

**ON THE IMPLEMENTATION OF STRUCTURE FUNCTION IN THE DIGITAL
SIGNAL PROCESSOR**

Mehmet Ergezer

I hereby release this thesis to the public; I understand that this thesis will be made available from the OhioLINK ETD Center and the Maag Library Circulation Desk for public access. I also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

Mehmet Ergezer

Signature:

Submitted in Partial Fulfillment of the Requirements

Mehmet Ergezer, Student

4/27/2006
Date

for the Degree of

Master of Science in Engineering

Approval:

in the

Electrical and Computer Engineering

Dr. Paramore Monroy, Program Advisor

4/27/06
Date

Dr. Jalal Jalali, Committee Member

4/27/2006
Date

YOUNGSTOWN STATE UNIVERSITY

Dr. Salvatore Panico, Committee Member

4/27/06
Date

May, 2006

Peter J. Karvinsky, Dean of Graduate Studies and Research

4/27/06
Date

**ON THE IMPLEMENTATION OF STRUCTURE FUNCTION IN THE DIGITAL
SIGNAL PROCESSOR**

Mehmet Ergezer

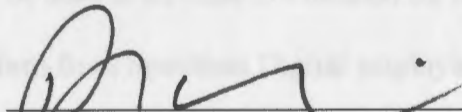
I hereby release this thesis to the public. I understand that this thesis will be made available from the OhioLINK ETD Center and the Maag Library Circulation Desk for public access. I also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

Signature:


Mehmet Ergezer, Student

04/27/2006
Date


Approvals:


Dr. Faramarz Mossayebi, Thesis Advisor

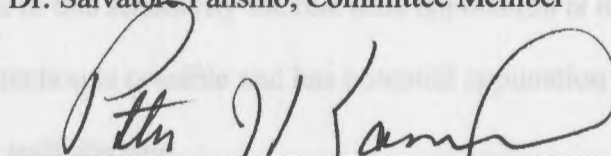
4/27/06
Date


Dr. Jalal Jalali, Committee Member

4/27/2006
Date


Dr. Salvatore Pansino, Committee Member

4/27/06
Date


Peter J. Kasvinsky, Dean of Graduate Studies and Research

4/27/06
Date

ABSTRACT

ON THE IMPLEMENTATION OF STRUCTURE FUNCTION IN THE DIGITAL SIGNAL PROCESSOR

The implementation of the structure function on a digital signal processor board to detect incipient stall and its applicability for online or near online applications were investigated. Previous research proved that the structure function successfully detects precursors to stall in axial compressors in off-line situations. The simplicity of the detection mechanism from a computational point of view suggests that the structure function could be used as an effective method for online stall detection. An eZdsp hardware platform from Spectrum Digital employing TMS320F2812 DSP by Texas Instruments was utilized to implement the structure function in this study. The ability to detect chaos in a time series was tested since the fundamental theory is affiliated with chaos theory. The experimental data obtained from high-speed compressors were then filtered with structure function implemented on the DSP board. The structure function applied to these systems showed sensitivity to the system dynamics and pre-filtering conditions. Due to this sensitivity the real time application is not feasible, although the near on-line detection is possible and has potential application in a supervisory control mechanism for stall warning.

ACKNOWLEDGMENTS

I would like to acknowledge the great support Dr. Paramariz Mossayehi has shown me during my graduate and undergraduate studies. Without his help, this work would not be possible. I would also like to thank Dr. Salvatore R. Passaro and Dr. Jalal Jafari for all their help and guidance over the years. Special thanks go to Rick Laughlin for all the valuable feedback he gave me during my research and Carolyn Denny-Schaefer for always being so understanding.

In addition, I would like to express my sincere thanks to Dr. Robert Fouikes, Dr. Phillip Mauro, Dr. Robert Kramer and many other professors at Youngstown State University for all the knowledge they passed on to me and my classmates. Finally, I would like to thank my parents, who never stopped believing in me.

ACKNOWLEDGMENTS

I would like to acknowledge the great support Dr. Faramarz Mossayebi has shown me during my graduate and undergraduate studies. Without his help, this work would not be possible. I would also like to thank Dr. Salvatore R. Pansino and Dr. Jalal Jalali for all their help and guidance over the years. Special thanks go to Rick Laughlin for all the valuable feedback he gave me during my research and Carolyn Denny-Schaefer for always being so understanding.

In addition, I would like to express my sincere thanks to Dr. Robert Foulkes, Dr. Phillip Munro, Dr. Robert Kramer and many other professors at Youngstown State University for all the knowledge they passed on to me and my classmates. Finally, I would like to thank my parents, who never stopped believing in me.

III. DIGITAL SIGNAL PROCESSING.....	21
3.1 INTRODUCTION TO DIGITAL SIGNAL PROCESSING.....	21
3.2 BASICS OF DIGITAL SIGNAL PROCESSING.....	23
3.3 CHARACTERISTICS OF SPECTRUM DIGITAL WITH DZDP	29
3.3.1 F2812 Interrupt System	33
3.3.2 F2812 Event Manager	34
3.3.3 Analog-to-Digital Converter	35
3.3.4 Fixed Point and Floating Point	36

IV. IMPLEMENTATION OF THE STRUCTURE FUNCTION WITH A DSP.....	37
4.1 CHUA AND ROSSLER SYSTEMS.....	37
4.2 ROTOR37 and T55.....	40

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION.....	1
II. STRUCTURE FUNCTION.....	6
2.1 BACKGROUND INFORMATION ON STRUCTURE FUNCTION.....	6
2.2 CHUA AND ROSSLER SYSTEMS.....	8
2.3 BIFURCATION DIAGRAM.....	12
2.4 STRUCTURE FUNCTION SIMULATION.....	14
2.4.1 <i>Chua and Rossler Systems</i>	14
2.4.2 <i>Stage 37 Compressor</i>	18
2.4.3 <i>T55 Multi-stage Engine</i>	18
III. DIGITAL SIGNAL PROCESSING.....	21
3.1 INTRODUCTION TO DIGITAL SIGNAL PROCESSING.....	21
3.2 BASICS OF DIGITAL SIGNAL PROCESSING.....	23
3.3 CHARACTERISTICS OF SPECTRUM DIGITAL'S EZDSP.....	29
3.3.1 <i>F2812 Interrupt System</i>	33
3.3.2 <i>F2812 Event Manager</i>	34
3.3.3 <i>Analog-to-Digital Converter</i>	35
3.3.4 <i>Fixed Point and Floating Point</i>	36

IV. IMPLEMENTATION OF THE STRUCTURE FUNCTION WITH A DSP.....	37
4.1 CHUA AND ROSSLER SYSTEMS.....	37
4.2 ROTOR37 AND T55	40
V. SUMMARY OF RESULTS AND DISCUSSION FOR FUTURE WORK.....	43
REFERENCES	45
APPENDICES.....	47
APPENDIX 1	48
APPENDIX 2	51
APPENDIX 3	54
APPENDIX 4	56
APPENDIX 6	58
APPENDIX 6	61
APPENDIX 7	64
APPENDIX 8	67
3-1 Common Applications of Signal Processing	21
3-2 Sampled Analog Signal	23
3-3 Frequency Response of a Complex Signal	24
3-4 Block Diagram of a Common Signal Processing System	26
3-5 xZap Connector Positions	30
3-6 F2812 Block Diagram	32
3-7 Peripheral Interrupt Expansion	33
3-8 F2812 Event Manager Block Diagram	34

3-4 Simplified Block Diagram of the ADC 35

4-1 Structure Function Running **LIST OF FIGURES** 37

4-2 Offset Plot of the Implemented SF for Chua System 37

FIGURE : Plot of the Implemented SF for Rossler System **PAGE**

2-1 Periodic Response of Chua's System with Parameter $A=8.60$ 9

2-2 Chaotic Response of Chua's system with Parameter $A=8.90$ 9

2-3 Periodic Response of Rossler's System with Parameter $c=5.50$ 10

2-4 Chaotic Response of Rossler's System with Parameter $c=4.50$ 10

2-5 Bifurcation Diagram of Chua's System 12

2-6 Bifurcation Diagram of Rossler's System 12

2-7 Response of SF to Various Step and Window Sizes 14

2-8 Structure Function for Chua's System 15

2-9 Structure Function for Rossler's System 16

2-10 Rotor 37 Simulation 18

2-11 T55 Simulation 19

3-1 Common Applications of Signal Processing 21

3-2 Sampled Analog Signal 23

3-3 Frequency Response of a Complex Signal 24

3-4 Block Diagram of a Common Signal Processing System 26

3-5 eZdsp Connector Positions 30

3-6 F2812 Block Diagram 32

3-7 Peripheral Interrupt Expansion 33

3-8 F2812 Event Manager Block Diagram 34

3-9 Simplified Block Diagram of the ADC	35
4-1 Structure Function Running on DSP	37
4-2 Offline Plot of the Implemented SF for Chua System	37
4-3 Offline Plot of the Implemented SF for Rossler System	38
4-4 DSP Running SF on Averaged and Normalized T55 Data	40
4-5 DSP Running SF on Averaged and Normalized Rotor 37 Data	41

LIST OF TABLES

TABLE	PAGE
3-1 eZdsp Memory Space	29
3-2 eZdsp Jumpers Factory Settings	31

Many engineering applications such as turbojet engines, refrigerators/air-conditioners and industrial compressors depend on compressors. Compressors decrease the volume of a gas in order to increase its pressure [1]. The most common types of compressors are positive displacement, centrifugal and axial. Axial compressors increase the kinetic energy by accelerating the velocity of the gases, which are then converted into pressure. This type of compressors provides much higher flow rate than positive displacement compressors. Gas turbine engines, which power most of the commercial jets, use axial compressors. These compressors are designed to operate under constant, non-fluctuating load. The performance of the engine can be improved by decreasing the mass flow rate, which then increases the pressure rise. However, changing the mass flow rate can cause the compressor to be unstable since the compressor is designed for steady and axisymmetric flow. These instability consequences are categorized as surge, rotating stall and classical surge.

Surge is an axisymmetric oscillation of the mass flow along the axial length of the compressor. It has a higher amplitude and lower frequency than rotating stall and both fan and compressor systems can be affected by surge. If this oscillation is strong enough to cause a temporary change in flow direction, it is referred to as deep surge.

I. INTRODUCTION

Rotating stall is a severely non-axisymmetric distribution of axial flow velocity, taking the form of a wave, which propagates steadily in the circumferential direction at a fraction of the rotor speed. These large vibratory stresses usually affect the loading of axial compressor and are, therefore, often unacceptable for structural reasons [2]. Furthermore, they can cause a significant drop in the engine performance (a drop of more than 40 percent is shown in [3]). In some cases, a restart of the engine is required to recover.

Many engineering applications such as turbojet engines, refrigerators/air-conditioners and industrial compressors depend on compressors. Compressors decrease the volume of a gas in order to increase its pressure [1]. The most common types of compressors are positive displacement, centrifugal and axial. Axial compressors increase the kinetic energy by accelerating the velocity of the gases, which are then converted into pressure. This type of compressors provides much higher flow rate than positive displacement compressors. Gas turbine engines, which power most of the commercial jets, use axial compressors. These compressors are designed to operate under constant, non-fluctuating load. The performance of the engine can be improved by decreasing the mass flow rate, which then increases the pressure rise. However, changing the mass flow rate can cause the compressor to be unstable since the compressor is designed for steady and axisymmetric flow. These instability consequences are categorized as surge, rotating stall and classical surge.

Surge is an axisymmetric oscillation of the mass flow along the axial length of the compressor. It has a higher amplitude and lower frequency than rotating stall and both fan and compressor systems can be affected by surge. If this oscillation is strong enough to cause a temporary change in flow direction, it is referred to as deep surge.

Rotating stall is a severely non-axisymmetric distribution of axial flow velocity, taking the form of a wave, which propagates steadily in the circumferential direction at a fraction of the rotor speed. These large vibratory stresses usually affect the blading of axial compressor and are, therefore, often unacceptable for structural reasons [2].

Furthermore, they can cause a significant drop in the engine performance (a drop of more than 80 percent is shown in [3]). In some cases, a restart of the engine is required to recover from rotating stall.

If during a surge cycle, the compressor goes through a rotating stall, this phenomenon is called a classic surge. By leaving a margin between the operating mass flow rate of the air compressor and the mass flow rate at which the stall occurs (i.e. stall line), the instabilities can be avoided. Because of the possible catastrophic consequences that could result from such instabilities, especially in aircraft engine applications [2], the designer must ensure that there is enough margin between the stall line and the operating line, known as stall margin (typically 25%). Keeping a large stall margin causes the performance penalty and decreases the operating range of the engine. And even when leaving a stall margin, in such situations as rotor speed transients, flow distortions, and unsteady inlet and exit pulsations, a stall can occur. Based on this fact, it is essential to detect the precursors to stall in order to be able to recover from stall condition.

Recently, much research has been done to reduce the stall margin to maximize the engine performance. This requires prior knowledge to approaching stall by locating stall precursors. Then, a feedback control system can be used to lower the working line or lower the mass flow to increase the system stability. Several techniques already exist in

literature today to detect the stall precursors. Tryfonidis et al. [6] used the traveling wave energy technique assuming that the stall was caused by the instability of small amplitude circumferential waves and Day [7] used a nonlinear method to argue that before a stall occurs, short length-scale pressure disturbances take place. Another approach, suggested by Bright et al. [8], is the use of chaotic time series analysis method, a correlation integral, to observe an approaching stall.

The nonlinear techniques are based on the idea that when stall approaches, a change in a compressor event will occur, and if some parameters change during this time, then the technique could be used to identify the stall. The correlation integral successfully detects stall precursors; however, it cannot not be easily implemented due to the data size and processing resources it requires. M. H. Vhora [9] argues that structure function could be seen as a single dimensional variation of the correlation integral and, therefore, requires less data and computing power that could be collected from a single sensor. Given this, structure function is the preferable method for online implementation to detect stall in real-time and then for informing the control scheme to prevent stall from happening. This thesis investigates the use of the structure function on simulated data running on a digital signal processor board to detect incipient stall. Previous research suggests that the structure function successfully detects precursors to stall in axial compressors, while its simple format provides effortless design. These reasons justify the use structure function as an effective method for online stall detection.

In order to test the success of the structure function to detect stall, the ability to detect chaos in a time series is tested since the fundamental theory is affiliated with chaos theory. The Chua and Rossler systems are selected to serve as the chaotic systems, as

their response displays various nonlinear dynamic behaviors based on the value of the chosen parameter. A time vector is created, and the structure function is applied to these systems as they are driven to chaos to observe online ability of structure function to detect chaos. An eZdsp platform from Spectrum Digital employing TMS320F2812 DSP by Texas Instruments is utilized to implement the structure function. The simulation data for Chua's and Rossler's systems is obtained by using MATLAB, which is then used by the DSP board, as if they were real-time measurements, to implement the structure function and to observe and quantify the chaotic nature of the systems. This is then followed by the analysis of several time series associated with a high-speed compressor to detect the incipient of stall. Thus, the primary objectives of this research are to apply the structure function to nonlinear systems in general and a high speed compressor data, detect dynamical changes, and finally, examine its capabilities and limitations for the real time implementation.

This work is structured as follows. Chapter 2 provides the background information on the mathematical foundation of structure function and random processes. In addition the chapter presents information on Chua's and Rossler's systems and their time series, as well as their bifurcation diagram. This chapter also displays simulated results of structure function to these time series as well as a high-speed compressor running at stage 37 and T55 multi-stage engine. A brief introduction to digital signal processing is presented in Chapter 3. The specifications for the eZdsp board used during this research are also provided in this chapter. Chapter 4 includes results of applying the structure function, as implemented on the eZdsp board, to the Chua's and Rossler's systems, as well as the high-speed compressor's data. The comparisons of results along

with the limitation of the technique used, as well as potential areas for future research are provided in Chapter 5.

II. STRUCTURE FUNCTION

2.1 Background Information on Structure Function

The structure function has been developed as a tool for random variable analysis and applied in electronics, automatic regulation and the theory of turbulence [15]. This section gives background information on the utilization of structure function in random variables.

When using continuous random variables, which are numeric results of operating a non-deterministic mechanism, random functions of time yield different results for various measurements taken under the same conditions. The observation results obtained using random functions are called stochastic processes.

Random functions cannot be analyzed applying classical theory of probability. Appropriate mathematical tools for such analysis were partially developed in 1940s by A.N. Kolmogorov (known as Kolmogorov extension), E.B. Slutsky and, A.Y. Khinchin. Kolmogorov extension makes it possible to construct stochastic processes with fairly arbitrary finite-dimensional distributions. Also, all issues related to a sequence would have a probabilistic answer if they were taken as a random sequence. Unfortunately, some questions about functions on continuous domain do not have a probabilistic answer.

II. STRUCTURE FUNCTION

2.1 Background Information on Structure Function

The structure function has been developed as a tool for random variable analysis and applied in electronics, automatic regulation and the theory of turbulence [15]. This section gives background information on the utilization of structure function in random variables.

When using continuous random variables, which are numeric results of operating a non-deterministic mechanism, random functions of time yield different results for various measurements taken under the same conditions. The observation results obtained using random functions are called stochastic processes.

Random functions cannot be analyzed applying classical theory of probability. Appropriate mathematical tools for such analysis were partially developed in 1940s by A.N. Kolmogorov (known as Kolmogorov extension), E.E. Slutsky and, A.Y. Khinchi. Kolmogorov extension makes it possible to construct stochastic processes with fairly arbitrary finite-dimensional distributions. Also, all issues related to a sequence would have a probabilistic answer if they were taken as a random sequence. Unfortunately, some questions about functions on continuous domain do not have a probabilistic answer.

If the mean values and variances of a process are constant, while the correlation function only depends upon the difference of $t_2 - t_1$, the process is considered to be stationary. The majority of random processes encountered in real life consist of stationary processes. Also, if parts of a process are stationary over intervals of time, τ , the process can be identified as quasi-stationary. However, if the value of τ is large enough, the mean can change significantly, causing the loss of stationary conditions.

For random processes $x(t)$ with stationary increments, $\Delta_\tau x(t) = x(t+\tau) - x(t)$, the processes $x(t)$ themselves are not stationary [17,18]. The structure function serves as a fundamental characteristic of random processes with stationary increments. This equation is used in various fields as correlation of two-point velocity differences. Kolmogorov introduced the structure function

$$SF(\tau) = \langle [x(t+\tau) - x(t)]^2 \rangle \quad (2.1)$$

to random fields [16]. The Eq. (2.1) can be described as the moment of the difference field $\Delta_\tau x(t)$ in the direction of τ . Kolmogorov employed the structure function to describe the dissipation of energy in the locally isotropic turbulence. He concluded that

$$\langle [u(x+r) - u(x)]^q \rangle = C_q \epsilon^{q/3} r^{q/3} \quad (2.2)$$

where u is the streamwise component of the turbulent fluctuating velocity, r is the separation of measuring points in the direction of flow, C_q is a constant, ϵ is energy dissipation rate and q is the order of the structure function [16].

The structure equation is also utilized in fluid mechanics to detect the intermittency in fluid flow behavior. Intermittency is defined as the random, rapid switching from tranquil to bursting behaviors.

M. H. Vhora [9] states that when dealing with stationary random processes, using

the structure function instead of the correlation function is more convenient if we are interested in the difference of the random processes in different times, not their absolute values. Vhora employs the following structure function:

$$SF(n) = \frac{1}{N-n} \sum_{i=1}^{N-n} [x(i+n) - x(i)]^2 \quad (2.3)$$

where the structure function is defined for any time series $x(k)$, and x_i is the i^{th} sample number in the time series. N is the data window size, and τ is the difference in number of samples for which the structure function is calculated.

2.2 Chua and Rossler Systems

Two systems of differential equations, Chua's circuit and Rossler attractor, are used to demonstrate the capability of the SF to detect the chaotic changes in a system. Both systems are simulated using MATLAB and are driven to chaos in order to analyze the correspondence between the structure function and the dynamic behavior of the system.

The Chua's system is given by Eqs. (2.4) to (2.6):

$$\frac{dx}{dt} = A \left(y + \frac{x - 2x^3}{7} \right) \quad (2.4)$$

$$\frac{dy}{dt} = x - y + z \quad (2.5)$$

$$\frac{dz}{dt} = \frac{-100}{7} y \quad (2.6)$$

Initially x , y and z are set to 0.1, 0.02 and -0.3, respectively. As displayed in Figure 2-1 and Figure 2-2, the system response changes from periodic to

chaotic as the value of parameter A is changed from 8.60 to 8.90. Appendix 1 displays the MATLAB commands used to create the system response. This script varies the parameter A from 8.65 to 8.95 in 200 equal steps. For each value of the parameter, first 12,000 data point obtained from this simulation are discarded to ignore the transient response and the following 5,000 data points are collected. At the end of the program, an array of one million data points (i.e., 5,000 points collected for each one of the 200 steps) is stored on the host machine.

Rossler's system is also simulated on MATLAB in a similar manner using the differential Eqs. (2.7) to (2.9):

$$\frac{dx}{dt} = -y - z \quad (2.7)$$

$$\frac{dy}{dt} = -x + ay \quad (2.8)$$

$$\frac{dz}{dt} = b + xz - cz \quad (2.9)$$

The parameter of interest, c, is swept from 4 to 6 in 200 equal steps and the first 20,000 transient data points are discarded in each step. The initial conditions for the variables are taken as x=0.01, y=0.1, z=0.03, a=0.2 and b=0.2. The script used for system simulation is shown in Appendix 2. Figure 2-3 and 2-4 show the periodic and chaotic response of Rossler's system, respectively.

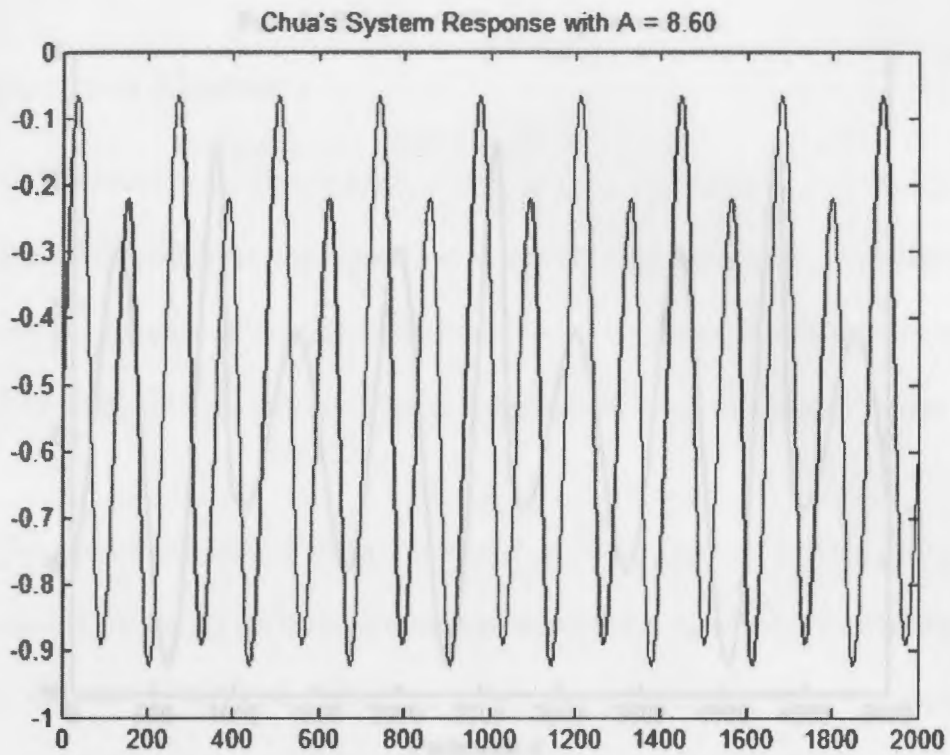


Figure 2-1 Periodic Response of Chua's System with Parameter $A=8.60$

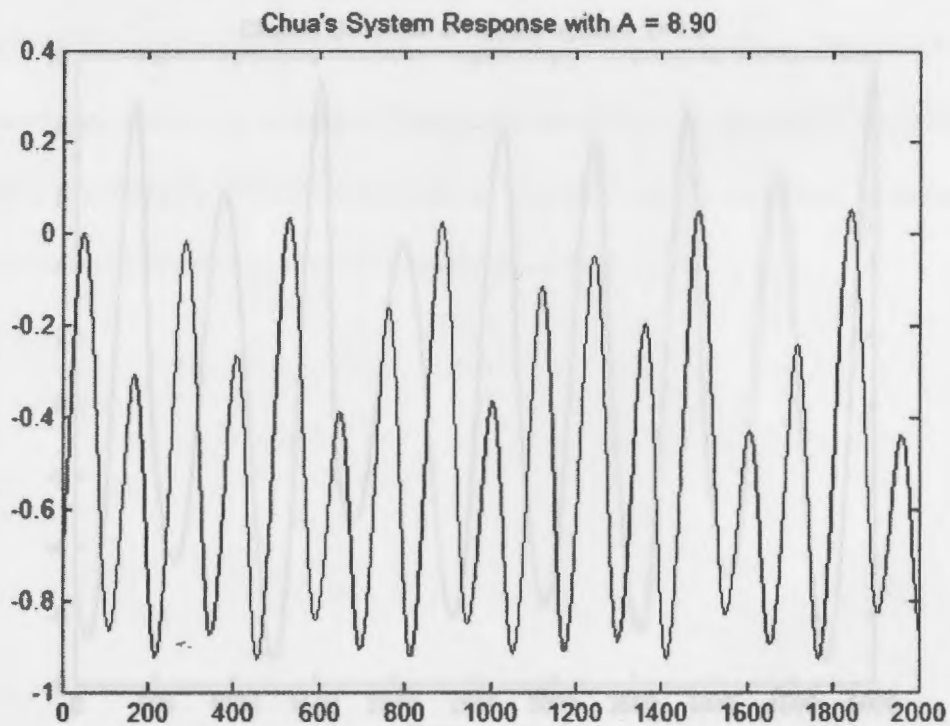


Figure 2-2 Chaotic Response of Chua's System with Parameter $A=8.90$

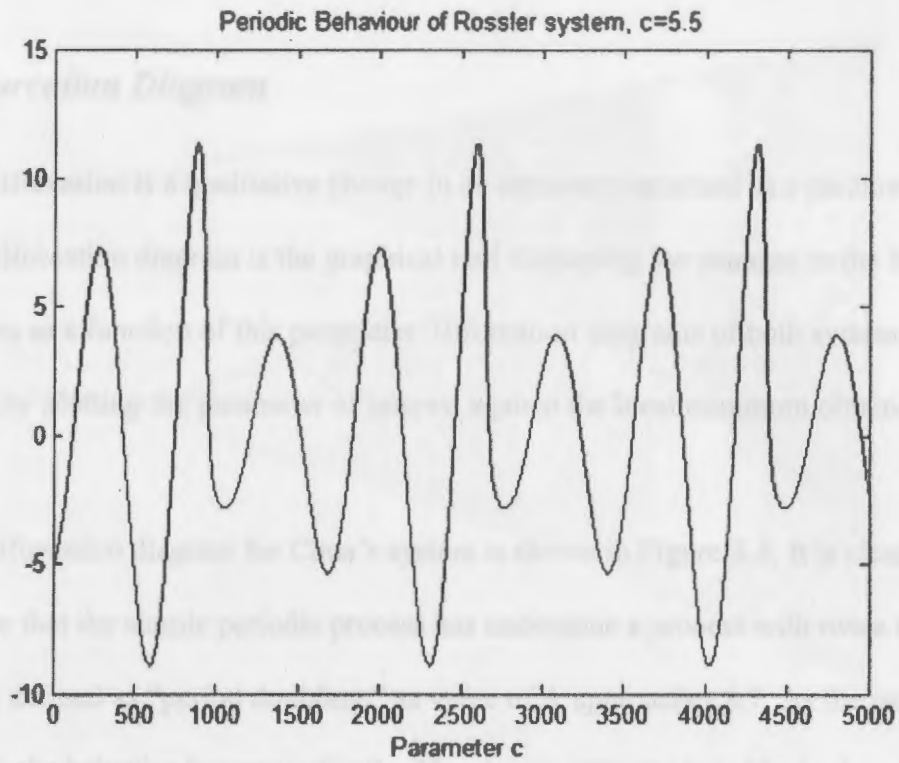


Figure 2-3 Periodic Response of Rossler's System with Parameter $c=5.50$

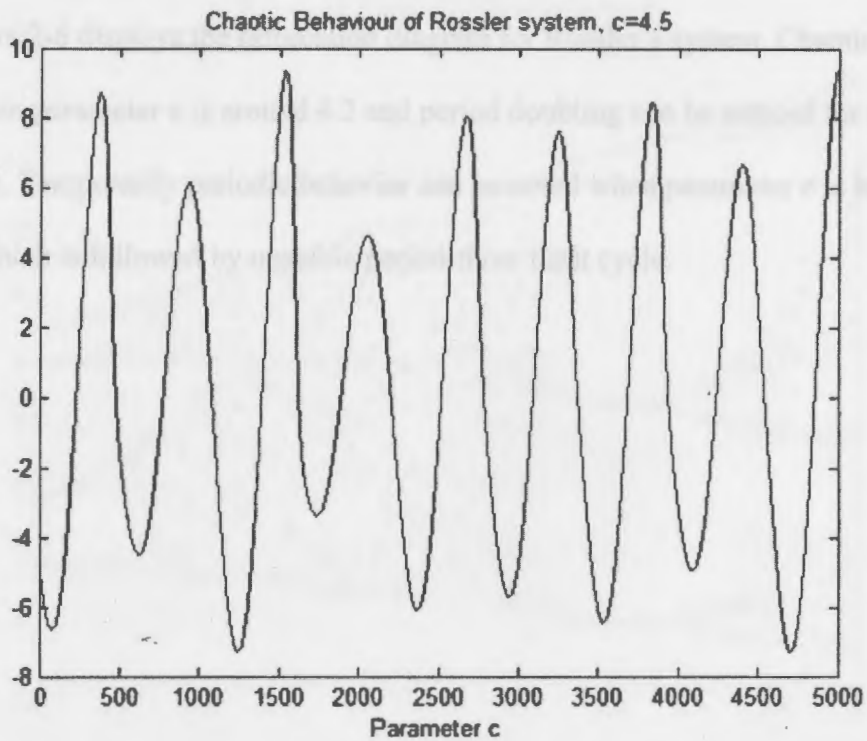


Figure 2-4 Chaotic Response of Rossler's System with Parameter $c=4.50$

2.3 Bifurcation Diagram

Bifurcation is a qualitative change in an attractor's structure as a parameter is varied. Bifurcation diagram is the graphical tool displaying the changes in the behavior of the system as a function of this parameter. Bifurcation diagrams of both systems are obtained by plotting the parameter of interest against the local minimum obtained in each step.

Bifurcation diagram for Chua's system is shown in Figure 2-5. It is clearly seen in this figure that the simple periodic process has undergone a process with twice the period, a process defined as "period doubling," as value of A approaches 8.7. As the parameter is increased, the behavior becomes chaotic. The chaotic attractor is evident when A is 8.78. Also, the system response temporally returns to periodic when A is between 8.87 and 8.88. Figure 2-6 displays the bifurcation diagram for Rossler's system. Chaotic attractor begins when parameter c is around 4.2 and period doubling can be noticed for smaller values of c . Temporarily periodic behavior can be noted when parameter c is between 5.2 and 5.6, which is followed by unstable period-three limit cycle.

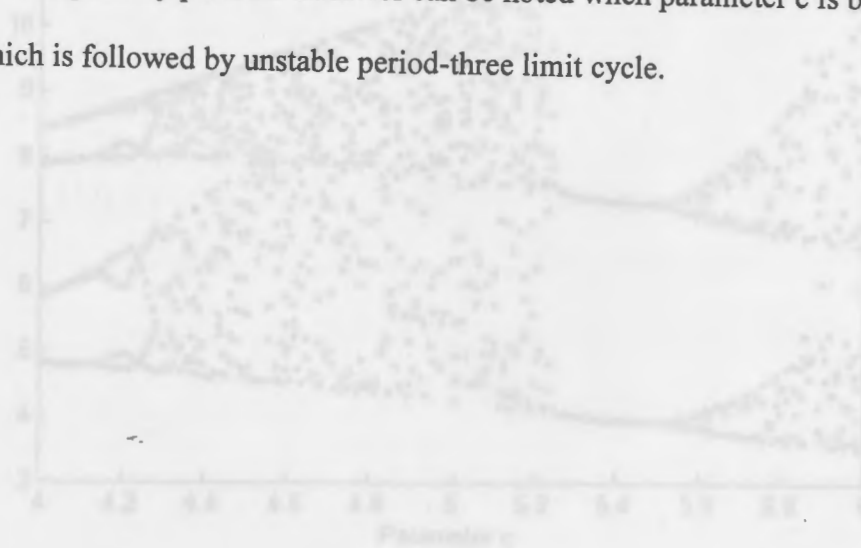


Figure 2-6 Bifurcation Diagram of Rossler's System

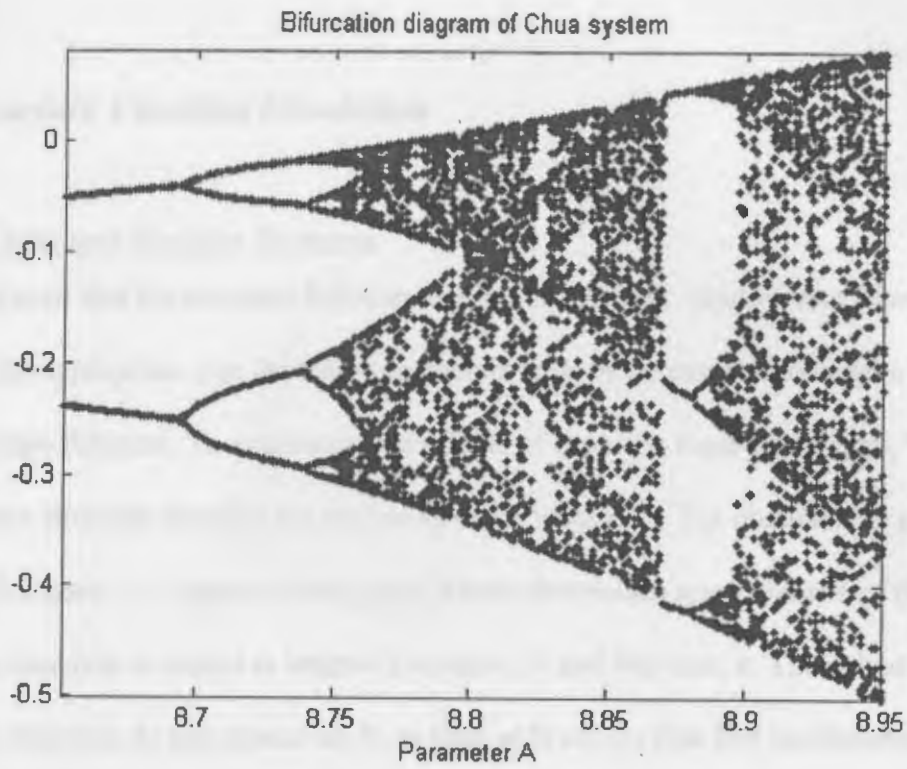


Figure 2-5 Bifurcation Diagram of Chua's System

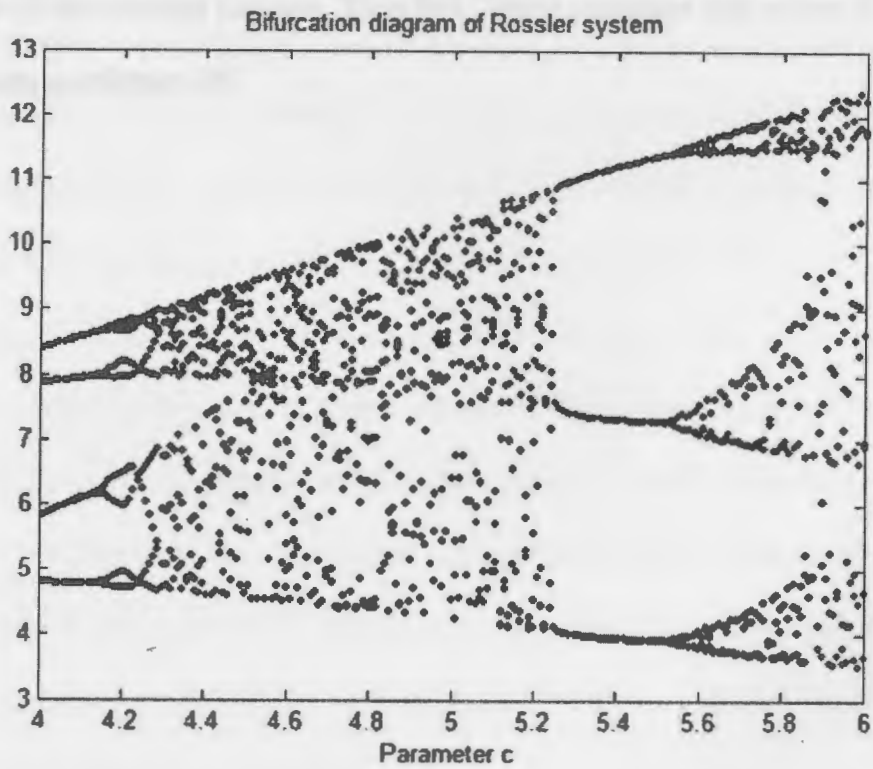


Figure 2-6 Bifurcation Diagram of Rossler's System

2.4 Structure Function Simulation

2.4.1 Chua and Rossler Systems

Recall that the structure function requires two values: window size, N and step size, n . An appropriate size for these parameters must be determined to be able to apply the structure function. To understand the effects of choosing these parameters, Vhora applies the structure function for various N and n values [9]. The obtained 3D graph is shown in Figure 2-7. Based on this graph, Vhora determines some features of the structure function in regard to length of window, N and step size, n . The values of the structure function do not depend on N , as long as N covers first few oscillations.

Secondly, any value n will work as good as the other. The only difference will be the amplitude of the resulting function. Therefore, Vhora concludes that values of N and n can be taken as arbitrary [9].

Structure function for Chua's system is presented in Figure 2-8. Window length of 100 and step size of 48 are selected for simulation (see Appendix Y for the MATLAB script). Comparing bifurcation diagram (Figure 2-7) and the structure function (Figure 2-8), we can see that the structure function is smooth in regions where the system dynamics are periodic, but it becomes wiggly as the system becomes a chaotic attractor. Also, notice that when A is 8.27, the system changes behavior from chaotic to periodic and this corresponds to a sudden drop in the structure function. Similarly, when A is around 8.9, a sudden jump occurs in the structure function as the system re-enters the chaotic region. From these responses, we can interpret that the structure function can detect dynamic changes and, therefore, is suitable for detecting dynamic system changes before stall.

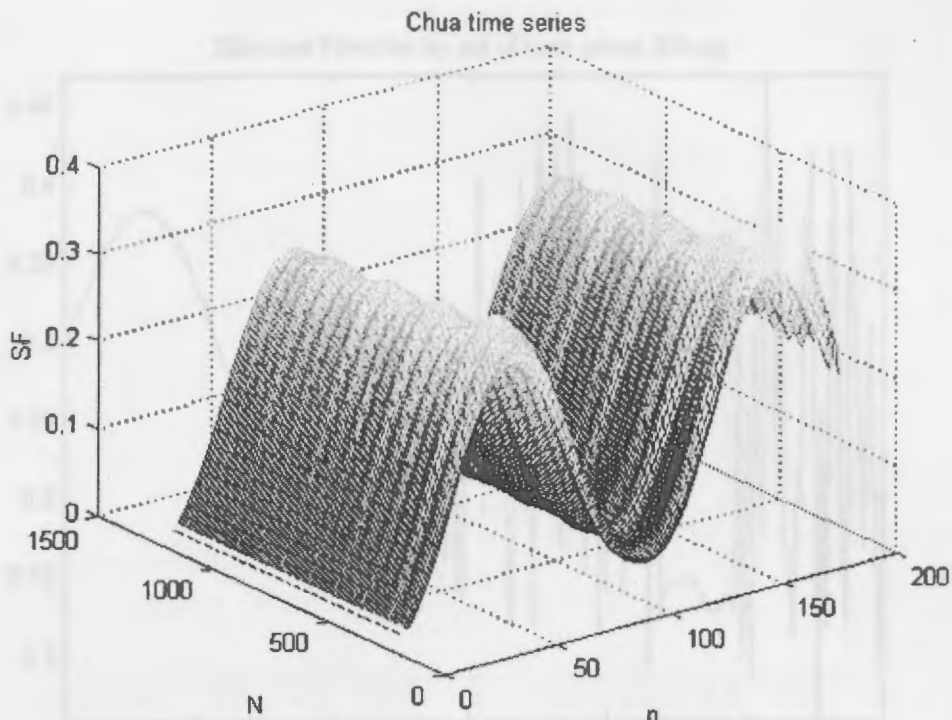


Figure 2-7 Response of SF to Various Step and Window Sizes

Structure function for Chua's system is presented in Figure 2-8. Window length of 100 and step size of 48 are selected for simulation (see Appendix 3 for the MATLAB script). Comparing bifurcation diagram (Figure 2-5) and the structure function (Figure 2-8), we can see that the structure function is sensitive to changes in the system dynamics as the response in periodic regions is smooth, but it becomes wiggly as the system becomes a chaotic attractor. Also, notice that when A is 8.87, the system changes behavior from chaotic to periodic and this corresponds to a sudden drop in the structure function. Similarly, when A is around 8.9, a sudden jump occurs at the structure function as the system re-enters the chaotic region. From these responses, we can interpret that the structure function can detect dynamic changes and, therefore, is suitable for detecting dynamic system changes before stall.

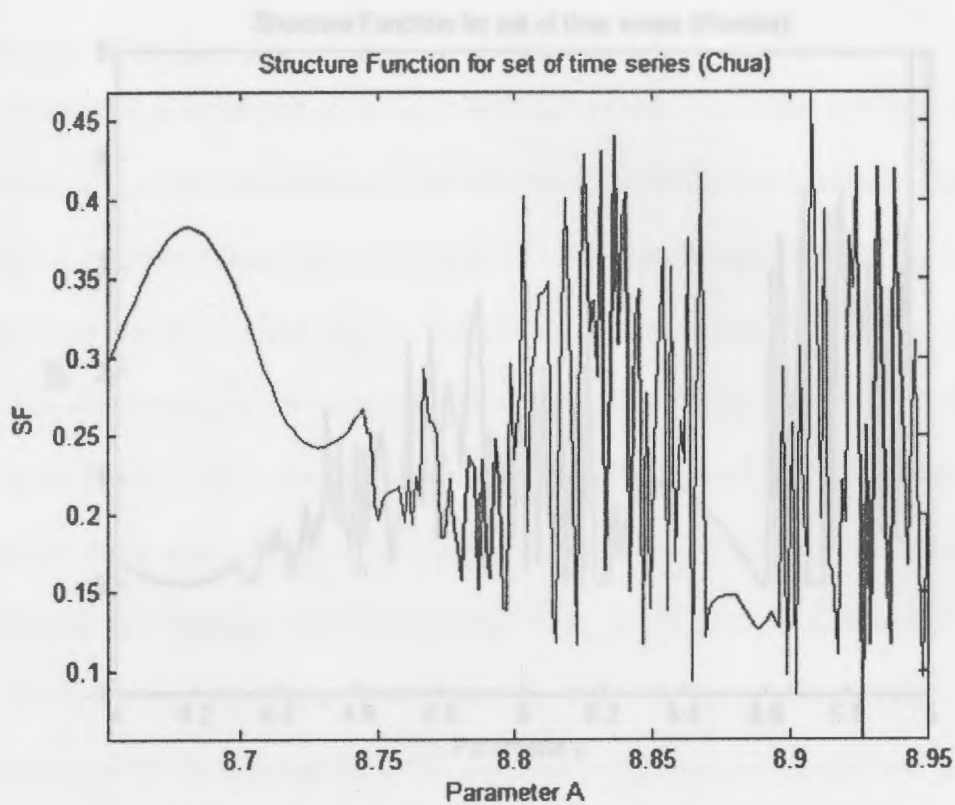


Figure 2-8 Structure Function for Chua's System

Figure 2-9 shows the structure function for Rossler's system obtained using window length of 50 and step size of 24 (see Appendix 4 for the MATLAB script). Once again, bifurcation diagram (Figure 2-6) and the structure function (Figure 2-9) for the system can be put side by side to observe that the structure function is sensitive to changes in the system dynamics. Note the smooth curve the SF produces until c reaches 4.3 and when c is between 5.2 and 5.6. Both of these smooth curves are followed by disproportionate lines caused by the chaotic response of the system. This simulation concludes the study of structure function using simulated data by detecting chaos in time series.

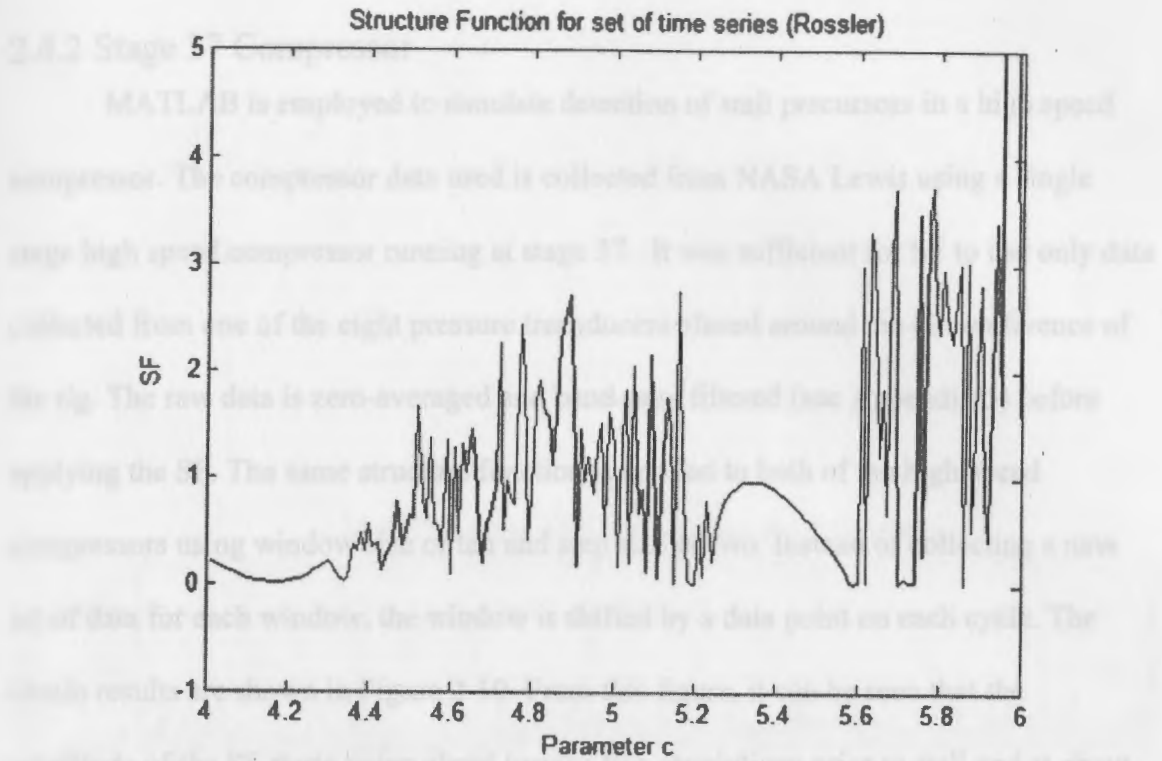


Figure 2-9 Structure Function for Rossler's System

2.4.3 T55 Multi-stage Engine

In this case, data collected from a multi-stage T55 rig with a single sensor is zero-averaged and low-passed-filtered at a low frequency, 50 Hz, before applying the structure function (see Appendix 6). The sliding structure function with window size of ten and step size of two is applied to filtered data. The results are displayed in Figure 2-11. The magnitude of the structure function becomes high enough to provide warning, twenty revolutions before the stall. This case concludes the simulation of structure function on simulated and high speed compressor data. In Chapter 4, the same data sets are collected by a digital signal processor to investigate the online implementation of the SF.

2.4.2 Stage 37 Compressor

MATLAB is employed to simulate detection of stall precursors in a high speed compressor. The compressor data used is collected from NASA Lewis using a single stage high speed compressor running at stage 37. It was sufficient for SF to use only data collected from one of the eight pressure transducers placed around the circumference of the rig. The raw data is zero-averaged and band-pass filtered (see Appendix 5) before applying the SF. The same structure function is applied to both of the high speed compressors using window size of ten and step size of two. Instead of collecting a new set of data for each window, the window is shifted by a data point on each cycle. The obtain results are shown in Figure 2-10. From this figure, it can be seen that the amplitude of the SF starts rising about twenty-five revolutions prior to stall and at about eighteen revolutions before stall it has grown enough to alert the necessary control systems.

2.4.3 T55 Multi-stage Engine

In this case, data collected from a multi-stage T55 rig with a single sensor is zero-averaged and low-passed-filtered at a low frequency, 8Hz, before applying the structure function (see Appendix 6). The sliding structure function with widow size of ten and step size of two is applied to filtered data. The results are displayed in Figure 2-11. The amplitude of the structure function increases high enough to provide warning, twenty revolutions before the stall. This case concludes the simulation of structure function on simulated and high speed compressor data. In Chapter 4, the same data sets are collected by a digital signal processor to investigate the online implementation of the SF.

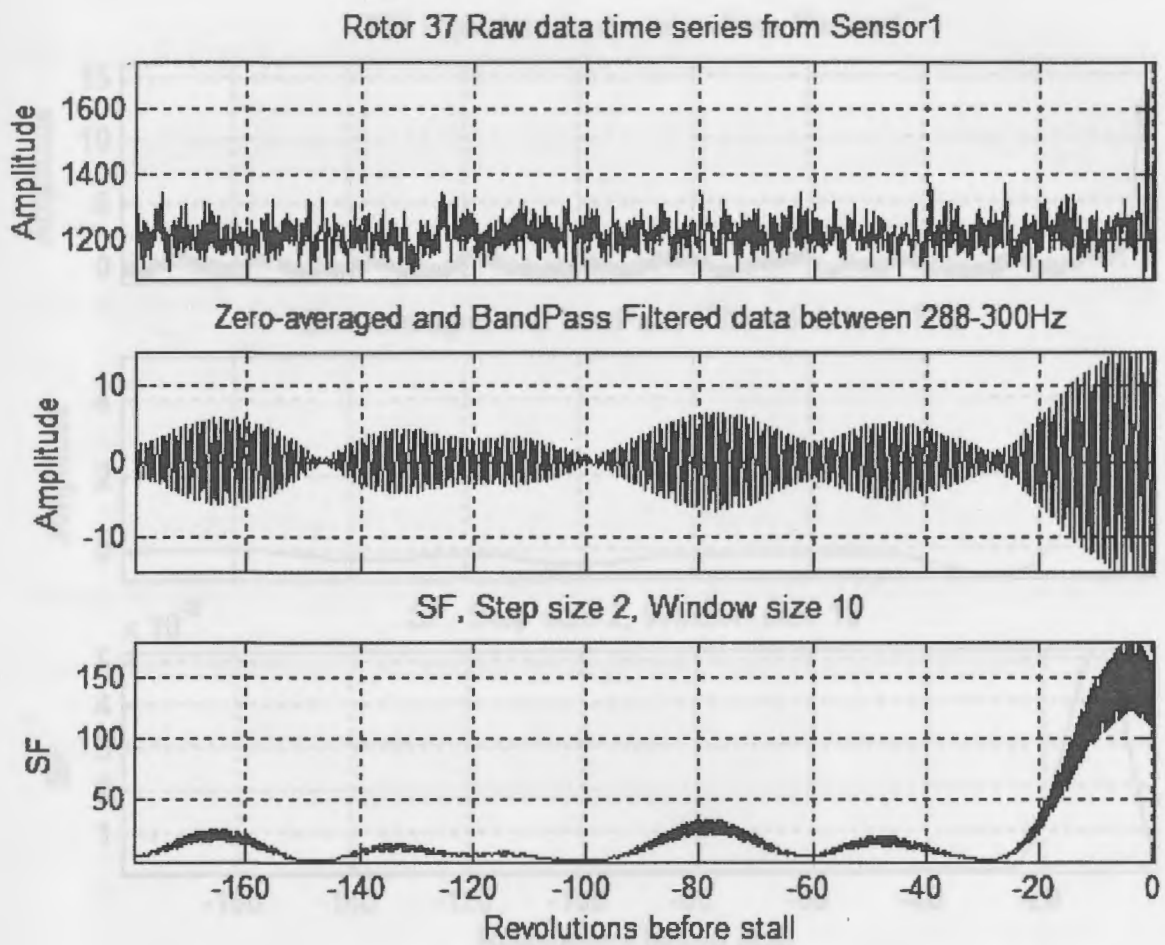


Figure 2-10 Rotor 37 Simulation

III. DIGITAL SIGNAL PROCESSING

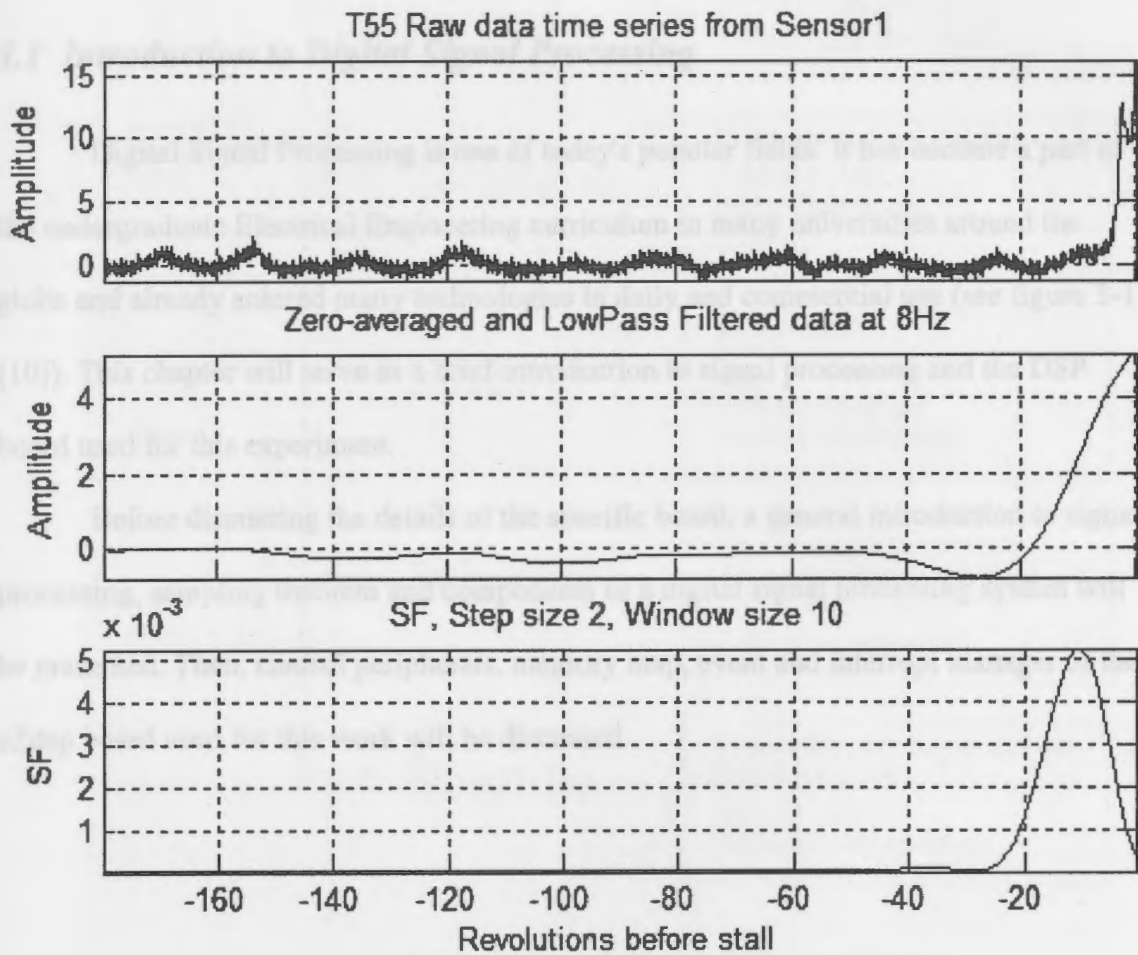


Figure 2-11 T55 Simulation

III. DIGITAL SIGNAL PROCESSING

3.1 Introduction to Digital Signal Processing

Digital Signal Processing is one of today's popular fields. It has become a part of the undergraduate Electrical Engineering curriculum in many universities around the globe and already entered many technologies in daily and commercial use (see figure 3-1 [10]). This chapter will serve as a brief introduction to signal processing and the DSP board used for this experiment.

Before discussing the details of the specific board, a general introduction to signal processing, sampling theorem and components of a digital signal processing system will be presented. Then, control peripherals, memory map, event and interrupt manager of the eZdsp board used for this work will be discussed.

Figure 3-1 Common Applications of Signal Processing

3.2 Basics of Digital Signal Processing

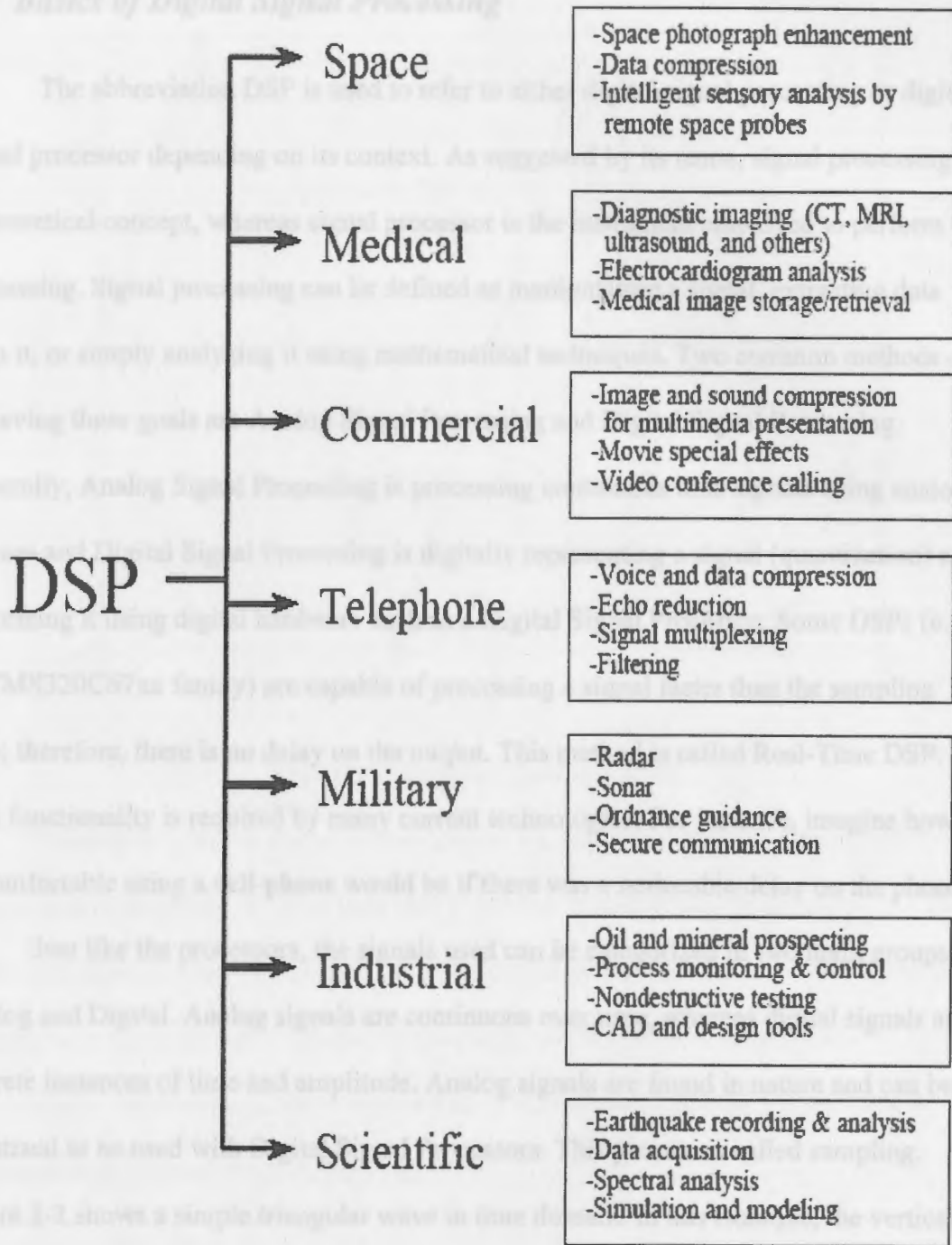


Figure 3-1 Common Applications of Signal Processing

3.2 Basics of Digital Signal Processing

The abbreviation DSP is used to refer to either digital signal processing or digital signal processor depending on its context. As suggested by its name, signal processing is a theoretical concept, whereas signal processor is the instrument employed to perform the processing. Signal processing can be defined as manipulating a signal, extracting data from it, or simply analyzing it using mathematical techniques. Two common methods of achieving these goals are Analog Signal Processing and Digital Signal Processing.

Generally, Analog Signal Processing is processing continuous time signals using analog devices and Digital Signal Processing is digitally representing a signal (quantization) and processing it using digital hardware such as a Digital Signal Processor. Some DSPs (e.g., TI TMS320C67xx family) are capable of processing a signal faster than the sampling time; therefore, there is no delay on the output. This method is called Real-Time DSP. This functionality is required by many current technologies: For instance, imagine how uncomfortable using a cell-phone would be if there was a noticeable delay on the phone.

Just like the processors, the signals used can be categorized in two main groups, Analog and Digital. Analog signals are continuous over time, whereas digital signals are discrete instances of time and amplitude. Analog signals are found in nature and can be quantized to be used with Digital Signal Processors. This process is called sampling.

Figure 3-2 shows a simple triangular wave in time domain. In this example, the vertical axis, y , represents voltage, and the horizontal axis, x , represents time. Other domains such as frequency and spatial (measure of space) are also commonly used in DSP applications.

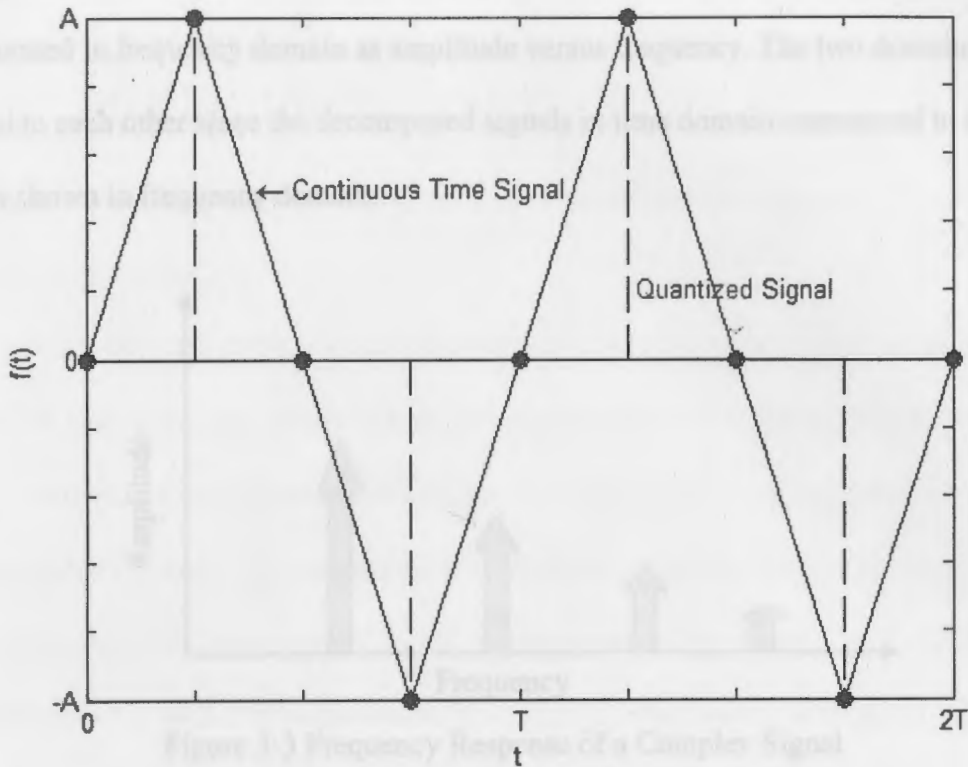


Figure 3-2 Sampled Analog Signal

In order to give the reader a clear idea of the sampling theorem, fundamentals of signals and relationship between time and frequency domains will be analyzed. As stated by Fourier theorem, any waveform can be generated by summing sine waves; in the same manner, any complex waveform can be decomposed into the sum of multiple sine waves. For example, the triangular wave represented in time domain, Figure 3-2, can also be represented in Fourier series by Eq (3.1).

$$f(t) = \frac{8A}{\pi^2} \sum_{n=1,3,5}^{\infty} \left[\frac{1}{n^2} \sin\left(\frac{n\pi}{2}\right) \right] \sin n\omega_0 t \quad (3.1)$$

In frequency domain, the signals are displayed as amplitude versus frequency, as shown in Figure 3-3. The signal shown in this figure is made up of the sum of four frequencies. Therefore, we can see that a complex signal in time domain can be

decomposed into simple sine waves analyzed as amplitude versus time and also be represented in frequency domain as amplitude versus frequency. The two domains can be related to each other since the decomposed signals in time domain correspond to the spikes shown in frequency domain.

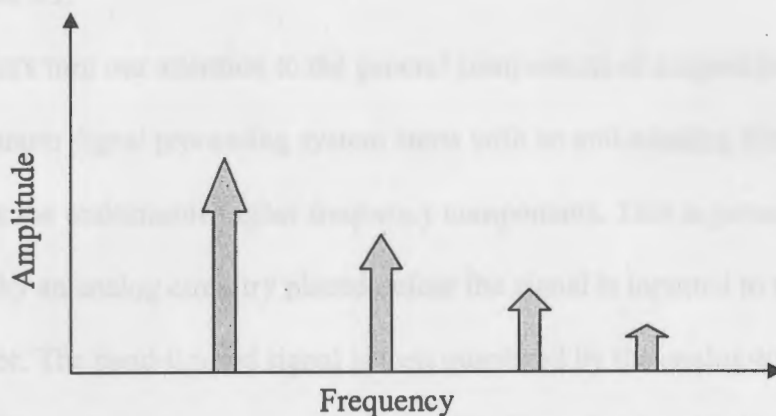


Figure 3-3 Frequency Response of a Complex Signal

Now that the relationship between the two domains has been established, let us examine the sampling theorem. Nyquist–Shannon sampling theorem states that when sampling a band-limited signal (e.g., converting from an analog signal to digital), the sampling frequency must be greater than twice the input signal bandwidth in order to be able to reconstruct the original perfectly from the sampled version. If we refer back to Figure 3-2, we can see that each period lasts 16 seconds and is sampled 16 times, thus, yielding a sampling frequency of 1 Hz. The frequency of the signal is 0.0625 Hz, $\frac{1}{16}$ of the sampling frequency; therefore, the original signal can be reconstructed from the collected data. However, if the sampling frequency were identical to the frequency of the triangular wave, the digital representation obtained would be no more than a DC value. Similarly, if the sampling frequency were 1.5 times the signal frequency, another

triangular waveform with a lower frequency would be obtained from the two periods of the original signal. This phenomenon is known as aliasing. To avoid this situation, the engineer must ensure that signal is appropriately *band-limited*: in other words, the difference between the frequencies of any two of its sinusoidal components must be strictly less than $s/2$.

Next, let's turn our attention to the general components of a signal processing system. A common signal processing system starts with an anti-aliasing filter, which is used to remove the undesirable higher frequency components. This is generally accomplished by an analog circuitry placed before the signal is inputted to the digital signal processor. The band-limited signal is then quantized by the analog-to-digital converter (ADC), so it can be processed by the DSP. Once the signal is in digital form, the processor can implement the desired functions. Most of the systems require the created response to be converted back to an analog signal, which is done by a digital-to-analog controller (DAC), whereas in others the digital form is used throughout the system, and no more conversions are necessary. A reconstruction filter can be used after the conversion back to analog to smooth the high frequency components. Figure 3-4 taken from the Texas Instruments web page [11] shows the block diagram of such a system.

Since the anti-aliasing filter removes the higher frequency components of a signal, the square wave already begins to resemble a sine wave. The signal is then sampled using an analog-to-digital converter, processed using the DSP and converted to continuous time with the DAC. The reconstruction filter then smooths the edges of the resulting signal, and the transformation is complete. This example outlines the general use of signal processing in real-world applications. In signal processing, many design decisions make

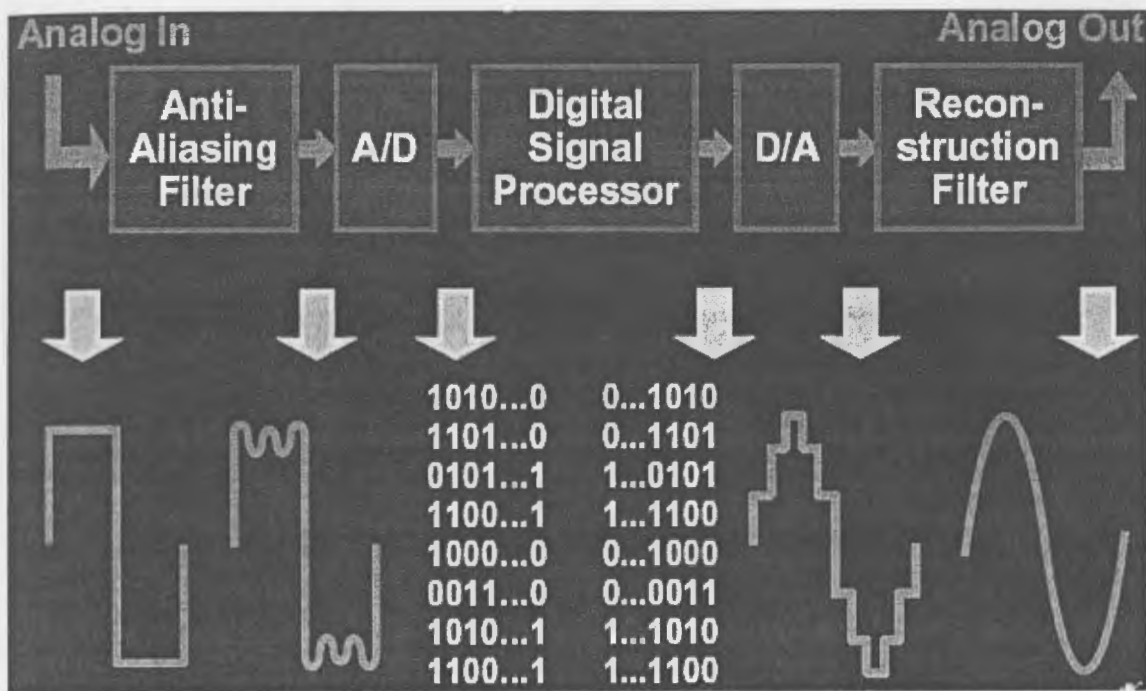


Figure 3-4 Block Diagram of a Common Signal Processing System

The general scenario described above can be illustrated with an example based on Figure 3-4. The objective of the system shown is to transform inputted square wave into a sine wave. As previously indicated, any signal can be represented by many simple sine waves. The Fourier series expansion of a square wave is given by Eq. (3.2).

$$f(t) = \frac{4}{\pi} \sum_{n=1,3,5}^{\infty} \frac{1}{n} \sin n\omega_0 t \quad (3.2)$$

Since the anti-aliasing filter removes the higher frequency components of a signal, the square wave already begins to resemble a sine wave. The signal is then sampled using an analog-to-digital converter, processed using the DSP and converted to continuous time with the DAC. The reconstruction filter then smoothes the edges of the resulting signal, and the transformation is complete. This example outlines the general use of signal processing in real-world applications. In signal processing, many design decisions made

are system dependent, thus required hardware and software must be chosen to meet the system demands.

When designing a filter or a spectrum analysis tool for a system, a decision must be made if analog or digital components will be used. Digital systems have become more and more popular in their usage due to their advantages. They are considered to be more stable since they do not age over time, and the response of a digital system will not be affected by changes in temperature. They also offer portability: system responses built of same digital components will be identical; on the other hand, responses obtained from analog systems might differ since analog components have tolerance. If using an analog system, system modification requires changes of components, which might not be practical in many cases, whereas a recompile might be all it takes to update a digital system. Given these facts, a digital system has been chosen to be used in this research.

The advantages of digital signal processors as a digital system are numerous. In order to achieve real-time performance, a fast instruction cycle is necessary. Since the digital signal processors have on-board dedicated multipliers that perform single cycle multiplication, they can process more data in a given time than a system that uses consecutive additions. Also, availability of multiple bus lines enables operations to be carried out in parallel. The fact that Texas Instruments boards have support for single-cycle signal processing specific functions presents even more reasons for using them in signal processing applications. TMS320F2812 manufactured by Texas Instruments meets all aforementioned requirements and, therefore, is the digital signal processor employed in this research.

3.3 Characteristics of Spectrum Digital's eZdsp

The structure function is processed on an eZdsp F2812 board by Spectrum Digital. The board is a stand-alone module based on Texas Instruments TMS320F2812 digital signal processor. It is created for developers to evaluate the TMS320F2812 DSP to determine the applicability of the processor to their design. The module has the following listed specifications:

- 150 MIPS operating speed
- 18K words on-chip RAM
- 128K words on-chip Flash memory
- 64K words off-chip SRAM memory
- 30 MHz. clock
- 2 Expansion Connectors
- Onboard IEEE 1149.1 JTAG Controller
- 5-volt only operation with supplied AC adapter
- TI F28xx Code Composer Studio tools driver

Two expansion slots provided, analog and input/output expansion, can be used to interface the board with any necessary circuitry. The JTAG emulation connector is for emulation and debugging of assembly or high-level language code. The code composer studio is the software development environment to be used with the DSP. eZdsp requires 5V on P6 in order to operate. Table 3-1 [12] displays the memory space of eZdsp F2812. This figure will be necessary to create the command file that maps the compiled program into the board's memory.

Block Start Address	On-Chip Memory		External Memory XINTF			
	Data Space	Prog Space	Data Space	Prog Space		
0x00 0000	M0 Vector - RAM (32 x 32) (Enabled if VMAP = 0)		Reserved			
0x00 0040	M0 SARAM (1K x 16)					
0x00 0400	M1 SARAM (1K x 16)					
0x00 0800	Peripheral Frame 0 (2K x 16)	Reserved				
0x00 0D00	PIE Vector - RAM (256 x 16) (Enabled if VMAP = 1, ENPIE = 1)					
0x00 1000	Reserved					
0x00 2000	Reserved					
0x00 6000	Peripheral Frame 1 (4K x 16, Protected)	Reserved			XINTF Zone 0 (8K x 16, XZCS0AND1)	0x00 2000
0x00 7000	Peripheral Frame 2 (4K x 16, Protected)				XINTF Zone 1 (8K x 16, XZCS0AND1) (Protected)	0x00 4000
0x00 8000	L0 SARAM (4K x 16, Secure Block)	Reserved				
0x00 9000	L1 SARAM (4K x 16, Secure Block)					
0x00 A000	Reserved					
				XINTF Zone 2 (0.5M x 16, XZCS2)	0x08 0000	
				XINTF Zone 6 (0.5M x 16, XZCS6AND7)	0x10 0000	
					0x18 0000	
0x3D 7800	OTP (2K x 16, Secure Block)					
0x3D 8000	FLASH (128K x 16, Secure Block)					
0x3F 0000	128-Bit Password					
0x3F 7FF8	H0 SARAM (8K x 16)					
0x3F 8000	Reserved					
0x3F A000	Reserved					
0x3F F000	Boot ROM (4K x 16) (Enabled if MP/MC = 0)		XINTF Zone 7 (16K x 16, XZCS5AND7) (Enabled if MP/MC = 1)	0x3F C000		
0x3F FFC0	BROM Vector - ROM (32 x 32) (Enabled if VMAP = 1, MP/MC = 0, ENPIE = 0)		XINTF Vector - RAM (32 x 32) (Enabled if VMAP = 1, MP/MC = 1, ENPIE = 0)			

LEGEND: Only one of these vector maps—M0 vector, PIE vector, BROM vector, XINTF vector—should be enabled at a time.

Table 3-1 eZdsp Memory Space

The connectors used on the board are displayed in Figure 3-5 [12], where P1 is the JTAG interface and P2 is the expansion slot. P2 can be used to connect external module for evaluating functionalities that are not present on the module. P3 is the parallel port interface used to communicate with the host PC. P4/P7/P8 connectors are the input/output interfaces. P4 and P8 used to capture inputs and output pulse-width modulated waveforms or SPI connections, while P7 is used for compare trips. P5/P9 are

channel A and B ADC inputs, and P6 is the power connector as mentioned above.

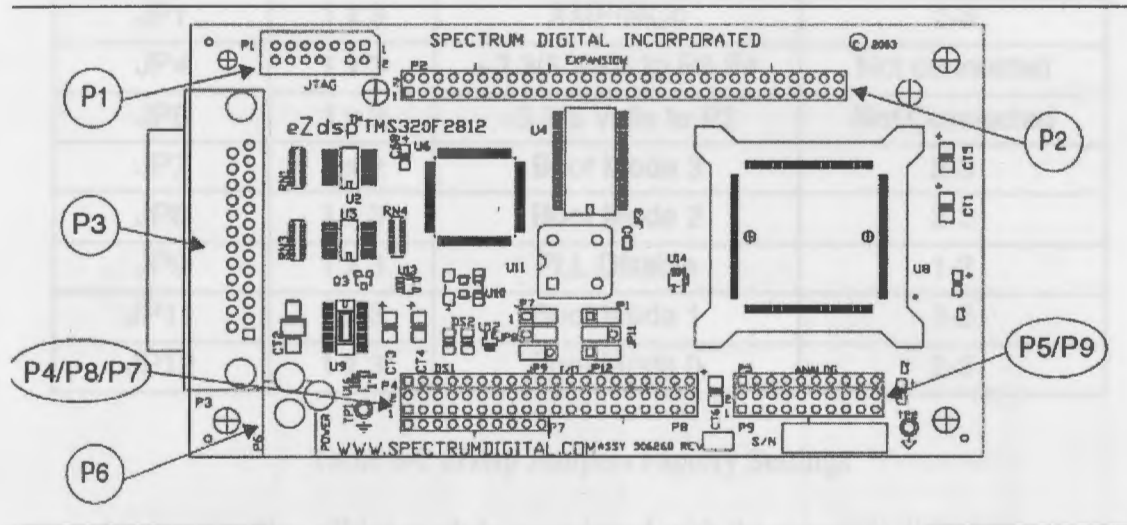


Figure 3-5 eZdsp Connector Positions

Various features of the DSP are determined by settings of the eight available jumpers. The factory defaults of these jumpers are shown in Table 3-2 [12]. Jumper 1 is used to select the operation mode for the processor as a microprocessor or a microcontroller. Jumpers 4 and 5 are unpopulated and used to provide 3.3V or 5V to P2, expansion slot, of the module. Similarly, jumper 4 and 5 provide 3.3V or 5V to pins 1 and 2 of P8 and P2, respectively. Jumpers 7, 8, 11 and 12 determine the boot mode for the processor. The default setting is H0; however, Flash, SPI, SCI, OTP or Parallel port could be selected to boot into. Finally, jumper 9 enables or disables the phase-locked loop.

Jumper #	Size	Function	Position As Shipped From Factory
JP1	1 x 3	XMP/MCn	2-3
JP4	1 x 3	+3.3/5 Volts to P8,P4	Not connected
JP5	1 x 3	+3.3/5 Volts to P2	Not Connected
JP7	1 x 2	Boot Mode 3	2-3
JP8	1 x 3	Boot Mode 2	2-3
JP9	1 x 3	PLL Disable	1-2
JP11	1 x 3	Boot Mode 1	1-2
JP12	1 x 3	Boot Mode 0	2-3

Table 3-2 eZdsp Jumpers Factory Settings

As stated earlier, eZdsp module is equipped with the powerful digital signal controller: TMS320C2812. Figure 3-6 [13] shows the block diagram of this controller. This figure partitions the DSP into four main functionalities: internal/external bus system, central processing unit (CPU), memory and peripherals. F2812 uses a “modified Harvard-Architecture” since it reads operands from data memory and program memory within a clock cycle using the Program and Data Bus. F2812 DSP CPU has the following capabilities:

- 32-bit fixed-point DSP
- 32 x 32 bit fixed-point MAC
- Dual 16 x 16 single-cycle fixed-point MAC (DMAC)
- 32-/64-bit saturation
- 64/32 and 32/32 modulus division
- Fast interrupt service time
- Single cycle read-modify-write instructions
- Unique real-time debugging capabilities

- Upward code compatibility
- MCU/DSP balancing code density & execution time
- Ability to support 32-bit instructions for improved execution time
- Ability to support 16-bit instructions for improved code efficiency

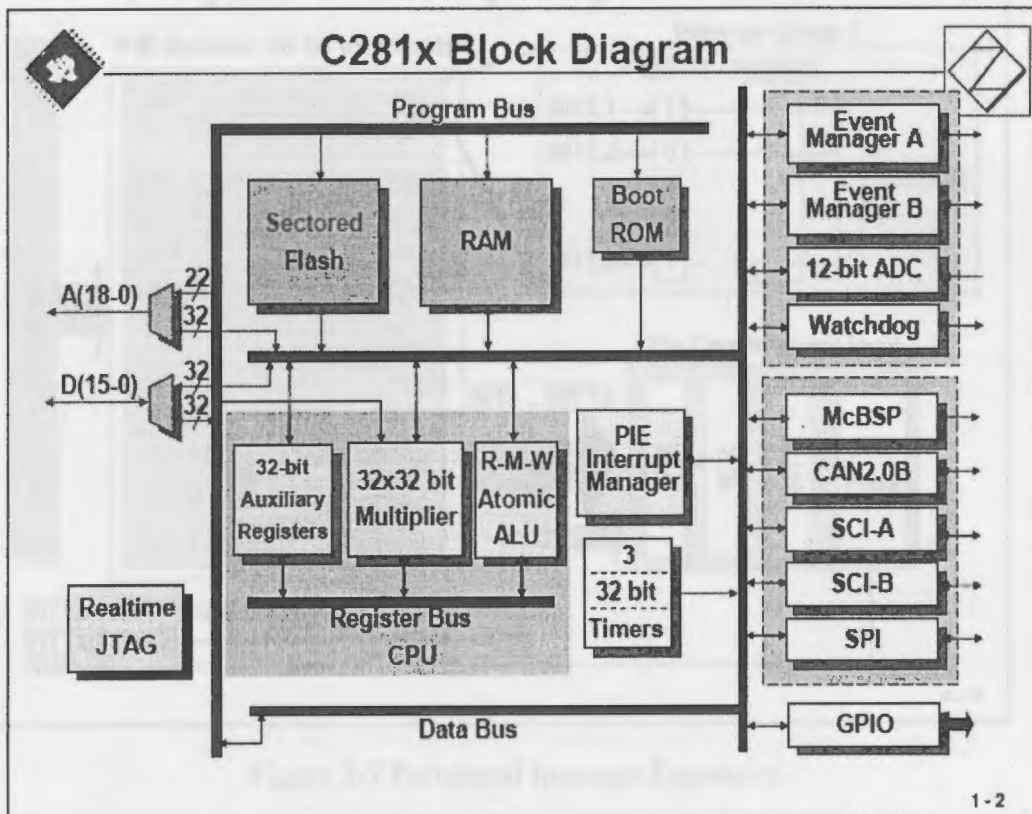


Figure 3-6 F2812 Block Diagram

3.3.1 F2812 Interrupt System

Interrupts are requests generated by an internal or external hardware to halt the current execution of the program to serve the hardware. This triggers the interrupt service routine to execute, and once the interrupt is served, CPU resumes processing of the program. C28x provides fourteen maskable interrupts; however, ninety-six interrupt sources exist on the DSP. This space issue is resolved by utilizing Peripheral Interrupt

Expansion, PIE. PIE partitions these ninety-six interrupts in twelve groups, each eight lines long as shown in Figure 3-7 [13]. The figure also shows the interrupt flag register (IFR), interrupt enable register (IER) and interrupt global mask bit (INTM) used to enable and acknowledge these interrupts.

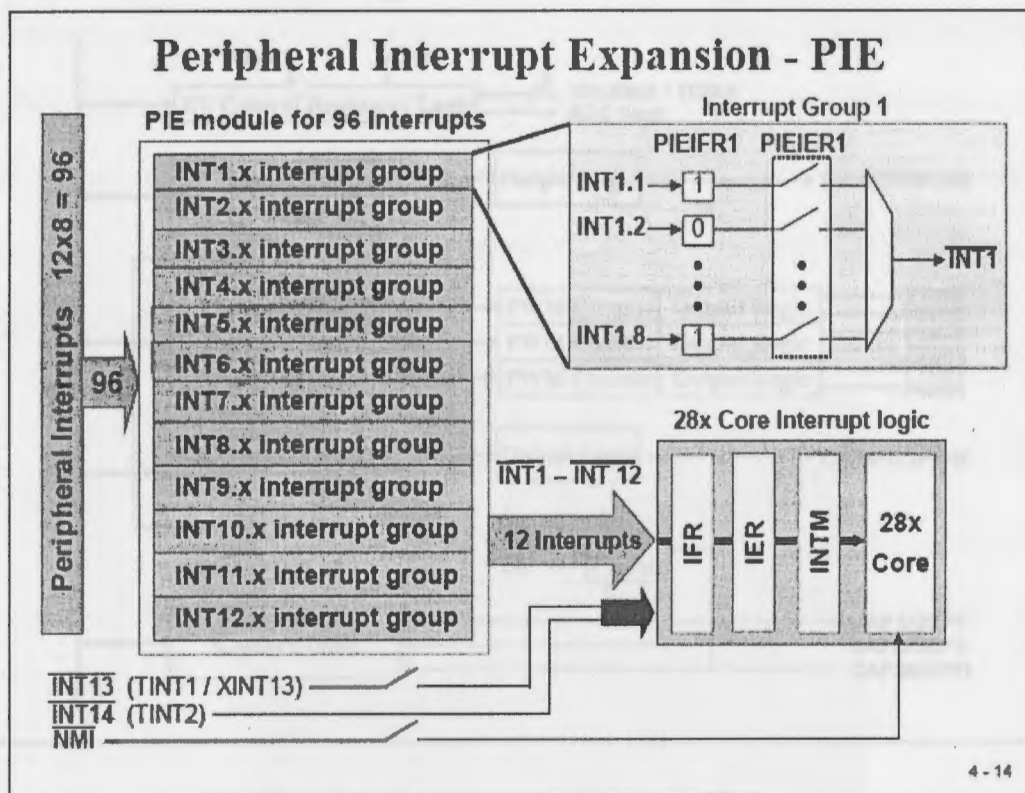


Figure 3-7 Peripheral Interrupt Expansion

3.3.2 F2812 Event Manager

The two event managers provided by F2812 (EVA and EVB) are control peripherals. Block diagram for EVA is shown in Figure 3-8. EVB is physically identical to EVA, but it uses a different naming convention (i.e., General Purpose Timer 3 and General Purpose Timer 4, instead of General Purpose Timer 1 and General Purpose Timer 2, etc.). The event managers provide sixteen pulse-width modulators generally used in motor control applications, six capture units used for logging of different events,

two quadrature encoders used for determining the position of a rotating shaft, and ten 16-bit compare units helpful for generating PWM waveforms using the four general-purpose timers.

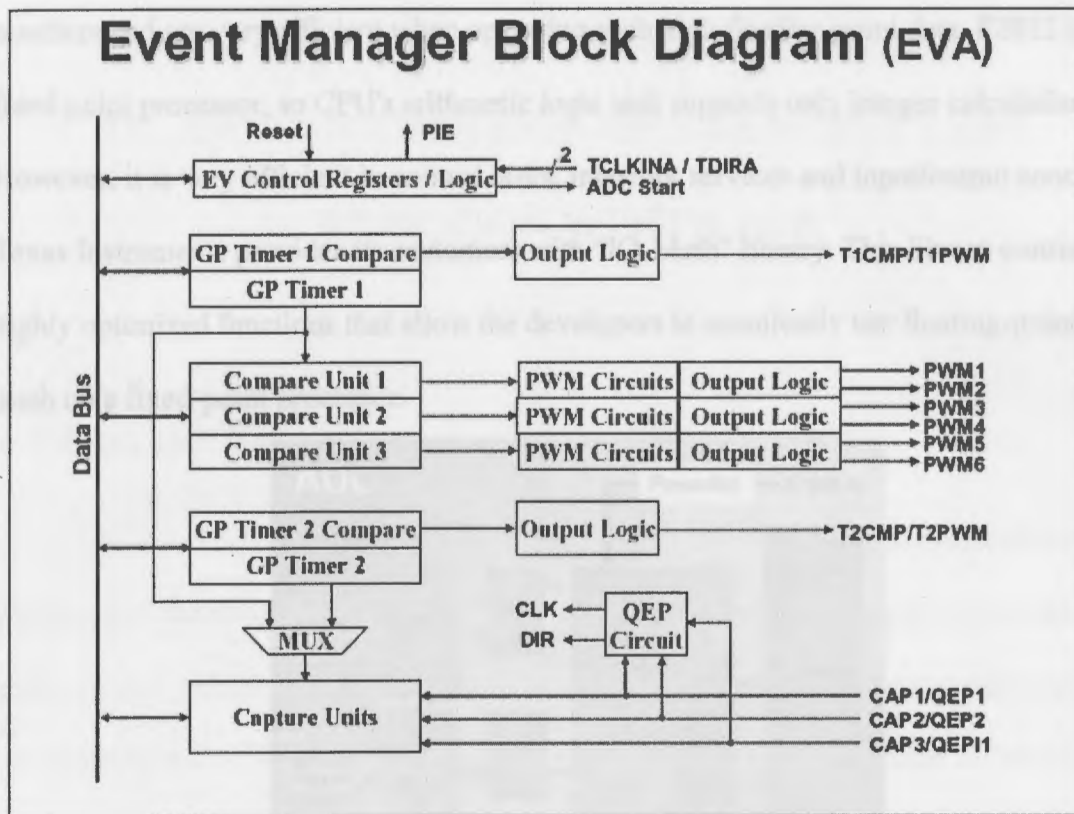


Figure 3-8 F2812 Event Manager Block Diagram

3.3.3 Analog-to-Digital Converter

The eZdsp module has on board a 12-bit 16 channel analog-to-digital convector. It provides two sample-and-hold circuits and supports simultaneous and sequential sampling modes. It has an analog input range of 0V to 3V and runs at 25 Mhz. Each result is stored individually, and autosequencer allows up to sixteen "autoconversations" in a single session. Figure 3-9 [14] shows a simplified block diagram of the unit.

3.3.4 Fixed Point and Floating Point

The processors can be categorized in two groups based on the type of operations the hardware can support. Floating type devices support the use of IEEE 754 real numbers and are very efficient when operating with high floating point data. F2812 is a fixed point processor, so CPU's arithmetic logic unit supports only integer calculations. However, it is very efficient in control tasks, interrupt services and input/output control. Texas Instruments provides its customers with "IQ-Math" library. This library consists of highly optimized functions that allow the developers to seamlessly use floating-point math on a fixed-point processor.

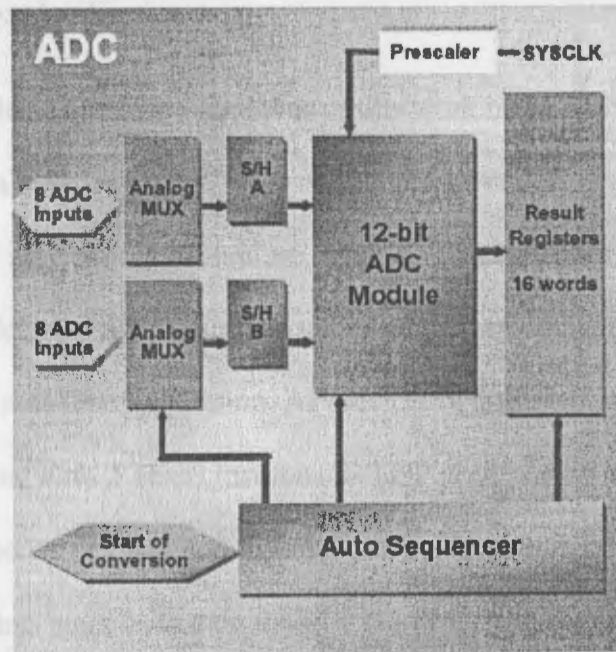
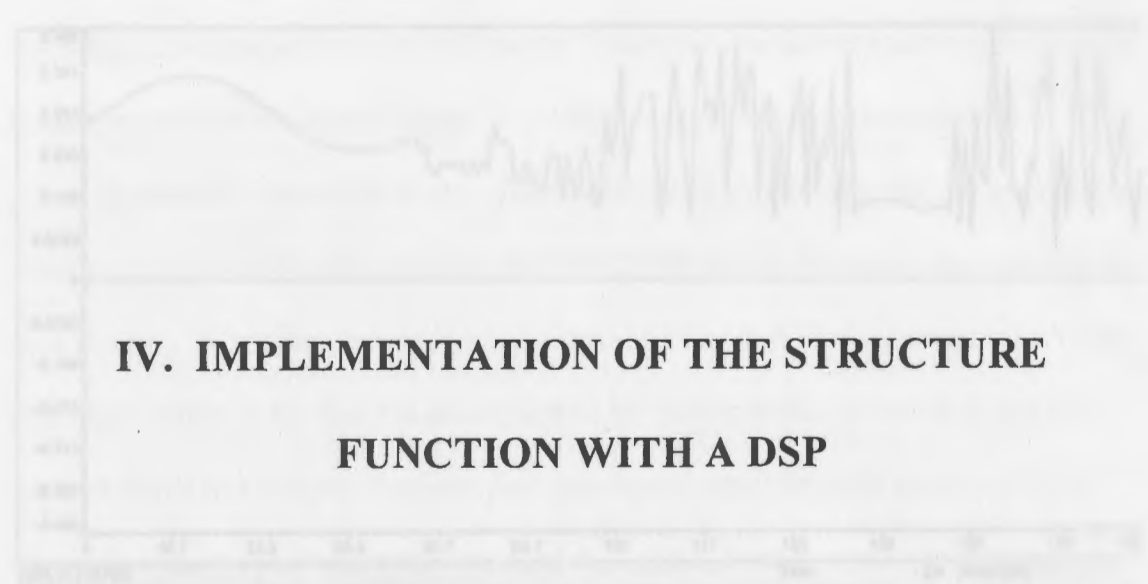


Figure 3-9 Simplified Block Diagram of the ADC

Thus, based on these listed characteristics of the signal processor, eZdsp module is qualified to implement the structure function.



IV. IMPLEMENTATION OF THE STRUCTURE FUNCTION WITH A DSP

4.1 *Chua and Rossler Systems*

As stated in the introduction the focus of this work is the DSP implementation of the structure function in order to explore a possibly effective stall detection technique for online or near online applications. Therefore, once the structure function is simulated successfully, and it detected the dynamic system changes, an online model is compiled and tested for a near real-time application. As previously discussed, an eZdsp board from Spectrum Digital using F2812 Texas Instruments DSP is employed to run the function.

The time response of Chua and Rossler systems described in Chapter II are stored as a data file in floating-point format on the PC. The processor continuously accesses the file as if collecting data from a sensor. The inputted data is then collected in a buffer and filtered using the structure function. The result is displayed in a live graph and also stored back on the computer to be analyzed later. A sample screenshot of this process for Chua system is presented in Figure 4-1. Notice that around the 44th sample, the system dynamics start changing. After this point, the system enters a chaotic stage until the 99th sample, where a sudden drop in amplitude occurs. During this period, the system is

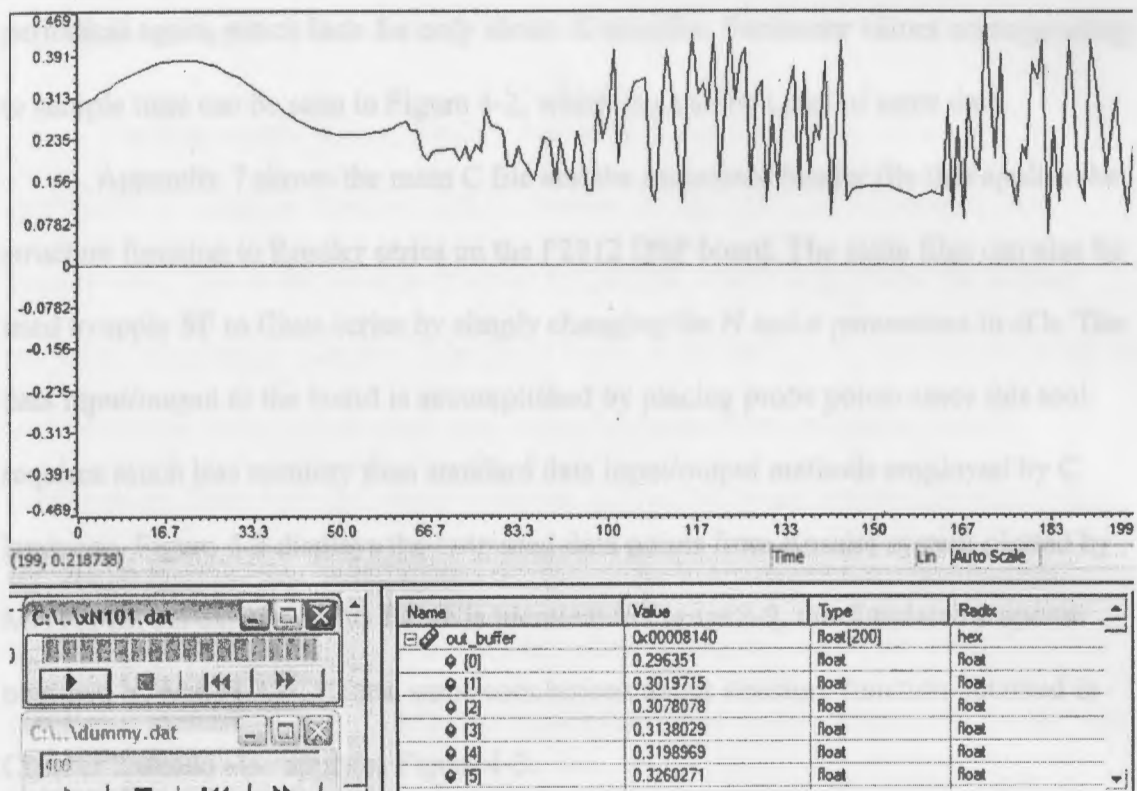


Figure 4-1 Structure Function Running on DSP

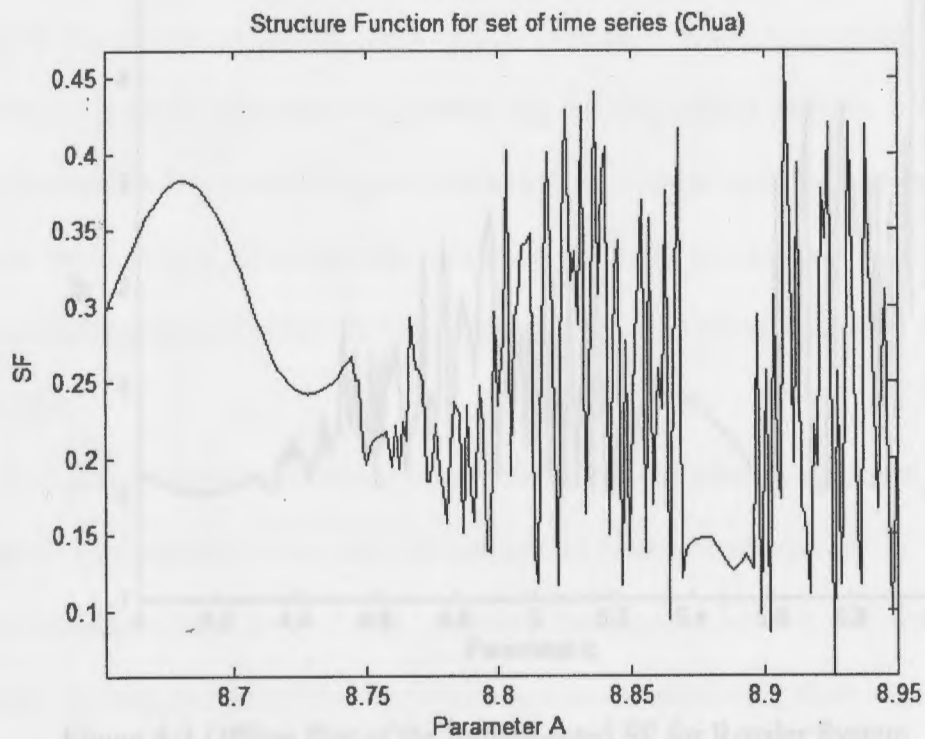


Figure 4-1 Offline Plot of the Implemented SF for Chua System

Figure 4-2 Offline Plot of the Implemented SF for Chua System

periodical again, which lasts for only about 10 samples. Parameter values corresponding to sample time can be seen in Figure 4-2, which is an offline plot of same data.

Appendix 7 shows the main C file and the associated header file that applies the structure function to Rossler series on the F2812 DSP board. The same files can also be used to apply SF to Chua series by simply changing the N and n parameters in sf.h. The data input/output to the board is accomplished by placing probe points since this tool requires much less memory than standard data input/output methods employed by C language. Figure 4-3 displays the outputted data points from Rossler system plotted by MATLAB. As expected, this figure is identical to Figure 2-9, the simulated response obtained by MATLAB. Hence, same conclusions about structure function obtained in Chapter 2 would also apply to Figure 4-3.

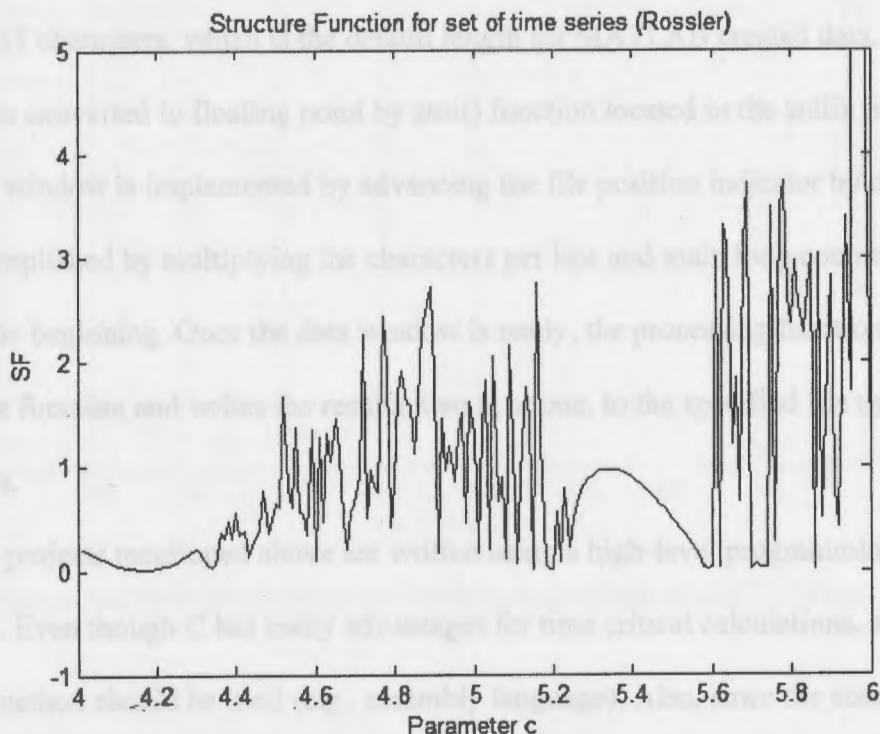


Figure 4-3 Offline Plot of the Implemented SF for Rossler System

4.2 Rotor37 and T55

The C program `Main.c`, which is included in Appendix 6, is associated with the application of structure function to data obtained for Rotor37 and T55 high-speed compressors. This is the main program of the project, which implements the sliding controller discussed in Chapter II on Texas Instruments DSP. Data input/output could not be achieved by probe points since they cannot accommodate such complicated functionality. Therefore, data input is accomplished by `fgets()` function available in `stdio.h`. Among the functions that are recognized by TI's C compiler, `fgets` is a data input/output function that requires the least resources. `Fgets()` reads the string from the stream for a specified length and stores it into the buffer. In this project, the stream was the specified data file containing the zero-averaged and filtered data points, while its length was 17 characters, which is the default length for MATLAB created data. This string is then converted to floating point by `atoi()` function located in the `stdlib.h` library. The sliding window is implemented by advancing the file position indicator by offset bytes (accomplished by multiplying the characters per line and main loop counter) with respect to the beginning. Once the data window is ready, the processing function applies the structure function and writes the results, two at a time, to the specified file using probe points.

The projects mentioned above are written using a high-level programming language C. Even though C has many advantages for time critical calculations, a more optimized method should be used (e.g., assembly language). Also, since the core of the DSP is fixed-point, declaring floating-point arrays and manipulating them requires numerous resources. A solution to this issue can be obtained by incorporating Texas

Instruments IQ math library to the project and using the IQ functions to do the calculations. This will increase the execution speed and might also free some memory.

The bottleneck of this system, however, was not the use of floating-point digits. Accessing data files over 15 MB and browsing through them took longer time and more memory than any other part of the project. Fortunately, this issue should not occur in a real-life system, since the data will most likely be collected through the analog to digital converter, not from a computer.

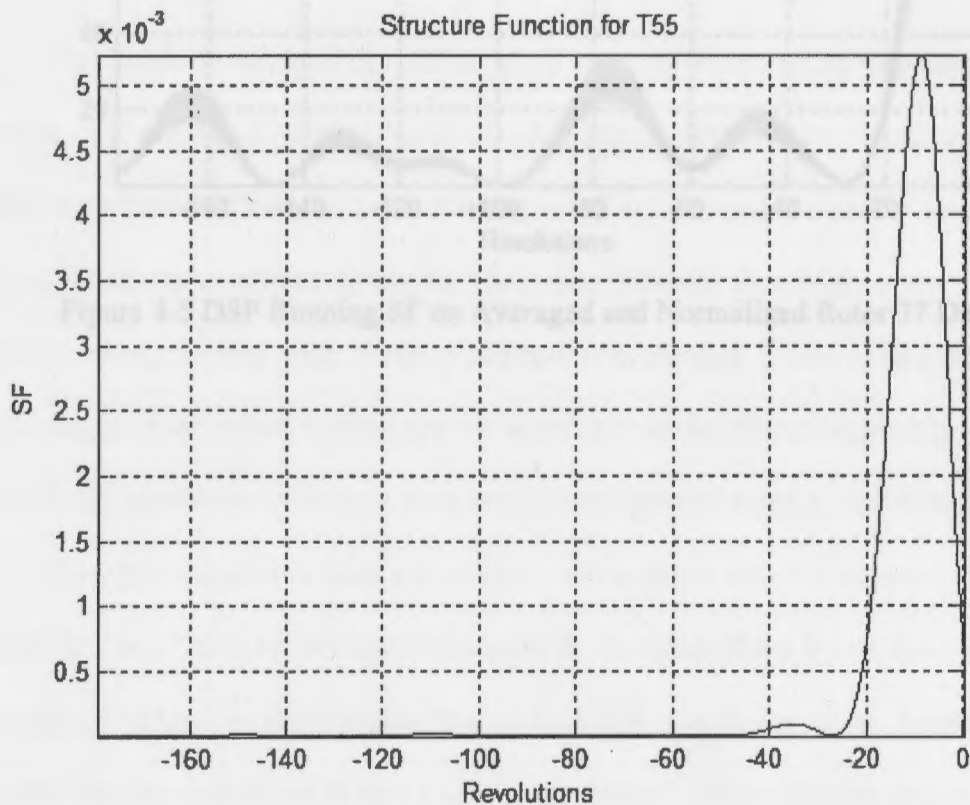


Figure 4-4 DSP Running SF on Averaged and Normalized T55 Data

Structure Function for Rotor 37

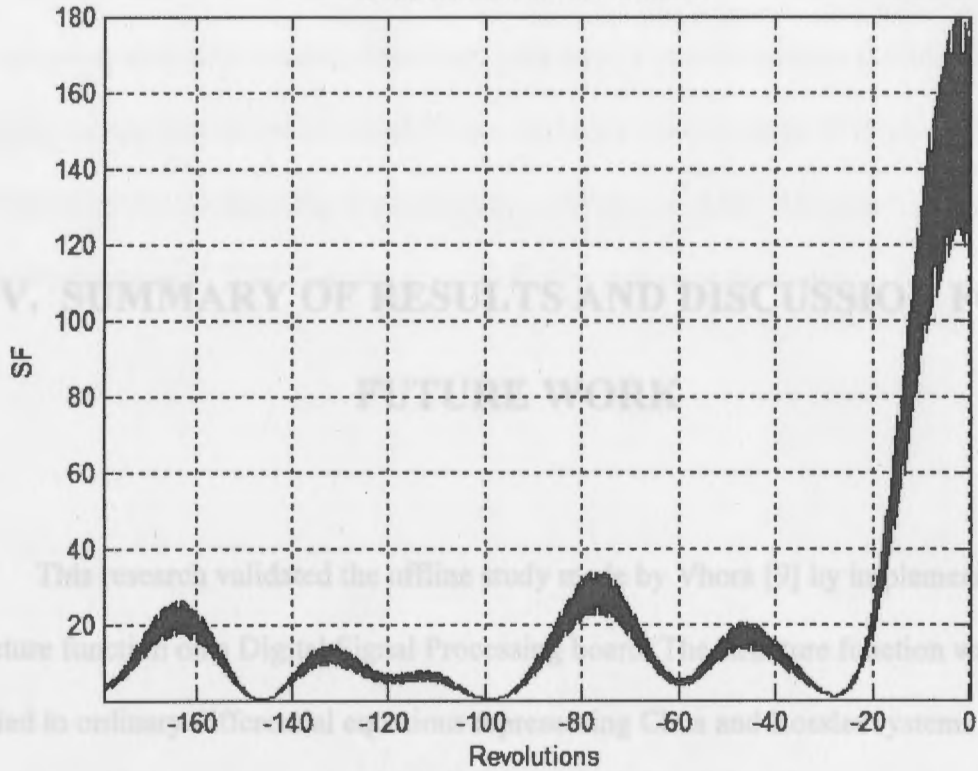


Figure 4-5 DSP Running SF on Averaged and Normalized Rotor 37 Data

This research validated the offline study by applying the structure function to the Digital Processor data. The structure function was applied to online data collected from a compressor and showed sensitivity to system dynamics. Then, data collected from high-speed compressors was filtered with structure function to detect stall. These results could be used to trigger a secondary control system to prevent the stall from happening. The simplicity of the function allows it to be easily implemented using a digital controller.

The objective of this work was to apply the structure function to detect dynamical changes in a non-linear system and investigate its implementation issues. Real-time processing could not be accomplished due to hardware interfacing issues; however, the structure function was found to be a successful method to detect changes in system dynamics in near-real-time applications. Since high precision mathematics is required to implement the structure function, a digital signal processor was used as the system core. For larger window sizes than the ones presented here, external memory would most likely be required due to large volumes of data that need to be stored and processed.

V. SUMMARY OF RESULTS AND DISCUSSION FOR FUTURE WORK

This research validated the offline study made by Vhora [9] by implementing the structure function on a Digital Signal Processing board. The structure function was applied to ordinary differential equations representing Chua and Rossler systems and showed sensitivity to system dynamics. Then, data collected from high-speed compressors was filtered with structure function to detect stall. These results could be used to trigger a secondary control system to prevent the stall from happening. The simplicity of the function allows it to be easily implemented using a digital controller.

The objective of this work was to apply the structure function to detect dynamical changes in a non-linear system and investigate its implementation issues. Real-time processing could not be accomplished due to hardware interfacing issues; however, the structure function was found to be a successful method to detect changes in system dynamics in near-real-time applications. Since high precision mathematics is required to implement the structure function, a digital signal processor was used as the system core. For larger window sizes than the ones presented here, external memory would most likely be required due to large volume of data that needs to be stored and processed.

The structure function has been successfully implemented, and additional studies to improve system performance have been proposed. The code written for this work is portable; hence, it could be executed from a real-time floating-point DSP such as Texas Instrument's 6711 to study the executing time differences. Also, IQ Math functions could be utilized instead of IEEE floating point to further optimize the code.

REFERENCES

- [1] compressor, (2005). *Encyclopedia Britannica*. Retrieved December 4, 2005, from [Encyclopedia Britannica Premium Service http://www.britannica.com/eb/article-9025037](http://www.britannica.com/eb/article-9025037)
- [2] Greitzer, E.M. (1981). The stability of pumping systems. *Journal of Fluids Engineering*, 103, p. 193.
- [3] Insa, T., and Rannie, W. D. (1954). Experimental investigations of propagating stall in axial-flow compressors. *Transactions ASME*, p. 463.
- [4] Greitzer, E. M. (1976). Surge and rotating stall in axial flow compressors, part 1, 2. *Transactions of the ASME, Journal of Engineering for Power*, 98, pp. 190-217.
- [5] Greitzer, E.M. (1980). Review - axial compressor stall phenomena. *Transactions of ASME, Journal of Fluids Engineering*, 102, pp. 134-151.
- [6] Tryfonidis, M., et al. (1995). Pre-stall behavior of several high-speed compressors. *Journal of Turbomachinery*, 117, pp. 62-80.
- [7] Day, I. J. (1993). Stall inception in axial flow compressors. *Journal of Turbomachinery*, 115(1).
- [8] Bright, M.M., et al. (1997). Stall precursor identification in high speed compressor stages using chaotic time series analysis methods. *Journal of Turbomachinery*, 119(3), pp. 491-499.
- [9] Vhoos, M. H. (1998). *Detection of stall precursors in high speed axial flow*

REFERENCES

- [1] compressor. (2005). *Encyclopædia Britannica*. Retrieved December 4, 2005, from Encyclopædia Britannica Premium Service <http://www.britannica.com/eb/article-9025037>
- [2] Greitzer, E.M. (1981). The stability of pumping systems. *Journal of Fluids Engineering*, 193, p. 193.
- [3] Iura, T., and Rannie, W. D. (1954). Experimental investigations of propagating stall in axial-flow compressors *Transactions ASME*, p. 463.
- [4] Greitzer, E. M. (1976). Surge and rotating stall in axial flow compressors, part 1, 2. *Transactions of the ASME, Journal of Engineering for Power*, 98, pp. 190–217.
- [5] Greitzer, E.M. (1980). Review - axial compressor stall phenomena. *Transactions of ASME, Journal of Fluids Engineering*, 102, pp. 134–151.
- [6] Tryfonidis, M., et al. (1995). Prestall behavior of several high-speed compressors. *Journal of Turbomachinery*, 117, pp. 62-80.
- [7] Day, I. J. (1993). Stall inception in axial flow compressors. *Journal of Turbomachinery*, 155(1).
- [8] Bright, M.M., et al. (1997). Stall precursor identification in high speed compressor stages using chaotic time series analysis methods. *Journal of Turbomachinery*, 119(3), pp. 491-499.
- [9] Vhora, M. H. (1998). *Detection of stall precursors in high speed axial flow*

compressors using structure function analysis technique. Akron, OH: University of Akron.

- [10] Smith, S. (1999). *The scientist and engineer's guide to digital signal processing.* San Diego, CA: California Technical Publishing.
- [11] Schachter, K. (No date.) An intuitive approach to digital signal processing. *Texas Instrumental.* Retrieved May 2005 from <http://www.ti.com>
- [12] *eZdsp F2812 Technical Reference.* (2003). Stafford, TX: Spectrum Digital.
- [13] Bormann, F. (2005). *Texas Instruments C2000 Teaching Materials* [CD-ROM]. Dallas, TX: Texas Instruments.
- [14] *Texas Instruments TMS320C28x 1-Day Workshop: Student Guide.* (2004). Dallas, TX: Texas Instruments.
- [15] Panchev, S. (1971). *Random functions and turbulence.* Oxford: Pergamon Press.
- [16] Kolmogorov, A. N. (1941). Dissipation of energy in locally isotropic turbulence. *Doklady Acad. Sci.,* 32(1).
- [17] Yaglom, A.M. (1955). Correlation theory of processes with random stationary n-th increments. *Math. Sbornik,* 37(1).
- [18] Yaglom, A.M. (1957). Some classes of random fields in n-dimensional space, related to the stationary random processes. *Prob. Theory and Its Application,* 11(3).

APPENDICES

Chua Bifurcation Diagram

```
%-----  
% Create Chua series data  
%-----  
  
% *****  
% SET VALUES AND DEFINE VARIABLES  
  
c=1;j=1;  
T=0.02; % TIME STEP FOR INTEGRATION  
temp_n=12000; % Transient steps to discard  
max_n=5000; % NUMBER OF INTEGRATION STEPS to record  
xdata=zeros(1,5000*200);  
store=zeros(10000,2); % temporary number  
  
% Main loop  
for R=8.6515:0.0015:8.95;  
  
% *****  
x=0.1; % INITIAL CONDITIONS  
y=0.02;  
z=0.3;  
% *****  
x_dot0=0;y_dot0=0;z_dot0=0;
```

Appendix 1

Chua Bifurcation Diagram

```
%-----  
% Create Chua series data  
%-----  
  
% *****  
% SET VALUES AND DEFINE VARIABLES  
  
c=1;j=1;  
T=0.02; % TIME STEP FOR INTEGRATION  
temp_n=12000; % Transient steps to discard  
max_n=5000; % NUMBER OF INTEGRATION STEPS to record  
xdata=zeros(1,5000*200);  
store=zeros(10000,2); % temporary number  
  
% Main loop  
for R=8.6515:0.0015:8.95;  
  
% *****  
x=0.1; % INITIAL CONDITIONS  
y=0.02;  
z=-0.3;  
% *****  
xdot0=0;ydot0=0;zdot0=0;
```

```

% Transient passage loop
for n=1:temp_n
    % Chua equations, R=A
    xdot1=R*(y+ x*(1.0 - 2.0*x*x)/7.0);
    ydot1=x-y+z;
    zdot1=-100.0*y/7.0;

    % FIND THE LOCAL MAXIMUM VALUES
    x=x + T/2.0*(3.0*xdot1-xdot0);
    y=y + T/2.0*(3.0*ydot1-ydot0);
    z=z + T/2.0*(3.0*zdot1-zdot0);

    xdot0=xdot1;
    ydot0=ydot1;
    zdot0=zdot1;
end % FOR

% Data recording loop
for n=1:max_n
    % Chua equations, R=A
    xdot1=R*(y+ x*(1.0 - 2.0*x*x)/7.0);
    ydot1=x-y+z;
    zdot1=-100.0*y/7.0;

    x=x + T/2.0*(3.0*xdot1-xdot0);
    y=y + T/2.0*(3.0*ydot1-ydot0);
    z=z + T/2.0*(3.0*zdot1-zdot0);

    xdot0=xdot1;
    ydot0=ydot1;
end

```

```

zdot0=zdot1;

xdata(j)=x;
j=j+1;
end % FOR
% Creates Rossler series data
% *****
% FIND THE LOCAL MAXIMUM VALUES
for n=j-max_n+1:j-2;
% SU
    if xdata(n-1)<xdata(n) & xdata(n)>xdata(n+1);
        store(c,1)=xdata(n);
        store(c,2)=R;
        c=c+1; % TIME STEP FOR INTEGRATION
    end % IF % Transient steps to discard
end % FOR % NUMBER OF INTEGRATION STEPS to record
% *****
store=zeros(10000,2); % temporary number

end %Main loop
%Main loop
store=store(1:c-1,:);
xdata=xdata(1:j-1);
% *****
store=store'; % INITIAL CONDITIONS
xdata=xdata';
z=0.03;

save c:\xdata.dat xdata -ascii
save c:\store.dat store -ascii
% *****

figure(1);plot(store(2,:),store(1:,'k.'),
title('Bifurcation diagram of Chua system'),
xlabel('Parameter A'), axis tight,

```

Appendix 2

Rossler Bifurcation Diagram

```
%-----  
% Creates Rossler series data  
%-----  
  
% *****  
% SET VALUES AND DEFINE VARIABLES  
  
c=1;j=1;  
T=0.01;          % TIME STEP FOR INTEGRATION  
temp_n=20000;    % Transient steps to discard  
max_n=5000;      % NUMBER OF INTEGRATION STEPS to record  
rdata=zeros(1,5000*200);  
store=zeros(10000,2); % temporary number  
  
%Main loop  
for R=4.01:0.01:6;  
  
% *****  
x=0.01;          % INITIAL CONDITIONS  
y=0.1;  
z=0.03;  
a=0.2;  
b=0.2;  
% *****  
  
% Transient passage loop  
for n=1:temp_n
```

```

% .....
% Rossler equations, R=c
for n=1:max_n
    xdot=-y-z;
    ydot=x+a*y;
    zdot=b+x*z-R*z;

    x=x + T*xdot;
    y=y + T*ydot;
    z=z + T*zdot;
end % FOR
% .....

```

```

% Data recording loop
for n=1:max_n
    store=store(1:n-1,:);

    % Rossler equations, R=c
    xdot=-y-z;
    ydot=x+a*y;
    zdot=b+x*z-R*z;

    x=x + T*xdot;
    y=y + T*ydot;
    z=z + T*zdot;

    figure(3);plot(store(2,:),store(1,:), 'r', MarkerSize 4)
    rdata(j)=x;
    j=j+1;

end % FOR

```

```

% *****
% FIND THE LOCAL MAXIMUM VALUES
for n=j-max_n+1:j-2;
    if rdata(n-1)<rdata(n) & rdata(n)>rdata(n+1);
        store(c,1)=rdata(n);
        store(c,2)=R;
        c=c+1;
    end % IF
end % FOR
% *****

end % Main loop

store=store(1:c-1,:);
rdata=rdata(1:j-1);

store=store';
rdata=rdata';

save c:\rdata.dat rdata -ascii
save c:\store.dat store -ascii

figure(3);plot(store(2,:),store(1,:),'k.','MarkerSize',4)
title('Bifurcation diagram of Rossler system')
xlabel('Parameter c')

save sfm2.dat sfm -ascii
figure(2)
plot(sfm(:,2),sfm(:,1),'k') %or 'b' for black
axis([8.65 8.95 0 0.5])

```

Appendix 3

Chua Structure Function Simulation

```
%-----  
% Implement SF to Chua's Series  
%-----  
load c:\xdata.dat -ascii;  
sf=zeros(85,2);  
c=1;k=0;  
n=48;  
N=100;  
total=0;  
  
for A=8.6515:0.0015:8.95;  
    x=xdata(1+5000*k:(k+1)*5000);  
    for im=1:1:N-n;  
        tempm1=(x(im+n) - x(im)).^2;  
        total=total+tempm1;  
    end;  
    sfm(c,1)=total/(N-n);  
    sfm(c,2)=A;  
    k=k+1;  
    c=c+1;  
    total=0;  
end; %FOR  
  
save sfm2.dat sfm -ascii  
figure(2)  
plot(sfm(:,2),sfm(:,1),'k') %or 'k' for black  
axis([8.65 8.95 0 0.5])
```



```
title('Structure Function for set of time series (Chua)')
```

```
xlabel('Parameter A')
```

```
ylabel('SF')
```

Appendix 4

Rosier Structure Function Simulation

```
%-----  
% Implements SF to Rosier series data  
%-----  
load c:\data.dat -ascii;  
sf=zeros(85,2);  
c=1;k=0;  
N=50;  
n=24;  
total=0;  
  
for A=4.01:0.01:6;  
    r=randn(1+5000*k:(k+1)*5000);  
    for im=1:1:N-n;  
        tempn1=(r(im+n) - r(im)).^2;  
        total=total+tempn1;  
    end;  
    sf(c,1)=total/(N-n);  
    sf(c,2)=A;  
    k=k+1;  
    c=c+1;  
    total=0;  
end; %FOR  
  
%save sf.dat sf -ascii  
figure(4)  
plot(sf(:,2),sf(:,1),X)  
axis([4.6 -1.5])
```

Appendix 4

Rossler Structure Function Simulation

```
title('Structure Function for set of time series (Rossler)')
xlabel('Parameter c')
ylabel('SF');

%-----
% Implements SF to Rossler series data
%-----

load c:\rdata.dat -ascii;
sf=zeros(85,2);
c=1;k=0;
N=50;
n=24;
total=0;

for A=4.01:0.01:6;
    r=rdata(1+5000*k:(k+1)*5000);
    for im=1:1:N-n;
        tempm1=(r(im+n) - r(im)).^2;
        total=total+tempm1;
    end;
    sfr(c,1)=total/(N-n);
    sfr(c,2)=A;
    k=k+1;
    c=c+1;
    total=0;
end; %FOR

%save sfr.dat sfr -ascii
figure(4)
plot(sfr(:,2),sfr(:,1),'k')
axis([4 6 -1 5])
```

title('Structure Function for set of time series (Rossler)')

xlabel('Parameter c')

ylabel('SF'),

Rotor 37 Structure Function Simulation

```
%-----  
% Filter and Normalize Rotor 37 Data and Apply SF  
%-----  
  
% *****  
% Genb data  
% *****  
load '.../rotor37ut1001';  
x1=Prestart(10001:20000,1);  
x=x1;  
% *****  
% Filter and Normalize data  
% *****  
% find zero avg of data and subtract from the original data  
% to form the zero avg data  
x_orig=x;  
[p,q]=size(x);  
avg=sum(x)/p;  
x_avg = x - repmat(avg,p,1);  
x=x_avg;  
data=x;  
low=288;hcut=500;  
%  
sampling_freq = 3000;  
nyq_freq = sampling_freq/2;  
wn=(low+hcut)/nyq_freq;  
[b,a]=butter(3,wn);
```

Appendix 6

Rotor 37 Structure Function Simulation

```
%-----  
% Filter and Normalize Rotor 37 Data and Apply SF  
%-----  
  
% *****  
%Grab data % sliding distance in successive windows  
% *****  
load '..\rotor37\st1001';  
x1=Pressure(10001:20000,1);  
x=x1;  
% *****  
%Filter and Normalize data  
% *****  
%find zero avg of data and subtract from the original data  
%to form the zero avg data  
x_orig=x;  
[p,q]=size(x);  
avg=sum(x)/p;  
x_avg = x - repmat(avg,p,1);  
x=x_avg;  
data=x;  
lcut=288;hcut=300;  
%  
sampling_freq = 3000;  
nyq_freq = sampling_freq/2;  
wn=[lcut hcut]/nyq_freq;  
[b,a]=butter(3,wn);
```

```

y=filtfilt(b,a,x);
x=y;
% *****
%Apply SF
% *****
y1=y;
start=1; % start location in the data array x
n=2;
N=10;
Shift=N/10; % sliding distance in successive windows
[mm,nn]=size(x);
count=1+fix((mm-N-start+1)/Shift);
sf=zeros(count,1);
for NN_count=1:count
    temp1=(x(n+start:N+start-1) - x(start:N-n+start-1)).^2;
    sf(NN_count)=mean(temp1);
    start=start+Shift;
end;
% *****
%Figure
% *****
figure(4)
plot_limit=5005;
rev_count=180;
plot_start=plot_limit-(10*rev_count); %10 data points are taken during each rev
subplot(3,1,1),plot(x1(plot_start:plot_limit),'k');
title('Rotor 37 Raw data time series from Sensor1')
ylabel('Amplitude')
set(gca,'XTick',0:200:1800)
set(gca,'XTickLabel',{0:0:0})
axis tight, grid on,

```

```

subplot(3,1,2),plot(y1(plot_start:plot_limit),'k');
title('Zero-averaged and BandPass Filtered data between 288-300Hz')
ylabel('Amplitude')
set(gca,'XTick',0:200:1800)
set(gca,'XTickLabel',{0:0:0})
axis tight, grid on,
subplot(3,1,3),plot(sf(plot_start+N-n:plot_limit+N-n),'k');
title('SF, Step size 2, Window size 10')
ylabel('SF'),xlabel('Revolutions before stall'),
set(gca,'XTick',0:200:1800)
set(gca,'XTickLabel',{-180:20:0})
axis tight, grid on,

```

```

%Grab at least 7000 samples before stall occurs
%Plot selected data
data=x;
lower=22e3; upper=33.1e3;
x=data(lower:upper);
data=x;
t=1:size(data);
figure(1)
plot(t,x)
title('raw data')
%find zero avg of data and subtract from the original data
%to form the zero avg data
p=length(data);
avg=sum(x)/p;
x_avg = x - repmat(avg,p,1);
x=x_avg;

```

Appendix 6

T55 Structure Function Simulation

```
%-----  
% Filter and Normalize T55 Rig1 Data and Apply SF  
%-----  
sampling_freq=3000;  
% *****  
%Grab data  
% *****  
load '...\t55\DATA1';  
%Grab at least 7000 samples before stall occurs  
%Plot selected data  
data=x;  
lower=22e3; upper=33.1e3;  
x=data(lower:upper);  
data=x;  
t=1:1:size(data);  
figure(1)  
plot(t,x)  
title('raw data')  
%find zero avg of data and subtract from the original data  
%to form the zero avg data  
p=length(data);  
avg=sum(x)/p;  
x_avg = x - repmat(avg,p,1);  
x=x_avg;  
end;  
save c:\t55_avg_1p.dat x -ascii
```

```

% *****
%Filter data
% *****

lcut=8;
sampling_freq = 3000;
nyq_freq = sampling_freq/2;
[b,a]=butter(3,lcut/nyq_freq);
y=filtfilt(b,a,x);
%plot filtered data
figure(2)
set(0,'DefaultFigureColor','white'),
ty=1:1:size(y);
plot(ty,y)
title('Filtered and Averaged Data')
% *****
%Apply SF
% *****
x=y;
start=1; % start location in the data array x
n=2;
N=10;
Shift=N/10; % sliding distance in successive windows
[mm,nn]=size(x);
count=1+fix((mm-N-start+1)/Shift);
sf=zeros(count,1);
for NN_count=1:count
    temp1=(x(n+start:N+start-1) - x(start:N-n+start-1)).^2;
    sf(NN_count)=mean(temp1);
    start=start+Shift;
end;
save c:\t55_avg_lp.dat x -ascii

```



```

% *****
%Figure
% *****
figure(6)
plot_limit=p-50; %p=length(data);
rev_count=180;
plot_start=plot_limit-(10*rev_count); %10 data points are taken during each rev
subplot(3,1,1),plot(data(plot_start:plot_limit),'k');
title('T55 Raw data time series from Sensor1')
ylabel('Amplitude')
set(gca,'XTick',0:200:1800)
set(gca,'XTickLabel',{0:0:0})
axis tight, grid on,
subplot(3,1,2),plot(x(plot_start:plot_limit),'k');
title('Zero-averaged and LowPass Filtered data at 8Hz')
ylabel('Amplitude')
set(gca,'XTick',0:200:1800)
set(gca,'XTickLabel',{0:0:0})
axis tight, grid on,
% subplot(3,1,3),plot(sf(plot_start-N+1:plot_limit-N+1),'k');
subplot(3,1,3),plot(sf(plot_start+N-n:plot_limit),'k');
title('SF, Step size 2, Window size 10')
ylabel('SF'),xlabel('Revolutions before stall'),
set(gca,'XTick',0:200:1800)
set(gca,'XTickLabel',{-180:20:0})
axis tight, grid on,
zoom on;

```

Appendix 7

Application of SF to Ross system using DSP

```
/*
 * ===== sf.h =====
 */

#define N 50 //window size
#define n 24 //step size
#define BUFSIZE N //length to be read from file

/*
 * ===== main.c =====
 */

#include <stdio.h>
#include "sf.h"

/* Global declarations */
float inp[BUFSIZE]; /* processing data buffers */
float out_buffer[200];
int k;
float total=0; //total
int im=0; //counter
float temp1;
float temp3; //dummy variables
int m;

/* Functions */
static int sf(void);
```

```

static void dataInput(void);

/* ===== main ===== */

void main()
{
/* ===== */
while(k<200)
{
/* PARAMETERS: none */
/* Read using a Probe Point connected to a host file. */
/* Read From: ../rN51.dat buffer:50 */
dataIO();
}

sf();
}
/* Write output to a graph connected through a probe-point. */
/* Write to: sf_ross_51.dat length: 400 */
k=0;
}

/* ===== sf ===== */
* FUNCTION: apply structure function to input signal.
* PARAMETERS: none */
static int processing()
{

    for(im=0;im<(N-n);im++)
    {
        temp1=(inp[im+n] - inp[im]);
        temp3=temp1*temp1;
        total=temp3;
    }
}

```

```

    }
    out_buffer[k]=(float) total/(N-n);
    k++;
    total=0;

    return;
}
/* ===== dataInput ===== */
* FUNCTION: read input data.
* PARAMETERS: none. */
static void dataInput()
{
    return;
}

/* ===== main.c ===== */
*/

#include <stdio.h>
#include <stdlib.h>
#include "sf.h"

/* Global declarations */
float inp_buffer[BUFSIZE]; /* processing data buffers */
float out_buffer[2];
char line[LENGTH]; /*17 chars per line on matlab output
int k_in=0;
float data;
int file_index;
int counter; /*main counter
float total=0;

```

Appendix 8

Application of SF to T55 data using DSP

```
#File IO
/*
 * ===== sf.h =====
 *
 * Functions */
*/
#define BUFSIZE N // length to be read from file
#define LENGTH 21 //matlab data has 17 char per line
#define N 10 // Window size
#define n 2 // Step size

void main()
/*
 * ===== main.c =====
 */
#include <stdio.h>
#include <stdlib.h>
#include "sf.h"

/* Global declarations */
float inp_buffer[BUFSIZE]; /* processing data buffers */
float out_buffer[2];
char line[LENGTH]; //17 chars per line on matlab output
int k_in=0;
float data;
int file_index;
int counter; //main counter
//
float total=0;
```

```

int im;
int k_out;

//File IO
FILE *filein, *fileout;

/* Functions */
static int processing(void);
static void dataIn(void);

/* ===== main =====
*/
void main()
{
/* Data file used for T55 Simulation
filein=fopen("c:\\t55.dat","r"); */
/* Data file used for Rotor37 Simulation */
filein=fopen("c:\\r37.dat","r");
/*
if (filein == NULL)
{
puts("file did not open\n");
}
*/
PARAMETERS: address of input and output buffers.
//puts("program started\n");
RETURN VALUE: TRUE
/* puts("Zeroing array...\n");
static int processing()
{
for (k_in = 0; k_in <=BUFSIZE; k_in++)
{

```

```

    inp_buffer[k_in] = 0;
}
*/
while (counter<1793) {
    /*Advance the file position indicator by offset bytes with
    respect to place, 0: beginning, 1: current position, 2: EOF; */
    fseek(filein,counter*17,0);
    // file_index=ftell(filein);

    dataIn();
    processing();
    dataOut();

    counter++;
}

fclose(filein);
}

/* ===== dataIO =====
* ===== processing =====
* FUNCTION: read input signal and write processed output signal.
* FUNCTION: apply signal processing transform to input signal.
* PARAMETERS: none.
* PARAMETERS: address of input and output buffers.
* RETURN VALUE: none.
* RETURN VALUE: TRUE.
*/
static int processing()
{
    float temp;

```

```

for(im=0;im<(N-n);im++)
    {
        data=(inp_buffer[im+n] - inp_buffer[im]);
        temp=data*data;
        total=total+temp;
    }
out_buffer[k_out]=(float) total/(N-n);
k_out++;
if (k_out >= 2)
    {
        k_out=0; //Add fileoutput probe here: add:out_buffer,
        // length:4
    }
}

```

```

return(1);

```

```

}

```

```

/*

```

```

* ===== dataIO =====

```

```

*

```

```

* FUNCTION: read input signal and write processed output signal.

```

```

*

```

```

* PARAMETERS: none.

```

```

*

```

```

* RETURN VALUE: none.

```

```

*/

```

```

static void dataIn()

```

```

{

```

```

    /* do data I/O */

```



```
k_in=0;

//      puts ("start reading file\n");
//*****
while (k_in< N) {
    fgets(line, LENGTH, filein);
    data=atof(line);
    inp_buffer[k_in]=data;
//      file_index=ftell(filein);
    k_in++;
}

return;
//*****
}
//
```