this limit does not undermine the precision of the system. By looking for a local optimal result instead of a global optimal result, this method reduces the computation time 10 times in most cases without sacrificing resolution accuracy.

5.4.3 CWT convolver

As shown in Chapter III, the CWT operation is achieved by employing matched FIR filters in the system. By feeding input signals into FIR filters with coefficients which represent the characteristics of the dilated mother wavelet, the CWT convolution is achieved at the hardware level.

The advantage of the CWT for high resolution and precision is the main reason for adopting it as target parameter extraction method in the system. As explained in chapter III, numerous FIR filters are required to attain its advantage. Each FIR filter is corresponding to one set of dilated mother wavelets. 121 sets of FIR filters are found

sufficient for the target extraction process to extract target motion parameters with high resolution.

Dilated mother wavelets used for target parameter comparison are realized by corresponding filter coefficients. Obviously the tap number of the filter coefficients will determine how precisely those dilated wavelets are represented. Filter coefficients with larger tap numbers will give more details of the dilated mother wavelets leading to more precise match results. However, this also takes more computation time for the system to get the result. Since a 2 ms transit pulse is used in all simulations and the sample frequency at the receiver is 100 kHz, the tap number is approximately 200 taps for a dilated version of the wavelet when discretely sampled at the sample frequency.

The tap number indicates how many multiply operation the system needs to perform on the CWT coefficients for every data sample. Obviously with such large tap numbers the multiply operations cannot be performed serially in the allowed frequency range of FPGA chip to ensure the system works in real time. Parallel multiplier architecture is therefore required in the system. However, the multiplier resources in the FPGA chip are also limited. Besides, the bandwidth of input and output ports for the memory are limited; more parallel multipliers means that more fractional memory units are needed and more time is needed to arrange memory mapping and timing relocation arrangement for each multiplier. Therefore, large multiplier bank architecture is not a solution, either.

In the proposed architecture a novel, extreme area-saving design algorithm is adopted. Only five multipliers and five adders are used to implement FIR filters with

adaptable taps. All the filter coefficients for different versions of wavelets are stored in a storage unit at different pre-located blocks. When the input signals are fed into the CWT convolver to obtain CWT coefficients, the control unit will determine which set of filter coefficients is going to be fed into the FIR filter according to the optimization block. At every system cycle, five input data samples and five filter coefficients are read from the corresponding storage unit into the FIR filter. Then the parallel multipliers in the FIR filter will give the multiplication result for each set of input data and corresponding coefficients. Next, these results are then loaded into the adder together with the addition result from the last cycle. This partial result is then stored into the register for the next loop until all data and coefficients are processed. The working clock frequency of the FIR filters should be set much higher than the sample frequency, which provides the input data, in order to get the final convolution result of the current input data sample before next input data sample is fed into the filter. If the CWT computation is only required once for the whole data processing the working frequency can be set to n/5 times higher than the sample frequency, where n is the maximum tap number for the filter coefficients. However, before the exact target motion parameters are obtained several iterations are needed for the CWT computation. Therefore, the working frequency of the FIR filter is set to m*n/5 times of the sample frequency so that it satisfies the word-serial model and operates correctly in real time, where m is the maximum allowed number of iterations of the CWT operation .

Since the entire range of possible filter coefficients are pre-stored, and taking the large tap

numbers and large number of different versions of the filter coefficients into consideration, the coefficients storage unit is the major area consumption unit in the hardware design. How many bits are used to represent the filter coefficients is then not only a factor of computation precision, but also a major factor of area consumption. Traditionally the filter coefficients are in the range of [-1, 1]. However, these fractional numbers might lead to unnecessary data post-processing for multiplication, i.e., for determining the rounding and truncation algorithm. Therefore, all coefficients are multiplied by 2048 and transferred into 12 bits signed 2's complement representation before being stored. This corresponds to left-shifting the fractional numbers by 11 bits. Therefore, all the filter coefficients can be considered as integer numbers which avoids data post-processing of fractional number multiplication. After the multiplication, right shifting the computation results by 11 bits and simply truncating the unnecessary bits, the width of the output results will be kept the same as the input data samples. The simulation process shows that 12 bit representation will not affect the computation precision and neither will lead to heavy hardware resource consumption. Other solutions are also adopted to save the area further. Since the mother wavelet chosen for simulation is symmetric, the filter coefficients which represent dilated mother wavelets are also symmetric, and therefore the size of the storage unit which contains all the possible versions of filter coefficients can be halved, but care must be taken when fetching data sequences in the control unit. Overall the hardware cost is significantly reduced.

From all the design considerations described above, the final CWT convolver is

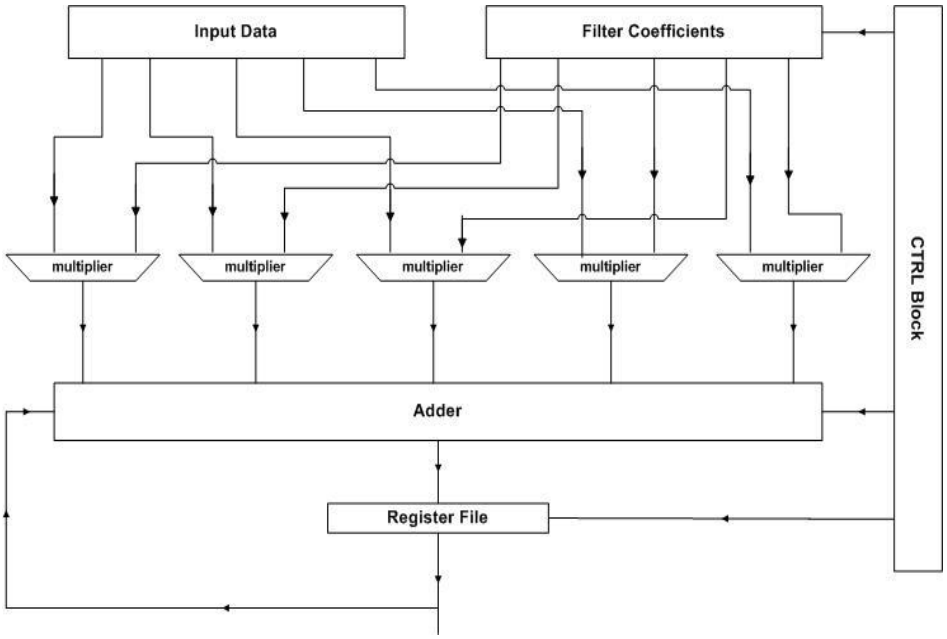


Figure 5.13. CWT Computation Unit architecture

implemented as shown in Fig 5.13. The optimization block shown in Fig 5.4 will give the control unit (which is shown in Fig 5.13 as CTRL Block) indication signals of which set of filter coefficients should be used for the current computation iteration. Then the control unit (CTRL Block shown in Fig 5.13) will determine which block of input data samples are going to be fed into the parallel multipliers during the current working cycle. The final result will be fed back to the optimization block to determine whether more iteration is required.

In the proposed design it is assumed the input data samples are 16 bits wide each and presented as 2's complement format and are sampled at 100k Hz. The sample window width is assumed to be 1 second. By deciding the sample frequency and window width of the input data, the size of the most area-consuming block, the data buffer between the

DWT denoising block and the CWT convolver is decided. Therefore the Xilinx FPGA chip XC2V30P was chosen as the implementation platform for its sufficient built-in block RAMs. Due to the area-saving algorithm implemented in this design, the whole design only required 9 of out 136 18x18 bit multipliers, 32% slices and 69% block RAMs on the XC2V30P chip.

As mentioned above, the working frequency of the CWT convolver is set to m*n/5 time of the sampling frequency, where m is the maximum allowed number of iterations of the system and n is the maximum number of taps of filter coefficients. Since the maximum iteration number is already set to 20 in the Optimization block and the maximum tap numbers is around 200 during all sets of pre-defined filter coefficients, the working frequency of the CWT convolver is set to150 MHz to ensure that the TME works in real time at the current sample frequency. The layout of the hardware is also specified to reduce the transmit delay between each function block in order to improve the system working speed. The data storage unit is placed at the center of the FPGA and all functional blocks are laid surrounding this. This layout algorithm reduces the transmit time between each functional blocks and therefore increases the working speed of the system.

5.5 Simulation Results

The proposed system is simulated in Modelsim XE 6.1e block by block in order to verify the functionality of the system.

As the first step, the functionality of the TME block is tested. A transit 2 ms long morlet

wavelet [1] depicted in Fig. 5.15(a) is adopted as the transmitted signal. Fig. 5.15(b) illustrates a returned echo signal where the target signal is buried in the noise with a SNR of -20dB. This signal is sampled at 32 kHz in the time range [0, 1] sec. The received signal processing is firstly performed using the ANFIS offline in the ANC operation. The output of the ANC part is then used as the input signal for the proposed hardware implementation. In the fig. 5.16 the first signal is the input signal fed into the hardware system, which is the output of the ANC with a reduced SNR of -1.8539 dB.

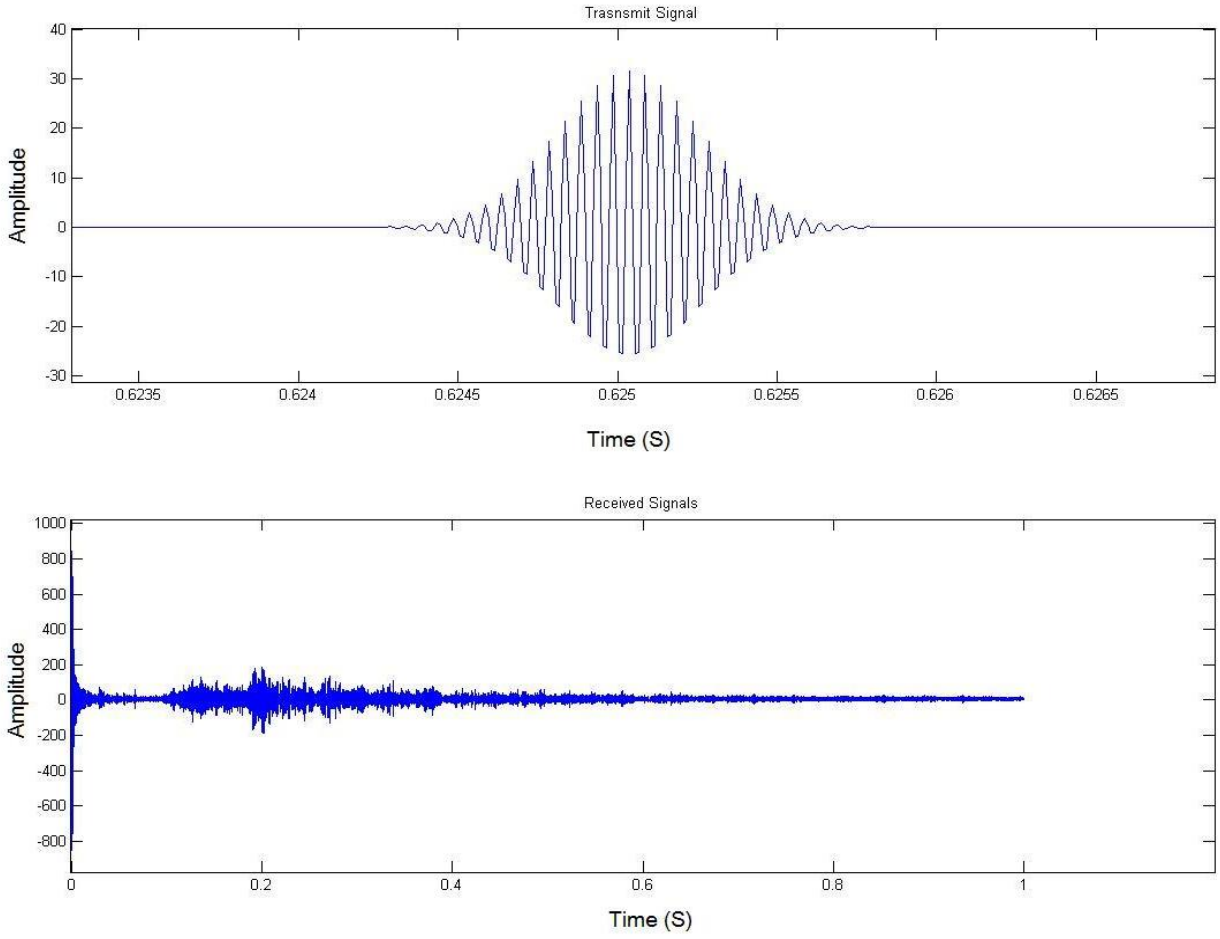


Figure 5.15. (a)Transit signal; (b) Received signal;

The input samples are represented in two's complement format with 16-bit width. The

following graphs in fig. 5.16 represent the output signal after the first, the second and the fourth TME iteration where the stop algorithm for TME has not been added into the system. It is clear that after each iteration the position of the target is more and more apparent; the target signal is extracted from the noise and its amplitude boosted afterwards. However, in the hardware only 16 bits are used to represent the coefficients, and the amplitude would eventually exceed the hardware limit. In order to prevent this, the output amplitude is deliberately decreased after every other iteration. In the graph shown above, the peak coefficient appears at the time 0.6809 s after only 4 iterations, which compares favorably with the theoretical result of 0.6794 s. Using the equations 3.5 and 3.6 in Chapter III, the calculated location of target is 475.93 meters away which compares favorably with the theoretical location of and the velocity of the detected target is 475 meters; and the calculated velocity is 19.5 knots towards to the receiver, which compares favorably with the theoretical velocity of 20 knots towards to the receiver. From the simulation results we can conclude that the hardware system is able to work correctly in a noisy environment [5].

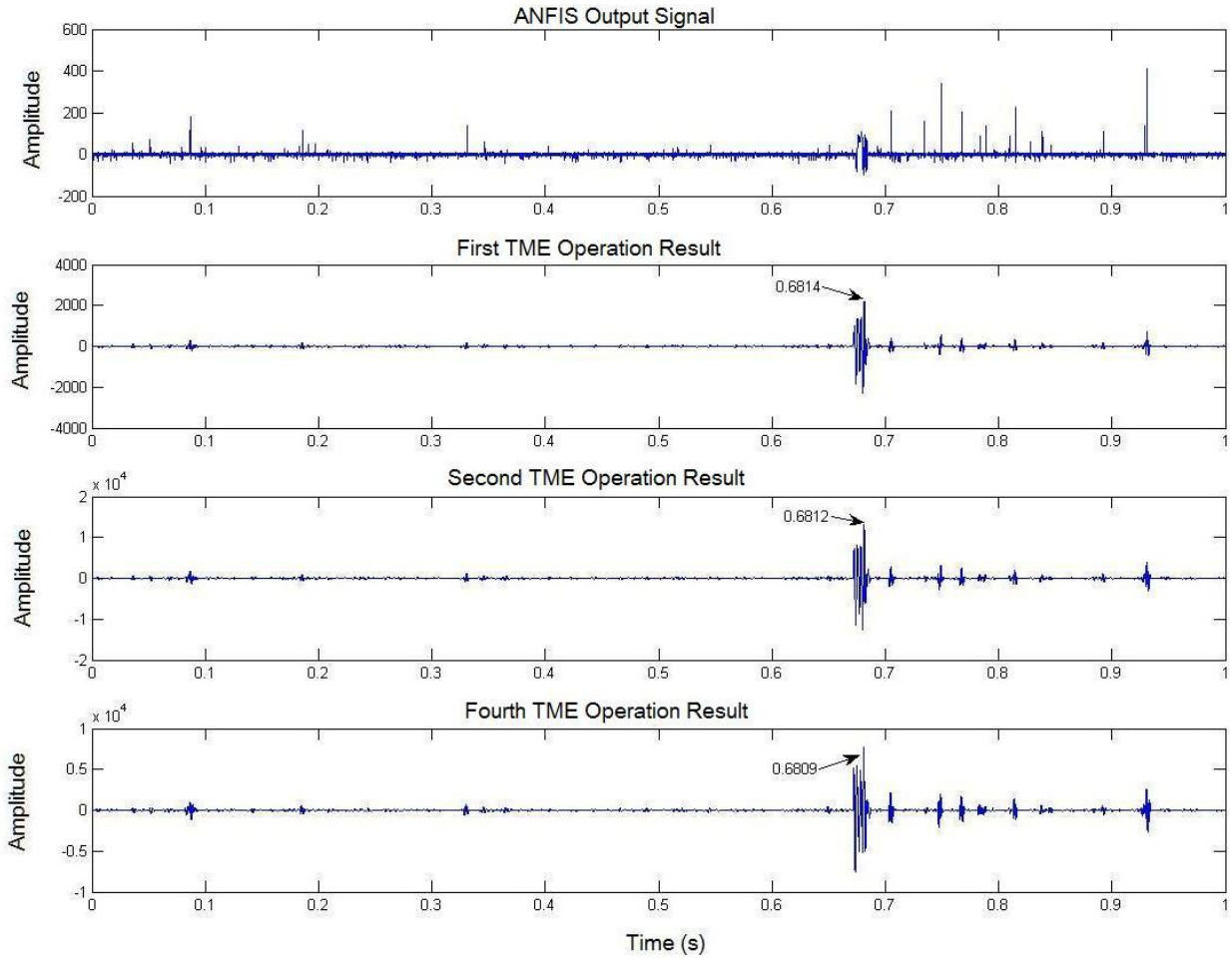


Figure 5.16. Signals after TME Operation

The hardware architecture was slightly modified as the result of the system simulation. Initially, the DWT decomposition level was set to 5 in order to simplify the hardware implementation. However, during the simulation process it was found that at this decomposition level the performance of the DWT denoising block cannot meet the requirement. As shown in Fig 5.17, the target is at 0.6918s. However, a false alarm at 0.2653s is not well suppressed by the DWT denoising block and the target cannot be detected. When the decomposition level was set to 7, the performance was significantly improved as shown in Fig 5.18, the noise was significantly suppressed by the DWT denoising block and the computation load was increased by less than 3% and only 40

more samples in the register files are needed for 2 more decomposition levels due to the description in chapter III. Therefore, the decomposition level is set to 7 for this particular hardware implementation.

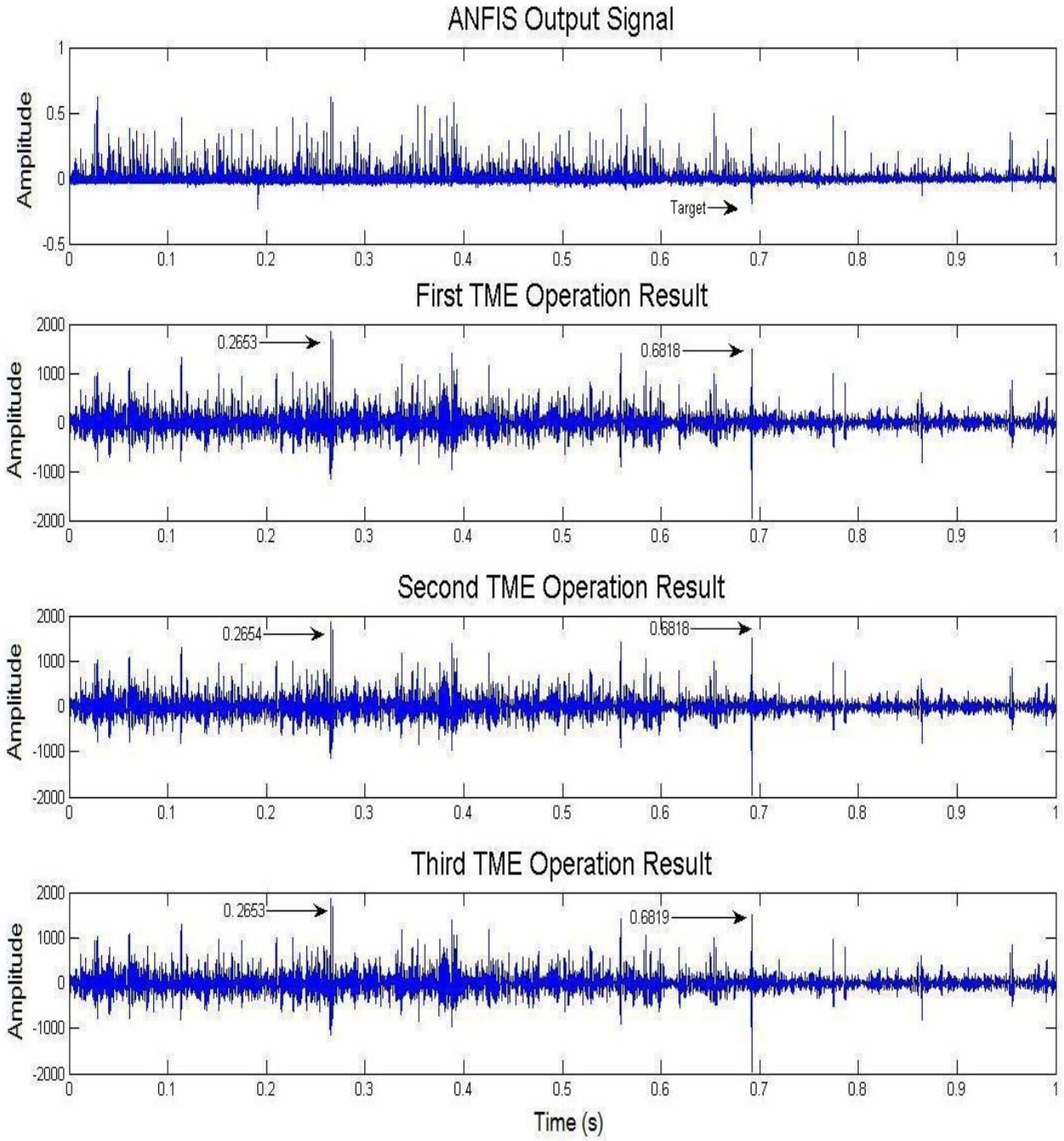


Figure 5.17. Signals after TME Operation J=5 SNR=-1.9 dB after ANC operation

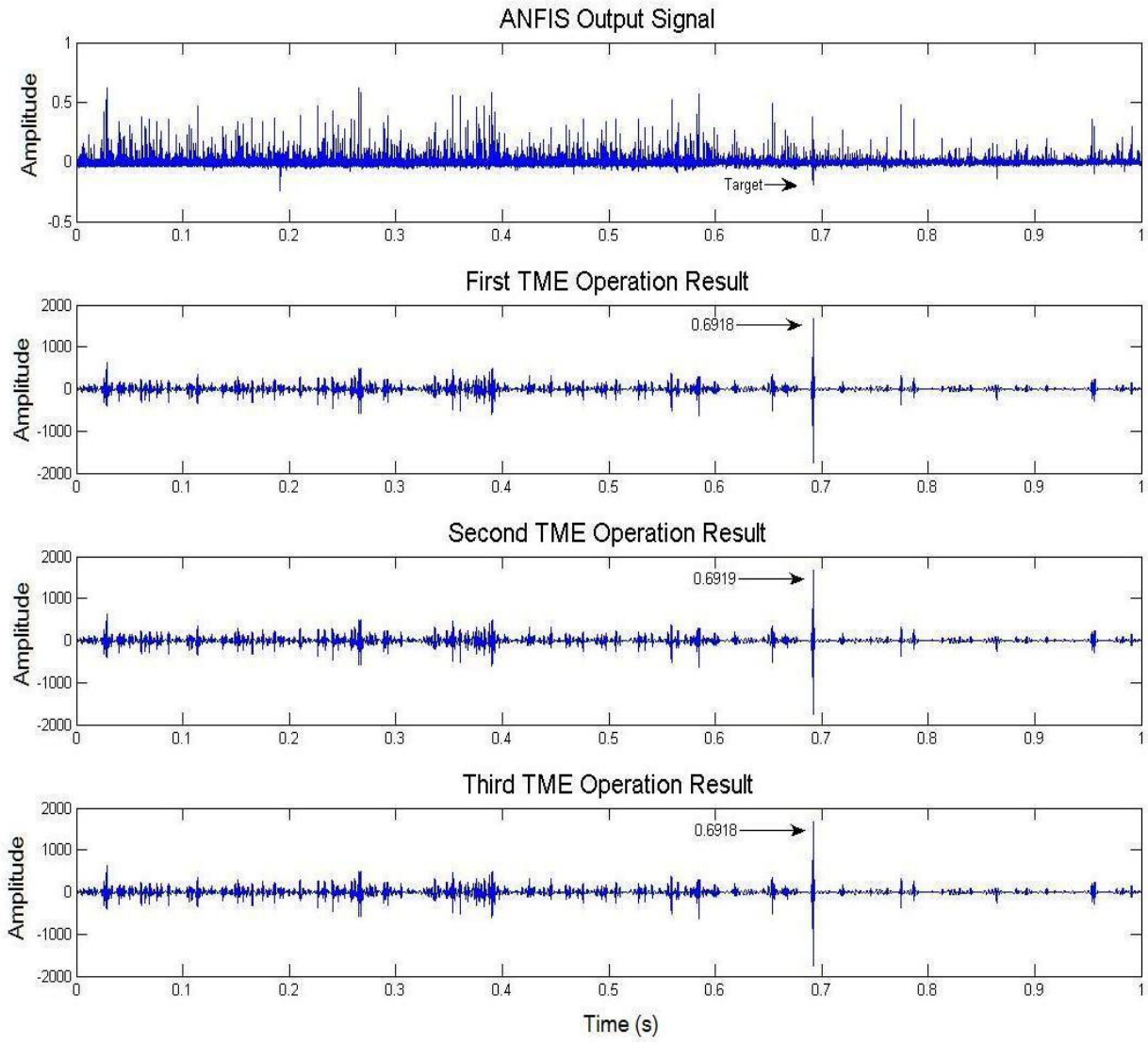


Figure 5.18. Signals after TME Operation J=7 SNR=-1.9 dB after ANC operation

Then the Local Optimum Search block was added into the system to reduce the target extraction time. For this simulation a transit 2ms long Morlet wavelet signal depicted in Fig. 5.19(a) was adopted as the transmitted signal. Fig. 15(b) illustrates the composite signal where the target signal is buried in the noise with the target strength of approximate -24dB at the position of 0.8359s. This signal is sampled at 100 kHz in the maximum time range [0, 1] sec.

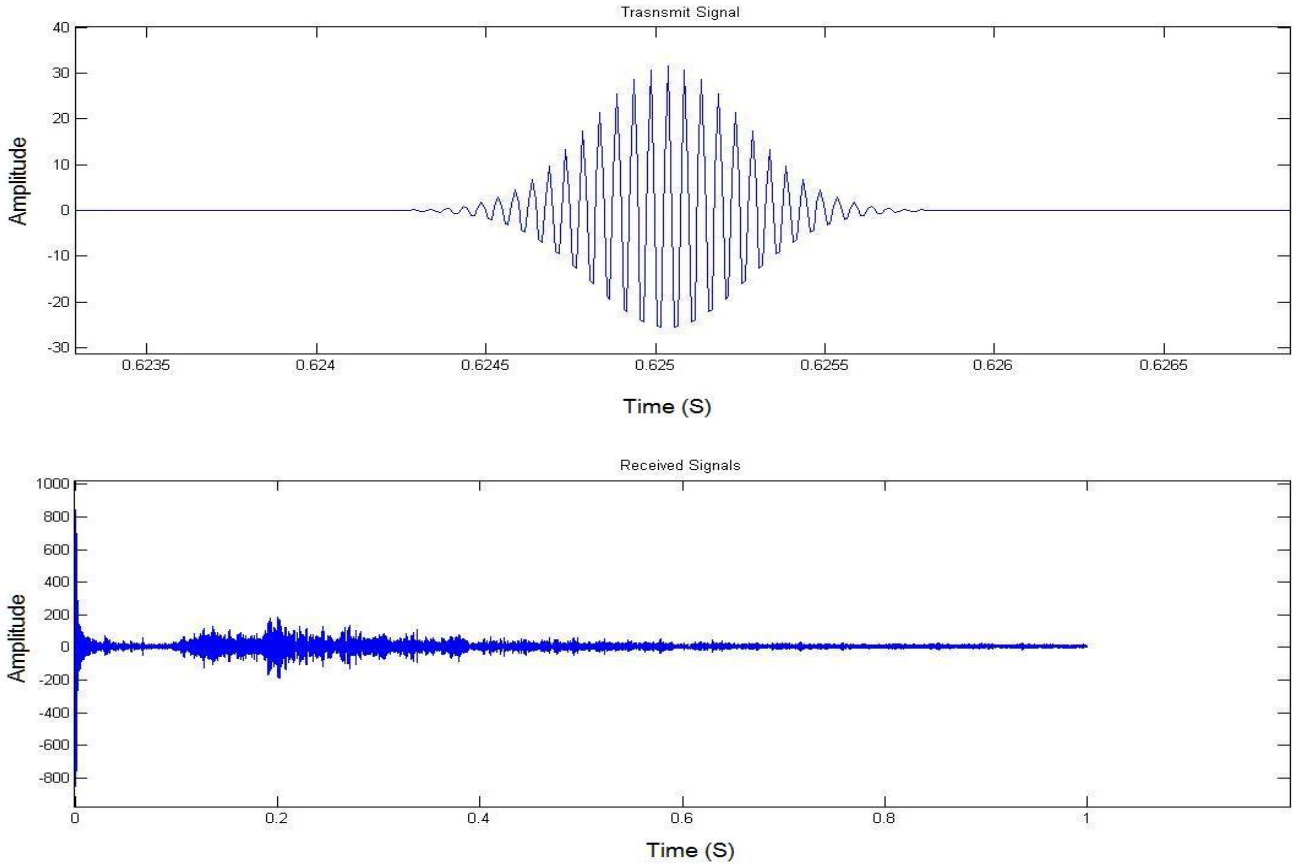


Figure 5.19. (a).Transmitted Signal(b)Received Signal

The simulation results are shown in Fig. 5.20 where the first signal is the input signal into the FPGA system, which is actually the output of the ANC. The input samples are represented in two's complement format with a 16-bit width. The graphs in Fig. 5.20 represent the output signal after the first, the second and the third TME loop. It is clear that the position of the target in each TME loop moves in a range of less than 0.001s, and in that case the position of the target can be found. In the graph shown in Fig 5.20, the peak coefficient takes only 3 loops of the TME to appear at the time axis 0.8359 s that is favorably compared to the theoretical result at 0.8368 s. By detecting the position of the peak coefficient, motion parameters can be determined. In this data set the target is

located in 585 meters away from the receiver and with velocity of 30 knots towards the receiver. According to the location and scale output from TME the detected target is 584.6 meter away from the receiver and the velocity is 19.5 knots towards the receiver. The error ratio against total range of speed is less than 4% and the location error ratio is less than 1%. Therefore the simulation results indicate that the system is capable to work under severe reverberation noisy environment.

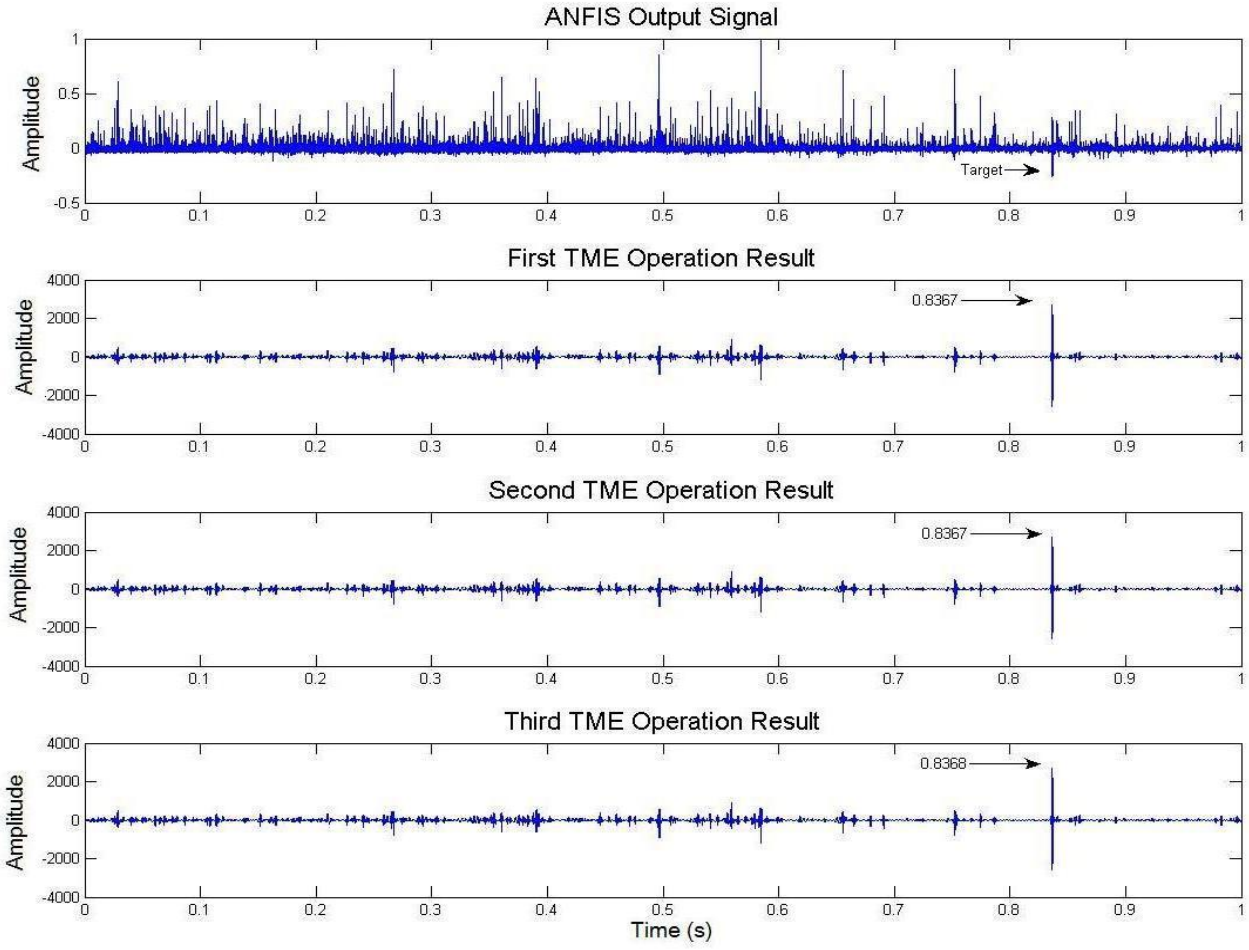


Figure 5.20. Signals after TME Operation SNR=-24dB before ANC Operation, -1.83dB after ANC

Finally, the ANFIS block was introduced into the simulation and the stop criterion for TME is proposed. The system was tested under 16 sets of single target data with a SNR range from 0dB to -30dB with intervals every 2dB. The noise constituted of white

Gaussian noise and a reverberation waveform which is provided by the MOD to simulate the real underwater environment amid the following environment settings: sea depth (=100m), sonar depth (=50m) wind speed (=6m/s ~ sea state 3), seabed type (=medium sand). Received signal is sampled at 100 kHz in the maximum time range [0, 1] sec.

A transit 2ms long morlet wavelet signal is adopted as the transmitted signal for simulation under environment above. A returned echo signal is buried at the place of 0.6908 s in the noise with SNR (signal-noise ratio) of -30dB.

As shown in the fig. 5.21, the first signal is the input signal fed into the TME block, which is the output of the ANFIS. The ANFIS has already reduced the SNR from -30dB to -2.65 dB after 47 epochs. As explained in Chapter IV, it is hard for ANFIS to go further since the input data used for ANFIS model are simulated ambient noise and experimental reverberation model. The following graphs in fig. 5.21 represent the output signal after the TME operation. In the graph shown in Fig 5.21, the peak coefficient appears at the time 0.6914 s, which compares favorably with the theoretical result of 0.6908 s. By detecting the position of the highest coefficient the time delay τ in Equation 3.1 is known. Also the corresponding scale s can be determined by examining which set of filter coefficients produces the highest CWT coefficient. Therefore, the motion parameters such as position and velocity can be determined using the definition of τ and s outlined in Eq. 3.2 and 3.3 in chapter III, which in this simulation the location of detected target is 483.28 meters away from receiver with velocity of 29 knots away from the receiver, which compares favorably with the theoretical result of 481.76 meters away in location

and 30 knots in velocity.

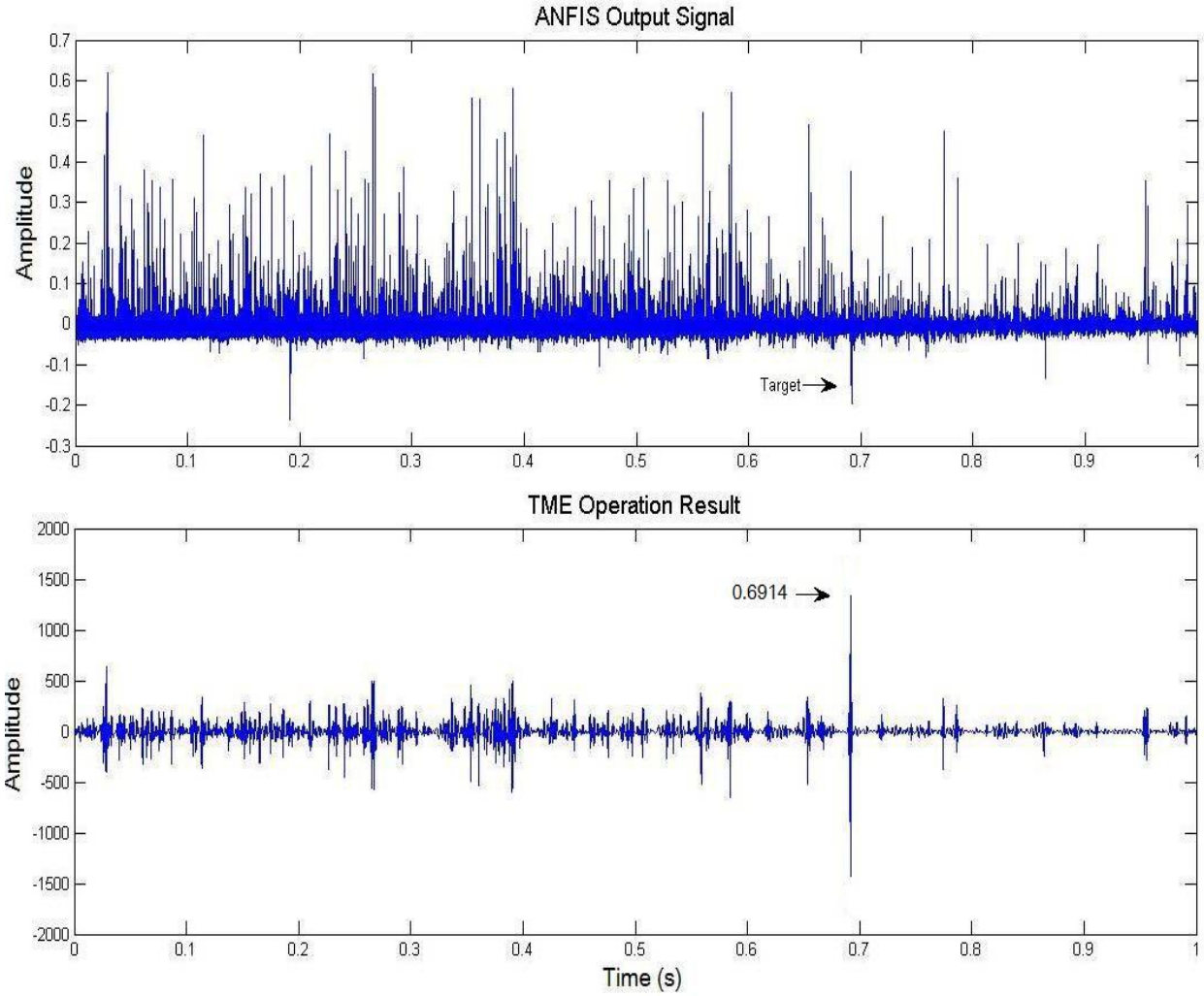


Fig 5.21 Signals after TME Operation SNR=-30dB before ANC Operation, -2.65dB after ANC

Fig 5.22 shows the performance of ANFIS model under each different SNR environment. The SNR achieved after ANFIS operation is -2.65 dB, a little less than results from high level simulation, which is -2.45dB in Chapter IV. However, this change is expected because by transferring high level model into practical implementation, some data precision will be scarified to ensure the platform has enough resources for computation. Fig 5.23 shows the error rate of simulation results for different sets of input signal data. In the figure it shows the maximum error rate in position estimation is less than 1% and the

error range of velocity estimation is from -1 knots to 1 knot. It is no surprise to see that the maximum location error appears at the environment with SNR of -30dB. Because under this environment with a lower SNR the noise has more effect on the target detection than in the other environments with a larger SNR. From the simulation results above it can be concluded that the hardware system's functionality is what is expected and the system is able to work correctly in real applications.

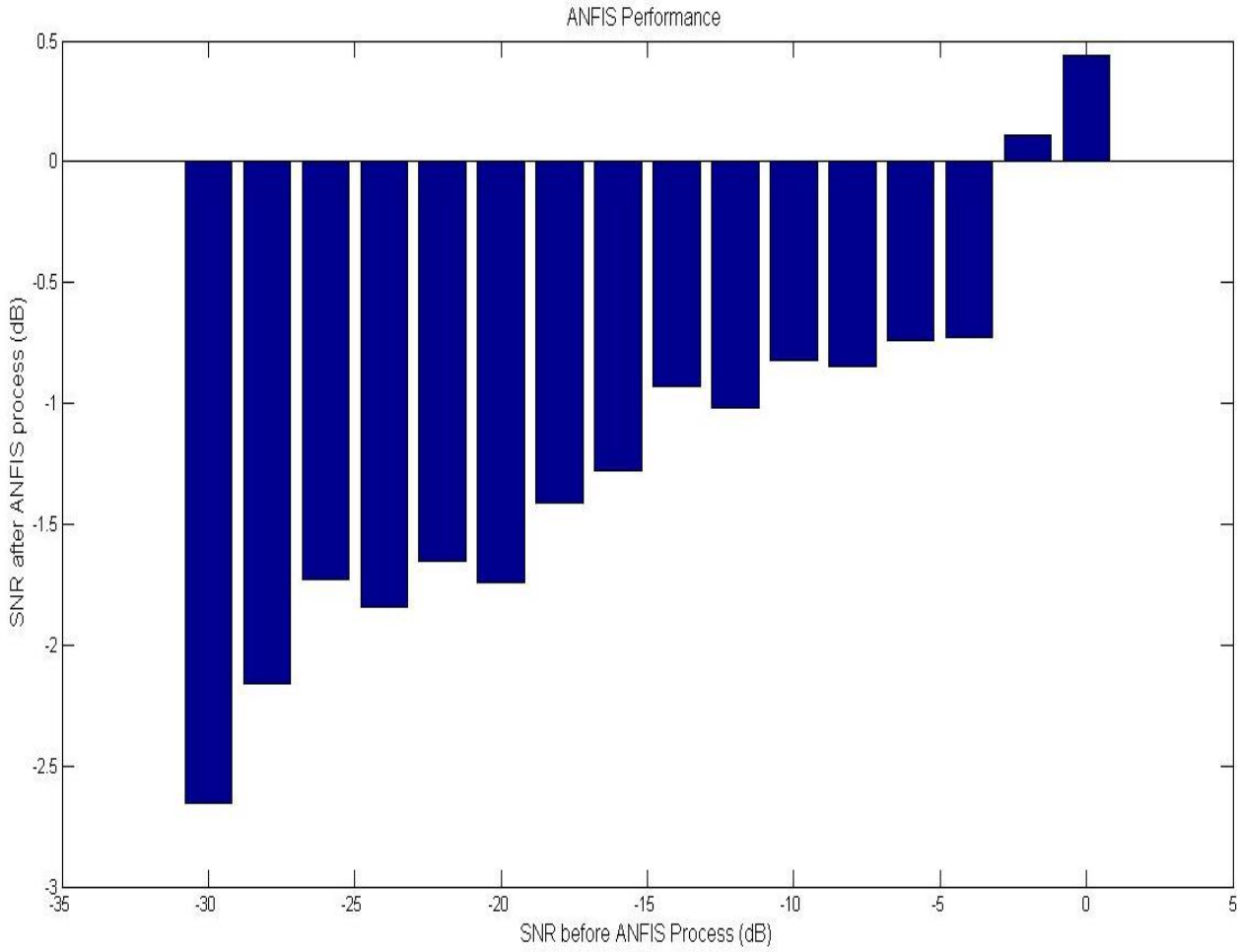


Fig 5.22 ANFIS performance

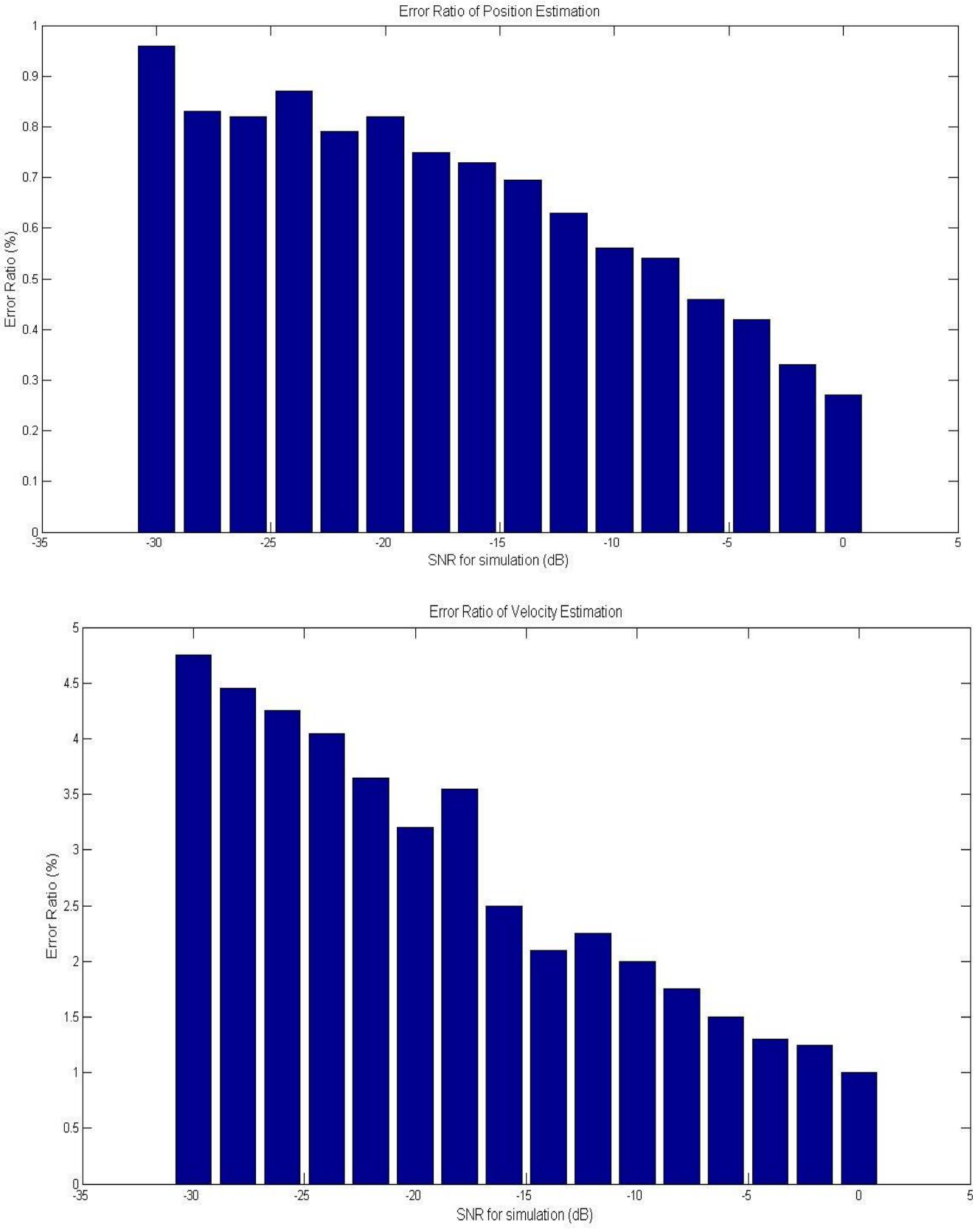


Fig 5.23 Position Error & Speed Error for Single Target

5.6 Multiple Target Detection

Following the single target detection, multiple target detection was investigated using the proposed system. For multiple target detection simulation, the composition of noise and the environment settings was the same as in single target detection test. The transmitted signal also remained unchanged. However, one more target signal is added and the returned echo signals therefore contain 2 targets with different location and different velocity. Fig 5.24 shows an example of a multiple target detection process.

The first graph shows the received signals after the ANFIS operation. After TME processing two target signals stand out. One positioned at 0.371s and the other at 0.751s. The amplitude of the first signal is stronger and is marked as the first target. This signal is removed and the TME operation will be executed again so that the information of the second target can be extracted more precisely. The last graph shows that the second target has also been extracted. Then this target is also been marked as a target and removed. Then the TME operation will examine the rest of received signals for the next target until no signal is beyond the amplitude threshold and then the system decides to stop target seeking and report information of previous two targets. In order to achieve the amplitude threshold for stop criteria of multiple target detection, signals without any target is fed into the sonar system and the results are examined. The maximum amplitude of the output of sonar system will be used as reference of the threshold. Fig 5.25 shows the result of this operation. Simulation results from other different sets of input data, as presented in fig 5.26, shows even under the severe noisy environment

with SNR of -30dB, the position error rate for both targets is still within 1% and the estimation speed error rate is still within 5%. Therefore, the system fulfills the original specifications in the case of multiple target detection as well.

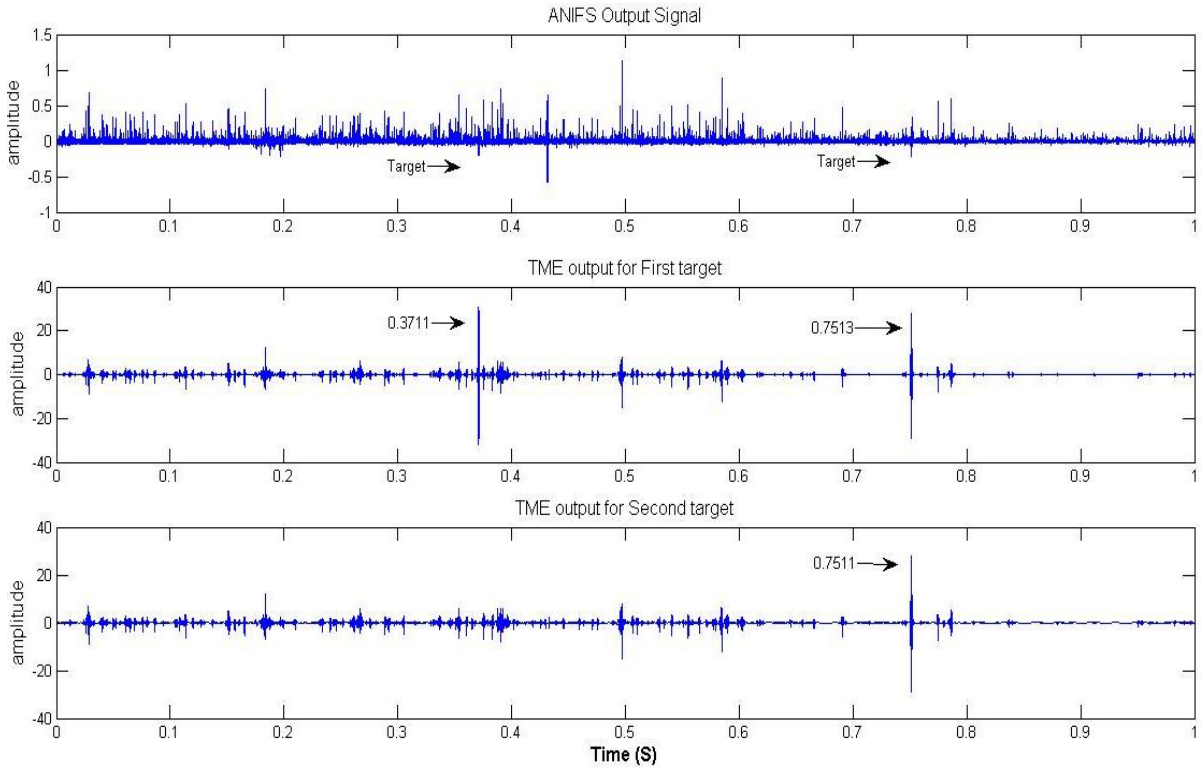


Fig 5.24 Multiple Target Detection under SNR=-26dB before ANC operation, -1.7 dB after ANC

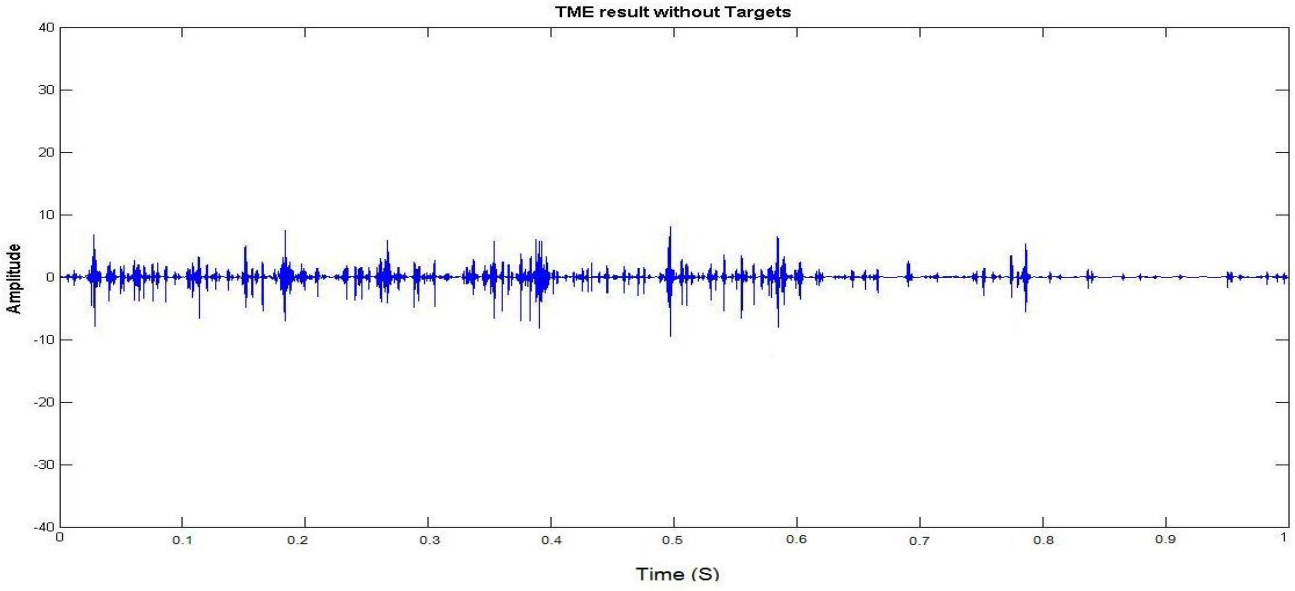


Fig 5.25 Multiple Target Detection with no target

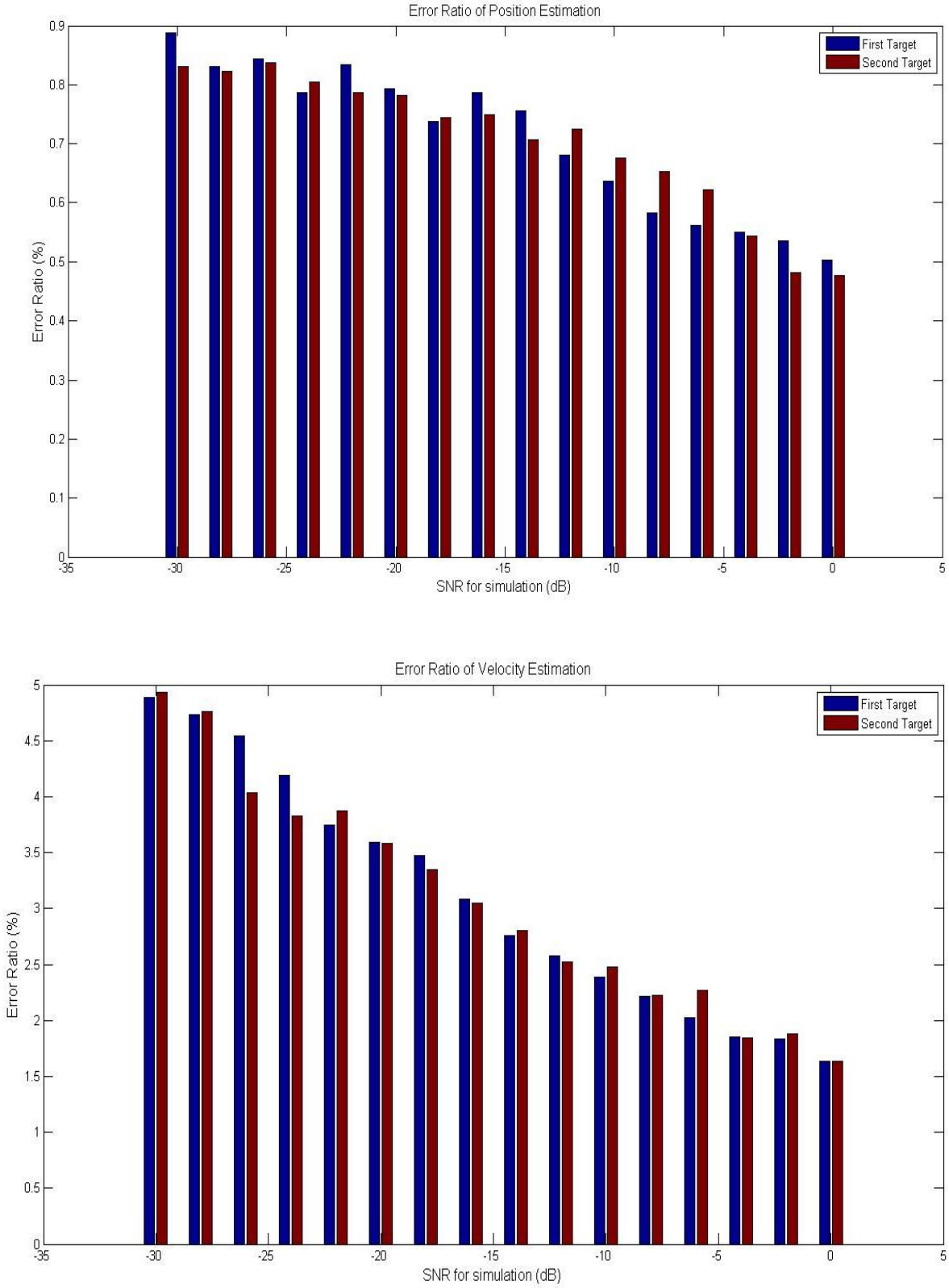


Fig 5.26 Error rate of Speed and Position Estimation

5.7 Conclusion

This chapter introduces the implementation details of the proposed system. The hardware block diagram is firstly presented. The hardware platform is the Xilinx XUP Virtex II Pro Development System which contains a Virtex II pro XC2VP30 FPGA chip with 2 powerPC 405 cores and a comprehensive collection of peripheral components that can be used to communicate with PCs. Then the ANFIS implementation using the Xilinx EDK 10.1 and ISE 10.1 environment is described. Following the design considerations and tradeoffs during the implementation, the TME block is also presented in detail. In total the whole design only occupied 9 of out 136 18x18 bit multipliers, 32% slices and 79% of the block RAM on the XC2V30P chip. At last, simulation results for functionality verification are given. Single target detection and multiple target detection are both investigated and simulated on the proposed system. The noise used in the simulation is provided by the DSTL of MOD to simulate the real underwater environment. Simulations are carried out under reverberation-limited underwater environments with different SNR ranging from 0 to 30dB. The simulation results for a single target detection show the maximum location error appears at the lowest SNR environment. In simulations for single target detection, the maximum location error is 6 meters and the speed error is 1 knot. In terms of error rate against total range for speed and location, the location error rate is less than 1% and the speed error rate is less than 5%. Therefore the proposed system satisfactory fulfils the specifications for this project. In multiple target detection the maximum location error is

10 meters and the speed error is 1 knot, still with error rate less than 1% for location and 5% for velocity. The simulation results clearly show the proposed system is able to fulfil the specifications of this project.

5.8 References

1. Jyh-Shing Roger Jang, "ANFIS: Adaptive-Network-Based Fuzzy Inference System", IEEE Transations on Systems,Man,and Cybernetics, Vol 23,No 3,May, 1993

2. C. H. Tseng and M. Cole, "A hybrid algorithm based on neuron-fuzz and wavelet transforms for wideband sonar detection in a reverberation-limited environment", schedued for publication in EUSIPCO 2007.

3. Stephane Mallat, "A Wavelet Tour of Signal Processing", Academic Press, 1997

4. Chaitali Chakrabarti, and Mohan Vishwanath, "Efficient Realizations of the Discrete and Continuous Wavelet Transforms: From Single Chip Implementations to Mappings on SIMD Array Computers", IEEE Transctions on Signal Processing, Vol. 43, No. 3, March 1995

5. Sheng Cheng, Chien-Hsun Tseng and Marina Cole, "An Efficient and Effective VLSI Architecture for a Wavelet-based Broadband Sonar Signal Detection System", IEEE International Conference on Electronics, Circuits, and Systems, 2007, pp593-596, 2007

6. Xilinx University Program Virtex-II Pro Development System-Hardware

CHAPTER VI

TESTING OF HARDWARE SYSTEM

6.1 Introduction

At the beginning of this chapter the testing environment of the proposed sonar detection system will be presented. The hardware platform will be introduced and details will be given. Two different tests of the proposed system on the hardware platform will be described. The first set is testing for single target detection, where input data only contains one single target buried in noise with different SNR ratios. The system will be tested under 16 sets of single target data with SNR ranging from 0dB to -30dB with intervals of every 2dB. Then the system will also be tested for its ability for multiple target detection. The input data contains 2 target signals and also the system will be tested for 16 sets of testing data with SNR ranging from 0dB to -30dB. Finally the

results of testing will be presented.

6.2 Testing Environment and Hardware Platform

The proposed system is implemented on a Xilinx University Program Virtex-II Pro Development System. It contains a Virtex II pro XC2VP30 FPGA chip with 2 powerPC 405 cores. The board provides a comprehensive collection of peripheral components that can be used to communicate with PCs. The block diagram is shown in Fig 6.1:

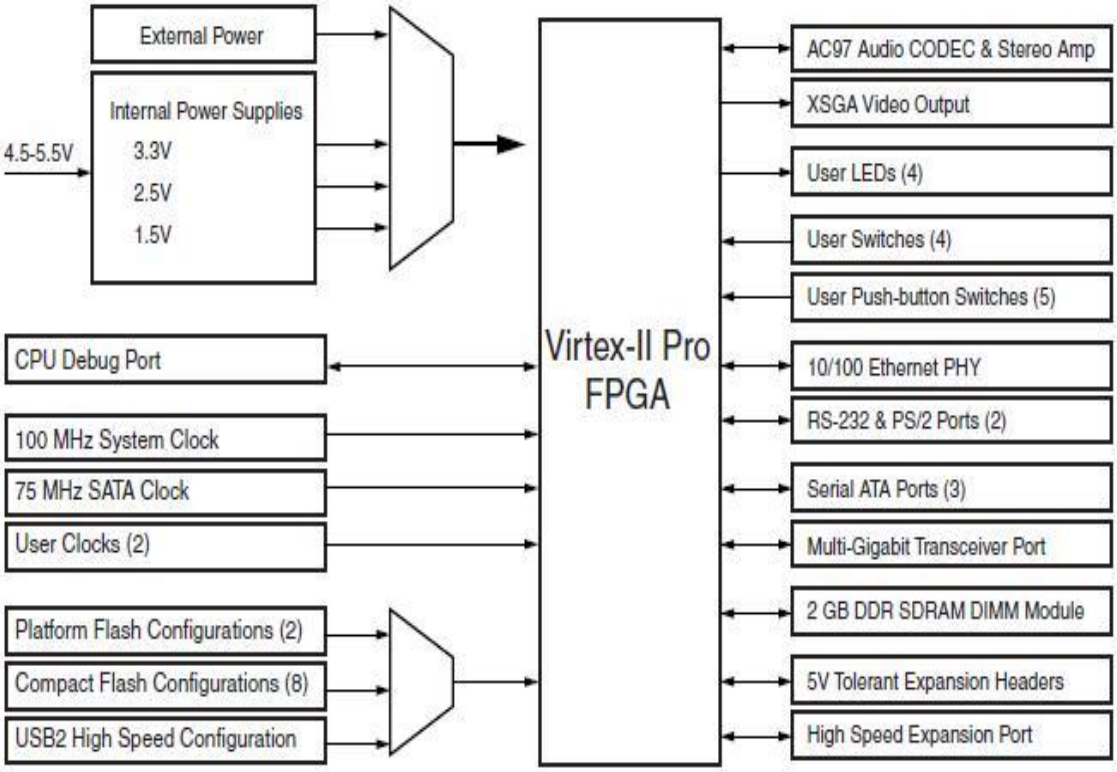


Fig 6.1 XUP Virtex-II Pro Development System Block Diagram

The on-board system clock is 100MHz. In the XC2VP30 FPGA chip on board there are 13969 slices, 428Kb distributed RAMs, 136 18bit multipliers, 2448Kb block RAMs, 8 Digital Clock Manager (DCM) and 2 embedded PowerPC 405 RISC cores. A 256MB DDR RAM is added to the DDR SDRAM DIMM socket on the board. This RAM is

used to store the input data for ANFIS block such as received signal as training data, ambient noise and reverberation noise, etc, output results from ANFIS block and TME block, membership function parameters for ANFIS, if-then rule parameters for ANFIS block and partial computation results from ANFIS and TME. All implementations are compiled, synthesised, placed and routed on the XCV2P30 using XILINX ISE 10.1 and XILINX EDK 10.1.

The test board is connected to a PC using a single RS-232 port. The PC will first send the input data into the onboard RAM through the RS-232 port, and then the ANFIS program compiled in instruction RAM on the FPGA chip will give instructions to the embedded RISC core-PowerPC 405 to fetch input data from the onboard RAM and perform the ANFIS operation. When the ANFIS operation is finished, all the data will be saved in the RAM, a LED light will be turned on and the TME logic will start working on the post-ANFIS data. When the target motion parameters are extracted, all information will be stored into the RAM and the output will be printed on the Hyper-terminal on the PC screen via the RS-232 port.

6.3 Single Target Detection Test

In order to characterize the performance of the system the transit 2ms long morlet wavelet signal is adopted as the transmitted signal. A returned echo signal where the target signal is buried at time of 0.4315 s in the noise with a SNR (signal-noise ratio) of -28dB is used as received signal. The noises are constituted of white Gaussian noise and a reverberation waveform to simulate the real underwater environment amid the same environment settings as used in simulation in Chapter IV and V. This signal is sampled at 100 kHz in the maximum time range [0, 1] sec. Besides the returned echo signal with SNR of -28dB, another 15 sets of returned echo signal with different SNR range from 0dB to -30dB are also tested on the hardware platform.

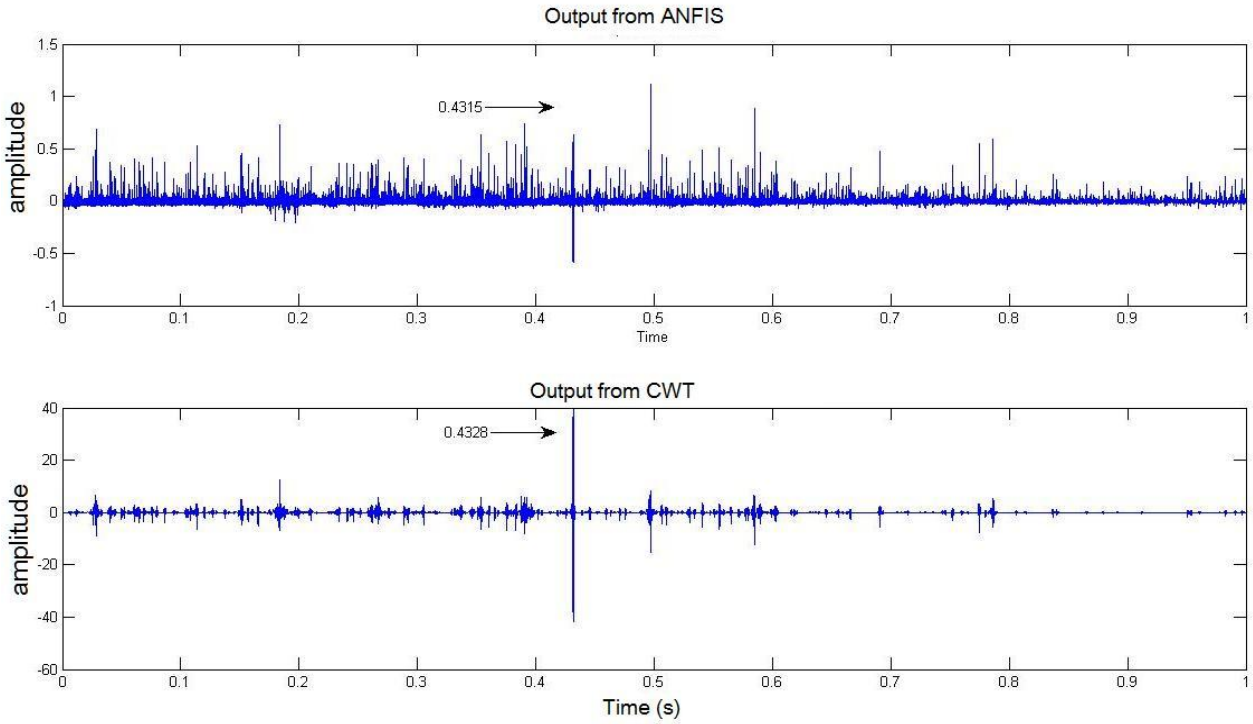


Fig 6.2 Signals after TME Operation SNR=-28dB before ANC operation, 2.16 dB after ANFIS

As shown in the fig. 6.2, the first signal is the input signal fed into the TME block, which

is the output of ANFIS with SNR reduction from -30dB to -2.16 dB. The following graph in fig. 6.2 represent output signal after the TME operation. In the graph shown above, the peak coefficient appears at the time 0.4328 s, which compares favorably with the theoretical result of 0.4315 s. By detecting the position of the highest coefficient, the time delay τ in Equation 4.2 is known. Also the corresponding scale s can be determined by examining which set of filter coefficients produces the highest CWT coefficient. Therefore, the motion parameters such as position and velocity can be determined using the definition of τ and s outlined in chapter IV. In this simulation, the calculated target location is 323.85 meters away, and velocity is 34 knots apart from the receiver, which compares favorably with real location of 322.85 meters away and 33 knots apart from the receiver. Furthermore, 15 more sets of input signals with different SNR range from 0 dB to -30dB are examined through hardware architecture. Fig 6.4 shows the error of testing results of different sets of input signal data. In the figure it shows the maximum error rate in position estimation is below 1% and the error rate of velocity estimation is below 5%. As shown in Fig 6.3, with the decrease of SNR the location error increases. This is because when the SNR decreases, the noise is increasing compared to the target signal. Therefore it is harder for the proposed system to eliminate the noise effect on target detection. However, with only 1% error rate in location error and less than 5% error in speed estimation it can be concluded that the hardware system is able to work correctly in real applications in an underwater environment for single target detection.

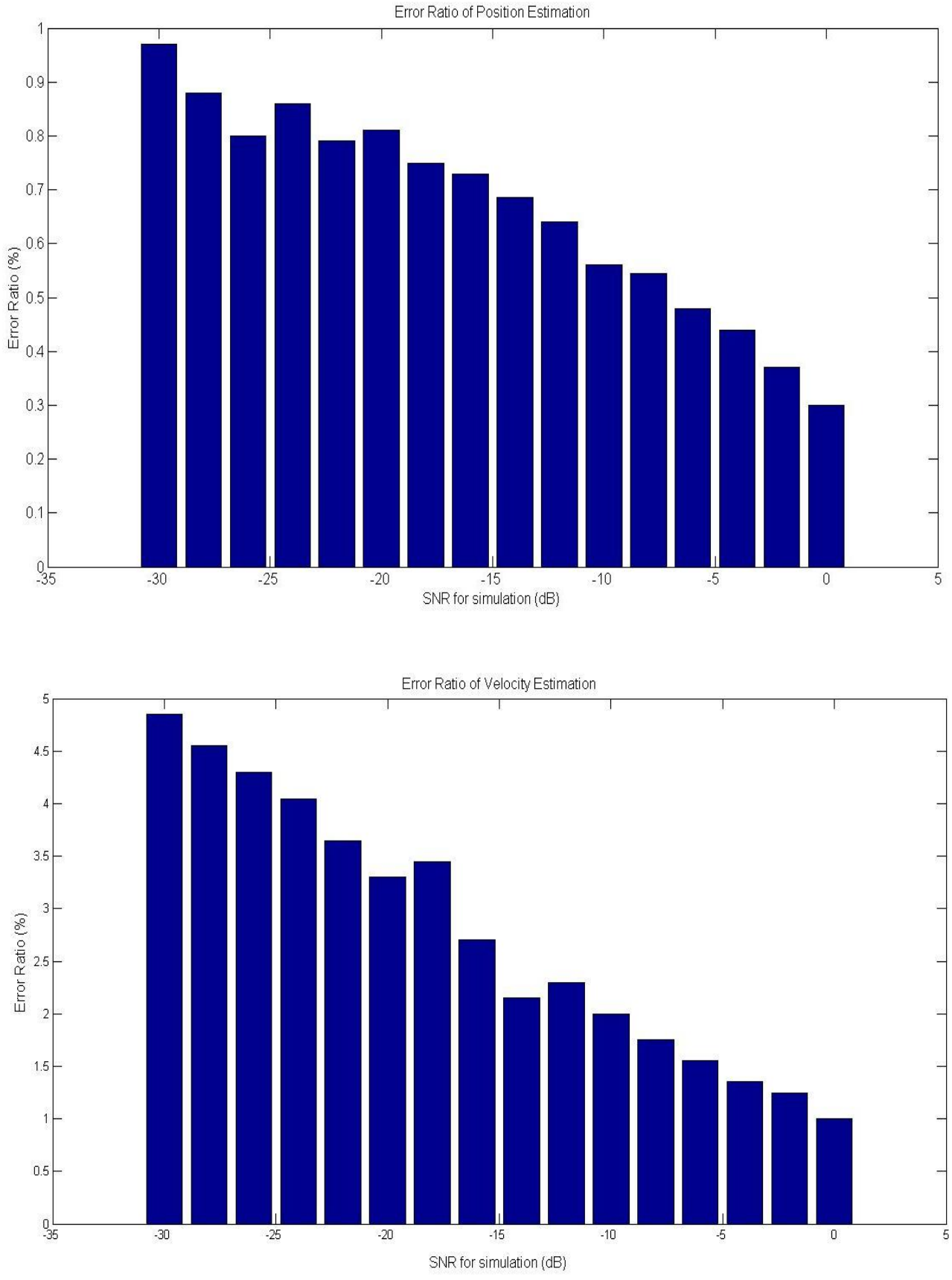


Fig 6.3 (a).Position Error (b) Speed Error

In Fig 6.4 shows for how much iteration the Optimization Block runs. As the SNR gets lower and lower, the iteration numbers increases. However, even at the lowest SNR, -30 dB, the optimization block only runs for 18 times, which is almost 10 times less than traditional method by examining all the possibilities. Therefore, the computation load is almost 10 times less than for the global search algorithm.

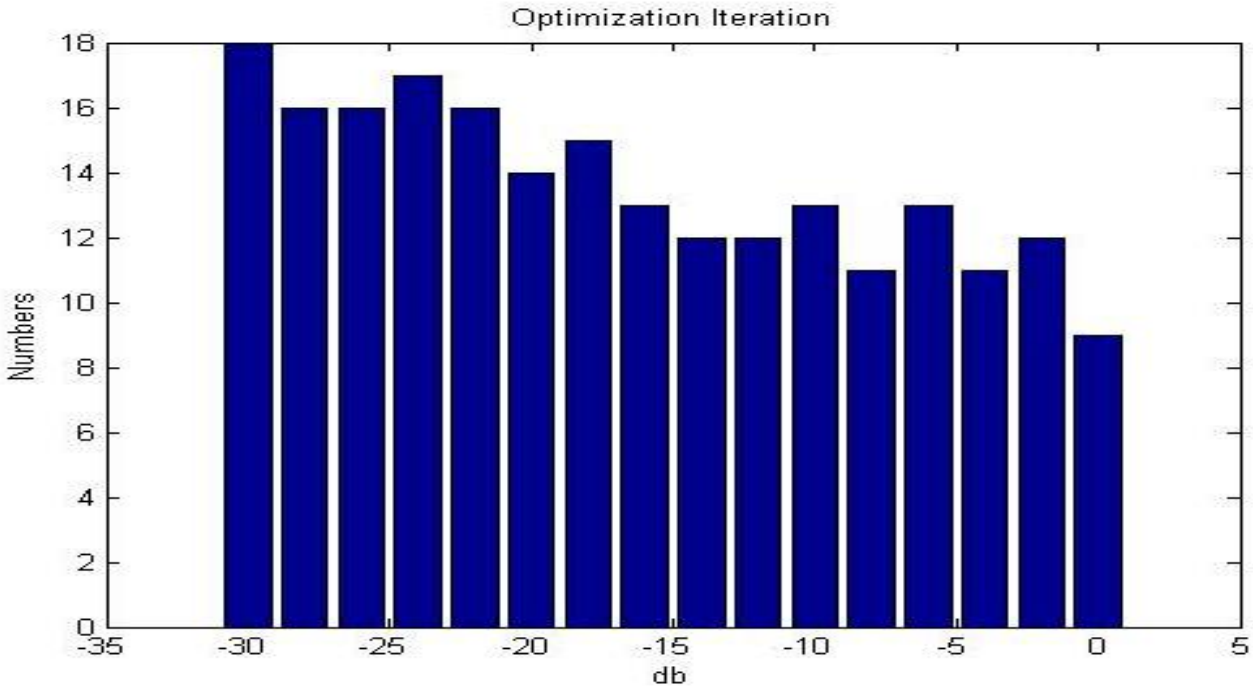


Fig 6.4 Optimization Iteration Numbers

Fig 6.5 shows how many ANFIS epochs have been executed at noise with different SNR. It is clear that when the SNR is high, it will take no time for the ANFIS to remove noise. However, when the SNR decreases, the computation iterations will increase to deal with this.

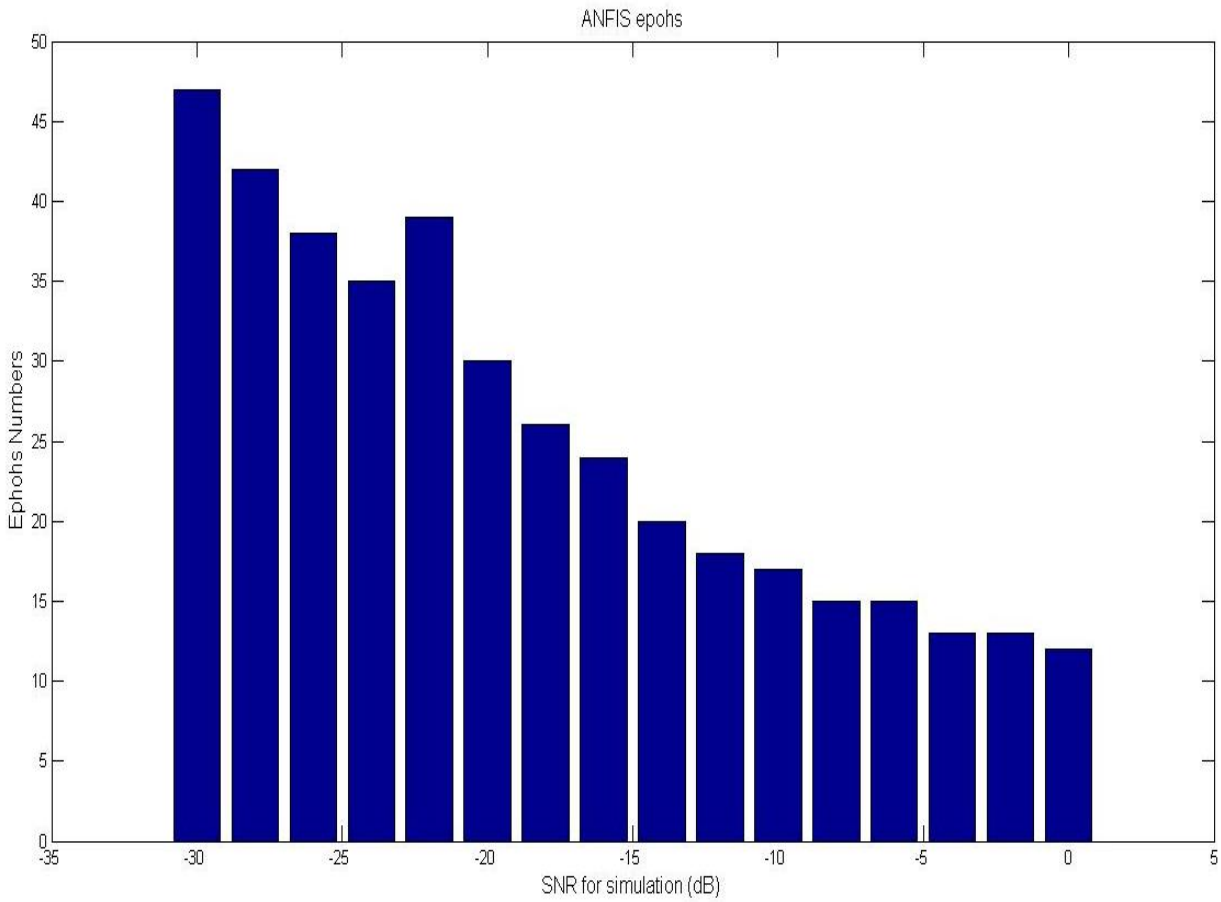


Fig 6.5 ANFIS Epochs

6.4 Multiple Target Detection Test

The noise still constitutes a white Gaussian noise and a reverberation waveform. The environment settings are still the same as in the single target detection test. The transmitted signal also remains unchanged. However, the returned echo signals will contain 2 targets with different locations and different velocity. Fig 6.6 and Fig 6.7 shows an example of a multiple target detection process in an environment of SNR of -20dB.

In Fig 6.6 two target signals without noise are shown. These two targets are assumed to be recorded by the receiver at 0.333s and 0.775s, respectively. In Fig 6.7 the first graph shows the output of the ANFIS operation. Then after the TME processing the two target signals stands out. One is at 0.336s and the other at 0.772s. The amplitude of the signal at 0.772s is stronger and then it is marked as the first target and this signal is removed. The TME operation will be executed again so that the information of the second target can be extracted more precisely.

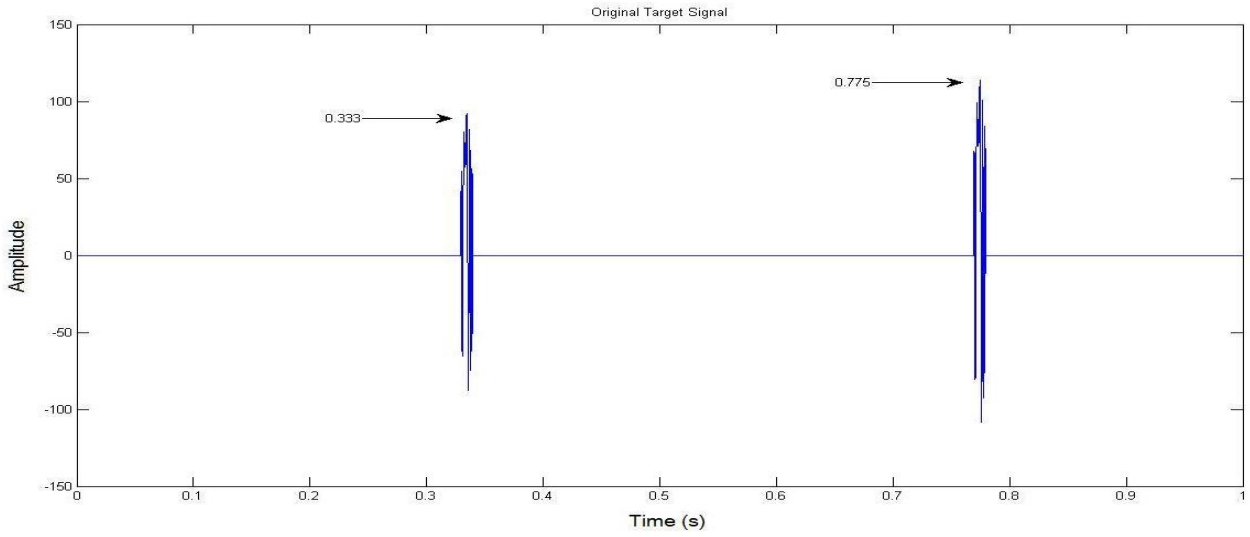


Fig 6.6 Original Target Position

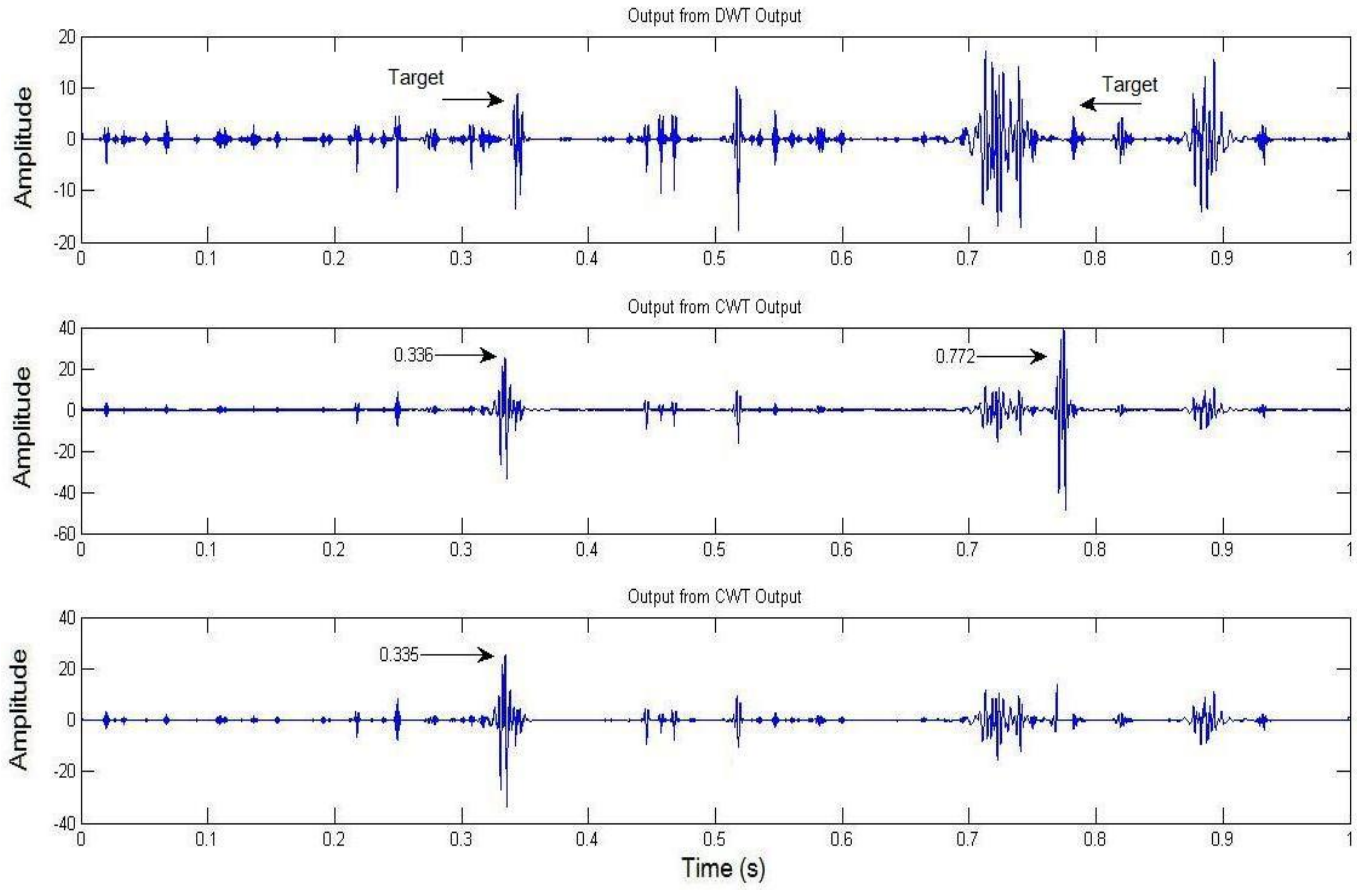


Fig 6.7 Multiple Target Detection under SNR=-20dB before ANC Operation, -1.7 dB after ANC

The last graph shows that the second target has also been extracted. Then this target has also been marked as a target and removed. Then the TME operation will examine the rest of received signals for next targets. If no signal is beyond the amplitude threshold as shown in Fig 6.8, then the system decides to stop target seeking and report the information of the previous two targets. Therefore, the two targets are estimated to be

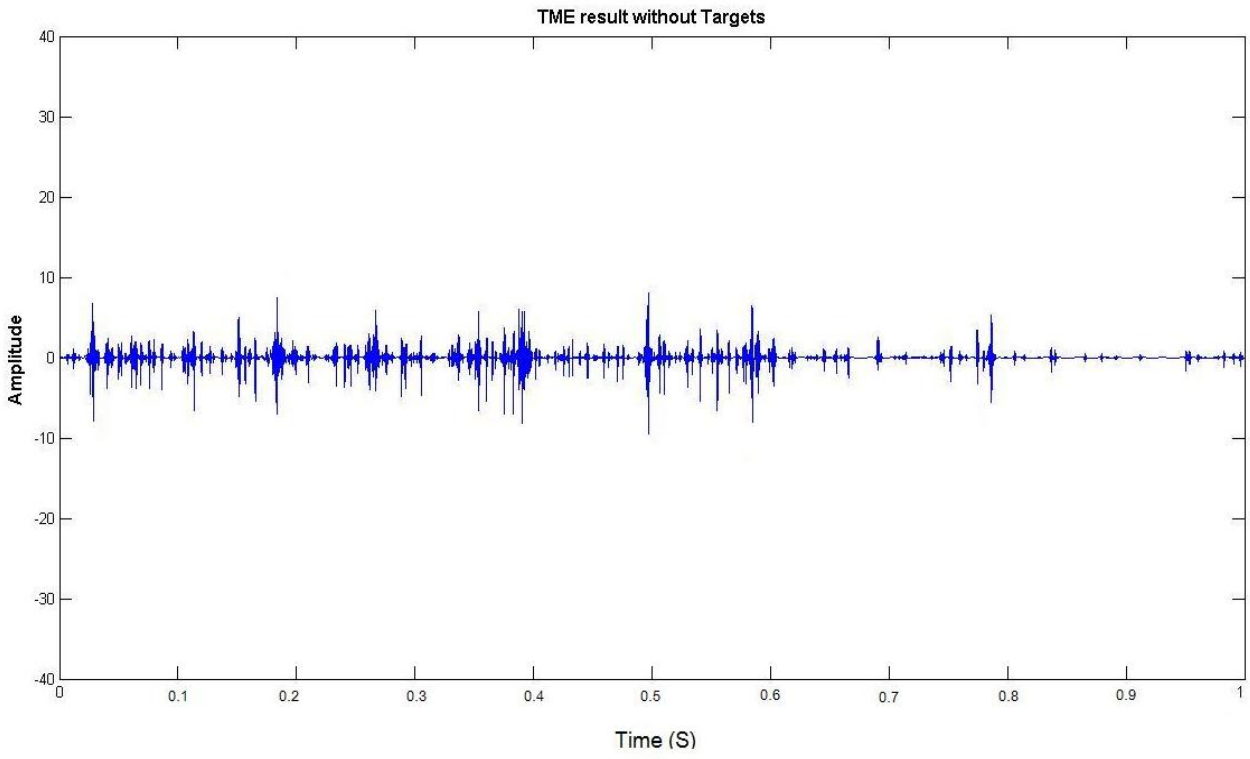


Fig 6.8 TME results for no target signal

at 0.772s and 0.335s, respectively. According to Eq 4.2, the location information can be easily computed. Since the sound speed underwater is 1500m/s, and the time estimation error is 0.003s, therefore the location error in this simulation is only around 2 to 3 meters.

Fig 6.9 and Fig 6.10 show the Location Estimation Error and Velocity Estimation Error rate of the system under different SNR environments for multiple target detection. It is clear that the speed error rate for both targets less than 5% and the location error rate is less than 1%. Therefore it is safe to claim the proposed sonar detection system for multiple target detection can fulfill the original specifications mentioned in Chapter III.

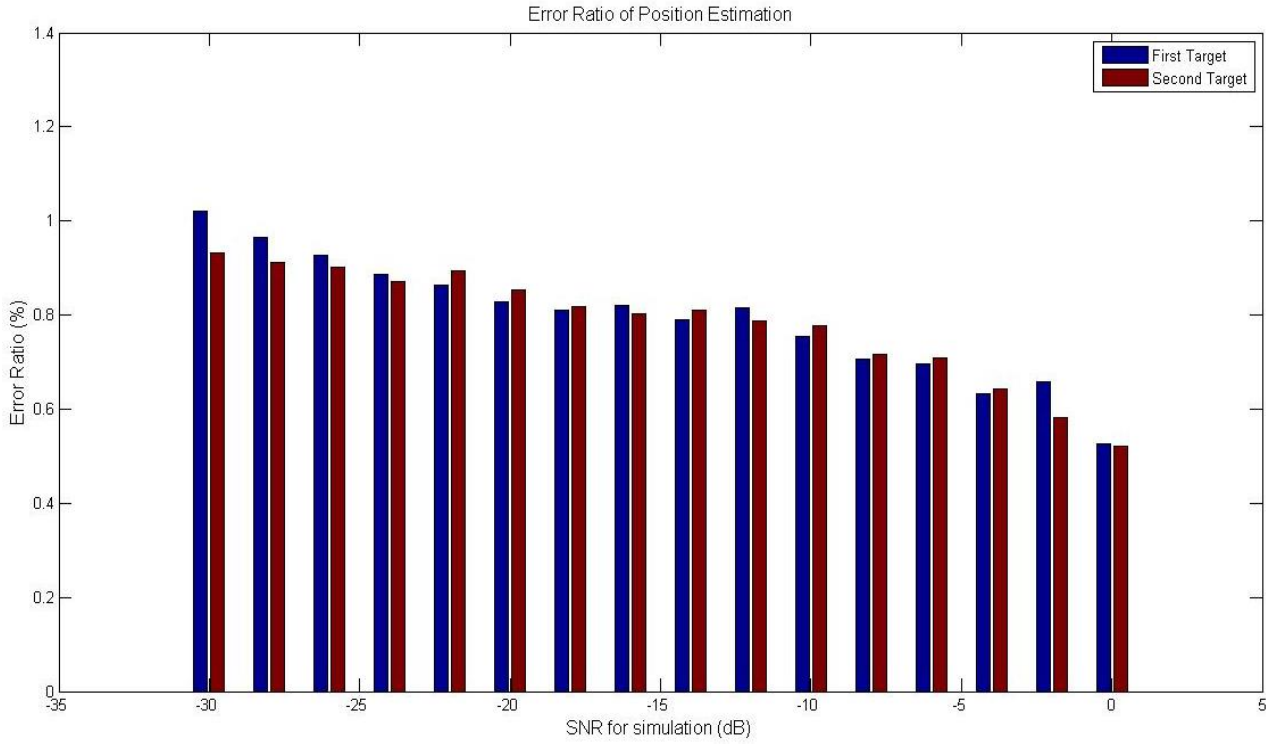


Fig 6.9 Position Error for Multiple Target Detection

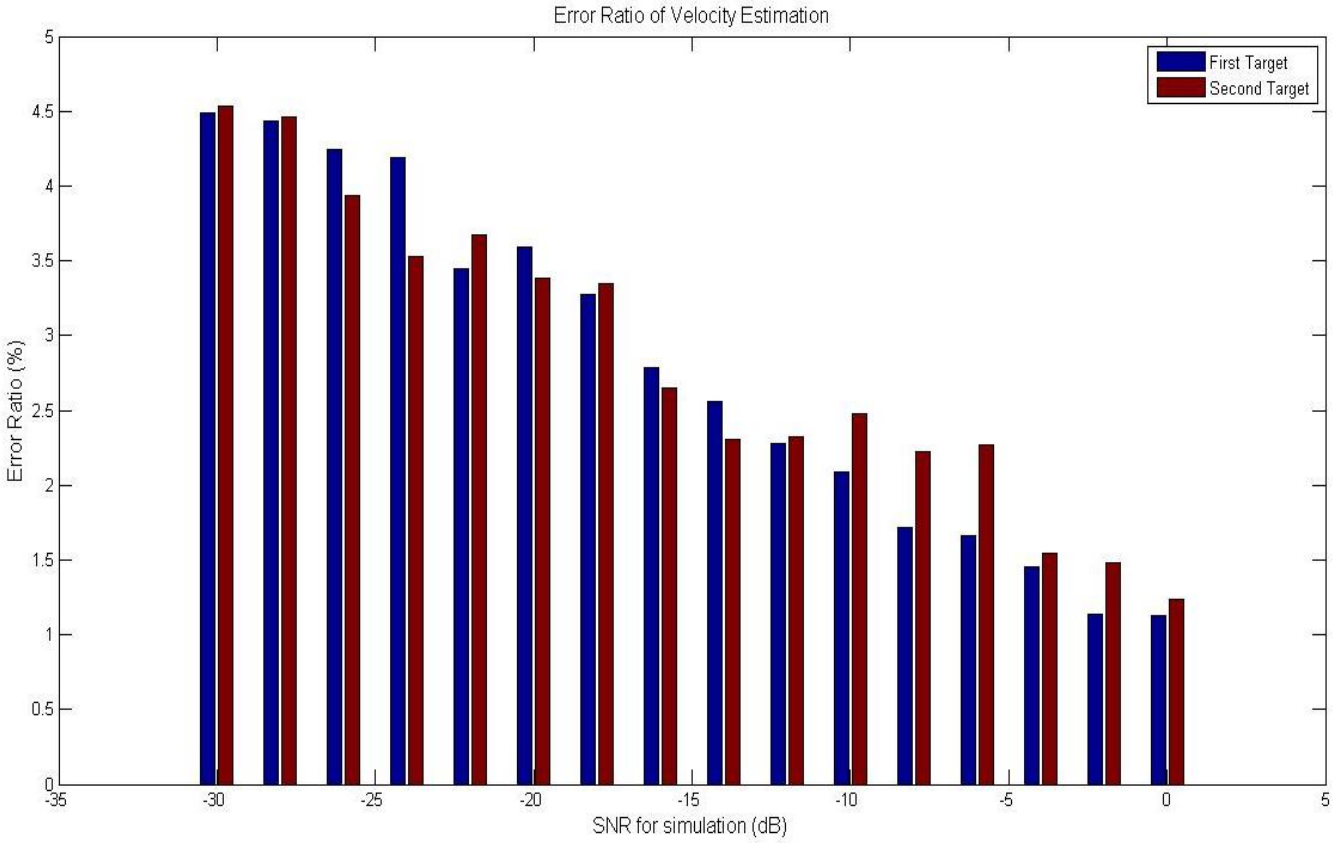


Fig 6.10 Speed Error for Multiple Target Detection

6.5 Conclusion

This chapter introduces specifications of hardware design platform for implementation of the hybrid algorithm. It gives a general test flow from generating stimulus data for hardware testing to data processing during the test. The single target detection is analyzed on the first instance. With the noise data provided by the DSTL (Defense Science and Technology Laboratory) of MOD (Ministry of Defense), real underwater environment with different SNR ranging from 0dB to -30dB is tested on the hardware platform. The proposed system gives satisfactory performance for target detection with a maximum location estimation error rate less than 1% and velocity error rate less than 5%. In absolute values, the error of location is within 10 meters and a velocity estimation error within 1 knot. The computation time is increased when the decrease in SNR because more iterations are required for the ANC to minimize the noise effect. The multiple target detection is also presented and simulated under same conditions for single target detection. The simulation results have demonstrated that the performance of the system is better than the original specifications given in chapter III.

CHAPTER VII

ANALYSIS & DISCUSSIONS

7.1 Introduction

During the design complete process, the system undergoes three design stages. Firstly, the high level system model is built in Matlab and high level simulation on the proposed system for single target detection and multiple target detection under real underwater environment is carried out to verify the performance of the system. The proposed system is then implemented on a hardware platform in the Xilinx design environment through XILINX ISE 10.1 and XILINX EDK 10.1. The hardware code is tested in Mentor Graphics Modelsim XE 6.4b using the same stimulus as the simulations used for the Matlab model for both single target detection and multiple target detection. The last step of design process is to test the proposed system on real hardware platform. The system is tested on the Xilinx University Program Virtex-II Pro Development System. For three levels of simulation and testing, the input data used are the same 16 sets of single target data with SNR ranging from 0dB to -30dB with intervals of every 2dB, and the same 16 sets of multiple target data with SNR range from 0 to -30dB. In this chapter,

the simulation results for single target detection achieved in the order of high level simulation, hardware simulation and hardware testing will be discussed. The simulation results of multiple target detection will be analyzed in the same order following the discussion of single target detection.

7.2 Single Target Detection

There are 16 sets of input data for single target detection with different SNR ranging from 0 to -30dB. The transit 2ms long morlet wavelet signal depicted in Fig. 7.1 is adopted as the transmitted signal. The sample frequency is 100 kHz and the central frequency is 20 kHz. Fig 4.5 illustrates the composite signal where the contact signal is buried in the noise with the target strength of approximate -24dB at the position of 0.8359s with velocity of 20 knots. This signal is sampled at 100 kHz in the maximum time range [0, 1] sec. The noise consists of white Gaussian noise and reverberation waveform to simulate the real underwater environment, which is provided by the MOD, for the following environment settings: sea depth (=100m), sonar depth

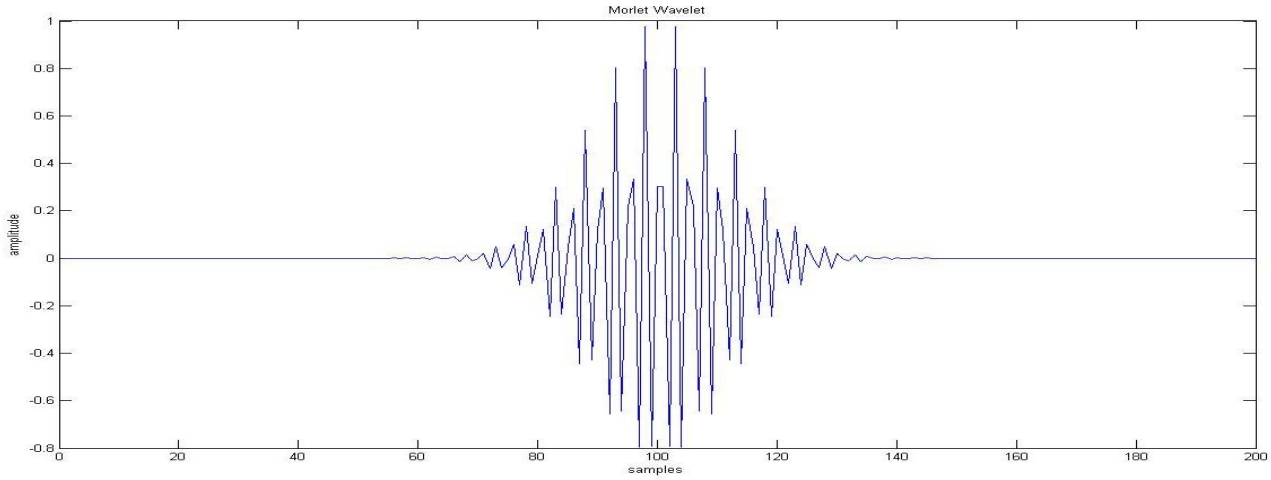


Fig 7.1 Transmit Signal

(=50m) wind speed (=6m/s ~ sea state 3), seabed type (=medium sand).

As explained in Chapter IV, the returned echo is fed into an Adaptive Noise Canceller to minimize the noise effect. To characterize the performance of the proposed system, the system is tested by 16 different sets of input data with different SNR ranging from 0dB to -30dB. The statistics of the location error and the speed error under different SNR conditions are shown in Fig 4.11 in Chapter IV. It is clear that the location error is less than 2 meters and the speed error is less than 1 knot. Since the original specification only requires the system to be able to determine the speed of a moving target up to 50 knots with the resolution of 1 knot . In terms of error rate against total range for velocity and location, the numbers are within 4% and 0.5%, correspondly. Therefore, it s safe to conclude that based on the Matlab simulations the proposed system can fulfil the specifications for this project for single target detection.

These 16 sets of data are also simulated in Modelsim to verify the functionality and performance of the proposed hardware-based system. The transmit signal remains the same as shown in Fig 7.1, and all the returned echoes are buried in severe noise. The statistics of all 16 simulation sets are presented in Fig 5.23 in Chapter V. In the figure it shows the maximum error in position estimation is below 6 meters and the error range of the velocity estimation is from -1 knots to 1 knot. It is no surprise to see that the maximum location error appears at the environment with SNR of -30dB. In environments with lower SNR, the noise amplitude is larger and it will be harder to remove the noise effect on target detection. It should also be noticed that even the speed resolution remains under

1 knot in the hardware based simulation; the resolution of speed estimation is decreased to 0.5 knot because there are only 121 sets of the filter coefficients to represent the velocity range from -60 knots to 60 knots, where negative velocity indicates the target is moving away from the receiver and positive velocity means the target is moving toward the receiver. In general the location error in the hardware-based simulations is bigger than the location error of simulations in matlab because in hardware the precision of the computation is compromised due to hardware limitations. Taking the TME implementation as an example, in matlab simulations there are 10000 versions of dilated mother wavelets for target detection. However, the hardware cost will not be affordable if all versions of dilated mother wavelets are transplanted into hardware platform. Therefore, only 121 versions of dilated mother wavelets are introduced in the proposed system. Same tradeoffs happen in implementations of membership functions in the ANFIS operation as described in Chapter V. Therefore, computation errors are inevitably increased.

The same test data are used for hardware testing on the real development board. The proposed system is realized on a Xilinx University Program Virtex-II Pro Development System. In chapter VI, the testing results are presented. The statistics of all 16 sets of testing data is shown in Fig 6.4. Most of the testing data fits the simulations on hardware codes in Modelsim.With only less than 1% error rate in location and 4.5% error rate in velocity, which in absolute values are 8 meters in the location error and 1 knot error in speed estimation, it can be concluded that the hardware system is able to work correctly

in real applications in an underwater environment for single target detection.

7.3 Multiple Target Detection

Multiple target detection is first simulated in Matlab on the high-level system.

The overall simulation results for multiple target detection in Matlab are shown in Fig 4.16, Chapter IV. The location error is still within 1 meters and velocity estimation error is still less than 1 knot for both targets.

The same simulations are carried out in Modelsim for functionality and performance verification of the proposed hardware implementation and in the testing procedure of hardware itself. The simulation error statistics are shown in Fig 6.10 in Chapter VI. The testing results fit the results in the Modelsim simulation very well. From Fig 6.10 it is clear that the speed error for both targets is less than 1 knot and the location error are less than 10 meters. The error rate for speed is within 5% and location error rate is within 1%. Therefore it is safe to claim the proposed sonar detection system for multiple target detection can fulfill the original specifications mentioned in Chapter III.

7.4 Discussions

As mentioned in previous chapters, Active side-scan sonar [1-4] is also commonly used for forming high resolution images of underwater scenes. Side-scan systems form an image of a target scene by transmitting a succession of sound pulses (referred to as ‘pings’) and recording the echoes from the target scene using a very narrow beam width receiver. An image of a target scene is built up a line at a time by moving the

transmitting / receiving platform past the target area. The azimuth resolution of a side-scan sonar system can be increased by using a higher pulse frequency and / or a larger physical aperture. The attenuation of sound in water increases with frequency, creating a tradeoff between azimuth resolution and maximum range of side-scan sonar systems. The maximum physical length of the sonar array is limited by practical constraints.

The advantage of side-scan sonar is that it is able to provide a through image for underwater scenes in a certain area, however, this image cannot provide any velocity information for a certain target and post-processing of imaging is required to extract certain target motion information such as location and velocity. The proposed wavelet-based active sonar system, on the other hand, provides another way to examine target motion parameters. With the similar representation form between the returned echo in a wideband sonar system and the transformed signal by the Continuous Wavelet Transform (CWT), the seeking of target motion parameters can be achieved by finding parameters of returned echo through CWT. Therefore, the proposed sonar system can get the target motion parameters more quickly than side-scan sonar and less resource are required because no array of receivers is needed.

Cross-correlation techniques [5-8] are also widely used in target parameter extraction in sonar system. Although cross-correlation technique works well dealing with ambient noise; its performance on reverberation noises is limited due to the similarity between reverberation noises and target echo. Therefore in environment with

severe reverberation cross-correlation is seldom used alone. With combination of Adaptive noise canceller and wavelet transform the proposed system can deal with both ambient noise and reverberation noises.

Fig 7.2 shows how many ANFIS epochs have been executed at noise with different SNRs. ANFIS is the most time consuming function block in the proposed system. Therefore, the process time of the ANFIS will determine the speed of the proposed system. It is clear that when the SNR is high, it will take no time for ANFIS to remove the noise. However, when the SNR decreases, the computation iterations will increase to deal with more complex noises. When the SNR reached -30dB, it took 47 epochs for the ANFIS to finish the process. In hardware testing, it takes up to 50 seconds to finish the processing. Therefore, reducing the process time is the key factor to improve the speed of the proposed system.

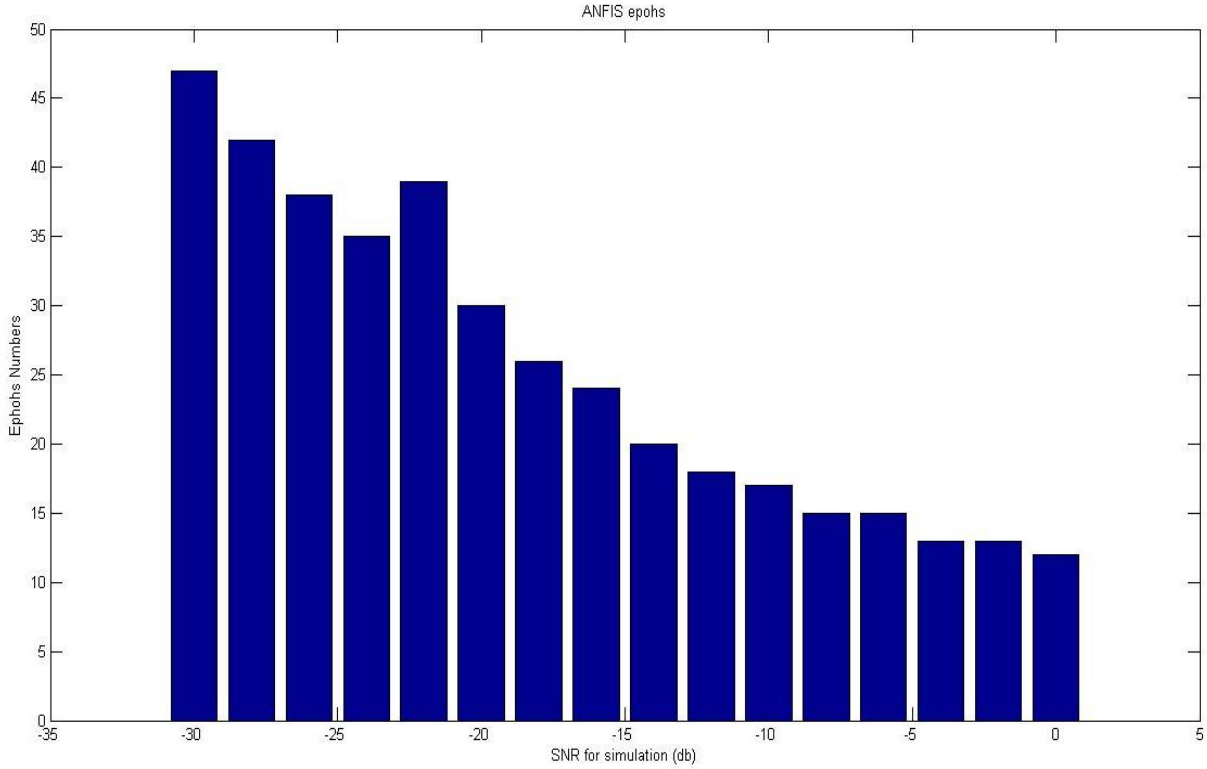


Fig 7.2 ANFIS Epochs

Under -30dB, it takes 2 minutes to complete the whole process on PC but only 58 seconds on FPGA platform. It is obvious FGPA implementation is faster than running the proposed system on PC. However, there are other platforms which can improve the performance of the system. Improving the hardware working frequency is helpful to accelerate the performance in terms of speed. The highest clock frequency of the embedded core for current platform is 300MHz. However, the highest frequency of DSP can reach 900MHz to 1100MHz and if this design is transplanted to platform with advanced DSP chip, the performance of the ANC will be increased significantly. The reason for choosing FGPA platform is the tradeoff between price and performance. The expense of highest performance DSP platform is 10 times higher than current FPGA

platform, but the performance improvements will not reach10 times. Meanwhile, FPGA implementation is closer to ASIC platform implementation, which gives higher speed and less cost. The working frequency of ASIC design can easily reaches 1000MHz or higher, in the meantime the cost is close to the cost of FPGA platform. Considering the possibility of moving proposed design into ASIC platform in the future is one reason to choose FPGA implementation as well.

Besides the ANFIS operation, though the local optimum search algorithm reduces the computation time of the CWT almost by half in this novel hybrid system, the CWT operation is still the most time-consuming block in the TME. As described in Chapter V, the CWT operation is implemented using only five multipliers. The initial aim is to reduce the area consumption as well as the power consumption. However, the synthesized results show that there are still tens of multipliers that have been saved. In order to increase the performance of the CWT, adding some parallel multipliers would be the most convenient method. However, more parallel multipliers mean more power will be required and more area is consumed. Also the hardware complexity is increased. The timing for memory access would become more complex. Some trade-offs can be introduced to increase the performance.

One possible solution and trade-off would be to transfer the hybrid system into ASIC platform. On ASIC platform the cost of the system would be lower and more importantly, the performance of the system will increase because the ASIC platform has faster working frequencies. However, transporting the design from FPGA platform to

ASIC platform would require some changes to the existing design. First of all the memory descriptions in the FPGA design should be changed according to the fabrication factory and fabrication technique since each technique or factory has its own memory libraries. Secondly since the ANFIS part is written in C and compiled by XILINX EDK 10.1, the ANFIS part should be rewritten into HDLs so that it can be used with the ASIC platform. Additionally all layouts of each functional block should be reconsidered to fit into the fabrication process. In the meantime, the design flow of the ASIC platform is more complex than the FPGA design since the functionality cannot be tested on real chips but only simulated in EDA tools. Therefore, the design flow of the ASIC platform requires more care and it might take considerable time to move the FPGA design to the ASIC platform. However, the performance of the current system would be improved significantly on ASIC platform.

References

1. Lianantonakis, M.; Petillot, Y.R,” Sidescan sonar segmentation using active contours and level set methods”, Oceans 2005 – Europe, vol 1, pp 719-724,2005
2. Tarlet, O.; Loussert, A.; Sintes, C,” Comparison of sonar images with well-known differences from raw data”, OCEANS 2007 – Europe, pp 1-3, 2007
3. McCarthy, E.M.; Sabol, B; “Acoustic characterization of submerged aquatic vegetation: military and environmental monitoring applications”, OCEANS 2000 MTS/IEEE Conference and Exhibition, vol 3, pp 1957 – 1961, 2000
4. Dura, E.; Yan Zhang; Xuejun Liao; Dobeck, G.J.; Carin, L,” Active learning for detection of mine-like objects in side-scan sonar imagery”, IEEE Journal of Oceanic Engineering, Vol 30, Issue 2, pp 360-371, 2005
5. Houser, D.; Martin, S.; Phillips, M.; Bauer, E.; Herrin, T.; Moore, P, “Signal processing applied to the dolphin-based sonar system”, Proceedings of OCEANS 2003, Vol.1, pp 297 – 303, 2003
6. Song, H.Y.; Golomb, S.W, “Two-dimensional patterns with optimal auto- and cross-correlation functions”, IEEE International Symposium on Information Theory, 1994
7. Thorne, P.D.; Holdaway, G.P,” Interpretation of acoustic backscatter measurements using a cross-correlation technique to evaluate near-bed current velocities”, OCEANS '97. MTS/IEEE Conference Proceedings, Vol 1, pp 457 – 461, 1997
8. Belmont, M.R.; Maskell, S.J. ,“Rapid scan sonar using generalised

cross-correlation”, Seventh International Conference on Electronic Engineering in Oceanography, pp 139 – 152, 1997

CHAPTER VIII

CONCLUSIONS AND FUTURE WORK

8.1 Aimed Reviewed

A novel hybrid algorithm developed for underwater active sonar echolocation system has been proposed, implemented and tested based on an FPGA technique on a Xilinx University Program Virtex-II Pro Development System. The test results clearly state that the proposed system is capable of extracting exact target location and velocity information under reverberation-limited and shallow-water environment.

A review of the aims of the project, given at the end of Chapter I, shows a positive view on the acheivements. In fact, not only are all the original goals satisfactionary reached, some additional possibilities were also investigated during the implementation of the proposed system.

The hybrid algorithm combing neural network, fuzzy logic, local optimum search and

cross-correlation techniques for underwater sonar echo detection is propsed in a theortical platform, simulated and verified in Matlab, and then implemented on a commonly used FPGA hardware platform. The details of each stage of implementation are descirbed in previous chapters.

As put forward from the aims of the project, simulation results in Matlab and test results feeding back from the hardware platform all indicate this proposed system can exactly extract the target location and velocity information under severe noisy underwater conditions. The ANFIS succesfully removes the majority of noise and significantly increases the SNR. The noises left did not affect the efficiency of target extraction by wavelet transform. The requirement of resolution of target information on location and velocity was then easily satisfied.

Furthermore, the combination of a continuous wavelet transform and local optimum search algorithm kept the resolution advantage of the CWT and signicantly reduces the traditonal heavy computation time and resource requirements. This novel implementation of CWT is able to reduce by around 8 to 10 times the computation load compared to the conventional design.

In the meanwhile, besides the original project specifications, multiple target detection has also been investigated. Once target information is extracted, the target signal will be removed from received echo signals and the TME function block will process the rest of the signal again to examine whether there is another target signal buried in it. If another target is found, then the target information will be stored and this target signal will also

be removed before executing the TME on rest of the signal to check for other target information. These iterations will go on and on until all wavelet coefficients of the rest of the signals are below the threshold, which is defined according to the amplitude of the wavelet coefficients of pure noise.

The implementation of the ANFIS on the hardware system is also an exploration of a new area. The ANFIS is very effective for dealing with the chaotic nature of impulse noise; however, the heavy computation load limits its use in practical applications. In this proposed design, the ANFIS system is implemented using the embedded RISC core in the FPGA chip. 2 inputs and 4 membership functions are adopted in the design. The ANFIS function is written in the C-language, compiled by XILINX EDK 10.1 and stored in instruction rams for the embedded RISC cores. Once the hardware system is processing the input signals, these instructions will direct embedded RISC cores to operate the arithmetic operations needed for the ANFIS and feed back the results to the hardware system. In this way, the translation time for the ANFIS hardware implementation from a high-level language to a hardware language is reduced by the compiler of the XILINX EDK 10.1. Since the embedded RISC cores take most of the ANFIS computations, it will not consume the ordinary FPGA resources. Therefore, the hardware consumption is reduced. Following this method, if the technique of embedded DSP core develops to a certain scale, a real time implementation of the ANFIS on a FPGA would be feasible.

In general, the implementation of the proposed novel hybrid algorithm for an

underwater active sonar echo location system presented in this project, demonstrated the following benefits:

- The novel implementation of CWT by combining CWT with local optimum search algorithm significantly saves computation time for CWT and makes it more feasible to practical applications and real time applications.

- The implementation of the ANFIS on an FPGA board indicates in the future a real-time ANFIS operation VLSI implementation would be possible.

- The hybrid system explore the possibility of using ANFIS and CWT in practical applications and the test results supports the capability of the hybrid system and it is a new way to adopt computation-consuming algorithms such as ANFIS and CWT in practical applications.

- The test for multiple target detection shows the proposed system is not only be able to capture the information of a single target, it can also detect multiple targets under severe noisy system precisely.

- The implementation on a commonly used hardware system indicates the cost of this proposed system is affordable for most institutions. Therefore resource cost would not be a limit to turn this proposed system into real applications.

- The test results show with limited hardware resources, the proposed hybrid sonar system is still able to produce satisfactory target information.

8.2 Future Work

Although the proposed system fulfills all the specifications, there are still some areas that can be investigated to improve the performance of the hybrid system.

8.2.1 Performance improvement of ANFIS

Though the embedded RISC cores take most of the operation load of the ANFIS，the huge computation load for ANFIS operation still eliminates the possibility for ANFIS to finish the task in real time . The highest core frequency of the embedded core is 300MHz. However, as discussed in Chapter VII, the latest technology already raises the working frequency of DSP chip to 900MHz to 1100MHz. Therefore, if the hybrid system is transferred to the latest DSP platform and interleaving the data process with parallel process flows, the performance of the system will be increased significantly.

8.2.2 Improving the speed for CWT operation

Though the local optimum search algorithm reduces the computation time of the CWT almost by half in this novel hybrid system, the CWT operation is still the most time-consuming block in the TME. As mentioned in Chapter V, the CWT operation is implemented using only five multipliers. The initial purpose

is to reduce the area as well as the power consumption. As discussed in Chapter VII, there are still some multipliers not used. Adding parallel multipliers into the system would be the most convenient method to improve the performance. However, more parallel multipliers mean more power consumption and more area cost. Also the hardware complexity is increased such as the coefficient addressing for each multiplier. Some trade-offs would be introduced to increase the performance.

8.2.3 ASIC implementation of proposed design

Though the FPGA board which this project is based on is already affordable for most institutions, the most efficient implementation of proposed system would be based on ASIC (Application Specific Integrated Circuits) platform. However, transporting the design from FPGA to ASIC platform will require modifications of hardware code. First of all the memory descriptions in the FPGA design should be changed according to the fabrication factory and fabrication technique since each technique or factory has its own memory libraries. Secondly since the ANFIS part is written in C and compiled by XILINX EDK 10.1, the ANFIS part should be translated into HDL (hardware Description Language) so that it can be incorporated onto the ASIC platform. Additionally all layouts of each functional block should be reconsidered to fit into the fabrication process. In the meantime, the design flow of the ASIC platform is more complex than the FPGA design since the functionality cannot be tested on

real chips but only simulated in the EDA tools. Therefore, the design flow of the ASIC platform requires more workload.

Taking all considerations above, it might take considerable time to move the FPGA design to the ASIC platform. However, the performance of current system would be improved significantly on the ASIC platform. Thus if possible, the following work should be carried out.

APPENDIX A DESIGN AND SIMULATION ENVIROMENT AND TOOLS

Software

Purpose	Software Package Used
High level System Modelling& Simulation	Matlab, 7.5
RTL design	Xilinx ISE 10.1
RTL simulation	Modsim XE III 6.4b
ANFIS implementation	Xilinx EDK 10.1
Communication between PC and development board	Xilinx EDK 10.1& Matlab

Hardware

Xilinx University Program Virtex-II Pro Development System

SOURCE CODE

```
function [estimated_gwave,SNR_new,t_anfis,anfis_para]=...

anfis_wideband(gwave,noise_data,disp,resp,wave,SNR,m_type,anfis_para,t
me)
global save_dir main_dir
% adaptive noise cancellation using the Fuzzy Toolbox functions ANFIS and
% GENFIS1

%% Building the ANFIS (adaptive neuro-fuzzy inference system) model to
% identify the nonlinear relationship between n1 (original noise source)
and n2 (unknown). Though
% n2 is unknown, we can take the 'P_sig=gwave+n2' as a 'contaminated' version
of n2
% for training. Thus, the noise-free 'gwave' is treated as 'noise' in this
% kind of nonlinear fitting.
% Since the order of the nonlinear channel set to D, we can use
% the number of D-input ANFIS for training.
% We assume 2 membership functions (default choice) per input, so the total
% number of fuzzy rules for learning is D^2=16.
% The FIS membership function parameters are trained based on a combination
of least-squares
% and backpropagation gradient descent methods to a given set of
% input/output data. The step-size for these optimization methods are set
% to 0.2 (default = 0.01)
% STEPSIZE: step size of LSE
% epoch.anr: maximum epoch number to run the training process
% EPOCH_FL: final epoch number to stop the training process
% ERROR_G: noise training error goal
%
fprintf('\nMaximum training epoch No. = %g.',anfis_para.epoch);
fprintf('\nInitial step-size = %g.',anfis_para.int_ss)
fprintf('\nThe noise training error goal = %g.',anfis_para.err_g);
fprintf('\nANFIS data training rate: %g%%, ANFIS data checking
rate: %g%%',anfis_para.data_rate(1),anfis_para.data_rate(2));
%
tic
Data=[noise_data gwave.gwave_map_total'];
trn_data=Data(1:floor(end/100*anfis_para.data_rate(1)),:); % training
data sets
chk_data=Data(floor(end/100*anfis_para.data_rate(2))+1:end,:); %
checking data sets
% generating the initial FIS using GENFIS1
```

```
inmf_type=anfis_para.mf.inmf;
outmf_type=anfis_para.mf.outmf;
mf_n=anfis_para.mf_n;
in_fismat=genfis1(trn_data,mf_n,inmf_type,outmf_type); % generating the
initial FIS applying the generalized bell function (gbellmf) or
                                        % the Gaussian membership function
(gaussmf)
% using ANFIS to fine-tune the initial FIS
trn_opt=[anfis_para.epoch,anfis_para.err_g,anfis_para.int_ss];
if disp==0
   disp_opt=[0,0,0,0];
else
   disp_opt=[1,1,1,1];
end
[out_fismat1,rmse_trn,stepsize,out_fismat2,rmse_chk]=anfis(trn_data,in
_fismat,trn_opt,disp_opt,chk_data);
% using EVALFIS to simulate the FIS
estimated_n2=evalfis(noise_data,out_fismat2);
% esimation of the noise-free return signal
estimated_gwave=gwave.gwave_map_total-estimated_n2';
% time consumption
t_anfis=toc;
diff_n2=estimated_gwave-gwave.gwave_map_m;
%
SNR_new=10*log10(max(abs(gwave.gwave_map_m).^2)/var(diff_n2));
I=find(rmse_trn>0);
epoch_fl=I(end);
anfis_para.epoch=epoch_fl;
fprintf('\nThe SNR is increased to %g dB.\n',SNR_new);
if disp==1
   % plot the input/out membership functions
   [x11,y11]=plotmf(in_fismat,'input',1);
   [x12,y12]=plotmf(in_fismat,'input',2);
   %
   [x21,y21]=plotmf(out_fismat2,'input',1);
   [x22,y22]=plotmf(out_fismat2,'input',2);
   %
   char_s=['-.','--','. ',': '];
   j=1;
   figure,
   subplot(221) % membership functions corresponding to the first input data
   for i=1:mf_n
```

```
        plot(x11(:,i),y11(:,i),char_s(j:j+1),[0 0],[0 1]);
        hold on
        j=j+1;
    end
    title('(a)Initial MFs on x');
    xlabel('x');
    hold off
    %
    j=1;
    subplot(222) % membership functions corresponding to the second input
data
    for i=1:mf_n
        plot(x12(:,i),y12(:,i),char_s(j:j+1),[0 0],[0 1]);
        hold on
        j=j+1;
    end
    title('(b)Initial MFs on y');
    xlabel('y');
    hold off
    %
    j=1;
    subplot(223) % membership functions corresponding to the first output
data
    for i=1:mf_n
        plot(x21(:,i),y21(:,i),char_s(j:j+1),[0 0],[0 1]);
        hold on
        j=j+1;
    end
    title('(c)Final MFs on x');
    xlabel('x');
    hold off
    %
    j=1;
    subplot(224) % membership functions corresponding to the second output
data
    for i=1:mf_n
        plot(x22(:,i),y22(:,i),char_s(j:j+1),[0 0],[0 1]);
        hold on
        j=j+1;
    end
    title('(d)Final MFs on y');
    xlabel('y');
```

```
   hold off
   % plotting the results
   figure,
   subplot(211), % RMSE of training and checking data
   semilogy(I,rmse_trn(I),'--',I,rmse_chk(I),'-');
   title('RMSE of interference');
   xlabel('Epoch number');
   ylabel('RMSE')
   legend('RMSE_{trn}','RMSE_{chk}');
   %
   subplot(212), % LSE step size
   plot(I,stepsize(I))
   title('Step-size for LSE');
   xlabel('Epoch number');
end
% save data into a file

if resp.saveMode=='y'
    cd(save_dir);
    if tme.epoch>=1
       switch resp.noise_type
           case 'rever'

save(strcat('rever_anfis_full_',m_type,'t_',wave.name,num2str(wave.tim
e*1.0e3),'_',num2str(SNR),'db'));
           case 'gauss'

save(strcat('gauss_anfis_full_',m_type,'t_',wave.name,num2str(wave.tim
e*1.0e3),'_',num2str(SNR),'db'));
           case 'rever_n_gauss'

save(strcat('rng_anfis_full_',m_type,'t_',wave.name,num2str(wave.time*
1.0e3),'_',num2str(SNR),'db'));
       end
    else
       switch resp.noise_type
           case 'rever'

save(strcat('rever_anfis_ANR_',m_type,'t_',wave.name,num2str(wave.time
*1.0e3),'_',num2str(SNR),'db'));
           case 'gauss'
```

```
save(strcat('gauss_anfis_ANR_',m_type,'t_',wave.name,num2str(wave.time
*1.0e3),'_',num2str(SNR),'db'));
         case 'rever_n_gauss'

save(strcat('rng_anfis_ANR_',m_type,'t_',wave.name,num2str(wave.time*1
.0e3),'_',num2str(SNR),'db'));
      end
   end
   cd(main_dir);
end
clear genfis1 anfis evalfis % clear up anfis, i.e. initialize anfis

function [coefs,filter_opt,scale_opt,it_no,dF] = cwt_s(SIG,opt_cwt)
%   computes the optimal continuous
%   wavelet coefficients of the vector S at real, positive
%   SCALES.
%   The signal S is real, the wavelet can be real or complex.
%
%   COEFS = CWT(S,SCALES,'wname','plot') computes
%   and, in addition, plots the continuous wavelet
%   transform coefficients.
%

val_SIG   = SIG(:)';
lenSIG=length(SIG);

if ~isnumeric(SIG)
   errargt(mfilename,'Invalid Value for Signal !','msg');
   error('*')
end
% calculate the optimal scale
cum_WAV=opt_cwt.P_opt.cum_psi;
xval=opt_cwt.P_opt.xval;
xWAV=xval-xval(1); % starting from 0
xMaxWAV = xWAV(end);
%
cwt_options=opt_cwt.options; % options from fminbnd
cwt_scale=[opt_cwt.scale_bnd(1),opt_cwt.scale_bnd(2)];
switch opt_cwt.method
   case 'fminbnd'
      [scale_opt,fval,exitflag,output]=fminbnd(@(scale)
cwtfun_scale(scale,SIG,cum_WAV,xWAV),cwt_scale(1),cwt_scale(2),cwt_opt
```

```
ions);
    otherwise ''
end
opt_tap =
[1+floor([0:scale_opt*xMaxWAV]/(scale_opt*xMaxWAV)*(numel(xWAV)-1))]; %
generate a scaled index
if opt_tap(end)>length(cum_WAV)
   op_tap=wkeep1(opt_tap,length(cum_WAV),'c');    % keep the central part
   op_tap=op_tap-(op_tap(end)-length(cum_WAV));   % shift to the left
   zo_idx=find(op_tap<=0);
   if ~isempty(zo_idx)
      op_tap=op_tap(zo_idx(end)+1:end);      % keep the positive index
   end
else
   op_tap=opt_tap;
end
if length(op_tap)==1 ,
    op_tap = [1 1];
end
% calculate the optimal filter coefficients
filter_opt = -sqrt(scale_opt)*diff(fliplr(cum_WAV(op_tap))); % filter
coefficients where cumsum(phi(l))dl=cumsum(Z(-l))dl
coefs = wkeep1(conv2(val_SIG(:)',filter_opt(:)','full'),lenSIG,'c');
% pad the filter with zero
tap_no=numel(filter_opt);
it_no=output.iterations;
dF=gradient(coefs.^2);
fprintf('\nThe maximun iteration = %g.\n',it_no);
fprintf('\nThe optimal scale = %g.\n',scale_opt);
fprintf('\nThe number of FIR filter taps = %g.\n',tap_no);
%------------------------------------------------------------------------
-----
% subfunction
%
function [F] = cwtfun_scale(scale,SIG,cum_WAV,xWAV)
% subfunction of fminbnd.m used to generate the least square cost

val_SIG = SIG(:)';
lenSIG  = length(val_SIG);
xMaxWAV = xWAV(end);

Jt = [1+floor([0:scale*xMaxWAV]/(scale*xMaxWAV)*(numel(xWAV)-1))];
```

```
if Jt(end)>length(cum_WAV)
   J=wkeep1(Jt,length(cum_WAV),'c');   % keep the central part
   J=J-(J(end)-length(cum_WAV));       % shift to the left
   zo_idx=find(J<=0);
   if ~isempty(zo_idx)
      J=J(zo_idx(end)+1:end);           % keep the positive index
   end
else
   J=Jt;
end
if length(J)==1 ,
   J = [1 1];
end
fir_filter = -sqrt(scale)*diff(fliplr(cum_WAV(J))); % filter coefficient
where cumsum(phi(l))dl=cumsum(Z(-l))dl
coefs = wkeep1(conv2(val_SIG(:)',fir_filter(:)','full'),lenSIG,'c');
F=-norm(coefs)^2; % pick the maximum value in the scale axis of CWT
coefficients for each sampling time point
%
function [F] = cwtfun_s(scale,SIG,val_WAV,xWAV)
% subfunction of fminbnd.m used to generate the least square cost

val_SIG = SIG(:)';
lenSIG  = length(val_SIG);
xMaxWAV = xWAV(end);
%coefs    = zeros(1,lenSIG);

j = [1+floor([0:scale*xMaxWAV]/(scale*xMaxWAV)*(numel(xWAV)-1))];
if length(j)==1 ,
   j = [1 1];
end
fir_filter = -diff(sqrt(scale)*fliplr(val_WAV(j))); % filter coefficient
where cumsum(phi(l))dl=cumsum(Z(-l))dl
coefs = wkeep1(conv2(val_SIG(:)',fir_filter(:)','full'),lenSIG,'c');
F=-norm(coefs)^2; % pick the maximum value in the scale axis of CWT
coefficients for each sampling time point
%F=-max(abs(coefs).^2);
if nargout >1
  DF=gradient(F);
end
%
```

```
function coefs=fcwt(SIG,scales,wname)
% CWT Real or Complex Continuous 1-D wavelet coefficients.
% COEFS=FCWT(F,SCALE,'WNAME')
% fast computation of the continuous wavelet coefficient of the input signal
f(t) based on the given mother wavelet.

% Check signal.
%--------------
val_SIG = SIG;
lenSIG  = length(val_SIG);
xSIG    = (1:lenSIG);
stepSIG = 1;

% Check wavelet.
%---------------
val_WAV = WAV;
lenWAV  = length(val_WAV);
xWAV = linspace(0,1,lenWAV);
stepWAV = 1/(lenWAV-1);
xMaxWAV = xWAV(end);
val_WAV = stepWAV*cumsum(val_WAV);
%-------------------------------------------------------------------------
-----
val_SIG   = val_SIG(:)';
nb_SCALES = length(scales);
coefs     = zeros(nb_SCALES,lenSIG);
ind  = 1;
for k = 1:nb_SCALES
    a = scales(k);
    a_SIG = a/stepSIG;
    j = [1+floor([0:a_SIG*xMaxWAV]/(a_SIG*stepWAV))];
    if length(j)==1
        j = [1 1];
    end
    f           = fliplr(val_WAV(j)); % flip matrix (or vector) left/right
    % keep central part of the vector, 1D convolution of val_SIG and f
    coefs(ind,:) = -sqrt(a)*wkeep1(diff(wconv1(val_SIG,f)),lenSIG,'c');
    ind         = ind+1;
end
%
function [npf,zz,xx,yy]=noise_functions(noise_fun,dn,domain)
%
```

```
% noise functions applied to the noise path filters
%
[xx,yy]=meshgrid(domain(1,:),domain(2,:));
x1=dn(1,:); % first parameter
x2=dn(2,:); % second parameter
nstr=strfind(noise_fun,' ');
if nstr~[]
   noise_fun=noise_fun(1:strfind(noise_fun,' ')-1);
end
switch noise_fun
    case 'rbf'
        npf=100*(x2-x1.^2).^2+(1-x1).^2; % rosenbrock banana function
        zz=100*(yy-xx.^2).^2+(1-xx).^2; % desampling
    case 'sine'
        npf=4.*x1.^2.*x2.^2.*(sin(x1+x2)+4*x2);
        zz=4.*xx.^2.*yy.^2.*(sin(xx+yy)+4*yy);
    case 'exp'
        npf=exp(x1).*(4*x1.^2+2*x2.^2+4*x1.*x2+2*x2+1);
        zz=exp(xx).*(4*xx.^2+2*yy.^2+4*xx.*yy+2*yy+1);
end

function [mcwt]=optimal_scale(SIG,mwave,options)

cum_WAV=mwave.cum_psi;
xWAV=mwave.xval;
%

if ~isnumeric(SIG)
    errargt(mfilename,'Invalid Value for Signal !','msg');
    error('*')
end

xMaxWAV = xWAV(end);
%
scale_ub=mwave.scale_ub;  % local upper bound of scale (34 for 10ms)
scale_lb=mwave.scale_lb;  % local lower bound of scale

[scale_opt,fval,exitflag,output]=fminbnd(@(scale)
cwtfun_s(scale,SIG,cum_WAV,xWAV),scale_lb,scale_ub,options);
%fprintf('\nThe optimal scale = %g.\n',scale_opt);

tap_range =
```

```
[1+floor([0:scale_opt*xMaxWAV]/(scale_opt*xMaxWAV)*(numel(xWAV)-1))]; %
generate a scaled index
if length(tap_range)==1 ,
    tap_range = [1 1];
end
mcwt.tap_range=tap_range;
mcwt.scale=scale_opt;
%
function
[map_coefs_den,time_hbd,filter_opt,scale_filter,tme,tar_ss,pflag_tme,t
arget,SIG_wdenX]=map_wden(varargin)
% postprocessing de-noising scheme: combination of CWT and WDeN to process
the noisy signal
% SNG: input signal

echo=varargin{1};
wden_options=varargin{2};
SIG=varargin{3};
tme=varargin{4};
%mwave=varargin{5};
t_N=varargin{5};
N=length(t_N);
target=varargin{6};
opt_map=varargin{7};
anfis_para=varargin{8};
pflag_tme=varargin{9};
tar_ss=varargin{10};
psd_total=varargin{11};
resp=varargin{12};
prange=varargin{13};
wave=varargin{14};

% WDEN options
if tme.epoch>0
   tptr=wden_options.tptr;
   sorh=wden_options.sorh;
   scale_th=wden_options.scale_th;
   dec_level=wden_options.dec_level;
   wav=wden_options.wav;
end

fs=echo.fs;
```

```
psi_step=t_N(2)-t_N(1);
tar_ss.test_failure=1;
%
ct=0; % count for WDen at the first and last loop
SIG_seg=numel(SIG)/echo.seg;
if isempty(target.locations_t)
    target.locations_t=target.locations(end);
end

psd_den=sqrt(sum(abs(SIG).^2.*psi_step));
psd_den_rate=(1-psd_den/psd_total); % power cut in percentage
fprintf('\nThe power cut rate: %g%%.\n',psd_den_rate);
rp100_alpha1(1)=tme.rp100; % TM-alpha 1
t1_anr=clock;
%
if resp.cross_test=='y'
    switch wave.name % scaled transmitted signal
        case 'morl'
            [psi_w]=morlet_wave(prange,echo,1,0);
        case 'mexh'
            [psi_w]=mexihat_wave(prange,echo,1,0);
        case 'glfm'
            [psi_w]=gauss_wlfm(prange,echo,1,0);
        otherwise
            errargt(mfilename,'Unknown transmitted signal!','msg');
            error('*');
    end
end
%
for j=1:echo.seg
    range=(j-1)*SIG_seg+1:j*SIG_seg;
    coefs_sig0=SIG(range);
    % ANC
    tme_count=1;
    if resp.cross_test=='n'
        % CWT
        [coefs,filter_opt,scale_opt,it_no,dF]=cwt_s(coefs_sig0,opt_map);
    else
        % cross correlation

[coefs,filter_opt,scale_opt,it_no]=xcross(coefs_sig0,opt_map,psi_w);
    end
```

```
    % collect data
    map_coefs_den(tme_count,range)=coefs;
    scale_filter(1,tme_count)=scale_opt;
    scale_filter(2,tme_count)=numel(filter_opt);
    scale_filter(3,tme_count)=it_no;
    peak_anc=find(map_coefs_den(1,:)==max(map_coefs_den(1,:)),1);
    target_abs=abs(peak_anc-target.locations_t);
    ind_target=find(target_abs==min(target_abs),1);
    anc_accuracy=target_abs(ind_target);
    dFM(1,range)=dF;
    d2F=gradient(dF); % 2nd order derivative
    d2FM(1,range)=d2F;
    F2(1,:)=abs(d2FM(1,:)-dFM(1,:));         % find active points
    max_peak_F2(1)=find(F2(1,:)==max(F2(1,:)))    % find the most active
point
    % TME
    if tme.epoch>0
        t1_tme=clock;
        tme_tr_count=1;
        redo_TM1=1;
        current_loc=[];
        if target.count>target.no
           tme.tm_alpha1=1;
           tme.tm_alpha2=1;
        end
        while redo_TM1==1
            redo_TM2=1;
            tme_count=1;
            tme_accuracy=tme.accuracy_eps+1; % points accuracy
            rp100_alpha2=rp100_alpha1(tme_tr_count);
            fprintf('\nThe basic threshold value: %g.\n',rp100_alpha2);
            coefs_sig=coefs_sig0;
            map_coefs_den=map_coefs_den(tme_count,:);
            scale_filter=scale_filter(:,tme_count);
            target_pM=[];
            dFM=dFM(1,:);
            d2FM=d2FM(1,:);
            max_peak_F2=max_peak_F2(1);;
            peak_tme=[];
            while redo_TM2==1 %tme_accuracy>tme.accuracy_eps
                % wavelet denoising: WDeN
                SIG_wden_old=wavelet_denoise(coefs_sig,tme);
```

```
                % TM normalization
                fprintf('\nThe threshold value: %g.\n',rp100_alpha2);
                SIG_wden=noise_gauge(SIG_wden_old,psi_step,rp100_alpha2);
                if resp.cross_test=='n'
                   % CWT

[coefs,filter_opt,scale_opt,it_no,dF]=cwt_s(SIG_wden,opt_map);
                else
                   % cross correlation

[coefs,filter_opt,scale_opt,it_no]=xcross(SIG_wden,opt_map,psi_w);
                end
                % collect data
                map_coefs_den(tme_count+1,range)=coefs;
                scale_filter(1,tme_count+1)=scale_opt;
                scale_filter(2,tme_count+1)=numel(filter_opt);
                scale_filter(3,tme_count+1)=it_no;
                peak_tme(tme_count)=find(coefs==max(coefs),1)
                target_abs=abs(peak_tme(tme_count)-target.locations_t);
                ind_target=find(target_abs==min(target_abs),1);
                tme_accuracy=target_abs(ind_target)
                dFM(tme_count+1,range)=dF;
                d2F=gradient(dF); % 2nd order derivative
                d2FM(tme_count+1,range)=d2F;

F2(tme_count+1,:)=abs(d2FM(tme_count+1,range)-dFM(tme_count+1,range));

max_peak_F2(tme_count+1)=find(F2(tme_count+1,:)==max(F2(tme_count+1,:)
)) % find the most active point
                %
                etarget_region1=peak_tme-floor(0.1*echo.fs/2);
                etarget_region2=peak_tme+floor(0.1*echo.fs/2);
                if target.count<=target.no
                    ntarget_region1=1:etarget_region1-1;
                    ntarget_region2=etarget_region2+1:N;
                    ntarget_region=[ntarget_region1,ntarget_region2];
                else
                    ntarget_region=1:N;
                end
                if max(map_coefs_den(tme_count+1,ntarget_region)) <
map_coefs_den(tme_count+1,peak_tme(tme_count))...
                        &
```

```
abs(max_peak_F2(tme_count+1)-peak_tme(tme_count))<=tme.accuracy_eps
                redo_TM2=0;
                redo_TM1=0;
                tar_ss.test_failure=0;
                current_loc=target.locations_t(ind_target);
                target.locations_t(ind_target)=0;
                target.locations_t=nonzeros(target.locations_t)';
            end
            if tme_count==1
                SIG_wdenX(1,range)=SIG_wden; % collect the first
iteration of WDeN
            end
            % update data
            coefs_sig=coefs;
            %peak_tme=peak2_tme;%peak2_tme=peak1_tme;
            tme_count=tme_count+1;
            rp100_alpha2=1-(1-rp100_alpha2)/2^(tme_count); % TM-alpha 2
            if tme_count>ceil(tme.tm_alpha2)
                fprintf('\nWarning: TME epoch has reached its
maximum: %g.\n',tme.tm_alpha2);
                redo_TM2=0;
            end
        end % end while loop
        % data collection
        tme.epoch=tme_count-1;
        SIG_wdenX(2,range)=SIG_wden; % collect the last iteration of
WDeN
        t2_tme=clock;
        time_hbd=etime(t2_tme,t1_tme); % time consumption for TME
operations
        % calculate the true peak
        if ~isempty(current_loc)
            estimate_current_loc=current_loc;
        else
            estimate_current_loc=target.locations_t(ind_target);
        end
        fprintf('\nThe overall TME epoch: %g.\n',tme.epoch);
        %for tk=1:tme.epoch+1
            target_pM=max_peak_F2;
        %    find(map_coefs_den(tk,:)==max(map_coefs_den(tk,:)),1);
        %end
        fprintf('\nThe nearest true current
```

```
location: %g.',estimate_current_loc);
          fprintf('\nThe estimated location(s):\n');target_pM'
          fprintf('\nTME
accuracy: %g.\n');abs(estimate_current_loc-target_pM(:))
          % data update and check-up
          if tme_tr_count<tme.tm_alpha1

rp100_alpha1(tme_tr_count+1)=rp100_alpha1(tme_tr_count)*(1-tme.rp100_r
TR*1.0e-2); % TM-alpha1 update (moving up)
             fprintf('\nWARNING: The %g-th trial with threshold
rate: %g.',tme_tr_count+1,rp100_alpha1(tme_tr_count+1));
          else
             redo_TM1=0;
             % force to reduce the target number when reaching the maximum
             if (target.count<=target.no) & ...

(anfis_para.redo_count(target.count)==anfis_para.redo_count_eps|anfis_
para.epoch>=anfis_para.epoch_max)
                errargt(mfilename,strcat('The ANFIS redo loops has
reached maximum. Target detection failed.'),'msg');
                tar_ss.test_failure=0;
             end
          end
          tme_tr_count=tme_tr_count+1;
       end
    else % ANC operation
       if (anc_accuracy>tme.accuracy_eps)
          tar_ss.test_failure=1;
          target.locations_t
          peak_anc
       else
          target.locations_t(ind_target)=0;
          target.locations_t=nonzeros(target.locations_t)';
       end
       t2_anr=clock; % time consumption for ANR operations
       time_hbd=etime(t2_anr,t1_anr);
       % calculate the true peak
       target_absx=abs(peak_anc-target.locations);
       true_peak_loc=find(target_absx==min(target_absx),1);
       % data update
       SIG_wdenX=[];
    end
```

```
end
%
if (pflag_tme.flag==1) && (tar_ss.test_failure==0)
    figure,
    subplot(2,1,1),

plot(t_N(target.locations(true_peak_loc)),SIG(target.locations(true_pe
ak_loc)),'ro:',t_N,SIG,'b-'),
    ylabel('Amp.');
    title(strcat('ANC output of
[',num2str(target.no-numel(target.locations_t)),']-th target with ANFIS
epoch number [',...
        num2str(anfis_para.epoch),']'));
    axis tight;
    %
    subplot(2,1,2),

plot(t_N(target.locations(true_peak_loc)),map_coefs_den(1,target.locat
ions(true_peak_loc)),'ro:',t_N,map_coefs_den(1,:),'b-'),
    hold on
    plot(t_N(peak_anc),map_coefs_den(1,peak_anc),'gd:'),
    title('ANC + CWT');
    ylabel('CWT coefs.');
    axis tight;
    xlabel('Time (sec)');
    hold off
    %
    if tme.epoch>0
        figure,
        if tme.epoch==1
            plot_tme=2;
        else
            plot_tme=4;
        end
        for i=1:plot_tme
            subplot(plot_tme,1,i),
            if i==1

plot(t_N(target.locations(true_peak_loc)),SIG_wdenX(1,target.locations
(true_peak_loc)),'ro:',t_N,SIG_wdenX(1,:),'b-');
                ylabel('Amp.');
```

```
title(strcat('[',num2str(target.no-numel(target.locations_t)),']-th
target, 1st-WDeN with TM-\alpha_1 = [',...
                num2str(rp100_alpha1(1)),']'));
          elseif i==2

peak1_tme=find(map_coefs_den(i,:)==max(map_coefs_den(i,:)),1);

plot(t_N(target.locations(true_peak_loc)),map_coefs_den(i,target.locat
ions(true_peak_loc)),'ro:')
             hold on

plot(t_N(peak1_tme),map_coefs_den(i,peak1_tme),'gd:',t_N,map_coefs_den
(i,:),'b-'),
             title('1st-TME');
             ylabel('CWT coefs.');
          elseif i==3

plot(t_N(target.locations(true_peak_loc)),SIG_wdenX(2,target.locations
(true_peak_loc)),'ro:',t_N,SIG_wdenX(2,:),'b-');
             ylabel('Amp.');
             title(strcat('[',num2str(tme.epoch),'] * WDeN with
TM-\alpha_1 = [',num2str(rp100_alpha1(end)),']'));
          else
             %peak2_tme=find(map_coefs_den(end,range)==max(map_coefs_d
en(end,range)));

plot(t_N(target.locations(true_peak_loc)),map_coefs_den(end,target.loc
ations(true_peak_loc)),'ro:'),
             hold on

plot(t_N(peak_tme),map_coefs_den(end,peak_tme),'gd:',t_N,map_coefs_den
(end,:),'b-'),
             title(strcat('[',num2str(tme.epoch),'] * TME'));
             ylabel('CWT coefs.');
          end
          axis tight;
          hold on
       end
       % PSD calculattion
       figure,

[y_g1,f_g,t_g,psd_ge1]=spectrogram(SIG_wdenX(1,:),128,120,128,fs);
```

```
[y_g2,f_g,t_g,psd_ge2]=spectrogram(SIG_wdenX(2,:),128,120,128,fs);
       subplot(211),
       surf(t_g,f_g,10*log10(abs(psd_ge1)),'EdgeColor','none');
       axis xy; axis tight; %colormap(jet); view(0,90);
       title(strcat('PSD (dB) of
[',num2str(target.no-numel(target.locations_t)),']-th target, 1st-WDeN
with TM-\alpha_1 = [',...
           num2str(rp100_alpha1(1)),']'));
       xlabel('Time (sec)');
       ylabel('Frequency (Hz)');
       colorbar,
       subplot(212),
       surf(t_g,f_g,10*log10(abs(psd_ge2)),'EdgeColor','none');
       axis xy; axis tight; %colormap(jet); view(0,90);
       title(strcat('PSD (dB) of
[',num2str(target.no-numel(target.locations_t)),']-th target',',
[',num2str(tme.epoch),...
           ']-WDeN with TM-\alpha_1 = [',num2str(rp100_alpha1(end)),']'));
       xlabel('Time (sec)');
       ylabel('Frequency (Hz)');
       colorbar,
   else
       subplot(2,1,1),
       %Apeak_tme=find(map_coefs_den(end,:)==max(map_coefs_den(end,:)),
1);

plot(t_N(target.locations(true_peak_loc)),map_coefs_den(end,target.loc
ations(true_peak_loc)),'ro:')
       hold on

plot(t_N(peak_anc),map_coefs_den(1,peak_anc),'gd:',t_N,map_coefs_den(1
,:),'b-');
       if anfis_para.epoch>0

title(strcat('[',num2str(target.no-numel(target.locations_t)),']-th
target with ANFIS [',...
             num2str(anfis_para.epoch),'] epoch(s)'))
       else
           title('Noise free test')
       end
       ylabel('CWT coefs.');
```

```
        axis tight
        %
        subplot(2,1,2),
        plot(),
        title('Optimal FIR filter');
        ylabel('Amp.');
        axis tight
    end
    xlabel('Time (sec)');
    hold off
end
xval=opt_map.P_opt.xval;
xWAV=xval-xval(1);
if min(opt_map.scale_bnd)>1
    % CWT filter coefficients where cumsum(phi(l))dl=cumsum(Z(-l))dl

filter_opt=[filter_opt(:)',zeros(1,max(opt_map.scale_bnd)*xWAV(end)+1-
numel(filter_opt))];
else
    % cross correlation

filter_opt=[filter_opt(:)',zeros(1,max(1./opt_map.scale_bnd)*xWAV(end)
+1-numel(filter_opt))];
end
%
%if tme.epoch>0
%   test_print
%end
%
```

```
/*
 *
 * Xilinx, Inc.
 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" AS A
 * COURTESY TO YOU.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION AS
 * ONE POSSIBLE   IMPLEMENTATION OF THIS FEATURE, APPLICATION OR
 * STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS IMPLEMENTATION
 * IS FREE FROM ANY CLAIMS OF INFRINGEMENT, AND YOU ARE RESPONSIBLE
 * FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE FOR YOUR IMPLEMENTATION
 * XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO
 * THE ADEQUACY OF THE IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO
 * ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE
 * FROM CLAIMS OF INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.
 */

/*
 * Xilinx EDK 10.1 EDK_K.15
 *
 * This file is a sample test application
 *
 * This application is intended to test and/or illustrate some
 * functionality of your system.  The contents of this file may
 * vary depending on the IP in your system and may use existing
 * IP driver functions.  These drivers will be generated in your
 * XPS project when you run the "Generate Libraries" menu item
 * in XPS.
 *
 * Your XPS project directory is at:
 *    D:\Work\EDK_ANFIS\
 */


// Located in: ppc405_0/include/xparameters.h
#include "xparameters.h"
#include "math.h"
#include "stdio.h"
#include "xio.h"
#include "xutil.h"
#define Input_length 0x00019000
#define Mn_n 4
#define Step 1
```

```
#define Rule_n 16
#define Epoch_n 50
#define norm 128


//=====================================================

Xint16 *input_X= (Xint16 *)XPAR_DDR_SDRAM_MPMC_BASEADDR;
Xint16 *input_Y= (Xint16 *)XPAR_DDR_SDRAM_MPMC_BASEADDR+2*Input_length;
Xint16 *Reference_data = (Xint16
*)XPAR_DDR_SDRAM_MPMC_BASEADDR+4*Input_length;
Xint16 *ANFIS_out=(Xint16 *)XPAR_DDR_SDRAM_MPMC_BASEADDR+6*Input_length;
Xint8 *weight = (Xint8 *)XPAR_DDR_SDRAM_MPMC_BASEADDR+8*Input_length;
Xint8 *mem_fun_addr=(Xint8
*)XPAR_DDR_SDRAM_MPMC_BASEADDR+10*Input_length;;
Xint16 trn_error;
Xint16 trn_error_last;
Xint8 step_size;
Xint16 Layer_2[Rule_n];
Xint16 Layer_3[Rule_n];
Xint8  Mem_para_X[4][2];
Xint8  Mem_para_Y[4][2];
Xint8  de_dp_x[4][2];
Xint8  de_dp_y[4][2];
Xint8  Conse_para[48];
Xint8  LSE_data[49];
Xint16 de_dout;
Xint8  mem_fun_add[20][20];

Xint8 In_X,In_Y,Ref_data,AN_out;

Xint8 Membership_fun(Xint16 input,Xint8 para[2])
{   int a,c;
    Xint8 temp;
    Xint8 addr;
    a=para[0]+norm;
    c=para[1]+norm;
    addr=mem_fun_add[a][c];
    temp=*(mem_fun_addr+addr);
    return (temp);
}
```

```
get_LSE_data(kalman_data, target)
Xint8 kalman_data[49];
Xint8 target;
{
    int i, k, j=0;
    for (i = 0; i < 16; i++) {

            kalman_data[j++] = Layer_3[i]*In_X;
            kalman_data[j++] = Layer_3[i]*In_Y;
            kalman_data[j++] = In_X;

    }

    LSE_data[j] = target;
}
```

```
int main (void) {

            int i,k,j,weight_norm,ep_n;
            void forward(double anfis_out);
            void update_parameter(int step_size);
            weight_norm=0;
     for (ep_n=0;ep_n<Epoch_n;ep_n++){

            for (i=0;i<Input_length;i++)
                   { In_X=*(input_X+i);
                     In_Y=*(input_Y+i);
                     Ref_data=*(Reference_data+i);

Layer_2[0]=Membership_fun(In_X,Mem_para_X[0])*Membership_fun(In_Y,Mem_
para_Y[0]);

Layer_2[1]=Membership_fun(In_X,Mem_para_X[0])*Membership_fun(In_Y,Mem_
para_Y[1]);
```

```
Layer_2[2]=Membership_fun(In_X,Mem_para_X[0])*Membership_fun(In_Y,Mem_
para_Y[2]);

Layer_2[3]=Membership_fun(In_X,Mem_para_X[0])*Membership_fun(In_Y,Mem_
para_Y[3]);

Layer_2[4]=Membership_fun(In_X,Mem_para_X[1])*Membership_fun(In_Y,Mem_
para_Y[0]);

Layer_2[5]=Membership_fun(In_X,Mem_para_X[1])*Membership_fun(In_Y,Mem_
para_Y[1]);

Layer_2[6]=Membership_fun(In_X,Mem_para_X[1])*Membership_fun(In_Y,Mem_
para_Y[2]);

Layer_2[7]=Membership_fun(In_X,Mem_para_X[1])*Membership_fun(In_Y,Mem_
para_Y[3]);

Layer_2[8]=Membership_fun(In_X,Mem_para_X[2])*Membership_fun(In_Y,Mem_
para_Y[0]);

Layer_2[9]=Membership_fun(In_X,Mem_para_X[2])*Membership_fun(In_Y,Mem_
para_Y[1]);

Layer_2[10]=Membership_fun(In_X,Mem_para_X[2])*Membership_fun(In_Y,Mem
_para_Y[2]);

Layer_2[11]=Membership_fun(In_X,Mem_para_X[2])*Membership_fun(In_Y,Mem
_para_Y[3]);

Layer_2[12]=Membership_fun(In_X,Mem_para_X[3])*Membership_fun(In_Y,Mem
_para_Y[0]);

Layer_2[13]=Membership_fun(In_X,Mem_para_X[3])*Membership_fun(In_Y,Mem
_para_Y[1]);

Layer_2[14]=Membership_fun(In_X,Mem_para_X[3])*Membership_fun(In_Y,Mem
_para_Y[2]);

Layer_2[15]=Membership_fun(In_X,Mem_para_X[3])*Membership_fun(In_Y,Mem
_para_Y[3]);
```

```
                        for (j=0;j<Rule_n;j++)
                        { weight_norm+=Layer_2[j];
                        }
                         for (j=0;j<Rule_n;j++)
                        { Layer_3[j]=Layer_2[j]/weight_norm;
                        }
                          get_LSE_data(LSE_data, Ref_data);
                          get_LSE_para(LSE_data,Conse_para);
                          forward(AN_out);
                          de_dout = -2*(Ref_data - AN_out);
                         calculate_de_do(de_dout);
                         update_de_dp();
                         }


                         trn_error_last=trn_error;
                          training_error_measure(Reference_data,
ANFIS_out,
           Input_length, trn_error);

                          if (trn_error_last<trn_error)
                              step_size=-step_size;
                         update_parameter(step_size);

     }
                  }



void
 forward(double anfis_out)

    { double p, q, r, wn, x, y;
       int i,j;
       x = In_X;
       y = In_Y;
       anfis_out=0;
    for (i=0;i<16;i++)
       { j=3*i;

          p=Conse_para[j];
        q=Conse_para[j+1];
```

```
          r=Conse_para[j+2];
          wn=Layer_3[i];
          anfis_out+=wn*(p*x+q*y+r);
}
        }


void
update_parameter(step_size)
int step_size;
{
    int i,j;
    double length ;
    double gradient_vector_length();
    length = gradient_vector_length();
    if (length == 0) {

        return;
    }
    for (i=0;i<4;i++)
       for (j=0;j<2;j++){
        Mem_para_X[i][j]+=step_size*de_dp_x[i][j]/length;
        Mem_para_X[i][j]+=step_size*de_dp_y[i][j]/length;
    }
}

double
gradient_vector_length()

{
    int i,j;
    double total = 0;
    for (i=0;i<4;i++)
       for (j=0;j<2;j++){
            total += de_dp_x[i][j]*de_dp_x[i][j];
    }

        for (i=0;i<4;i++)
       for (j=0;j<2;j++){
            total += de_dp_y[i][j]*de_dp_y[i][j];
    }
```

```
    return(sqrt(total));
}
#include "math.h"
#include "xparameters.h"
#include "math.h"
#include "stdio.h"
#include "xio.h"
#include "xutil.h"

training_error_measure(Reference_data, anfis_output, data_n, error_rmse)
double *Reference_data, *anfis_output;
int data_n;
double error_rmse;
{
    double RMSE, NDEI, ARV, APE;
    int i;
    static double target_total, abs_target_total, target_mean;
    static double target_VAR, target_STD, tmp;
    static int initialization;
    double total_abs_error, total_squared_error;
    double diff;


    /* initialization for the first call of this function */
    if (initialization != 7527474) {
        initialization = 7527474;
        target_total = 0;
        abs_target_total = 0;
        target_VAR = 0;
        target_STD = 0;
        for (i = 0; i < data_n; i++) {
            target_total += *(Reference_data+i);
            abs_target_total += abs(*(Reference_data+i));
        }
        target_mean = target_total/data_n;
        tmp = 0;
        for (i = 0; i < data_n; i++)
            tmp += (*(Reference_data+i) - target_mean)*
                (*(Reference_data+i) - target_mean);
        target_VAR = tmp/data_n;
        target_STD = sqrt(target_VAR);
```

```
    }

    total_abs_error = 0;
    total_squared_error = 0;
    tmp = 0;
    for (i = 0; i < data_n; i++) {
        diff = *(Reference_data+i) - *(anfis_output+i);
        total_abs_error += abs(diff);

        total_squared_error += (diff*diff);
        tmp += (abs(diff)/(*(Reference_data+i)));
    }
    RMSE = sqrt(total_squared_error/data_n);
    NDEI = RMSE/target_STD;
    ARV = NDEI*NDEI;
    APE = total_abs_error/abs_target_total;
    error_rmse=RMSE;
    /*
    error_index[3] = tmp/data_n;
    */
}
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;
--use UNISIM.all;


entity TME IS
port ( reset:               in  std_logic;
              fpga_0_RS232_Uart_RX_pin : IN std_logic;
       sys_clk_pin : IN std_logic;
       sys_rst_pin : IN std_logic;
       fpga_0_DDR_SDRAM_DDR_DQS : INOUT std_logic_vector(7 downto 0);
       fpga_0_DDR_SDRAM_DDR_DQ : INOUT std_logic_vector(63 downto 0);
       fpga_0_LEDs_4Bit_GPIO_IO_pin : INOUT std_logic_vector(0 to 3);
       fpga_0_DDR_SDRAM_DDR_Clk_pin : OUT std_logic_vector(2 downto 0);
       fpga_0_DDR_SDRAM_DDR_Clk_n_pin : OUT std_logic_vector(2 downto 0);
       fpga_0_DDR_SDRAM_DDR_Addr_pin : OUT std_logic_vector(12 downto 0);
       fpga_0_DDR_SDRAM_DDR_BankAddr_pin : OUT std_logic_vector(1 downto
0);
       fpga_0_DDR_SDRAM_DDR_CAS_n_pin : OUT std_logic;
       fpga_0_DDR_SDRAM_DDR_CE_pin : OUT std_logic;
       fpga_0_DDR_SDRAM_DDR_CS_n_pin : OUT std_logic;
       fpga_0_DDR_SDRAM_DDR_RAS_n_pin : OUT std_logic;
       fpga_0_DDR_SDRAM_DDR_WE_n_pin : OUT std_logic;
       fpga_0_DDR_SDRAM_DDR_DM_pin : OUT std_logic_vector(7 downto 0);
       fpga_0_RS232_Uart_TX_pin : OUT std_logic;
       fpga_0_net_gnd_pin : OUT std_logic;
       fpga_0_net_gnd_1_pin : OUT std_logic;
       fpga_0_net_gnd_2_pin : OUT std_logic;
       fpga_0_net_gnd_3_pin : OUT std_logic;
       fpga_0_net_gnd_4_pin : OUT std_logic;
       fpga_0_net_gnd_5_pin : OUT std_logic
              clk:                    in std_logic;




);
end TME;
```

```
architecture TME of TME is


COMPONENT system
    PORT(
        fpga_0_RS232_Uart_RX_pin : IN std_logic;
        sys_clk_pin : IN std_logic;
        sys_rst_pin : IN std_logic;
        fpga_0_DDR_SDRAM_DDR_DQS : INOUT std_logic_vector(7 downto 0);
        fpga_0_DDR_SDRAM_DDR_DQ : INOUT std_logic_vector(63 downto 0);
        fpga_0_LEDs_4Bit_GPIO_IO_pin : INOUT std_logic_vector(0 to 3);
        fpga_0_DDR_SDRAM_DDR_Clk_pin : OUT std_logic_vector(2 downto 0);
        fpga_0_DDR_SDRAM_DDR_Clk_n_pin : OUT std_logic_vector(2 downto 0);
        fpga_0_DDR_SDRAM_DDR_Addr_pin : OUT std_logic_vector(12 downto 0);
        fpga_0_DDR_SDRAM_DDR_BankAddr_pin : OUT std_logic_vector(1 downto
0);
        fpga_0_DDR_SDRAM_DDR_CAS_n_pin : OUT std_logic;
        fpga_0_DDR_SDRAM_DDR_CE_pin : OUT std_logic;
        fpga_0_DDR_SDRAM_DDR_CS_n_pin : OUT std_logic;
        fpga_0_DDR_SDRAM_DDR_RAS_n_pin : OUT std_logic;
        fpga_0_DDR_SDRAM_DDR_WE_n_pin : OUT std_logic;
        fpga_0_DDR_SDRAM_DDR_DM_pin : OUT std_logic_vector(7 downto 0);
        fpga_0_RS232_Uart_TX_pin : OUT std_logic;
        fpga_0_net_gnd_pin : OUT std_logic;
        fpga_0_net_gnd_1_pin : OUT std_logic;
        fpga_0_net_gnd_2_pin : OUT std_logic;
        fpga_0_net_gnd_3_pin : OUT std_logic;
        fpga_0_net_gnd_4_pin : OUT std_logic;
        fpga_0_net_gnd_5_pin : OUT std_logic;
        user_input:                 input std_logic_vector(15 downto 0);
        user_output:            output std_logic_vector(15 downto 0);
        enable:                     output std_logic
        );
    END COMPONENT;

component data_exchange is
port ( clk: input std_logic;
        data_in: input std_logic_vector(15 downto 0);
        data_out: output std_logic_vector(15 downto 0);
        enable: input std_logic;
```

```
        dwt_input: output std_logic_vector(15 downto 0);
        dwt_output: input std_logic_vector(15 downto 0);
        dwt_enable: output std_logic;

        cwt_input: output std_logic_vector(15 downto 0);
        cwt_output: input std_logic_vector(15 downto 0);
        cwt_enable: output std_logic

);
end component;

component S_to_P IS
port ( reset:    in  std_logic;

             clk:          in std_logic;
             data_in:   in  std_logic_vector(15 downto 0);

       data_out:  out std_logic_vector(79 downto 0)

);
end component;

component coeff_mem
   port (
   clka: IN std_logic;
   dina: IN std_logic_VECTOR(79 downto 0);
   addra: IN std_logic_VECTOR(12 downto 0);
   wea: IN std_logic_VECTOR(0 downto 0);
   douta: OUT std_logic_VECTOR(79 downto 0));
end component;

component convolution IS
port ( reset:    in  std_logic;
             start:    in  std_logic;
             clk:          in std_logic;
             data_in:   in  std_logic_vector(79 downto 0);
             coeff_in: in  std_logic_vector(79 downto 0);
       data_out:  out std_logic_vector(15 downto 0);
          data_enable: in std_logic

);
end component;
```

```
component control_CWT IS
port ( reset:               in  std_logic;

              clk:                    in std_logic;

              data_in:             in  std_logic_vector(15 downto 0);
              start:               out std_logic;
              reset_conv:          out std_logic;
              address_data:     out std_logic_vector(14 downto 0);
              we_data:             out std_logic;
              address_coeff:    out std_logic_vector(12 downto 0);

              data_out:            out std_logic_vector(15 downto 0)



);
end component;

component dwt_denoising is
port( clk: in std_logic;
     data_in: in std_logic_vector(15 downto 0);
     data_out: out std_logic_vector(15 downto 0);
     reset: in std_logic;
     data_enable: in std_logic

);
end component;

signal data_to_dsp: std_logic_vector(15 downto 0);
signal dsp_to_data: std_logic_vector(15 downto 0);
signal data_enable: std_logic;
signal dwt_enable : std_logic;
signal cwt_enable : std_logic;
signal data_to_dwt:std_logic_vector(15 downto 0);
signal data_to_cwt:std_logic_vector(15 downto 0);
signal data_out_DWEN: std_logic_vector(15 downto 0);
signal we_data:       std_logic_VECTOR(0 downto 0);
signal reset_conv:       std_logic;
signal address_data: std_logic_vector(14 downto 0);
signal address_coeff:    std_logic_vector(12 downto 0);
```

```
signal start:               std_logic;

signal dina:                std_logic_vector(79 downto 0);
signal data_pa_CWT:  std_logic_vector(79 downto 0);
signal coeff:               std_logic_vector(79 downto 0);
signal data_out_buff:    std_logic_vector(15 downto 0);
signal data_out_temp: std_logic_vector(15 downto 0);


begin


Inst_system: system PORT MAP(
       fpga_0_DDR_SDRAM_DDR_Clk_pin =>fpga_0_DDR_SDRAM_DDR_Clk_pin ,
       fpga_0_DDR_SDRAM_DDR_Clk_n_pin => fpga_0_DDR_SDRAM_DDR_Clk_n_pin,
       fpga_0_DDR_SDRAM_DDR_Addr_pin => fpga_0_DDR_SDRAM_DDR_Addr_pin,
       fpga_0_DDR_SDRAM_DDR_BankAddr_pin
=>fpga_0_DDR_SDRAM_DDR_BankAddr_pin ,
       fpga_0_DDR_SDRAM_DDR_CAS_n_pin => fpga_0_DDR_SDRAM_DDR_CAS_n_pin,
       fpga_0_DDR_SDRAM_DDR_CE_pin => fpga_0_DDR_SDRAM_DDR_CE_pin,
       fpga_0_DDR_SDRAM_DDR_CS_n_pin =>fpga_0_DDR_SDRAM_DDR_CS_n_pin  ,
       fpga_0_DDR_SDRAM_DDR_RAS_n_pin =>fpga_0_DDR_SDRAM_DDR_RAS_n_pin ,
       fpga_0_DDR_SDRAM_DDR_WE_n_pin =>fpga_0_DDR_SDRAM_DDR_WE_n_pin ,
       fpga_0_DDR_SDRAM_DDR_DM_pin =>fpga_0_DDR_SDRAM_DDR_DM_pin ,
       fpga_0_DDR_SDRAM_DDR_DQS => fpga_0_DDR_SDRAM_DDR_DQS,
       fpga_0_DDR_SDRAM_DDR_DQ => fpga_0_DDR_SDRAM_DDR_DQ,
       fpga_0_RS232_Uart_RX_pin =>fpga_0_RS232_Uart_RX_pin ,
       fpga_0_RS232_Uart_TX_pin => fpga_0_RS232_Uart_TX_pin,
       fpga_0_LEDs_4Bit_GPIO_IO_pin =>fpga_0_LEDs_4Bit_GPIO_IO_pin ,
       fpga_0_net_gnd_pin => fpga_0_net_gnd_pin,
       fpga_0_net_gnd_1_pin => fpga_0_net_gnd_1_pin,
       fpga_0_net_gnd_2_pin => fpga_0_net_gnd_2_pin,
       fpga_0_net_gnd_3_pin => fpga_0_net_gnd_3_pin,
       fpga_0_net_gnd_4_pin => fpga_0_net_gnd_4_pin,
       fpga_0_net_gnd_5_pin => fpga_0_net_gnd_5_pin,
       sys_clk_pin => sys_clk_pin,
       sys_rst_pin =>sys_rst_pin;
       user_input =>data_to_dsp;
       user_output=> dsp_to_data;
       enable=>   data_enable

   );
```

```
data_change: data_exchange
port map
(clk,dsp_to_data,data_to_dsp,data_enable,data_to_dwt,data_out_DWEN,
dwt_enable,data_to_cwt,data_out_temp,cwt_enable);

coeff_unit : coeff_mem
       port map (
           clka => clk,
           dina => dina,
           addra => address_coeff,
           wea => we_data,
           douta => coeff);

DWEN: dwt_denoising port
map(clk,data_to_dwt,data_out_DWEN,reset,dwt_enable);

s_to_p_0: S_to_P port map(reset, clk, data_to_cwt,data_pa_CWT);

conv_0: convolution port map
(reset_conv,start,clk,data_pa_CWT,coeff,data_out_temp,cwt_enable);

control_cwt_0: control_CWT port
map(reset,clk,data_out_temp,start,reset_conv,address_data,we_data(0),a
ddress_coeff,data_out);

end TME;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synthesis translate_off
Library XilinxCoreLib;
-- synthesis translate_on
ENTITY coeff_mem IS
    port (
    clka: IN std_logic;
    dina: IN std_logic_VECTOR(79 downto 0);
    addra: IN std_logic_VECTOR(12 downto 0);
    wea: IN std_logic_VECTOR(0 downto 0);
    douta: OUT std_logic_VECTOR(79 downto 0));
```

```
END coeff_mem;

ARCHITECTURE coeff_mem_a OF coeff_mem IS
-- synthesis translate_off
component wrapped_coeff_mem
    port (
    clka: IN std_logic;
    dina: IN std_logic_VECTOR(79 downto 0);
    addra: IN std_logic_VECTOR(12 downto 0);
    wea: IN std_logic_VECTOR(0 downto 0);
    douta: OUT std_logic_VECTOR(79 downto 0));
end component;

-- Configuration specification
    for       all       :       wrapped_coeff_mem       use       entity
XilinxCoreLib.blk_mem_gen_v2_7(behavioral)
       generic map(
           c_has_regceb => 0,
           c_has_regcea => 0,
           c_mem_type => 0,
           c_prim_type => 1,
           c_sinita_val => "0",
           c_read_width_b => 80,
           c_family => "virtex2p",
           c_read_width_a => 80,
           c_disable_warn_bhv_coll => 0,
           c_write_mode_b => "WRITE_FIRST",
           c_init_file_name => "no_coe_file_loaded",
           c_write_mode_a => "WRITE_FIRST",
           c_mux_pipeline_stages => 0,
           c_has_mem_output_regs_b => 0,
           c_load_init_file => 0,
           c_xdevicefamily => "virtex2p",
           c_has_mem_output_regs_a => 0,
           c_write_depth_b => 6144,
           c_write_depth_a => 6144,
           c_has_ssrb => 0,
           c_has_mux_output_regs_b => 0,
           c_has_ssra => 0,
           c_has_mux_output_regs_a => 0,
           c_addra_width => 13,
           c_addrb_width => 13,
```

```
            c_default_data => "0",
            c_use_ecc => 0,
            c_algorithm => 1,
            c_disable_warn_bhv_range => 0,
            c_write_width_b => 80,
            c_write_width_a => 80,
            c_read_depth_b => 6144,
            c_read_depth_a => 6144,
            c_byte_size => 9,
            c_sim_collision_check => "ALL",
            c_use_ramb16bwer_rst_bhv => 0,
            c_common_clk => 0,
            c_wea_width => 1,
            c_has_enb => 0,
            c_web_width => 1,
            c_has_ena => 0,
            c_sinitb_val => "0",
            c_use_byte_web => 0,
            c_use_byte_wea => 0,
            c_use_default_data => 0);
-- synthesis translate_on
BEGIN
-- synthesis translate_off
U0 : wrapped_coeff_mem
       port map (
            clka => clka,
            dina => dina,
            addra => addra,
            wea => wea,
            douta => douta);
-- synthesis translate_on

END coeff_mem_a;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;

entity dwt_denoising is
port( clk: in std_logic;
      data_in: in std_logic_vector(15 downto 0);
```

```
        data_out: out std_logic_vector(15 downto 0);

        reset: in std_logic

);

end dwt_denoising;

architecture DWT_WDen of dwt_denoising is

component high_pass_filter_reconstruct
port (      reset:          in    std_logic;
            start:      in    std_logic;
            clk:            in std_logic;
            data_in:    in    std_logic_vector(15 downto 0);
            data_out:   out std_logic_vector(15 downto 0)

);
end component;


component high_pass_filter IS
port (      reset:          in    std_logic;
                        start:      in    std_logic;
                        clk:            in std_logic;
                        data_in:    in    std_logic_vector(15 downto 0);
            data_out:   out std_logic_vector(15 downto 0)

);
end component;

component low_pass_filter IS
port (      reset:          in    std_logic;
                        start:      in    std_logic;
                        clk:            in std_logic;
                        data_in:    in    std_logic_vector(15 downto 0);
            data_out:   out std_logic_vector(15 downto 0)

);
end component;
```

```
component low_pass_filter_reconstruct
port (      reset:          in    std_logic;
            start:      in    std_logic;
            clk:            in std_logic;
            data_in:    in    std_logic_vector(15 downto 0);
            data_out:  out std_logic_vector(15 downto 0)

);
end component;

component selector_2 is
port(clk: in std_logic;
      select_sig: in std_logic;
      data_in_0: in std_logic_vector(15 downto 0);
      data_in_1: in std_logic_vector(15 downto 0);

      data_out:   out std_logic_vector(15 downto 0)
      );
end component;

component selector is
port(clk: in std_logic;
      select_sig: in std_logic_vector(2 downto 0);
      data_in_0: in std_logic_vector(15 downto 0);
      data_in_1: in std_logic_vector(15 downto 0);
      data_in_2: in std_logic_vector(15 downto 0);
      data_in_3: in std_logic_vector(15 downto 0);
      data_in_4: in std_logic_vector(15 downto 0);
      data_in_5: in std_logic_vector(15 downto 0);
      data_in_6: in std_logic_vector(15 downto 0);

      data_out:   out std_logic_vector(15 downto 0)
      );
end component;

component decimator
port (          clock:          in    std_logic;
                reset:          in    std_logic;
                data_in:    in    std_logic_vector(15 downto 0);
                data_out:  out   std_logic_vector(15 downto 0)
```

```
);

end component;

component clock_gen
port (      reset:          in    std_logic;
            clk:            in    std_logic;
            clk_out_fast:   out std_logic;
            clk_out_slow:   out std_logic

);
end component;

component denoising_block
port( clk: in std_logic;
      data_in: in std_logic_vector(15 downto 0);
      data_out: out std_logic_vector(15 downto 0);
      reset: in std_logic;
      start: in std_logic

);

end   component;


component controller_1 is

port(
     clk: in std_logic;
     reset:          in std_logic;
     ctrl_high_pass:      out std_logic;
     ctrl_low_pass:       out   std_logic;
     ctrl_selector_1:     out   std_logic;
     ctrl_selector_2:     out   std_logic_vector(2 downto 0);
     enable_denoising:    out   std_logic;
     input_addr:                out std_logic_vector(13 downto 0);
     memory_read:         out   std_logic;
     memory_write_0:            out   std_logic;
     memory_write_1:            out   std_logic;
     memory_write_2:            out   std_logic;
     memory_write_3:            out   std_logic;
     memory_write_4:            out   std_logic;
```

```
        memory_write_5:         out   std_logic;
        memory_write_6:         out   std_logic



);

end component;

component controller_2 is

port(
        clk: in std_logic;
        reset:          in std_logic;
        ctrl_high_pass:     out std_logic;
        ctrl_low_pass:      out   std_logic;
        ctrl_selector_1:    out   std_logic;
        ctrl_selector_2:    out   std_logic_vector(2 downto 0);

        input_addr:             out std_logic_vector(13 downto 0);
        memory_read:        out   std_logic;
        memory_write_0:         out   std_logic;
        memory_write_1:         out   std_logic;
        memory_write_2:         out   std_logic;
        memory_write_3:         out   std_logic;
        memory_write_4:         out   std_logic;
        memory_write_5:         out   std_logic;
        memory_write_6:         out   std_logic

);

end component;

component FIFO is
generic(size: integer range 0 to 140);
port(clk_in: in std_logic;
data_in: in std_logic_vector(15 downto 0);
data_out:    out std_logic_vector(15 downto 0);
clk_out:    in std_logic;
read_en:    in std_logic;
reset:          in std_logic
        );
```

```
end component;



component combination is
	Port ( clk : in	STD_LOGIC;
		data_high_pass : in	STD_LOGIC_VECTOR (15 downto 0);
		data_low_pass : in	STD_LOGIC_VECTOR (15 downto 0);
		data_out : out	STD_LOGIC_VECTOR (15 downto 0));
end component;

component up_sampler is
port (clock:		in	std_logic;
		reset:		in	std_logic;
		data_in:	in	std_logic_vector(15 downto 0);
		data_out:	out	std_logic_vector(15 downto 0)

);

end component;

component buffer_input
	port (
	clka: IN std_logic;
	dina: IN std_logic_VECTOR(15 downto 0);
	addra: IN std_logic_VECTOR(13 downto 0);
	wea: IN std_logic_VECTOR(0 downto 0);
	douta: OUT std_logic_VECTOR(15 downto 0));
end component;



signal clk_internal_fast:	std_logic;
signal clk_internal_slow:	std_logic;
signal wea: std_logic_vector(0 downto 0);
signal data_external:	std_logic_vector(15 downto 0);
signal data_selector:	std_logic_vector(15 downto 0);
signal data_selector_idwt:	std_logic_vector(15 downto 0);
signal data_selector_2:	std_logic_vector(15 downto 0);
signal data_selector_idwt_2:	std_logic_vector(15 downto 0);
```

```
signal data_decomosing:  std_logic_vector(15 downto 0);
signal data_denoised:    std_logic_vector(15 downto 0);

signal data_low_pass:    std_logic_vector(15 downto 0);
signal data_high_pass:   std_logic_vector(15 downto 0);
signal data_low_pass_reconstruct:   std_logic_vector(15 downto 0);
signal data_high_pass_reconstruct:  std_logic_vector(15 downto 0);

signal data_buffed:      std_logic_vector(15 downto 0);
signal data_filtered:    std_logic_vector(15 downto 0);
signal data_cont:            std_logic_vector(15 downto 0);
signal data_combined:            std_logic_vector(15 downto 0);

signal data_decimated_low_pass: std_logic_vector(15 downto 0);
signal data_decimated_high_pass: std_logic_vector(15 downto 0);

signal data_upsampled_low_pass: std_logic_vector(15 downto 0);
signal data_upsampled_high_pass: std_logic_vector(15 downto 0);

signal ctrl_selector_1:    std_logic;
signal ctrl_selector_2:    std_logic_vector(2 downto 0);
signal ctrl_selector_idwt_1:    std_logic;
signal ctrl_selector_idwt_2:    std_logic_vector(2 downto 0);

signal ctrl_low_pass:    std_logic;
signal ctrl_high_pass:   std_logic;

signal ctrl_low_pass_reconstruct:   std_logic;
signal ctrl_high_pass_reconstruct:  std_logic;

signal ctrl_denoising:   std_logic;
signal memory_read_dwt: std_logic;
signal memory_write_0_dwt: std_logic;
signal memory_write_1_dwt: std_logic;
signal memory_write_2_dwt: std_logic;
signal memory_write_3_dwt: std_logic;
signal memory_write_4_dwt: std_logic;
signal memory_write_5_dwt: std_logic;
signal memory_write_6_dwt: std_logic;

signal memory_read_idwt: std_logic;
signal memory_write_0_idwt: std_logic;
```

```
signal memory_write_1_idwt: std_logic;
signal memory_write_2_idwt: std_logic;
signal memory_write_3_idwt: std_logic;
signal memory_write_4_idwt: std_logic;
signal memory_write_5_idwt: std_logic;
signal memory_write_6_idwt: std_logic;

signal mem_out_0_dwt: std_logic_vector(15 downto 0);
signal mem_out_1_dwt: std_logic_vector(15 downto 0);
signal mem_out_2_dwt: std_logic_vector(15 downto 0);
signal mem_out_3_dwt: std_logic_vector(15 downto 0);
signal mem_out_4_dwt: std_logic_vector(15 downto 0);
signal mem_out_5_dwt: std_logic_vector(15 downto 0);
signal mem_out_6_dwt: std_logic_vector(15 downto 0);

signal mem_out_0_idwt: std_logic_vector(15 downto 0);
signal mem_out_1_idwt: std_logic_vector(15 downto 0);
signal mem_out_2_idwt: std_logic_vector(15 downto 0);
signal mem_out_3_idwt: std_logic_vector(15 downto 0);
signal mem_out_4_idwt: std_logic_vector(15 downto 0);
signal mem_out_5_idwt: std_logic_vector(15 downto 0);
signal mem_out_6_idwt: std_logic_vector(15 downto 0);

signal input_signal: std_logic_vector(15 downto 0);
signal input_signal_idwt: std_logic_vector(15 downto 0);

signal input_addr: std_logic_vector(13 downto 0);
signal input_addr_idwt: std_logic_vector(13 downto 0);



begin

high_pass_reconstuct:     high_pass_filter_reconstruct     port
map(reset,ctrl_high_pass_reconstruct,clk_internal_fast,data_selector_idwt_2,data_high_pass_reconstruct)
;
low_pass_reconstruct:     low_pass_filter_reconstruct     port
map(reset,ctrl_low_pass_reconstruct,clk_internal_fast,data_selector_idwt_2, data_low_pass_reconstruct);



high_pass:     high_pass_filter     port
map(reset,ctrl_high_pass,clk_internal_fast,data_selector_2,data_high_pass);
```

low_pass: low_pass_filter port map(reset,ctrl_low_pass,clk_internal_fast,data_selector_2,data_low_pass);

decimator_high_pass: decimator port map(clk_internal_slow, reset, data_high_pass, data_decimated_high_pass);
decimator_low_pass: decimator port map(clk_internal_slow, reset, data_low_pass, data_decimated_low_pass);

upsampler_high_pass: up_sampler port map(clk_internal_fast,reset,data_filtered,data_upsampled_high_pass);
upsampler_low_pass: up_sampler port map(clk_internal_fast,reset,data_selector_idwt,data_upsampled_low_pass);

controller_dwt: controller_1 port map(clk,reset,ctrl_high_pass,ctrl_low_pass,ctrl_selector_1,ctrl_selector_2,ctrl_denoising,input_addr,memory_read_dwt,memory_write_0_dwt,memory_write_1_dwt,memory_write_2_dwt,memory_write_3_dwt,memory_write_4_dwt,memory_write_5_dwt,memory_write_6_dwt);
controller_idwt: controller_2 port map(clk,reset,ctrl_high_pass_reconstruct,ctrl_low_pass_reconstruct,ctrl_selector_idwt_1,ctrl_selector_idwt_2,input_addr_idwt,memory_read_idwt,memory_write_0_idwt,memory_write_1_idwt,memory_write_2_idwt,memory_write_3_idwt,memory_write_4_idwt,memory_write_5_idwt,memory_write_6_idwt);

clock_generator: clock_gen port map(reset,clk,clk_internal_fast, clk_internal_slow);

denoising_block_0:denoising_block port map(clk_internal_slow,data_decimated_high_pass,data_denoised,reset,ctrl_denoising);

selector_dwt_1: selector_2 port map(clk_internal_fast,ctrl_selector_1,data_decimated_low_pass,input_signal,data_selector);
selector_dwt_2: selector port map(clk_internal_fast,ctrl_selector_2,mem_out_0_dwt,mem_out_1_dwt,mem_out_2_dwt,mem_out_3_dwt,mem_out_4_dwt,mem_out_5_dwt,mem_out_6_dwt,data_selector_2);

selector_idwt_1: selector_2 port map(clk_internal_fast,ctrl_selector_idwt_1,data_combined,input_signal_idwt,data_selector_idwt);
selector_idwt_2: selector port map(clk_internal_fast,ctrl_selector_idwt_2,mem_out_0_idwt,mem_out_1_idwt,mem_out_2_idwt,mem_out_3_idwt,mem_out_4_idwt,mem_out_5_idwt,mem_out_6_idwt,data_selector_idwt_2);

fifo_0_dwt: FIFO generic map (20) port map(memory_write_0_dwt,data_selector,mem_out_0_dwt,clk,memory_read_dwt,reset);

```
fifo_1_dwt: FIFO generic map (20) port map(memory_write_1_dwt,data_selector,mem_out_1_dwt,clk,memory_read_dwt,reset);
fifo_2_dwt: FIFO generic map (20) port map(memory_write_2_dwt,data_selector,mem_out_2_dwt,clk,memory_read_dwt,reset);
fifo_3_dwt: FIFO generic map (20) port map(memory_write_3_dwt,data_selector,mem_out_3_dwt,clk,memory_read_dwt,reset);
fifo_4_dwt: FIFO generic map (20) port map(memory_write_4_dwt,data_selector,mem_out_4_dwt,clk,memory_read_dwt,reset);
fifo_5_dwt: FIFO generic map (20) port map(memory_write_5_dwt,data_selector,mem_out_5_dwt,clk,memory_read_dwt,reset);
fifo_6_dwt: FIFO generic map (20) port map(memory_write_6_dwt,data_selector,mem_out_6_dwt,clk,memory_read_dwt,reset);

fifo_0_idwt: FIFO generic map (20) port map(memory_write_0_idwt,data_selector_idwt,mem_out_0_idwt,clk,memory_read_idwt,reset);
fifo_1_idwt: FIFO generic map (20) port map(memory_write_1_idwt,data_selector_idwt,mem_out_1_idwt,clk,memory_read_idwt,reset);
fifo_2_idwt: FIFO generic map (20) port map(memory_write_2_idwt,data_selector_idwt,mem_out_2_idwt,clk,memory_read_idwt,reset);
fifo_3_idwt: FIFO generic map (20) port map(memory_write_3_idwt,data_selector_idwt,mem_out_3_idwt,clk,memory_read_idwt,reset);
fifo_4_idwt: FIFO generic map (20) port map(memory_write_4_idwt,data_selector_idwt,mem_out_4_idwt,clk,memory_read_idwt,reset);
fifo_5_idwt: FIFO generic map (20) port map(memory_write_5_idwt,data_selector_idwt,mem_out_5_idwt,clk,memory_read_idwt,reset);
fifo_6_idwt: FIFO generic map (20) port map(memory_write_6_idwt,data_selector_idwt,mem_out_6_idwt,clk,memory_read_idwt,reset);
```

```
input_buffer_dwt : buffer_input
        port map (
            clka => clk,
            dina => data_in,
            addra => input_addr,
            wea => wea,
            douta => input_signal);

input_buffer_idwt : buffer_input
        port map (
            clka => clk,
```

```
                dina => data_denoised,
                addra => input_addr_idwt,
                wea => wea,
                douta => input_signal_idwt);


combine:        combination                                                     port
map(clk_internal_fast,data_high_pass_reconstruct,data_low_pass_reconstruct,data_combined);

data_out<=data_combined;

end DWT_WDen;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;



entity high_pass_filter IS
port (      reset:          in      std_logic;
                    start:      in      std_logic;
                    clk:            in std_logic;
                    data_in:    in      std_logic_vector(15 downto 0);
            data_out:  out std_logic_vector(15 downto 0)

);
end high_pass_filter;

architecture db_20_high_pass of high_pass_filter is


constant coeff_0:std_logic_vector(15 downto 0) := conv_std_logic_vector(0, 16);

constant coeff_1:std_logic_vector(15 downto 0):=conv_std_logic_vector(0, 16);

constant coeff_2:std_logic_vector(15 downto 0):=conv_std_logic_vector(0, 16);

constant coeff_3:std_logic_vector(15 downto 0):=conv_std_logic_vector(4, 16);
```

```
constant coeff_4:std_logic_vector(15 downto 0):=conv_std_logic_vector(12, 16);

constant coeff_5:std_logic_vector(15 downto 0):=conv_std_logic_vector(-8, 16);

constant coeff_6:std_logic_vector(15 downto 0):=conv_std_logic_vector(-47, 16);

constant coeff_7:std_logic_vector(15 downto 0):=conv_std_logic_vector(-21, 16);

constant coeff_8:std_logic_vector(15 downto 0):=conv_std_logic_vector(192, 16);

constant coeff_9:std_logic_vector(15 downto 0):=conv_std_logic_vector(171, 16);

constant coeff_10:std_logic_vector(15 downto 0) := conv_std_logic_vector(-413, 16);

constant coeff_11:std_logic_vector(15 downto 0):=conv_std_logic_vector(-539, 16);

constant coeff_12:std_logic_vector(15 downto 0):=conv_std_logic_vector(738, 16);

constant coeff_13:std_logic_vector(15 downto 0):=conv_std_logic_vector(-1135, 16);

constant coeff_14:std_logic_vector(15 downto 0):=conv_std_logic_vector(-1447, 16);

constant coeff_15:std_logic_vector(15 downto 0):=conv_std_logic_vector(-1629, 16);

constant coeff_16:std_logic_vector(15 downto 0):=conv_std_logic_vector(3988, 16);

constant coeff_17:std_logic_vector(15 downto 0):=conv_std_logic_vector(-3053, 16);

constant coeff_18:std_logic_vector(15 downto 0):=conv_std_logic_vector(1090, 16);

constant coeff_19:std_logic_vector(15 downto 0):=conv_std_logic_vector(-155, 16);




type data_buff is array (integer range <>) of std_logic_vector(15 downto 0);
type multi_buff is array(integer range<>) of std_logic_vector(31 downto 0);

signal data_in_buff:        data_buff(19 downto 0);
signal multi_out:      std_logic_vector(33 downto 0);
signal coeff_vector : data_buff(19 downto 0);
signal i: integer range 0 to 31;
```

```
begin

process(clk, reset)

begin
if reset='0' then
        data_out<=(others=>'0');
        data_in_buff<=(others=>"0000000000000000");
        multi_out(15 downto 0)<=(others=>'0');
        multi_out(33 downto 16)<=(others=>'0');
        i<=0;
        data_in_buff(0)<=data_in;
        coeff_vector(0)<=coeff_0;
        coeff_vector(1)<=coeff_1;
        coeff_vector(2)<=coeff_2;
        coeff_vector(3)<=coeff_3;
        coeff_vector(4)<=coeff_4;
        coeff_vector(5)<=coeff_5;
        coeff_vector(6)<=coeff_6;
        coeff_vector(7)<=coeff_7;
        coeff_vector(8)<=coeff_8;
        coeff_vector(9)<=coeff_9;
        coeff_vector(10)<=coeff_10;
        coeff_vector(11)<=coeff_11;
        coeff_vector(12)<=coeff_12;
        coeff_vector(13)<=coeff_13;
        coeff_vector(14)<=coeff_14;
        coeff_vector(15)<=coeff_15;
        coeff_vector(16)<=coeff_16;
        coeff_vector(17)<=coeff_17;
        coeff_vector(18)<=coeff_18;
        coeff_vector(19)<=coeff_19;



     elsif clk'event and clk='1' then
    if start='1' then
        if i<19 then
          i<=i+1;
        multi_out<=multi_out+ (data_in_buff(i)* coeff_vector(i));
    elsif i=19 then
```

```
            for j in 19 downto 1 loop
            data_in_buff(j)<=data_in_buff(j-1);
            end loop;

            data_in_buff(0)<=data_in;

            data_out<=multi_out(31 downto 16);
            multi_out(15 downto 0)<= (others=>'0');
            multi_out(33 downto 16)<=(others=>'0');

            if start = '1' then
            i<=0;
            end if;

end if;

end if;

end if;

end process;

end db_20_high_pass;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;



entity low_pass_filter IS
port (      reset:          in    std_logic;
                    start:      in    std_logic;
                    clk:            in std_logic;
                    data_in:    in    std_logic_vector(15 downto 0);
            data_out:  out std_logic_vector(15 downto 0)

);
end low_pass_filter;
```

```
architecture db_20_low_pass of low_pass_filter is

constant coeff_0:std_logic_vector(15 downto 0) := conv_std_logic_vector(155, 16);

constant coeff_1:std_logic_vector(15 downto 0):=conv_std_logic_vector(1090, 16);

constant coeff_2:std_logic_vector(15 downto 0):=conv_std_logic_vector(3054, 16);

constant coeff_3:std_logic_vector(15 downto 0):=conv_std_logic_vector(3988, 16);

constant coeff_4:std_logic_vector(15 downto 0):=conv_std_logic_vector(1629, 16);

constant coeff_5:std_logic_vector(15 downto 0):=conv_std_logic_vector(-1447, 16);

constant coeff_6:std_logic_vector(15 downto 0):=conv_std_logic_vector(-1135, 16);

constant coeff_7:std_logic_vector(15 downto 0):=conv_std_logic_vector(738, 16);

constant coeff_8:std_logic_vector(15 downto 0):=conv_std_logic_vector(539, 16);

constant coeff_9:std_logic_vector(15 downto 0):=conv_std_logic_vector(-413, 16);

constant coeff_10:std_logic_vector(15 downto 0) := conv_std_logic_vector(-171, 16);

constant coeff_11:std_logic_vector(15 downto 0):=conv_std_logic_vector(192, 16);

constant coeff_12:std_logic_vector(15 downto 0):=conv_std_logic_vector(21, 16);

constant coeff_13:std_logic_vector(15 downto 0):=conv_std_logic_vector(-47, 16);

constant coeff_14:std_logic_vector(15 downto 0):=conv_std_logic_vector(8, 16);

constant coeff_15:std_logic_vector(15 downto 0):=conv_std_logic_vector(12, 16);

constant coeff_16:std_logic_vector(15 downto 0):=conv_std_logic_vector(-4, 16);

constant coeff_17:std_logic_vector(15 downto 0):=conv_std_logic_vector(0, 16);

constant coeff_18:std_logic_vector(15 downto 0):=conv_std_logic_vector(0, 16);
```

```
constant coeff_19:std_logic_vector(15 downto 0):=conv_std_logic_vector(0, 16);



type data_buff is array (integer range <>) of std_logic_vector(15 downto 0);
type multi_buff is array(integer range<>) of std_logic_vector(31 downto 0);

signal data_in_buff:        data_buff(19 downto 0);
signal multi_out:      std_logic_vector(33 downto 0);
signal coeff_vector : data_buff(19 downto 0);
signal i: integer range 0 to 31;

begin

process(clk, reset)

begin
if reset='0' then
      data_out<=(others=>'0');
      data_in_buff<=(others=>"0000000000000000");
      multi_out(33 downto 16)<=(others=>'0');
      multi_out(15 downto 0)<=(others=>'0');
      data_in_buff(0)<=data_in;
      i<=0;
      coeff_vector(0)<=coeff_0;
      coeff_vector(1)<=coeff_1;
      coeff_vector(2)<=coeff_2;
      coeff_vector(3)<=coeff_3;
      coeff_vector(4)<=coeff_4;
      coeff_vector(5)<=coeff_5;
      coeff_vector(6)<=coeff_6;
      coeff_vector(7)<=coeff_7;
      coeff_vector(8)<=coeff_8;
      coeff_vector(9)<=coeff_9;
      coeff_vector(10)<=coeff_10;
      coeff_vector(11)<=coeff_11;
      coeff_vector(12)<=coeff_12;
      coeff_vector(13)<=coeff_13;
      coeff_vector(14)<=coeff_14;
      coeff_vector(15)<=coeff_15;
      coeff_vector(16)<=coeff_16;
      coeff_vector(17)<=coeff_17;
```

```
      coeff_vector(18)<=coeff_18;
      coeff_vector(19)<=coeff_19;


    elsif clk'event and clk='1' then
      if i<19 and start = '1' then
        i<=i+1;
        multi_out<=multi_out+ (data_in_buff(i)* coeff_vector(i));
  elsif i=19 then

            for j in 19 downto 1 loop
              data_in_buff(j)<=data_in_buff(j-1);
            end loop;

            data_in_buff(0)<=data_in;

            data_out<=multi_out(31 downto 16);
            multi_out(15 downto 0)<=(others=>'0');
            multi_out(33 downto 16)<=(others=>'0');

            if start = '1' then
            i<=0;
            end if;

end if;

end if;

end process;

end db_20_low_pass;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity decimator is
port (clock:          in    std_logic;
                reset:          in    std_logic;
                data_in:   in    std_logic_vector(15 downto 0);
                data_out:  out   std_logic_vector(15 downto 0)
```

```
);

end decimator;

architecture dec_2 of decimator is

signal i: integer range 0 to 3;


begin

process(reset, clock)
begin
if reset='0' then
i<=0;
data_out<=(others=>'0');
elsif clock'event and clock='1' then
        if i=1 then
                data_out<=data_in;
                i<=0;
        else i<=i+1;
    end if;
end if;

end process;


end dec_2;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity up_sampler is
port (clock:        in    std_logic;
                reset:          in    std_logic;
                data_in:   in    std_logic_vector(15 downto 0);
                data_out:  out   std_logic_vector(15 downto 0)

);
```

```
end up_sampler;

architecture upsample_2 of up_sampler is

signal i: integer range 0 to 3;


begin

process(reset, clock)
begin
if reset='0' then
i<=0;
data_out<=(others=>'0');
elsif clock'event and clock='1' then
        if i=1 then
                data_out<=data_in;
                i<=0;
        else i<=i+1;
                data_out<=(others=>'0');
   end if;
end if;

end process;

end upsample_2;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;

entity controller_1 is

port(
        clk: in std_logic;
        reset:              in std_logic;
        ctrl_high_pass:     out std_logic;
        ctrl_low_pass:      out   std_logic;
        ctrl_selector_1:    out   std_logic;
```

```
        ctrl_selector_2:      out   std_logic_vector(2 downto 0);
        enable_denoising:     out   std_logic;
        input_addr:                 out std_logic_vector(13 downto 0);
        memory_read:          out   std_logic;
        memory_write_0:             out   std_logic;
        memory_write_1:             out   std_logic;
        memory_write_2:             out   std_logic;
        memory_write_3:             out   std_logic;
        memory_write_4:             out   std_logic;
        memory_write_5:             out   std_logic;
        memory_write_6:             out   std_logic



);

end controller_1;

architecture controller_DWT of controller_1 is


signal count_selector:      integer range 0 to 6;
signal count_sample:        integer range 0 to 10240;
signal count_loop:    integer range 0 to 20400;
signal count_level_0: integer range 0 to 1;
signal count_level_1: integer range 0 to 2;
signal count_level_2: integer range 0 to 3;
signal count_level_3: integer range 0 to 4;
signal count_level_4: integer range 0 to 5;
signal count_level_5: integer range 0 to 6;
signal count_memory:              integer range 0 to 6;


begin

process (clk,reset)

begin

if reset='0' then
count_selector<=0;
count_sample<=0;
```

```
count_loop<=0;
count_memory<=0;
count_level_0<=1;
count_level_1<=2;
count_level_2<=3;
count_level_3<=4;
count_level_4<=5;
count_level_5<=6;




elsif clk'event and clk='1' then
        count_sample<=count_sample+1;
                    ctrl_high_pass<='1';
                ctrl_low_pass<='1';
        if count_loop=20400 then
                memory_read<='0';
                ctrl_selector_2<="111";
                enable_denoising<='0';
                ctrl_high_pass<='0';
                ctrl_low_pass<='0';

        else
                ctrl_high_pass<='1';
                ctrl_low_pass<='1';
        case count_loop is
         when 0 to 19=> ctrl_selector_2<="000";

         when 20 to 40=>ctrl_selector_2<=conv_std_logic_vector(count_level_0, 3);

count_level_0<=count_level_0+1;

count_memory<=count_level_0;
                                                                    if    count_level_0=0
then


enable_denoising<='1';
                                                                    else
enable_denoising<='0';
                                                                    end if;
```

```
        when 41 to 101=>ctrl_selector_2<=conv_std_logic_vector(count_level_1, 3);
count_level_1<=count_level_1+1;
                                                                    if  count_level_1<2
then
enable_denoising<='1';
                                                                    else
enable_denoising<='0';
                                                                    end if;
count_memory<=count_level_1;
        when 102 to 182=>ctrl_selector_2<=conv_std_logic_vector(count_level_2, 3);
count_level_2<=count_level_2+1;
                                                                    if  count_level_2<3
then
enable_denoising<='1';
                                                                    else
enable_denoising<='0';
                                                                    end if;

count_memory<=count_level_2;
        when 183 to 283=>ctrl_selector_2<=conv_std_logic_vector(count_level_3, 3);
count_level_3<=count_level_3+1;
                                                                        if
count_level_3<4 then
enable_denoising<='1';
                                                                    else
enable_denoising<='0';
                                                                    end if;
count_memory<=count_level_3;
```

```
        when 284 to 404=>ctrl_selector_2<=conv_std_logic_vector(count_level_4, 3);

count_level_4<=count_level_4+1;
                                                                    if    count_level_4<5
then

enable_denoising<='1';
                                                                    else
enable_denoising<='0';
                                                                    end if;

count_memory<=count_level_4;
        when 405 to 10239=>ctrl_selector_2<=conv_std_logic_vector(count_level_5, 3);

count_level_5<=count_level_5+1;

enable_denoising<='1';

count_memory<=count_level_5;

       when others=>null;



   end case;
      if count_memory=0 then
 count_sample<=count_sample+1;
 input_addr<=conv_std_logic_vector(count_sample, 14);
 end if;
        end if;

end if;

end process;

memory_write_0<=clk when count_memory=0 else
                                    '0';
memory_write_1<=clk when count_memory=1 else
                                    '0';
memory_write_2<=clk when count_memory=2 else
                                    '0';
memory_write_3<=clk when count_memory=3 else
```

```
                                    '0';
memory_write_4<=clk when count_memory=4 else
                                    '0';
memory_write_5<=clk when count_memory=5 else
                                    '0';
memory_write_6<=clk when count_memory=6 else
                                    '0';

ctrl_selector_1<='1' when count_memory=0 else
                                    '0';
end controller_DWT;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;

entity clock_gen IS
port (      reset:          in    std_logic;
            clk:            in    std_logic;

            clk_out_fast:   out std_logic;
            clk_out_slow:   out std_logic

);
end clock_gen;

architecture clock_generator of clock_gen is

signal i : integer;


begin
process (clk,reset)
begin
if reset= '0' then
            i<=0;
            elsif clk'event and clk='1' then
                        i<=i+1;
            if i<10 then
            clk_out_slow<='1';
                  elsif i>9 and i<19 then
```

```
                clk_out_slow<='0';
                elsif i=19 then
                i<=0;
            end if;
end if;

end process;

clk_out_fast<=clk;

end clock_generator;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;


entity denoising_block is
port( clk: in std_logic;
      data_in: in std_logic_vector(15 downto 0);
      data_out: out std_logic_vector(15 downto 0);
      reset: in std_logic ;
      start: in std_logic

);

end   denoising_block;


architecture Denoising_fun of denoising_block is

constant thres:std_logic_vector(15 downto 0) := conv_std_logic_vector(14, 16);
signal data_denoise: std_logic_vector(15 downto 0);
signal data_temp: std_logic_vector(15 downto 0);

begin

process (reset, clk)

begin
```

```
if reset='0' then

data_denoise<=(others=>'0');
data_out<=(others=>'0');
data_temp<=(others=>'0');

elsif reset='1' then

	if clk'event and clk='1' and start='1' then
		data_temp<=data_in;

		if	thres > data_temp then
			data_denoise<=(others=>'0');
		else data_denoise<=data_temp;
		end if;
	data_out<=data_denoise;
	end if;
end if;

end process;

end Denoising_fun;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;

entity selector_2 is
port(clk: in std_logic;
	select_sig: in std_logic;
	data_in_0: in std_logic_vector(15 downto 0);
	data_in_1: in std_logic_vector(15 downto 0);

	data_out:	out std_logic_vector(15 downto 0)
	);
end selector_2;

architecture selector_2_1 of selector_2 is
```

```
begin

data_out<=data_in_0 when select_sig='1' else
                                data_in_1;

end selector_2_1;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;

entity selector is
port(clk: in std_logic;
      select_sig: in std_logic_vector(2 downto 0);
      data_in_0: in std_logic_vector(15 downto 0);
      data_in_1: in std_logic_vector(15 downto 0);
      data_in_2: in std_logic_vector(15 downto 0);
      data_in_3: in std_logic_vector(15 downto 0);
      data_in_4: in std_logic_vector(15 downto 0);
      data_in_5: in std_logic_vector(15 downto 0);
      data_in_6: in std_logic_vector(15 downto 0);

      data_out:    out std_logic_vector(15 downto 0)
      );
end selector;

architecture selector_7_1 of selector is

begin

process(clk, select_sig)

begin

if clk'event and clk='1' then
case select_sig is
when "000"=>data_out<=data_in_0;
when "001"=>data_out<=data_in_1;
when "010"=>data_out<=data_in_2;
when "011"=>data_out<=data_in_3;
```

```
when "100"=>data_out<=data_in_4;
when "101"=>data_out<=data_in_5;
when "110"=>data_out<=data_in_6;
when others=>null;
end case;

end if;

end process;


end selector_7_1;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;

entity FIFO is
generic(size: integer range 0 to 140);
port(clk_in: in std_logic;
data_in: in std_logic_vector(15 downto 0);
data_out:    out std_logic_vector(15 downto 0);
clk_out:    in std_logic;
read_en:    in std_logic;
reset:              in std_logic
      );
end FIFO;

architecture behav_FIFO of FIFO is
type shift_reg is array(integer range <>) of std_logic_vector(15 downto 0);
signal content:  shift_reg(size downto 0);
signal in_counter: integer range 0 to size;
signal out_counter: integer range 0 to size;

begin

process(clk_in, reset)

begin
if reset ='0' then
```

```
        in_counter<=0;
                content<=(others=>"0000000000000000");

        elsif clk_in'event and clk_in='1' then
        content(in_counter)<=data_in;
           if in_counter=size then
                        in_counter<=0;
           else in_counter<=in_counter+1;
           end if;

end if;

end process;

process(clk_out, reset)

begin
if reset ='0' then
        out_counter<=size;

        elsif clk_out'event and clk_out='1' then
                        if read_en='1' then
                        if
                              out_counter=size then
                                        out_counter<=0;
                        else out_counter<=out_counter+1;
                        end if;
                        end if;
end if;

end process;

data_out<=content(out_counter);

end behav_FIFO;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synthesis translate_off
Library XilinxCoreLib;
-- synthesis translate_on
ENTITY buffer_input IS
```

```
	port (
	clka: IN std_logic;
	dina: IN std_logic_VECTOR(15 downto 0);
	addra: IN std_logic_VECTOR(13 downto 0);
	wea: IN std_logic_VECTOR(0 downto 0);
	douta: OUT std_logic_VECTOR(15 downto 0));
END buffer_input;

ARCHITECTURE buffer_input_a OF buffer_input IS
-- synthesis translate_off
component wrapped_buffer_input
	port (
	clka: IN std_logic;
	dina: IN std_logic_VECTOR(15 downto 0);
	addra: IN std_logic_VECTOR(13 downto 0);
	wea: IN std_logic_VECTOR(0 downto 0);
	douta: OUT std_logic_VECTOR(15 downto 0));
end component;

-- Configuration specification
	for all : wrapped_buffer_input use entity XilinxCoreLib.blk_mem_gen_v2_7(behavioral)
		generic map(
			c_has_regceb => 0,
			c_has_regcea => 0,
			c_mem_type => 0,
			c_prim_type => 1,
			c_sinita_val => "0",
			c_read_width_b => 16,
			c_family => "virtex2p",
			c_read_width_a => 16,
			c_disable_warn_bhv_coll => 0,
			c_write_mode_b => "WRITE_FIRST",
			c_init_file_name => "no_coe_file_loaded",
			c_write_mode_a => "WRITE_FIRST",
			c_mux_pipeline_stages => 0,
			c_has_mem_output_regs_b => 0,
			c_load_init_file => 0,
			c_xdevicefamily => "virtex2p",
			c_has_mem_output_regs_a => 0,
			c_write_depth_b => 10240,
			c_write_depth_a => 10240,
			c_has_ssrb => 0,
```

```
                c_has_mux_output_regs_b => 0,
                c_has_ssra => 0,
                c_has_mux_output_regs_a => 0,
                c_addra_width => 14,
                c_addrb_width => 14,
                c_default_data => "0",
                c_use_ecc => 0,
                c_algorithm => 1,
                c_disable_warn_bhv_range => 0,
                c_write_width_b => 16,
                c_write_width_a => 16,
                c_read_depth_b => 10240,
                c_read_depth_a => 10240,
                c_byte_size => 9,
                c_sim_collision_check => "ALL",
                c_use_ramb16bwer_rst_bhv => 0,
                c_common_clk => 0,
                c_wea_width => 1,
                c_has_enb => 0,
                c_web_width => 1,
                c_has_ena => 0,
                c_sinitb_val => "0",
                c_use_byte_web => 0,
                c_use_byte_wea => 0,
                c_use_default_data => 0);
-- synthesis translate_on
BEGIN
-- synthesis translate_off
U0 : wrapped_buffer_input
        port map (
                clka => clka,
                dina => dina,
                addra => addra,
                wea => wea,
                douta => douta);
-- synthesis translate_on

END buffer_input_a;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity combination is
      Port ( clk : in    STD_LOGIC;
                  data_high_pass : in    STD_LOGIC_VECTOR (15 downto 0);
                  data_low_pass : in    STD_LOGIC_VECTOR (15 downto 0);
                  data_out : out    STD_LOGIC_VECTOR (15 downto 0));
end combination;

architecture Behavioral of combination is

begin

process(clk)
begin
if clk'event and clk='1' then

data_out<=data_high_pass+data_low_pass;
end if;
end process;

end Behavioral;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;



entity S_to_P IS
port (      reset:            in    std_logic;

                        clk:              in std_logic;
                        data_in:    in    std_logic_vector(15 downto 0);

            data_out:  out std_logic_vector(79 downto 0)
```

```
);
end S_to_P;

architecture connect_5 of S_to_P is

signal temp_out:      std_logic_vector(79 downto 0);
signal i: integer range 0 to 7;
begin

process(clk, reset)

begin
if reset='0' then
      data_out<=(others=>'0');
      temp_out<=(others=>'0');
      i<=0;



    elsif clk'event and clk='1' then
      case i is
      when 0 =>

                                    temp_out(15 downto 0)<=data_in;
                                    i<=i+1;

      when 1 =>
                                    temp_out(31 downto 16)<=data_in;
                                    i<=i+1;

      when 2 =>
                                    temp_out(47 downto 32)<=data_in;
                                    i<=i+1;

      when 3 =>
                                    temp_out(63 downto 48)<=data_in;
                                    i<=i+1;

      when 4 =>
                                    temp_out(79 downto 64)<=data_in;
                                    i<=0;
```

```
                                        data_out<=temp_out;


  when others=>null;
       end case;

       end if;

end process;

end connect_5;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;



entity convolution IS
port (       reset:              in     std_logic;
                        start:       in     std_logic;
                        clk:                in std_logic;
                        data_in:    in     std_logic_vector(79 downto 0);
                        coeff_in: in      std_logic_vector(79 downto 0);
             data_out:  out std_logic_vector(15 downto 0)

);
end convolution;

architecture convolution_5 of convolution is

signal multi_out:      std_logic_vector(27 downto 0);


begin

process(clk, reset)

begin
if reset='0' then
```

```
        data_out<=(others=>'0');

        multi_out(27 downto 16)<=(others=>'0');
        multi_out(15 downto 0)<=(others=>'0');




      elsif clk'event and clk='1' then
        if    start = '1' then
          multi_out(27 downto 0)<=data_in(79 downto 64)*coeff_in(49 downto 40)
          +data_in(63 downto 48)*coeff_in(39 downto 30)+data_in(47 downto 32)*coeff_in(29 downto 20)
          +data_in(31 downto 16)*coeff_in(19 downto 10)+data_in(15 downto 0)*coeff_in(9 downto 0);

        end if;

end if;

data_out<=multi_out(15 downto 0);

end process;

end convolution_5;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synthesis translate_off
Library XilinxCoreLib;
-- synthesis translate_on
ENTITY coeff_memory IS
        port (
        clka: IN std_logic;
        dina: IN std_logic_VECTOR(79 downto 0);
        addra: IN std_logic_VECTOR(12 downto 0);
        wea: IN std_logic_VECTOR(0 downto 0);
        douta: OUT std_logic_VECTOR(79 downto 0));
END coeff_memory;

ARCHITECTURE coeff_memory_a OF coeff_memory IS
-- synthesis translate_off
component wrapped_coeff_memory
```

```
        port (
        clka: IN std_logic;
        dina: IN std_logic_VECTOR(79 downto 0);
        addra: IN std_logic_VECTOR(12 downto 0);
        wea: IN std_logic_VECTOR(0 downto 0);
        douta: OUT std_logic_VECTOR(79 downto 0));
end component;

-- Configuration specification
        for all : wrapped_coeff_memory use entity XilinxCoreLib.blk_mem_gen_v2_7(behavioral)
                generic map(
                        c_has_regceb => 0,
                        c_has_regcea => 0,
                        c_mem_type => 0,
                        c_prim_type => 1,
                        c_sinita_val => "0",
                        c_read_width_b => 80,
                        c_family => "virtex2p",
                        c_read_width_a => 80,
                        c_disable_warn_bhv_coll => 0,
                        c_write_mode_b => "WRITE_FIRST",
                        c_init_file_name => "no_coe_file_loaded",
                        c_write_mode_a => "WRITE_FIRST",
                        c_mux_pipeline_stages => 0,
                        c_has_mem_output_regs_b => 0,
                        c_load_init_file => 0,
                        c_xdevicefamily => "virtex2p",
                        c_has_mem_output_regs_a => 0,
                        c_write_depth_b => 6144,
                        c_write_depth_a => 6144,
                        c_has_ssrb => 0,
                        c_has_mux_output_regs_b => 0,
                        c_has_ssra => 0,
                        c_has_mux_output_regs_a => 0,
                        c_addra_width => 13,
                        c_addrb_width => 13,
                        c_default_data => "0",
                        c_use_ecc => 0,
                        c_algorithm => 1,
                        c_disable_warn_bhv_range => 0,
                        c_write_width_b => 80,
                        c_write_width_a => 80,
```

```
                c_read_depth_b => 6144,
                c_read_depth_a => 6144,
                c_byte_size => 9,
                c_sim_collision_check => "ALL",
                c_use_ramb16bwer_rst_bhv => 0,
                c_common_clk => 0,
                c_wea_width => 1,
                c_has_enb => 0,
                c_web_width => 1,
                c_has_ena => 0,
                c_sinitb_val => "0",
                c_use_byte_web => 0,
                c_use_byte_wea => 0,
                c_use_default_data => 0);
-- synthesis translate_on
BEGIN
-- synthesis translate_off
U0 : wrapped_coeff_memory
        port map (
                clka => clka,
                dina => dina,
                addra => addra,
                wea => wea,
                douta => douta);
-- synthesis translate_on

END coeff_memory_a;
```

PUBLICATIONS