12-2006

Microprocessor Implementation of Autoregressive Analysis of Process Sensor Signals

Swetha Priyanka Pakala
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Swetha Priyanka Pakala entitled "Microprocessor Implementation of Autoregressive Analysis of Process Sensor Signals." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Gregory D. Peterson, Major Professor

We have read this thesis and recommend its acceptance:

Belle R. Upadhyaya, Syed Islam

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Swetha Priyanka Pakala entitled "Microprocessor Implementation of Autoregressive Analysis of Process Sensor Signals." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science, with a major in Electrical Engineering.

Gregory D. Peterson
Major Professor

We have read this thesis
and recommend its acceptance:

Belle R. Upadhyaya

Syed Islam

Accepted for the council

Linda painter
Interim Dean of Graduate Studies

(Original signatures are on file with official student records)

Microprocessor Implementation of Autoregressive Analysis of Process Sensor Signals

A Thesis

Presented for

Master of Science Degree

University of Tennessee

Swetha Priyanka Pakala

December 2006

Dedicated To
My Advisor Dr. Gregory D. Peterson
And
My Parents

ACKNOWLEDGEMENT

"*Behind every successful achievement lies great contribution by those, without which it could not have been achieved*". Although mere words of gratitude are insufficient for their unlimited contribution, I take this opportunity to convey my thanks to all of them who supported me in doing my thesis work.

First, I would like to sincerely thank my advisor Dr. Gregory D. Peterson for giving me this opportunity to work in his research group and for guiding me throughout my graduate studies. This work is a result of his constant encouragement, advice and support. I thank him and Dr. Belle R. Upadhyaya for giving me an opportunity to work on the project "Microprocessor Implementation of Autoregressive Analysis of Process Sensor Signals" and Emerson Process Management for sponsoring the project.

Again, I would like to thank my committee members Dr. Belle R. Upadhyaya and Dr. Syed Islam for reviewing my thesis and helping me in understanding some of the concepts related to my work.

Special Thanks to Saumil Merchant, Akila Godhandaraman and Derek Rose for sharing their knowledge with me and for helping me in debugging the design modules at various stages of my project. I would also like to thank my friends Shailaja and Nanditha for helping me in doing this work.

Finally, I would like to thank my parents, Sri Pakala Mallaiah, Srimati Pakala Ramakumari and my sisters Keerthi and Divya and all other friends for their support and encouragement throughout my education.

ABSTRACT

Automated signal analysis can help for effective system surveillance and also to analyze the dynamic behavior of the system such as impulse response, step response etc. Autoregressive analysis is a parametric technique widely used for system surveillance and diagnosis. The main aim objective of this research work is to develop an embedded system for autoregressive analysis of sensor signals in an online fashion for monitoring system parameters. This thesis presents the algorithm, data representation and performance of the optimized microprocessor implementation of autoregressive analysis.

In this work an autoregressive (AR) model is generated as a solution to a linear system of equations called Yule-Walker linear equations. The generated model is then implemented on Motorola PowerPC MPC555 processor. The embedded software for autoregressive analysis is written in the C programming language using fixed point arithmetic. It includes estimation of the autoregressive parameters, estimation of the noise variance recursively using the AR parameters, determination of the optimal model order and the model validation.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Objective

The objective of this work is to estimate the autoregressive (AR) parameters of the sensor noise signals for system surveillance and monitoring of system parameters. The mathematical model for the estimation of the AR parameters is implemented on a microprocessor using the CodeWarrior development tool, which offers a complete integrated development environment for hardware bring-up through programming applications like C/C++ and Java. It also includes the estimation of the autoregressive parameters, the determination of the optimal model order and the model validation.

The dynamic behavior of the system such as its impulse response, step and ramp responses can be estimated in a simple and fast manner using the autoregressive parameters. The power spectral density can also be estimated using the AR parameters for the frequency analysis of the system. Initially, the data is collected from the pressure sensor signals and an AR model is fitted to it for the estimation of the autoregressive parameters. Autoregressive parameter estimation model is chosen for the parameter estimation because the autoregressive spectrum contains sharp peaks which are used for the high resolution spectral analysis. The autoregressive parameter estimation is very simple and can be generated as a solution to a linear system of equations called the Yule-Walker linear equations. The linear equations are solved using Yule-Walker recursive algorithm of autoregressive parameter estimation. A single block of 1024 samples is used at a time for the estimation of the autoregressive parameters. Noise variance is calculated using autoregressive parameters recursively and also using the residual error sequence of the time series. The optimal model order is decided based upon the Akaike Information Criterion (AIC) [1]. The model order, for which the AIC function is minimized, is considered to be the optimal model order.

1.2 Time series analysis

A time series is a set of samples generated sequentially in time. It performs both univariate and multivariate analysis of signals. If a time series is a set of scalar quantities, it is called a univariate series and if it is a set of n-dimensional vector quantities, it is called a multivariate series of data. A time series can be either continuous or discrete. A discrete time series can be generated in two ways:

a) By sampling a continuous time series.

b) By accumulating a variable over a period of time.

1.2.1 Deterministic and statistical time series

If the future values of the time series are exactly determined by some mathematical function or equation, such a time series is called a deterministic time series. If the future values can be described only in terms of a probability function, such a time series is called a statistical time series [2].

Time series analysis is used in the applications such as:

- Predicting the future values of a time series from current and past values
- Determining the transfer function of a system
- Designing feed forward and feedback control schemes
- Sales forecasting
- Inventory studies
- Census analysis
- Budgetary analysis
- Stock market analysis
- Economic forecasting

Using time series and different mathematical models, one can read, plot and convert the raw time series data into a form suitable for model fitting of the data. In many problems we have to consider a time dependent phenomenon for which it is not possible to write a deterministic model that allows exact calculation of the future behavior of the phenomenon. However, it is possible to derive a model that can be used to calculate the probability of a future value lying between two specified limits and such a model is

called a stochastic model. Stochastic models are very much useful for forecasting and control.

An important class of stochastic models for describing time series is stationary models in which the process remains in equilibrium about a constant mean level. Some of the stationary stochastic processes in modeling time series are autoregressive, moving average and autoregressive moving average processes [2].

1.2.2 Autoregressive models

An autoregressive model is a stochastic model which is used to estimate the current value of the model using its previous values by expressing it as a linear combination of the previous values of the process as given by the equation 1.1.

$$z_t = \phi_1 z_{t-1} + \phi_2 z_{t-2} + \dots\dots\dots\dots + \phi_p z_{p-1} + a_t \qquad 1.1$$

The above equation is called an autoregressive (AR) process of order 'p'. Autoregressive processes may be stationary or non-stationary. In this project a simple stochastic stationary process with added white noise is considered for the AR model fitting to the time series data.

1.2.3 Mean and variance of a stationary process

The mean μ and variance of the stochastic process can be estimated with the following equations 1.2 and 1.3 respectively. Generally, the mean value of the time series data is subtracted from each sample of the time series sequence for the elimination of the DC components.

$$z = \frac{1}{N}\sum_{t=1}^{N} z_t \qquad 1.2$$

$$\sigma_z^2 = \frac{1}{N}\sum_{t=1}^{N} (z_t - z)^2 \qquad 1.3$$

1.3 Microprocessor implementation of autoregressive analysis

The mathematical model for autoregressive analysis of the process sensor signals is implemented in the 'C' programming language on a microprocessor with the help of CodeWarrior integrated development environment (IDE) software. The microprocessor

used is a Freescale PowerPC MPC 555 [12]. The block diagram for the microprocessor implementation of the autoregressive analysis of process sensor signals is shown in Figure 1.1.

The analog sensor signal is first band limited using the Butterworth low pass and high pass filters and is amplified to obtain high resolution. The amplified signal is then fed to the queued analog-to-digital converter of the PowerPC MPC 555 for the generation of the digital time series data. The sampled time series data is generated using the periodic interrupt timer of the PowerPC. Both the timer and the QADC modules are programmed as per the desired specifications. The sampled data is fed to the software module for the estimation of the AR parameters and the optimal model order. The source code for the implementation of the autoregressive parameter estimation is written in 'C' and is compiled using the CodeWarrior software which is used for the translation of the source code written in C into the assembly language.

1.4 Autoregressive model generation

The basic steps involved in the generation of the autoregressive model are shown in the form of a block schematic in Figure 1.2 [2]. The steps involved are:

- Initially, a useful class of models is chosen based upon the requirements.
- Depending upon the data, rough methods for identifying the subclass of these models are developed. The identification process can be used for the rough estimate of the initial parameters for model fitting.
- The selected model is fitted to data and its parameters are estimated. Rough estimated values determined during the identification of one particular model to be evaluated are used as the initial values for the iterative methods for the estimation of the parameters.
- The model order is selected depending upon various criteria. Most of the criteria depend on minimizing the error variance.
- Model validation is done to determine whether the developed model properly describes the physical phenomenon.

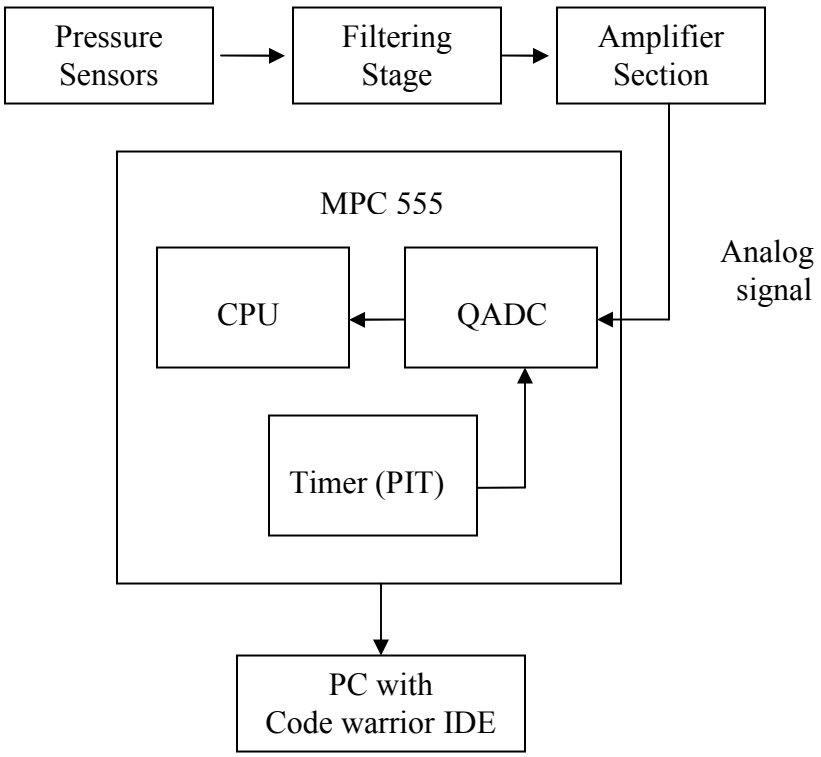

Figure1.1 Microprocessor implementation of the autoregressive analysis

Steps in model building

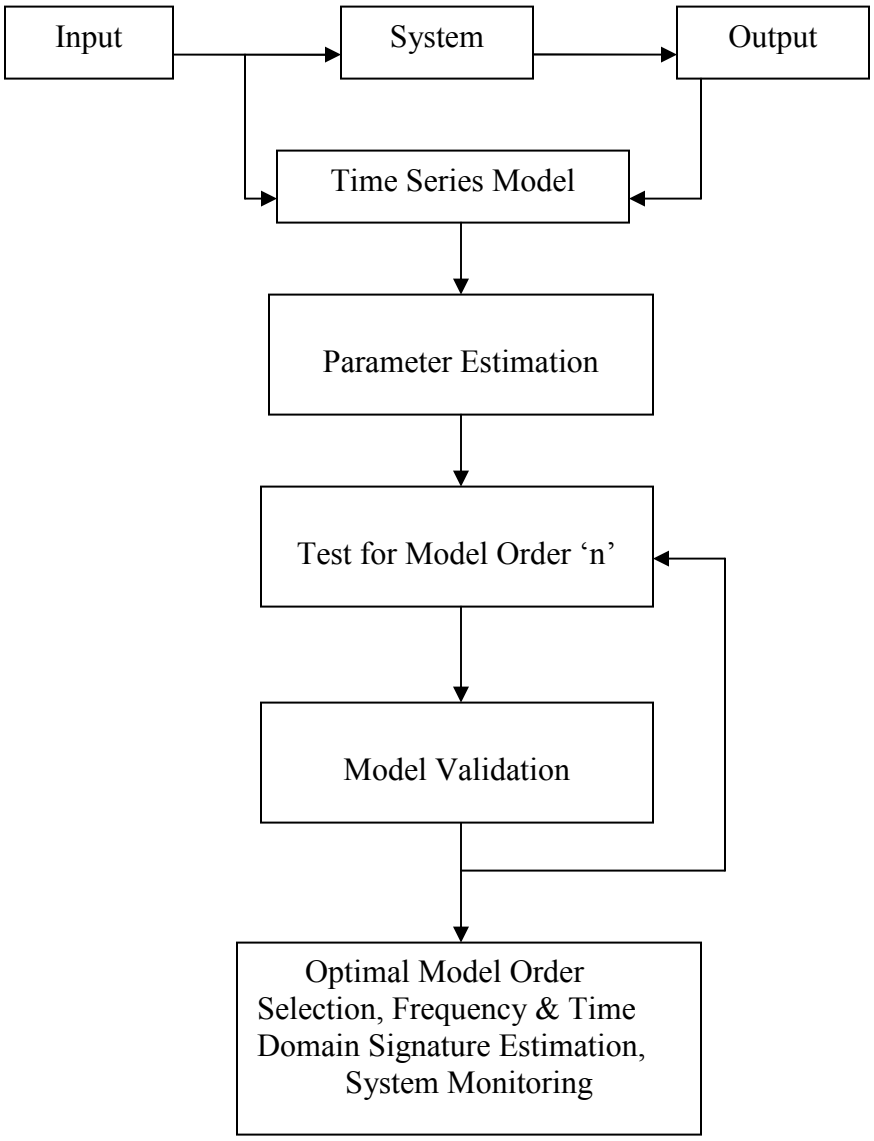

Figure 1.2 Block diagram for the autoregressive model building

This process of model building is repeated until an appropriate model for the estimation of the parameters and an optimal model order are determined.

1.5 Thesis outline

Chapter 1 presents the objectives of this thesis and an outline of the implementation of the AR model. It also discusses the fundamentals of time series and autoregressive analysis. Chapter 2 presents the literature review of various parametric models. It discusses the autoregressive, moving average, and autoregressive moving average models in detail. It also presents various properties of autoregressive processes and the relationship between AR, MA and ARMA models. Various methods of AR parameter estimation methods such as Yule Walker method, Geometric method and Burgs method are explained in detail. Different model order estimation selection criteria are discussed in this chapter.

Chapter 3 presents a mathematical approach to the AR parameter estimation, which is implemented using software and the model order selection criterion. The initial conditions for the model order estimation are also stated in this chapter.

Chapter 4 describes the hardware modules of the microprocessor implementation of the AR model for process signals. It gives detailed information about the MPC555 PowerPC and its various divisions QADC and timers. Chapter 5 describes the data acquisition apparatus and brings out some of the details of its components. In Chapter 6, the final results, graphs and discussion of the details of various specifications of the software are presented. Chapter 7 discusses the future work and conclusions. Appendix A presents the software code and Appendix B gives MATLAB code for the generation of the test data. Appendix C presents the results.

CHAPTER 2

REVIEW OF LITERATURE

2.1 Parametric models

Parametric estimation of the power spectral density (PSD) can be done assuming a time series model of random process. It can be calculated using the model parameters. Consider a random process with added white Gaussian noise. The parametric models that can be used for the estimation of the power spectral density are:

(a) Autoregressive process model (AR model)

(b) Moving average process model (MA model)

(c) Autoregressive moving average process model (ARMA model)

The output process of these above models is completely dependent on the model parameters and the variance on the white noise and has the following features:

(a) Ability to achieve better PSD estimation depending on the model

(b) Better spectral resolution

The degree of the improvement in resolution and spectral fidelity is determined by the appropriateness of the selected model and the ability of the model to fit the measured data with few parameters.

2.2 AR, MA, and ARMA Models

If spectra with sharp peaks but no deep nulls are required then the autoregressive analysis is appropriate for the estimation of the power spectral density. If spectra with deep nulls, but no sharp peaks, are required then moving average analysis is appropriate. The autoregressive model required less computation when compared to moving average and autoregressive moving average models.

2.2.1 Mathematical analysis

Let us consider a time series model as given in equation 2.1 [1] that approximates many discrete time deterministic and stochastic processes,

$$x[n] = -\sum_{k=1}^{p} a[k]x[n-k] + \sum_{k=0}^{q} b[k][n-k] \\ = \sum_{k=0}^{\infty} h[k]u[n-k] \qquad 2.1$$

where,

X [n] is the output sequence of a casual filter (h[k] =0 for k<0).

U [n] is an input driving sequence.

The above equation represents an autoregressive moving average (ARMA) model for the time series x [n] with u [n] as the white noise sequence. ARMA model is given in the Figure 2.1. The a[k] parameters form the auto regressive parameters and b[k] parameters form the moving average parameters.

The ARMA power spectral density is given by the equation 2.2 [1].

$$P_{ARMA}(f) = T_{\rho\omega}\left|\frac{B(f)}{A(f)}\right|^2 \qquad 2.2$$

where the polynomials A(f) and B(f) are given by the Equation 2.3 [1]

$$A(f) = 1 + \sum_{k=1}^{p} a[k]\exp(-j2\pi fkT) \\ B(f) = 1 + \sum_{k=1}^{q} b[k]\exp(-j2\pi fkT) \qquad 2.3$$

Figures 2.2 and 2.3 represent autoregressive and moving average models which are deduced from the autoregressive moving average model. If all the autoregressive parameters are zero except a [0] = 1 then

$$x[n] = \sum_{k=1}^{q} b[k]u[n-k] + u[n] \qquad 2.4$$

is strictly a moving average (MA) process of order q [1].

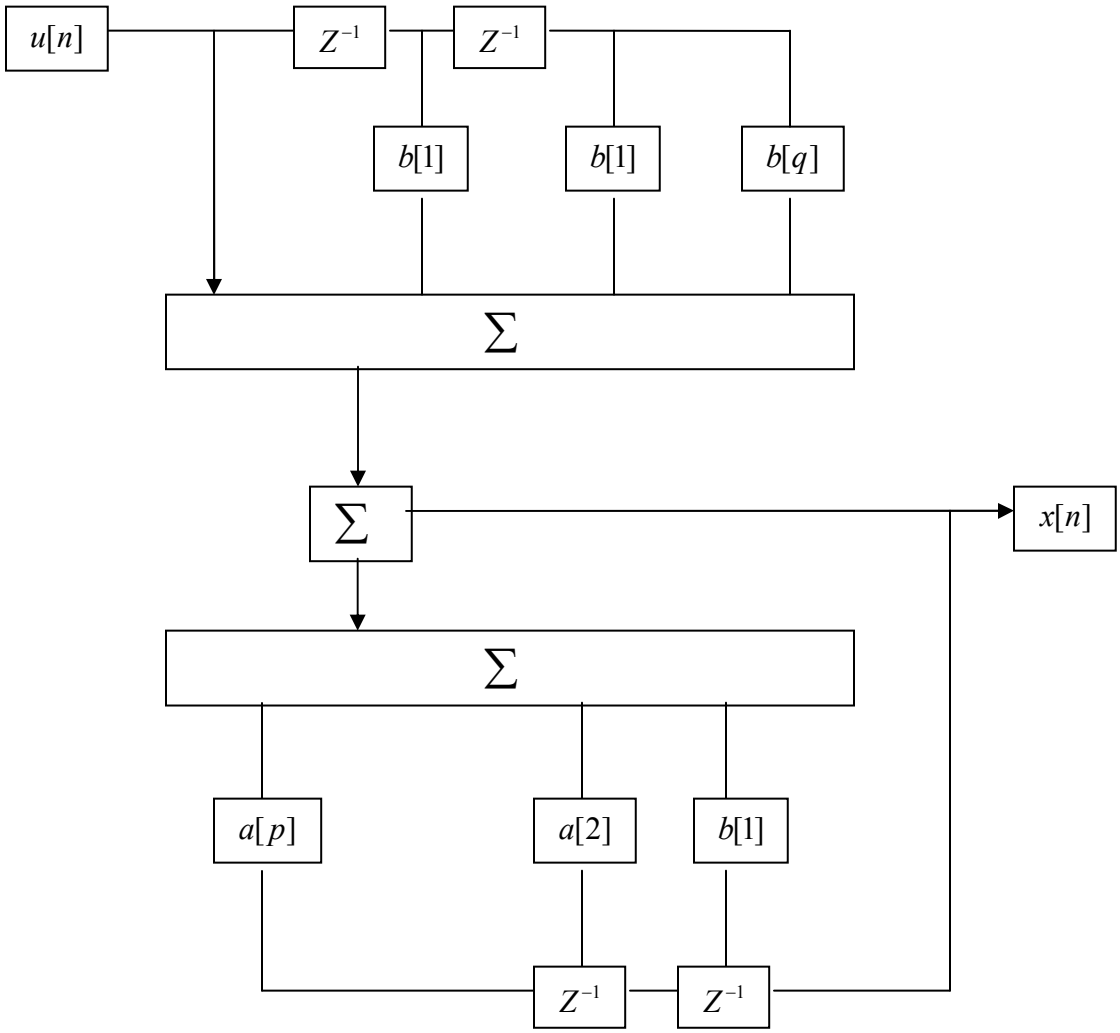

Figure 2.1 ARMA Filter of order (p, q) (from [1])

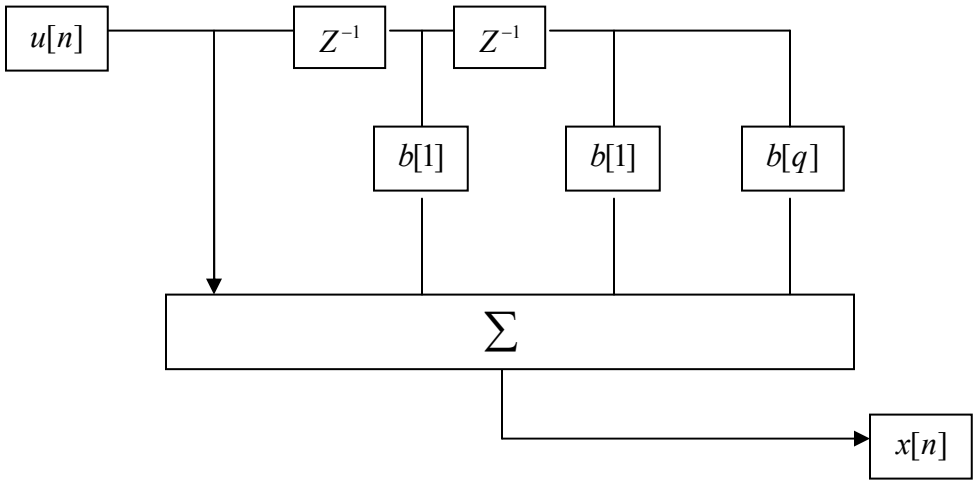

Figure 2.2 Moving average filter of order q (from [1])

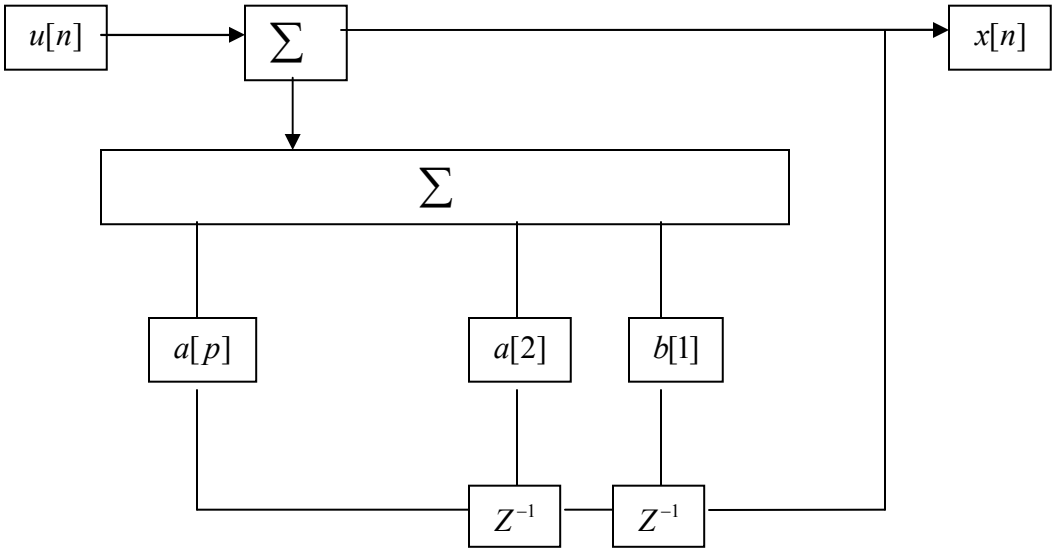

Figure 2.3 Autoregressive filter of order p (from [1])

If all the moving average parameters are zero except b [0] = 1 then

$$x[n] = -\sum_{k=1}^{p} a[k]x[n-k] + u[n] \qquad 2.5$$

is strictly an autoregressive (AR) process of order p.

2.3 Determination of the model parameters with autocorrelation sequence

Autoregressive parameters of an ARMA model are related by a set of linear equations to the autocorrelation sequence and is given in the matrix form for p lag indices $q+1 \le m \le q+p$ as [1]

$$\begin{bmatrix} r_{xx}[q] & r_{xx}[q-1] & \dots & r_{xx}[q-p+1] \\ r_{xx}[q+1] & r_{xx}[q] & \dots & r_{xx}[q-p+2] \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ r_{xx}[q+p-1] & r_{xx}[q+p-2] & \dots & r_{xx}[q] \end{bmatrix} \begin{bmatrix} a[1] \\ a[2] \\ . \\ . \\ . \\ a[p] \end{bmatrix} = -\begin{bmatrix} r_{xx}[q+1] \\ r_{xx}[q+2] \\ . \\ . \\ . \\ r_{xx}[q+p] \end{bmatrix} \qquad 2.6$$

Autoregressive parameters can be calculated from the moving average parameters as the solution to the simultaneous equations given in the above matrix. This relationship is called ARMA Yule-Walker normal equations. The number of computations required is proportional to p^2 [1].

The relationship between the autocorrelation sequence and autoregressive model can be obtained by setting q = 0 yielding [1],

$$\begin{aligned} r_{xx}[m] &= -\sum_{k=1}^{p} a[k]r_{xx}[m-k] && for\ m > 0 \\ &\quad -\sum_{k=1}^{p} a[k]r_{xx}[-k] + \rho_{\omega} && for\ m = 0 \\ &\quad r_{xx}^{*}[-m] && for\ m < 0 \end{aligned} \qquad 2.7$$

This relationship may be evaluated for the p+1 indices $0 \le m \le p$ and formed into the matrix expression as given by the equation 2.8 [1].

$$\begin{bmatrix} r_{xx}[0] & r_{xx}[-1] & \dots r_{xx}[-p] \\ r_{xx}[1] & r_{xx}[0] & \dots r_{xx}[-p+1] \\ . & . & . \quad . \\ . & . & . \quad . \\ . & . & . \quad . \\ r_{xx}[p] & r_{xx}[p-1] & \dots \quad r_{xx}[0] \end{bmatrix} \begin{bmatrix} 1 \\ a[1] \\ . \\ . \\ . \\ a[p] \end{bmatrix} = \begin{bmatrix} \rho_\omega \\ 0 \\ . \\ . \\ . \\ 0 \end{bmatrix} \qquad 2.8$$

The above relationship forms the AR Yule-Walker normal equations. The autocorrelation lags 0 to p uniquely describe the autoregressive process of order p, as the lags for $|k| > p$ are obtained recursively using the equation 2.9 [1].

$$r_{xx}[m] = -\sum_{k=1}^{p} a[k] r_{xx}[m-k] \qquad 2.9$$

The relationship between the autocorrelation sequence and moving average model can by obtained by setting p = 0 and is given by

$$\begin{aligned} r_{xx}[m] &= 0 && for\ m > q \\ &\rho_\omega \sum_{k=m}^{q} b[k] b^*[k-m] && for\ 0 \le m \le q \\ &r_{xx}^*[-m] && for\ m < 0 \end{aligned} \qquad 2.10$$

2.4 Autoregressive parameter estimation methods

In general autocorrelation sequence is not given for the estimation of the autoregressive parameters. It has to be estimated using the available data. The techniques to estimate the autoregressive power spectral density are basically divided into two types

(a) Algorithms for block of data.

(b) Algorithms for sequential data.

Algorithms based on the block of data can be described as fixed time recursive in order which operate on a fixed block of data and recursively calculate the higher order parameters based upon the lower order parameters. The simplest procedure to estimate the AR parameters from the block of data would to estimate the auto correlation sequence first from the data. This autocorrelation sequence is then used to estimate the AR

parameters by substituting them in the Yule Walker linear equations. Various techniques for the estimation of the AR parameters are:

(a) Yule-Walker Method

(b) Burg Method

(c) Covariance Method

(d) Modified Covariance Method.

Once the AR parameters are estimated then the AR power spectral density is given by the equation 2.11 [1].

$$P_{AR}(f) = \frac{T\rho_{\omega}}{\left|1 + \sum_{n=1}^{p} a[n]\exp(-j2\pi fnT)\right|^2} \qquad 2.11$$

The AR model order needs to be given as the initial input data for the computations along with the input data samples for the estimation of the parameters. The Order determines the trade off between resolution and estimate variance in AR spectra [2].

2.4.1 Yule-Walker method

In this method an autocorrelation sequence is first estimated from the data samples and then the AR parameters are estimated using the Yule-Walker linear equations. For large data records, Yule walker method of AR parameters estimation produces reasonable results. For short data records, the use of the Yule-Walker approach results in the poor resolution of the spectra when compared to the other techniques of estimation. This method of estimation of the parameters is explained in detail in Section 3.1.

2.4.2 Reflection coefficient methods

In reflection coefficient method only the reflection is directly dependent on the autocorrelation function. AR parameters are then estimated using the reflection coefficients recursively the following equations 2.12 and 2.13 [1].

$$a_p[n] = a_{p-1}[n] + k_p a^*_{p-1}[p-n] \qquad 2.12$$

where,

$$k_p = a_p[p] = \frac{-\sum_{n=0}^{p-1} a_{p-1}[n] r_{xx}[p-n]}{\rho_{p-1}} \qquad 2.13$$

The recursive for driving white noise variance is given by equation 2.14 [1].

$$\rho_p = p_{p-1}(1 - |k_p|^2) \qquad 2.14$$

Various methods exist for the estimation of the AR parameters based on the reflection coefficient concept such as:

(a) Geometric Algorithm: Based geometric mean of the forward and backward prediction errors.

(b) Burgs method: Based on the harmonic mean of the forward and backward prediction errors.

The forward linear prediction error and the backward linear prediction error are given by the following equations:

$$e^f_p[n] = x[n] + \sum_{m=1}^{p} a^f_p[m] x[n-m] \qquad 2.15$$

$$e^b_p[n] = x[n-p] + \sum_{m=1}^{p} a^{f*}_p[m] x[n+m-p] \qquad 2.16$$

$$\begin{aligned} e^f_p[n] &= e^f_p[n] + k_p e^b_{p-1}[n-1] \\ e^b_p[n] &= e^b_{p-1}[n-1] + k^*_p e^f_{p-1}[n] \end{aligned} \qquad 2.17$$

The reflection coefficient based on geometric mean of forward and backward prediction errors is determined as

$$k_p = \frac{-\sum_{n=p+1}^{N} e^f_{p-1}[n] e^{b*}_{p-1}[n-1]}{\left(\sum_{n=p+1}^{N} \left|e^f_{p-1}[n]\right|^2\right)^{1/2} \left(\sum_{n=p+1}^{N} \left|e^{b*}_{p-1}[n-1]\right|^2\right)^{1/2}} \qquad 2.18$$

The reflection coefficient based on harmonic mean of forward and backward prediction errors is determined as

$$k_p = \frac{-2\sum_{n=p+1}^{N} e_{p-1}^{f}[n]e_{p-1}^{b*}[n-1]}{\sum_{n=p+1}^{N}\left|e_{p-1}^{f}[n]\right|^2 + \sum_{n=p+1}^{N}\left|e_{p-1}^{b}[n-1]\right|^2} \qquad 2.19$$

The reflection coefficient method will then uses the Levinson algorithm for the computation of the AR parameters with known autocorrelation function values and the basic Levinson algorithm is provided with initial values at order zero given below.

$$e_0^f[n] = e_0^b[n] = x[n] \qquad 2.20$$

$$\rho_o = \frac{1}{N}\sum_{n=1}^{N}\left|x[n]\right|^2 \qquad 2.21$$

2.5 Autoregressive model order selection

For the estimation of the model order we need to consider some error criterion for choosing optimal model order. Too low estimation of the model order results in a highly smooth spectral estimate. Too high estimation of model order increases the resolution and introduces spurious details into the spectrum. Hence the issue of model order selection is a trade off of increased resolution and decreased variance for AR parameter estimation. Some of the criteria for the estimation of the model order are given below.

2.5.1 Final prediction error (FPE)

The optimal model order of the autoregressive process is selected based upon the average error variance. The model order, for which the average error variance is minimum, is selected as optimal model order. FPE for an AR process is defined as

$$FPE[p] = \rho_p\left(\frac{N+(p+1)}{N-(p+1)}\right) \qquad 2.22$$

where,

N is the number of data samples

P is the order

ρ_p is the estimated white noise variance.

2.5.2 Akaike information criterion (AIC)

The AIC determines the model order by minimizing an information theoretic function. AIC of a AR process is given by

$$AIC[p] = N\ln(\rho_p) + 2p \qquad 2.23$$

where

N is the number of sample points

p is the AR model order

ρ_p is the prediction error variance

The model order is selected based upon the approach of minimizing the AIC. The model order which has minimum AIC is the required optimal model order. As $N \rightarrow \infty$, the AIC and FPE are asymptotically equal. AIC is applied for the pure AR processes. AIC is also considered to be too low for the real signals. AIC over estimates the model order as the data set length increases.

CHAPTER 3

ALGORITHM IMPLEMENTATION

In this chapter, we discuss the mathematical approach to the autoregressive (AR) parameter estimation of the process sensor signals and its software implementation in the 'C' language.

3.1 Autoregressive analysis

The Yule-Walker method is used in this project for the estimation of the AR parameters, because of its simplicity. The Yule-Walker method can be used to estimate the parameters of an autoregressive model for a given model order 'p'. It is also used for the estimation of the power spectral density (PSD) of the input signal. This method is called the autocorrelation method. It fits an autoregressive (AR) model to the input data by minimizing the forward prediction error. This approach leads to the Yule-Walker equations, which are solved by Levinson Durbin recursion method [2]. Consider a random stationary sequence y_k with added white noise v_k ,

$$y_k = \sum_{i=0}^{n} a_i y_{k-i} + v_k \qquad 3.1$$

From the above sequence, the autocorrelation functions are estimated for different lags. Let $y_1, y_2, y_3, ..., y_t$ be the time series observations of the random process y_k. Then the auto-covariance between any observations y_t and y_{t+k} separated by a lag of k intervals is estimated using the equation 3.2 [2].

$$c_k = \frac{1}{N} \sum_{t=1}^{N-k} (y_t - \overline{y})(y_{t+k} - \overline{y}) \qquad 3.2$$

where lag k = 0, 1, 2, 3….k

c_k is the estimated autocovariance between y_t and y_{t+k}

$\overline{y}$ is the mean of the random time series sequence

Here, K is not larger than N/4 and in this project is considered to be the maximum model order for which the AR parameter estimation is done.

Now the autocorrelation function is estimated as $R_k = \frac{c_k}{c_0}$ [2] where c_k s auto-covariance between two observations with lag k and c_0 is the auto-covariance with lag 0 which is the variance of the time series sequence. Consider a general stationary stochastic process

$$y_t = \sum_{i=0}^{n} a_i y_{t-i} + v_t \qquad 3.3$$

The autocorrelation functions with different lags k = 0, 1, 2 …… K where K is the maximum model order, can be obtained by multiplying the equation by y_{t-k}, where $k > 0$, computing the expected value and normalizing both the sides. The observation y_{t-k} is independent of the white noise. Let the autocorrelation function [3] be defined as

$$R_k = E[y_t y_{t+k}] \qquad 3.4$$

Now,

$$y_{t-k} y_t = \sum_{i=1}^{n} a_i y_{t-i} y_{t-k} + y_{t-k} v_k$$

$$\Rightarrow E[y_{t-k} y_t] = \sum_{i=1}^{n} a_i E[y_{t-i} y_{t-k}]$$

$$\Rightarrow R_k = \sum_{i=1}^{n} a_i R_{k-i} \;\; where\, k > 0 \qquad 3.5$$

The above equation represents a set of linear equations called Yule Walker linear equations. These linear equations can be represented in a matrix form as given in equation 3.6 [3].

$$\begin{bmatrix} R(1) \\ R(2 \\ . \\ . \\ . \\ R(n) \end{bmatrix} = \begin{bmatrix} R(0) & R(1) & \ldots & R(n-1) \\ R(1) & R(0) & \ldots & R(n-2) \\ . & . & & \\ . & . & & \\ . & . & & \\ R(n-1) & R(n-2) & \ldots & R(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ . \\ . \\ . \\ a_n \end{bmatrix} \qquad 3.6$$

where R (0), R (1)….. , etc are autocorrelations

$a_1, a_2, ...etc$ are autoregressive parameters

$$R_n = P_n.a \quad \text{where } P_n \text{ is } (n\times n) \text{ Toeplitz matrix} \qquad 3.7$$

These Yule Walker equations are solved using the Levinson-Durbin recursive algorithm for the estimation of the autoregressive parameters. Generally the Levinson-Durbin algorithm is preferred over a direct solution of the Yule-Walker equations as it also gives the partial correlations and this helps us select the appropriate autoregressive model in the process of AR parameter estimation.

3.1.1 Levinson Durbin recursive algorithm for AR analysis

The Levinson Durbin Recursive Algorithm solves n^{th} order system of linear equations of the form $R * a = b$ involving a Hermitian, positive-definite, Toeplitz matrix(R). In the Yule Walker method, the resulted Yule Walker linear equations can be written in the matrix form as given in equation 3.7. These equations can be solved for Autoregressive parameters using Levinson Durbin Recursive Algorithm resulting in the Levinson Durbin recursive formulae 3.8 and 3.9 [3],

$$a_{n+1,n+1} = \frac{R_{n+1} - \sum_{i=1}^{n} a_{n,i} R_{n+1-i}}{R_0 - \sum_{i=1}^{n} a_{n,i} R_i} \qquad 3.8$$

$$a_{n+1,j} = a_{n,j} - a_{n+1,n+1} * a_{n,n-j+1} \qquad 3.9$$

3.1.2 Estimation of noise variance

Consider a random stationary sequence y_k with added white noise v_k ,

$$y_k = \sum_{i=0}^{n} a_i y_{k-i} + v_k$$

On multiplying both the sides of the above equation by y_k and taking the expectation value, it results in the equation 3.10 [3].

$$E[y_k^2] = \sum_{i=1}^{n} a_i E[y_k y_{k-i}] + E[y_k v_k]$$
$$\Rightarrow \quad \sigma_v^2 = C_0 - \sum_{i=1}^{n} \hat{a}_i C_i \qquad 3.10$$

We observe from the above equation that the noise variance is expressed as a function of autoregressive parameters. The noise variance of higher orders is determined with the estimated autoregressive parameters recursively by using the equation 3.11 [2].

$$\sigma_{n+1}^2 = \sigma_n^2 (1 - a_{n+1,n+1}^2) \, where \, a_{n+1,n+1} \leq 1 \qquad 3.11$$

σ_n^2 is variance of model order n

$a_{n,1}, a_{n,2} .....a_{n,n}$ are autoregressive parameters of model order n

3.1.3 Residual error sequence

Noise variance can also be estimated using the residual error sequence. The noise variance estimated using the recursive formulae for a given model order should be approximately equal to the noise variance estimated using residual error sequence. Let $x(k)$ be a sequence of time series data. The residual error sequence $v(k)$ for the given time series is given by the equation

$$v(k) = x(k) - \sum_{i=1}^{n} a_i x(k-i) \qquad 3.12$$

The noise variance [2] is then estimated as

$$\hat{\sigma}_v^2 = \frac{1}{N} \sum_{k=1}^{N} \hat{v}_k^2 \qquad 3.13$$

where $\hat{\sigma}_v^2$ is the noise variance

$\hat{v}_k^2$ is the residual error sequence for model order k

The whiteness of the sequence v is checked by computing the correlation of v and its spectrum.

3.1.4 Initial conditions

The Levinson Durbin recursive algorithm is supplemented with certain initial conditions prior to the computation of the AR parameters of the higher orders. In the Levinson Durbin recursive method, the first order parameters are initialized to [3]

$$a_{1,1} = \frac{R_1}{R_0}$$

where R_1 is the normalised autocorrelation function of first order

R_0 is the normalised autocorrelation with zero order.

$$\sigma_1^2 = \mathrm{R}_0 - a_{1,1} R_1$$

Where σ_1^2 is the first order noise variance

R_1 is the Normalised Autocorrelation function of first order

R_0 is the Normalised Autocorrelation with zero order

3.2 Model order estimation

Generally speaking, there is no definite way to determine the correct model order. The predicted model order needs to be accurate. An estimated model order, if too low, will not represent the properties of the signal and if too high, will include noise and inaccuracies. The selection of the model order is done depending upon the prediction of the residual error. The order, for which the prediction error is the least, is estimated as the optimal model order. Akaike information criterion is used for the estimation of the model order in this project.

3.2.1 Akaike information criterion

Akaike information criterion (AIC) function [1] is given by the equation 3.14:

$$AIC = N \ln(\sigma^2) + 2n \qquad 3.14$$

where

N is the number of sample points

n is the AR model order

σ^2 is the prediction error variance

The model order is selected based upon the approach of minimizing the Akaike Information Criterion function. The model order which has minimum AIC is the estimated model order. As $N \rightarrow \infty$, the AIC and final prediction error are asymptotically equivalent. Final prediction error estimates the model order depending upon the average error variance. AIC is applied to pure AR processes. AIC is also considered to be too low for real signals. AIC over-estimates the model order as the length of the data set increases.

3.2.2 Minimum description length (MDL) criterion

In order to overcome the above mentioned drawbacks of the AIC, the MDL criterion is used. MDL stands for Minimum Description Length. The MDL criterion is statistically consistent because $n\ln(N)$ increases faster with 'N' than with 'n' [1].

$$MDL[n] = N\ln(\sigma_n^2) + n\ln(N) \qquad 3.15$$

where N is total no. of observations in a given time series

n is the model order

σ_n^2 is the variance of model order n

3.3 Software implementation of the autoregressive analysis

Sections 3.1 and 3.2 discuss the mathematical algorithm that has to be implemented on a microprocessor for the estimation of the autoregressive parameters of a process signal. In this section, we discuss the software implementation of the algorithm in the 'C' language. The software for the implementation of the autoregressive model is developed in two stages. Initially, the algorithm is implemented in the 'C' language on a PC with floating point representation of the variables. In the second stage, the algorithm is implemented in 'C' with fixed point representation of the variables on CodeWarrior software, an integrated development environment for developing microprocessor applications, and run on Power PC MPC555.

3.3.1 Code structure

There are three modules in the software. The block diagram representing the three modules of the code is given in Figure 3.1.

1. **Timer Module:** The timer of the PowerPC MPC 555 is programmed in such a way that it throws an interrupt at sampling frequency of 100 Hz [12].
2. **QADC Module:** The queued analog-to-digital converter of the PowerPC MPC 555 is programmed to scan the analog signal at the 50th channel of the QADC [12]. Single scan mode of operation is used.

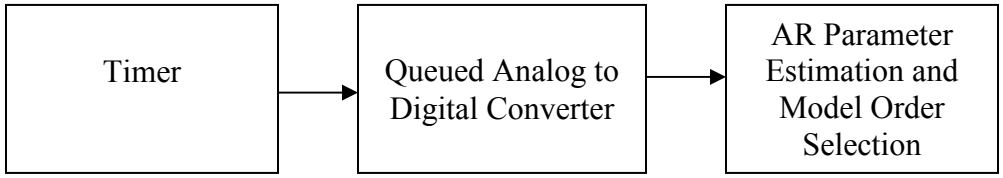

Figure 3.1 Modules of the embedded software

3. **AR parameter Estimation and Model Order Selection:** Once 1024 samples are collected at a sampling frequency of 100 Hz, the block of data is passed to the AR parameter estimation module and the autoregressive parameters are estimated by calculating the autocorrelation functions for different lags. The Yule-Walker linear system of equations is solved using the Yule-Walker method of AR parameter estimation.

The higher model order parameters are estimated recursively using the lower order parameters and the optimal model order is selected based on the Akaike Information Criterion. The noise variance for different model orders is calculated recursively as well as using residual error sequences.

3.3.2 Input and output variables

A time series data sequence to which an autoregressive model needs to be fitted is given as the input to the software. The test data is generated from pressure sensors which are considered to have white noise.

Input variables:

- A block of data containing 1024 time series observations
- Maximum order
- Input channel for the QADC
- Sampling frequency for the PIT timer
- Radix: Number of bits allotted for the decimal par

Output variables:

- Autocorrelation functions with the lag varying from 0 to maximum order
- AR parameters for different model orders
- Noise variance using recursion
- Noise variance using residual error sequence for different model orders
- Akaike Information Criterion functions for different model order.

3.3.3 Flowchart

The flow chart of the source code is given in the Figure 3.2.

Description of the Flowchart:

- Declaration of the variables: The signal is passed through the 50th channel of the QADC of PowerPC MPC 555. The sampling frequency is set to 100 Hz. Single scan mode of operation is used. The input variables are set to their initial values. A block of data containing 1024 samples is read into an array. Radix is set to 24, 8 or 10 based on the precision required. The maximum order is set to 25.
- Mean of the block of data is calculated and subtracted from each sample in order to get rid of the DC components from the time series observations.
- Normalized autocorrelation functions are calculated using the equation 3.2.
- AR parameter estimation: Now the autoregressive parameters are calculated using the following recursive equations 3.16 and 3.17 respectively [3].

$$a_{n+1,n+1} = \frac{R_{n+1} - \sum_{i=1}^{n} a_{n,i} R_{n+1-i}}{R_0 - \sum_{i=1}^{n} a_{n,i} R_i} \qquad 3.16$$

$$a_{n+1,j} = a_{n,j} - a_{n+1,n+1} * a_{n,n-j+1}$$

where j=1,2,3,.....n 3.17

$a_{1,n}, a_{2,n}, ..., a_{n,n}$ are AR parameters of model order n

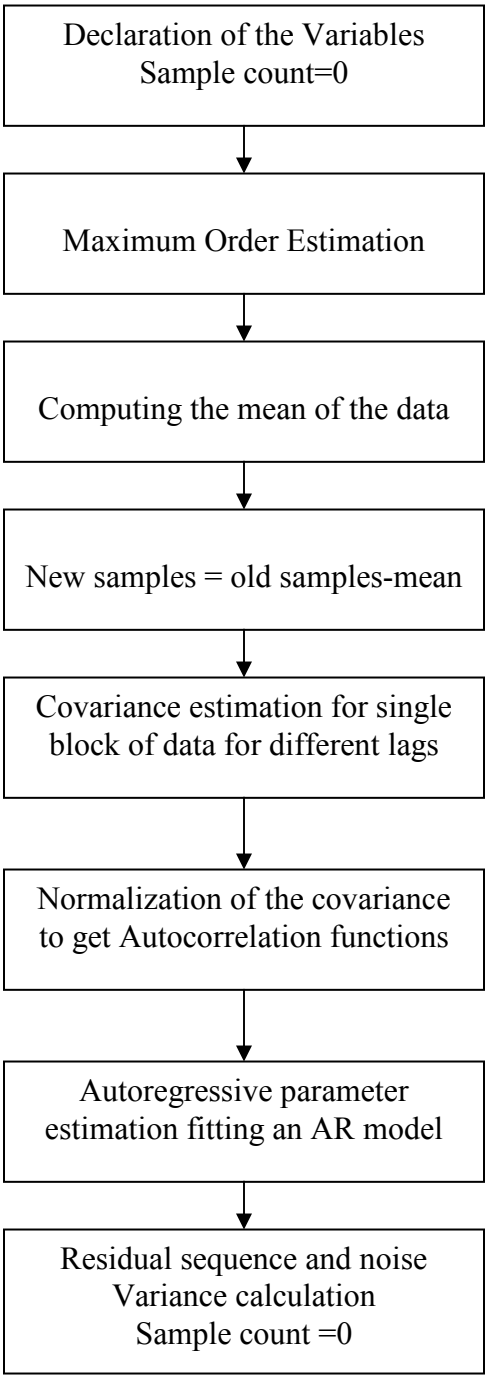

Figure 3.2 Flowchart for the implementation of the AR Model

- The optimal model order is estimated using the AIC function values for different model orders using the formula

$$AIC = N\ln(\sigma^2) + 2n$$

where

N is the number of sample points

n is the AR model order

σ^2 is the prediction error variance

- Noise variance is calculated using the formula.

$\sigma_{n+1}^2 = \sigma_n^2(1 - a_{n+1,n+1}^2)$ where $a_{n+1,n+1} \le 1$

σ_n^2 is variance of model order n

$a_{n,1}, a_{n,2} \ldots\ldots a_{n,n}$ are Autoregressive parameters of model order n

- Residual variance is calculated using the residual error sequence using the formula

$$v(k) = x(k) - \sum_{i=1}^{n} a_i x(k-i)$$

$\hat{\sigma}_v^2 = \frac{1}{N}\sum_{k=1}^{N} \hat{v}_k^2$ where $\hat{\sigma}_v^2$ is the noise variance

$\hat{v}_k^2$ is the residual error sequence for model order k

- Noise variance calculated in both the ways should be almost equal. There should not be much difference between the two values of variance.
- Now, the sample count is again set to zero and the next block of 1024 samples is collected and the AR model is fitted to it for updating the AR parameters.

3.3.4 Arithmetic operations

For embedded processors, the computations are often done using a fixed point representation of the variables. The software for the microprocessor implementation is developed in two stages. Initially, the source code for the autoregressive parameter estimation is developed with floating point representation of the variables on a PC in 'C'. Later, for implementing it on the embedded processor the source code is rewritten with fixed point representation of the variables and the results obtained are cross checked.

A separate library of arithmetic operations – addition, multiplication, division and logarithmic function are implemented using the fixed point integer operations. The next section discusses some of the basic differences between the number representation systems:

1. Floating point representation
2. Fixed point representation.

Floating point numbers:

Floating point representation is defined in IEEE standard 754. The IEEE standard defines both 32-bit single and 64-bit double formats. It also defines extended single precision and extended double precision numbers. Floating point representation is similar to scientific notation with a number multiplied by a base raised to some power. A floating point number consists of three parts: the sign bit, the exponent, and the mantissa. The sign bit is 0 if the number is positive and 1 if the number is negative. The format of floating point numbers is given in the Figure 3.3.

Fixed point numbers:

Fixed point numbers have a fixed radix point. The format of the fixed point numbers is given in Figure 3.4.There are a fixed number of bits to the right of the radix point called fractional bits and a fixed number of bits to the left of the radix point called integer bits. Fixed point arithmetic includes only integer operation. Thus there is no requirement for the additional hardware in the arithmetic logic unit to have floating point unit. Fixed point representation of numbers can produce efficient embedded code when performing mathematically huge operations. The disadvantage of the fixed point numbers is that they have a limited range of values; so fixed point numbers have inaccuracies and these inaccuracies depend upon the number of fractional bits.

The major advantage of using fixed point numbers is high efficiency. The processing speed is high if the application includes only fixed point computations and numbers.

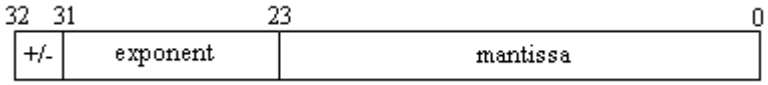

Figure 3.3 IEEE 754 format of 32 bit floating point numbers

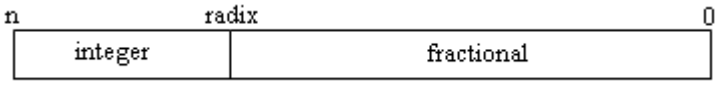

Figure 3.4 Bit format for the fixed point representation

In this research work, the software model for the implementation of autoregressive analysis is developed using fixed point numbers. Though the dynamic range of numbers that can be presented using the floating point numbers is higher than the fixed point numbers, fixed point numbers are preferred in embedded processors as applications developed with fixed point numbers have high efficiency and high processing speed. Most of the DSP applications require specific dynamic range which can be achieved using certain fixed point number representations. In this work 8.24 fixed point representation is used and the precision up to 6th decimal point is achieved.

CHAPTER 4

MICROPROCESSOR IMPLMENTATION

In this chapter, the microprocessor implementation of the autoregressive model for autoregressive parameter estimation of a process signal is discussed. The PowerPC MPC 555 is used for the implementation of the AR model. Various modules of the MPC 555 such as the queued analog to digital converter and timer module are described which are used for the generation digital time series data from analog process signal.

The MPC 555 is interfaced with the PC on which the C source code of the AR model implementation is run using CodeWarrior software which generates the assembly language code for the microprocessor implementation. CodeWarrior is an integrated development tool which provides C/C++ level debugging of the code. It is used to design, create and implement applications for microprocessors.

4.1 Block diagram for the microprocessor implementation of AR model

The hardware required for the microprocessor implementation of the autoregressive model for the estimation of the autoregressive parameters can be seen in Figure 4.1, which shows the block diagram of the microprocessor implementation of the AR model. The basic building blocks for the microprocessor implementation are:

1. PowerPC MPC 555
2. Queued analog to digital converter module of MPC 555
3. Periodic interrupt timer module of MPC 555
4. Pressure sensors
5. Filters and amplifier
6. PC with CodeWarrior software, which provides an integrated development environment for the generation of the code in ‘C’ and translates it into machine-understandable assembly language.

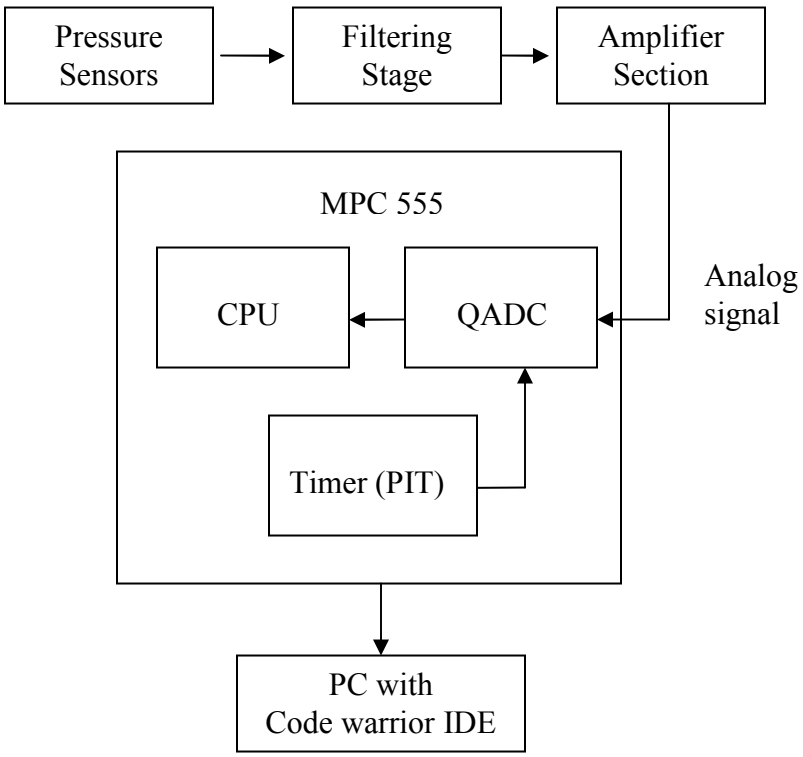

Figure 4.1 Microprocessor implementation of autoregressive analysis

Noise signals generated from the water flow pressure sensors are considered for the autoregressive analysis. The generated noise signals from the pressure sensors are first filtered using Butterworth low pass and high pass filters to get rid of the high frequency components band limiting the overall desirable frequencies. Then the analog signal is fed to the queued analog to digital converter module of the PowerPC MPC 555 for the generation of the digital time series data. The PowerPC is interfaced with the PC with the CodeWarrior software installed. QADC is programmed in such a way that it generates the digital data with desired sampling frequency with the help of Programmable Interrupt Timer (PIT) which is set to throw an interrupt at sampling rate. Then the time series data is fed to the autoregressive parameter estimation software module written in 'C', run on the CodeWarrior software which generates the machine-understandable assembly language code for programming the PowerPC. The PowerPC is interfaced with the PC through BDM connector.

4.2 PowerPC MPC 555

MPC 555 is a RISC processor and a member of Motorola MPC 500 family. It is built with Power PC core technology. Figure 4.2 gives the block diagram of the MPC 555 which represents different modules of it and their organization. The salient features of the MPC 555 processor are listed below [12].

1. PowerPC core with floating-point unit
2. 26 Kbytes fast RAM and 6 Kbytes TPU microcode RAM
3. 448 Kbytes Flash EEPROM with 5V programming
4. 5V I/O system
5. Serial system: queued serial multi-channel module (QSMCM), dual CAN 2.0B controller modules
6. 0-channel timer system: dual time processor units (TPU3), modular I/O system (MIOS1)
7. 32 analog inputs: dual queued analog-to-digital converters (QADC64)
8. Submicron HCMOS (CDR1) technology
9. 272-pin plastic ball grid array (PBGA) packaging
10. 40-MHz operation, -40 °C to 125 °C with dual supply (3.3 V, 5 V) (-55 °C to 125°C for the suffix A device)
11. 32-bit architecture (PowerPC ISA architecture compliant)
12. Core performance measured at 52.7-Kbyte Dhrystones (v2.1) @ 40 MHz
13. Fully static, low power operation
14. Integrated double-precision floating-point unit
15. Precise exception model
16. On–chip watch points and breakpoints.
17. BDM on chip emulation development interface with peripherals or PC.

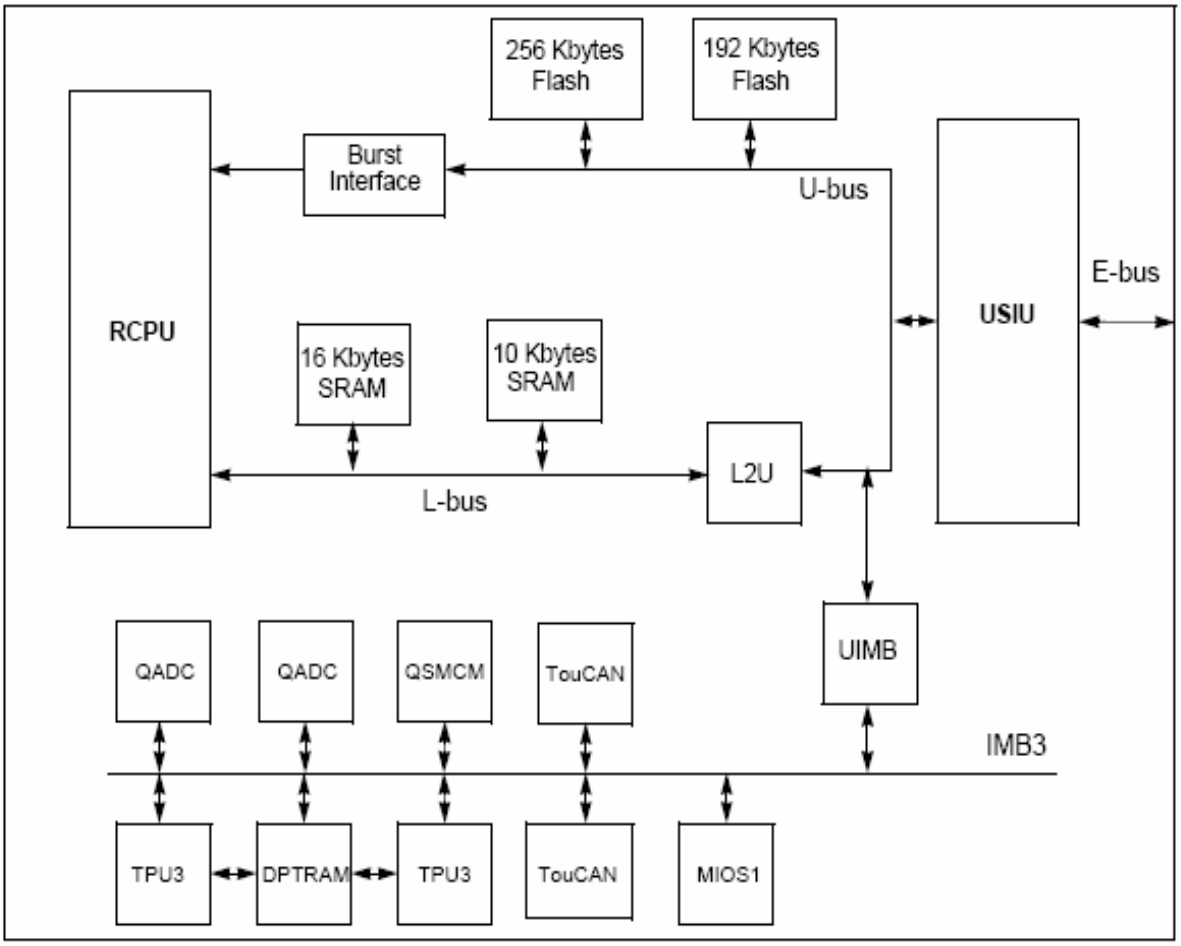

Figure 4.2 Block diagram of MPC 555 (from [12])

MPC 555 has a 32- bit RISC architecture. It works with SRAM, EPROM, FLASH EEPROM and other peripherals. It has an external bus interface with real time clock registers and two types of interrupt timer routines namely decrementer and time base. In this project the module periodic interrupt timer has been used for the sampling the digital data.

4.3 Queued analog to digital converter modules

MPC 555 is provided with 2 queued analog to digital converter modules and each QADC has the following features [12]:

1. 16 analog input channels, using internal multiplexing
2. 41 total input channels, using internal and external multiplexing
3. 10-bit A/D converter with internal sample/hold
4. Typical conversion time of 10 µs (100,000 samples per second)
5. Two conversion command queues of variable length
6. Single-scan or continuous-scan of queues
7. 64 result registers
8. Output data readable in three formats:
    - Right-justified unsigned
    - Left-justified signed
    - Left-justified unsigned
9. Automated queue modes initiated by:
    - External edge trigger/level gate
    - Software command
10. 5V reference and range

The block diagram of the QADC module of MPC 555 is given in Figure 4.3. The QADC consists of

1. Analog front-end
2. Digital control subsystem.

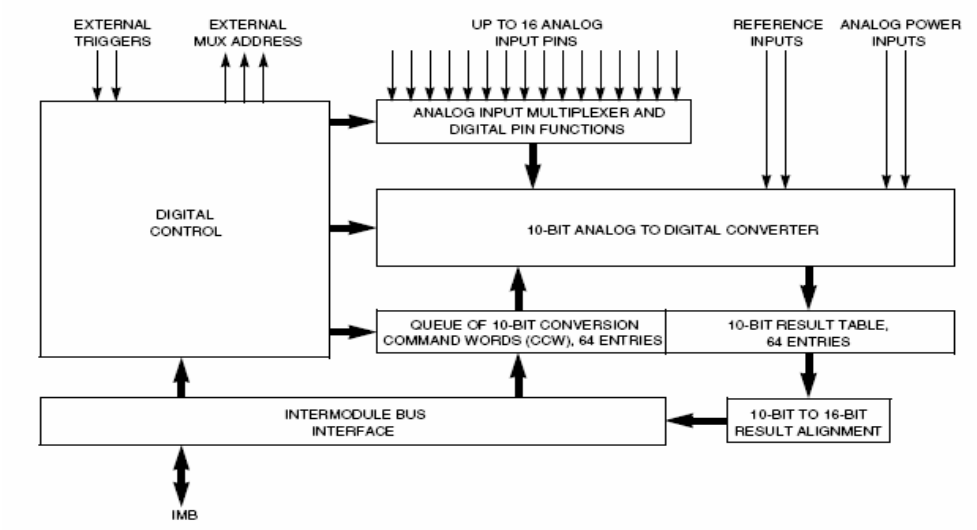

Figure 4.3 Block diagram of QADC (from [12])

It also includes an inter module bus (IMB3) interface block. The analog section includes input pins, channel selection logic, an analog multiplexer, and one sample-and-hold analog circuit. The analog conversion is performed by the digital-to-analog converter (DAC) resistor-capacitor array, a high-gain comparator, and a successive approximation register (SAR).The digital control section contains the conversion sequencing logic. It also includes periodic/interval timer, control and status registers, the conversion command word (CCW) table RAM, and the result word table RAM. The QADC performs 8-bit, 16-bit, and 32-bit data transfers, at both even and odd addresses.

4.3.1 Operation modes

The QADC64 module configuration register (QADC64MCR) defines

1. Freeze mode of operation
2. Stop mode operation
3. Supervisor space access
4. Interrupt arbitration priority.

Stop mode operation:

When the STOP bit in QADC64MCR is set, the clock signal to the A/D converter is disabled and analog to digital conversion analog circuitry is turned off. The STOP bit must be cleared to read results from RAM.

Freeze mode of operation:

The QADC64 enters freeze mode when background debug mode is enabled and a breakpoint is processed.

4.3.2 Analog input channels

The number of analog channels depends on whether or not external multiplexing exists.16 analog channels are present and supported by the internal multiplexing circuitry of the QADC.

4.3.3 Scan modes

The QADC provides several scanning input channels. In single-scan mode, a single pass through a sequence of analog to digital conversions is performed. In continuous-scan mode, multiple passes through a sequence of analog to digital conversions are executed. The different modes present are given below:

1. Disabled and reserved mode
2. Software initiated single-scan mode
3. External trigger single-scan mode
4. External gated single-scan mode
5. Interval timer single-scan mode
6. Software initiated continuous-scan mode
7. External trigger continuous-scan mode
8. External gated continuous-scan mode
9. Interval timer continuous-scan mode.

Software Initiated Single-Scan Mode: Execution of a scan sequence for queue 1 or 2 can be initiated by software by selecting single-scan mode. Upon trigger event, QADC immediately begins execution of the first CCW in the queue. If a pause occurs, another

trigger event occurs and then execution continues without pausing. The QADC then automatically performs the conversions in the queue until an end-of queue condition is encountered. The software initiated single-scan mode is useful for the following:

• Enables software to completely control the queue execution

• Enables software to easily alternate between several queue sequences

In this project single scan mode of scanning sequence of data is followed, which is initialized by the software. The analog signal from the pressure sensors is applied to one of the analog input channels. The samples are collected at a frequency of 100Hz. Once a block of 1024 digital samples are collected, the QADC stops scanning and is paused till the execution of the software module of estimation of the autoregressive parameters. The QADC is enabled once the Periodic Interrupt Timer throws a level zero interrupt for the QADC to perform the conversion.

4.4 Periodic interrupt timer

The periodic interrupt timer has16-bit counter clocked by the PITRCLK clock supplied by the clock module of the processor. The 16-bit counter counts down till zero when provided with a value from the PITC. When the timer reaches zero value, the PS bit is set and an interrupt is generated. The software should be able to read the PS bit and change it to zero to stop the interrupt. At the next input clock edge, the value in the PITC is again loaded into the counter and the process continues with the divider reset and the counter begins counting again. If the PS bit is not cleared, an interrupt request is generated. The request remains pending until PS is cleared. If the PITC value is changed, the counting is stopped and the count resumes with the new value in PITC. If the PTE bit is not set, the PIT is unable to count and retains the old count value. The block diagram of the PIT is given by the Figure 4.4.

The timeout period is calculated as given by the equation 4.1 [12].

$$PIT_{period} = \frac{PITC+1}{F_{pitrtclk}} = \frac{PITC+1}{\{\frac{External\ clock}{4\ or\ 256}\}} \qquad 4.1$$

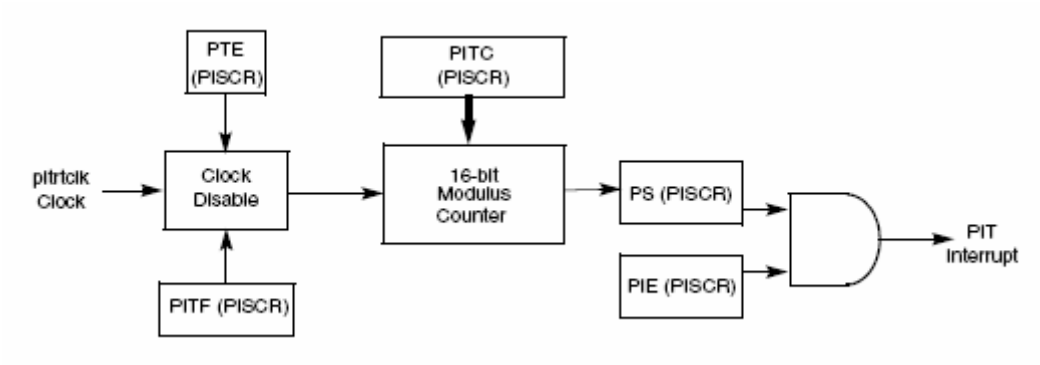

Figure 4.4 Block diagram of PIT (from [12])

For example solving the above equation using 4 MHz external clock and a pre divider of 256 gives:

$$PIT_{period} = \frac{PITC + 1}{15625} \qquad 4.2$$

4.5 Pressure Sensors and Filters

The QADC of the MPC 555 microprocessor is interfaced with the pressure sensors through filters. The analog signal from the pressure sensors are first passed through the Butterworth low pass filter to set a lower cut off frequency and then passed through Butterworth high pass filter to get rid of the all high frequency components as the noise signal is present only in the low frequency components of the signal. The filter signal is then amplified (20-dB amplification).

Till this chapter design and development of an embedded system for the implementation of autoregressive analysis is discussed. In the next chapters verification and testing of the developed prototype model of the embedded system for implementing AR analysis is focused.

CHAPTER 5

TWO TANK FLOW CONTROL LOOP EXPERIMENT

In this chapter we discuss about the test data generation and describe the apparatus used for the generation of the data. The experimental flow control loop has been used to develop multivariate control algorithms that are being applied to a space reactor system. For the collection of test data we mainly concentrate on acquiring the noise signals generated using pressure sensors during the flow control.

5.1 General description

The 2-Tank loop is built on a wheeled table-like seven-foot long, four-foot wide and six-foot high steel frame structure. This structure holds all sensors, piping, pump, sump tank, and aircraft aluminum table top, cables, control valves, manual valves and two tanks and can be easily moved around.

5.2 Component description

Tanks:

For the level control there are 2 similar acrylic tanks installed on this loop called Tank 1 and Tank 2, respectively, and their dimensions are: 5-3/4" in diameter and 3-foot long and a 27-gallon stainless steel tank is also installed underneath the table top to provide the necessary water for the circuit. Figure 5.1 shows a detail of the 2 acrylic tanks used in this project.

Sensors:

There are 11 sensors installed in the 2 tank loop - 4 differential pressure sensors, 4 thermocouples and 3 turbine meters. In addition to this, there are 2 primary orifice flow meters.

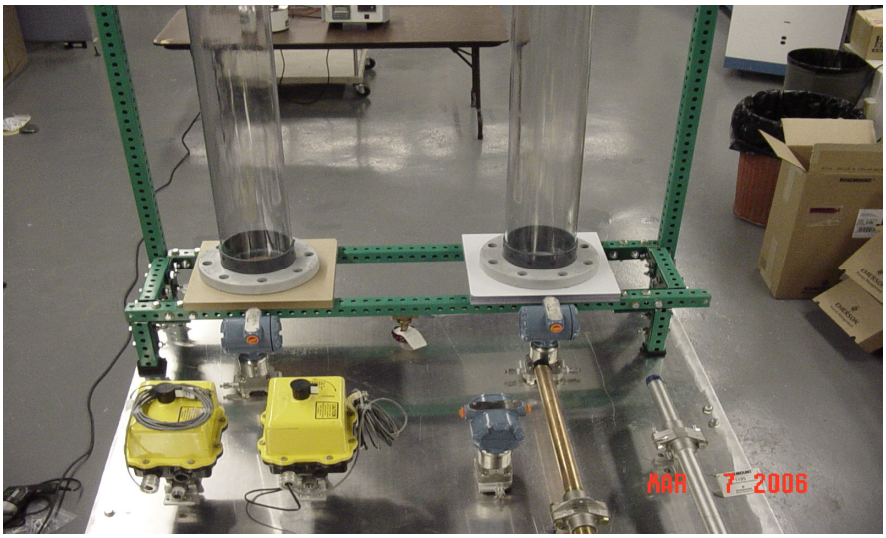

Figure 5.1 Level tanks details

Pressure Transmitters:

Four Rosemount differential pressure sensors are currently installed. Two of them are used to measure the water level in each tank. The two other pressure sensors are connected to orifice meters and are used to measure the water flow going into each of the level tanks. Information about the pressure sensors is given in Table 5.1 and Figure 5.2 gives details of two sensors. To calibrate the pressure sensors range, a software package called AMS Suite from Emerson® was used. This software package can provide us an easy way to calibrate, zero-trim, schedule maintenance and keep record of each and every calibration performed. It comes with an RS-232 modem and cable to plug into the computer serial port. On the other end, a pair of probes is used to connect the computer to the sensor terminals.

Flow meters:

Two different types of flow rate sensors are used in the loop: turbine and orifice meters. The orifice meters were provided by Rosemount™ and the turbines were manufactured by Omega™. Also, there are 3 turbines installed in the loop: one at each tank outlet and one in the bypass and all of them are factory calibrated. Two primary orifice plates are installed at the tank inlets to measure the inlet flow rate.

Table 5.1 Pressure Sensors Information

Sensor ID	Function	Output	Calibration Range
Tank 1	Measures Level in Tank 1	4 – 20mA	0 – 900 mmH_2O
Tank 2	Measures Level in Tank 2	1 – 5 Volts	0 – 900 mmH_2O
Flow 1	Tank 1 Inlet flow	4 – 20mA	0 – 6165 mmH_2O
Flow 2	Tank 2 Inlet flow	4 – 20mA	0 – 6303 mmH_2O

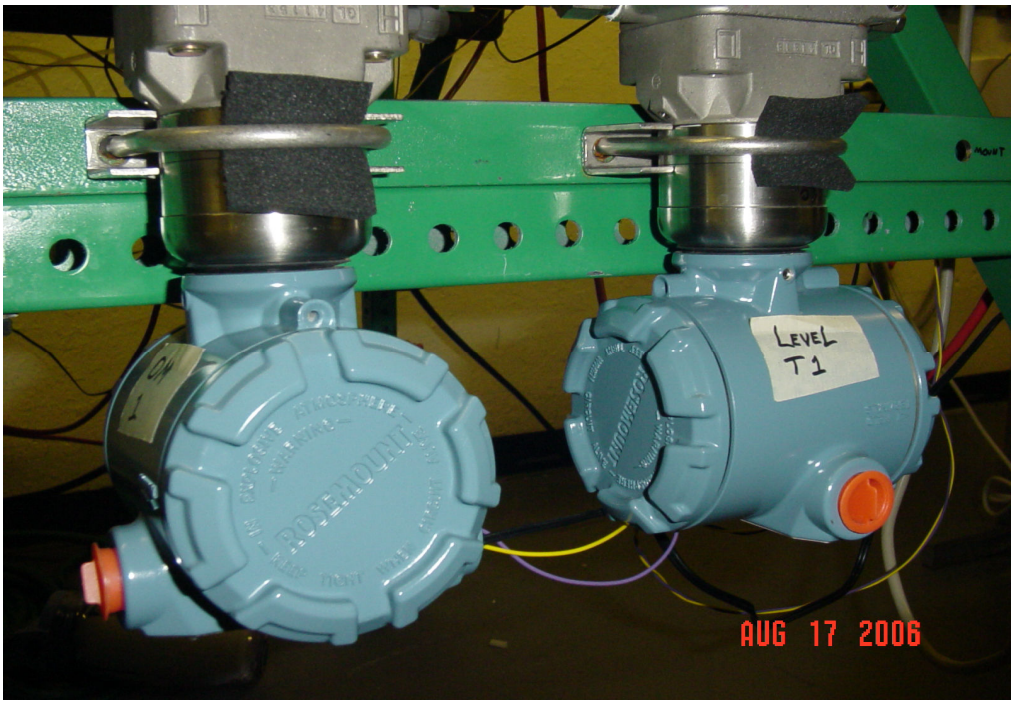

Figure 5.2 Pressure sensors used in the control loop

Thermocouples:

There are four type K thermocouples installed to monitor the temperature. These sensors are not calibrated; instead a standard calibration curve is being used.

Data Acquisition Software:

The data acquisition is conducted in two ways.

Method 1:

Data acquisition for the two tank loop for autonomous control experiment is written in Visual Basics software. The noisy digital data from the pressure sensors is collected using the data acquisition software. The graphical user interface for collecting the digital data is given below in the Figure 5.3. The graphical interface allows entering the required parameters for the data collection like sampling rate, cut off frequencies of low pass and the high pass filters.

Method 2:

First noise signal is filtered using the low pass and high pass Butterworth filters and then amplified. The amplified signal is then fed to the Analog to digital converter of the Power PC 555. The ADC of the PowerPC 555 is programmed in such a way that it reads the digital samples at the given sampling frequency.

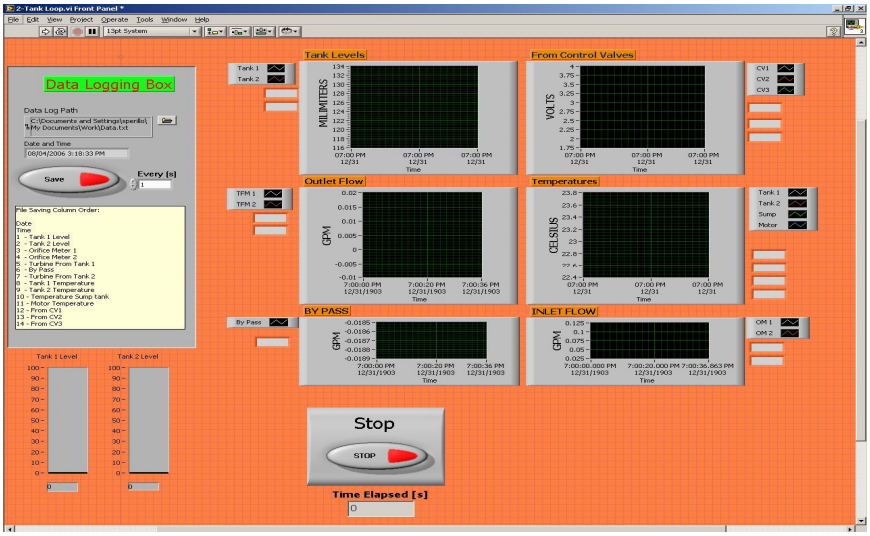

Figure 5.3 GUI of data acquisition software (from [11])

CHAPTER 6

RESULTS

In this chapter we discuss about the results obtained by running the software for estimating the autoregressive parameters on the microprocessor and analysis the parameters such as power spectral density and impulse response. Initially autoregressive parameters are estimated using the floating-point computations, the results of which are taken as the reference for the estimation of the parameters using fixed-point computations. The software is implemented with fixed numbers because embedded processors can perform only integer operations.

Autoregressive model (AR) is developed and Tested in three stages. They are:

1. Floating point implementation of the AR model on a PC and testing with the predefined data model.
2. Fixed point implementation of the AR model on a PC and testing it with the synthetic data generated using oscilloscope.
3. Fixed point implementation of the AR model on a PowerPC and testing it with a data generated from pressure sensors.

6.1 Test data generation

The test data for testing the developed autoregressive (AR) model is generated in three different ways.

Initially for the floating point implementation of the AR model on a PC, the test data is generated using a predefined 10th order model using MATLAB. The block diagram for the test data generation using MATLAB is given in Figure 6.1. The MATLAB code for the generation of the test data is given in Appendix B. The results obtained by testing the AR model using the data generated using the predefined 10th order AR model are considered to the desired results for the fixed point implementation of the model.

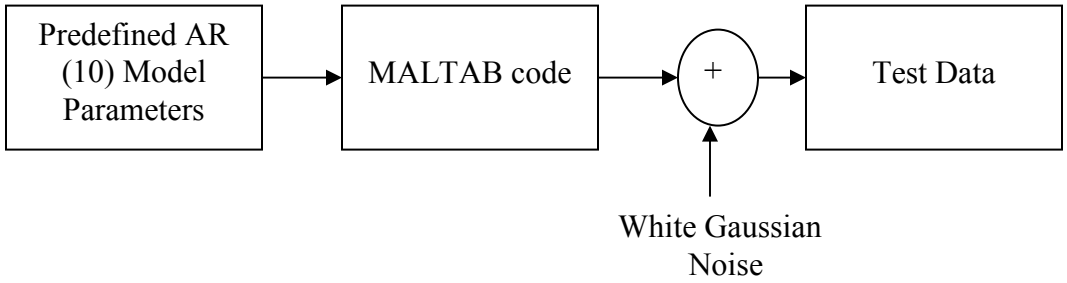

Figure 6.1 Test data generation using MATLAB

In the second stage the autoregressive (AR) model is testing using the synthetic data generated using oscilloscope and function generator. In this stage fixed point implementation of the AR model is considered and is implemented on a PowerPC. A sine wave signal with added noise is generated at sampling frequency of 100 Hz for testing the model. Analog to digital converter model and the timer modules of PowerPC are testing by giving different sampling frequencies.

In the third stage the AR model is testing using the data generated using real pressure sensors as explained in the Chapter 5. In this stage fixed point implementation of the AR model is considered and is implemented on a PowerPC. The AR parameters estimated in this stage have accuracy to the 6th decimal point with reference to the floating point results in the first stage. 8.24 fixed point representation format is considering in this stage.

The results presented in this chapter are obtained by fitting an autoregressive model to the following listed test data:

1. A test data set generated using predefined AR model of order 10 as given below:

$$\begin{aligned} x(t) = {} & 1.15620x(t-1) - 0.60582x(t-2) + 0.69749x(t-3) - 0.32119x(t-4) \\ & +0.26957x(t-5) - 0.17467x(t-6) + 0.09010x(t-7) - 0.09005x(t-8) \\ & +0.01647x(t-9) - 0.05458x(t-10) \end{aligned}$$

Plots of the generated test data using predefined 10 order model and pressure sensors are given in Figure 6.2 and 6.3 respectively.

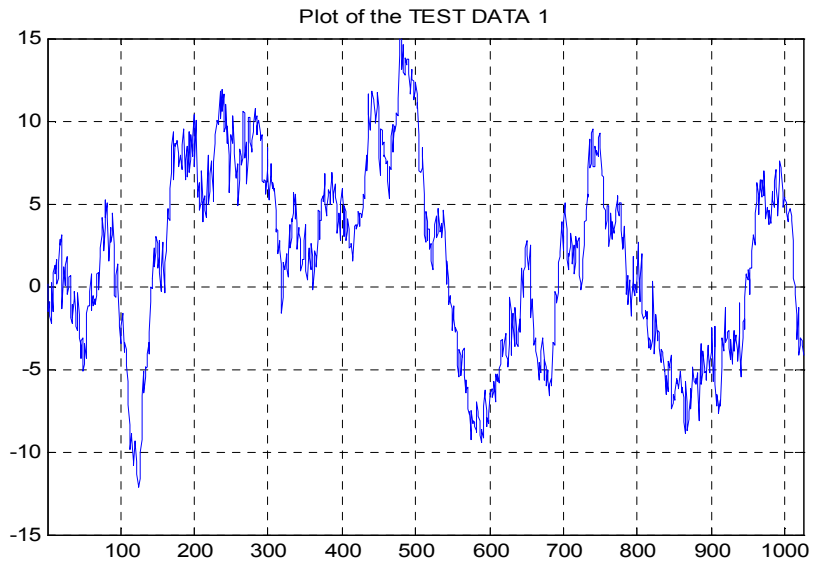

Figure 6.2 Plot of the test data generated using a 10th order AR model

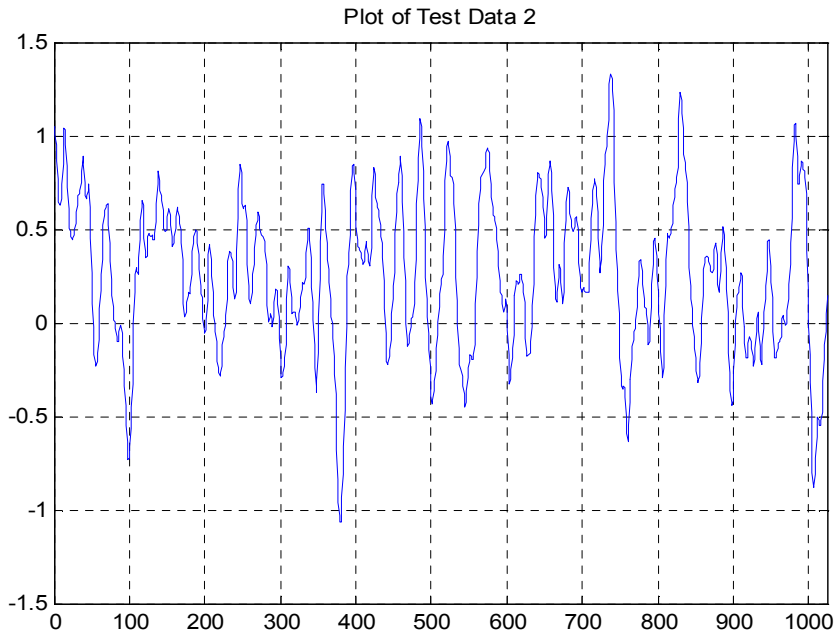

Figure 6.3 Plot of the test data generated using pressure sensors

2. A test data set of 1024 points generated at a sampling frequency of 30 Hz, lower cutoff frequency of 0.01 Hz and upper cutoff frequency of 10 Hz.

6.2 Estimated parameters using floating point numbers

Using Test data 1:

Total number of samples: 10000

Maximum order: 25. Generally up to 5 % of the total data set is considered to be the maximum model order for which the AR parameters are estimated recursively.

The auto correlation sequence generated using the test data set 1 on a PC is:

0.981312	0.980456	0.958669	0.950477	0.939282	0.922813
0.904767	0.886751	0.867483	0.846133	0.824165	0.802691
0.780979	0.759306	0.738271	0.717586	0.696871	0.675628
0.655006	0.634519	0.614240	0.595468	0.577337	0.558824
0.540262	0.522255				

Figure 6.4 gives the plot of autocorrelation sequence against lag. The optimal model order is estimated based upon the AIC functions. The model order, for which AIC function is minimized, is considered to be the optimal model order. Figure 6.5 gives the plot of AIC against model order. From Figure 6.5 we consider the optimal model order to be 10. The plot of the variance verse model order is given by the Figure 6.6. We observe from the variance plot that the variance decreases as the model order increases and around the optimal model order it tends to be nearly constant.

$$P_{xx}^{YW}(f) = \frac{\hat{\sigma}_{wp}^2}{\left|1 + \sum_{k=1}^{p} \hat{a}_p(k) e^{-j2\pi fk}\right|^2} \qquad 6.1$$

$$where \quad \hat{\sigma}_{wp}^2 = \hat{E}_p^f = r_{xx}(0) \prod_{k=1}^{p} [1 - |\hat{a}_k(k)|^2]$$

p is the model order

$r_{xx}(0)$ is the autocorrelation function with lag' 0 '

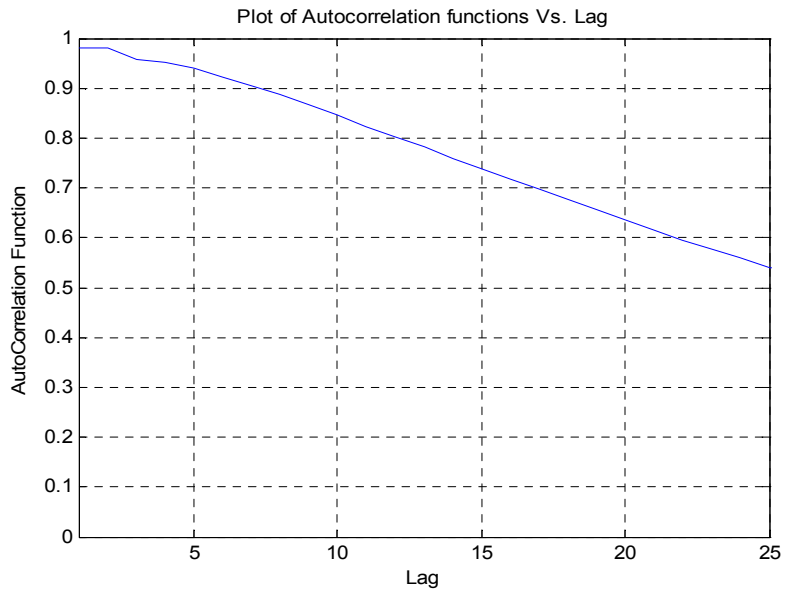

Figure 6.4 Plot of autocorrelation function Vs lag

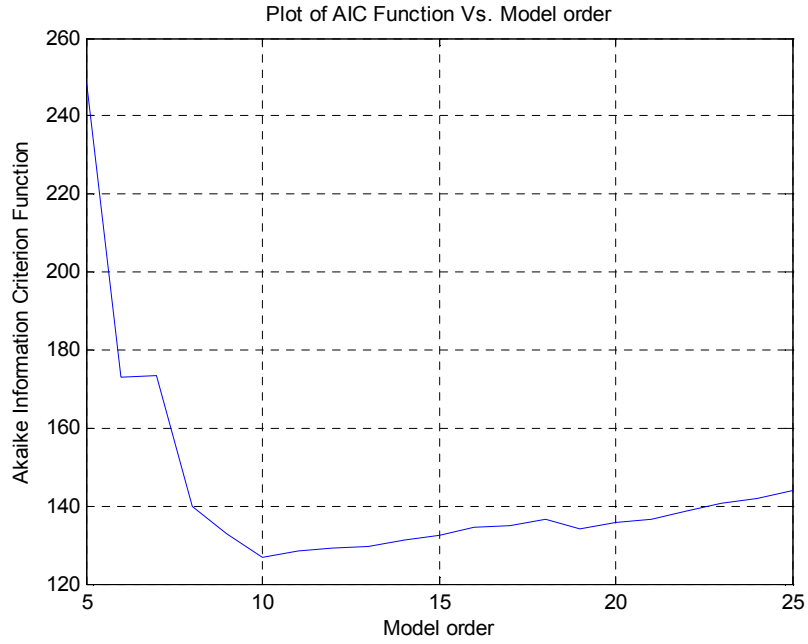

Figure 6.5 Plot of Akaike information criterion function vs. model order

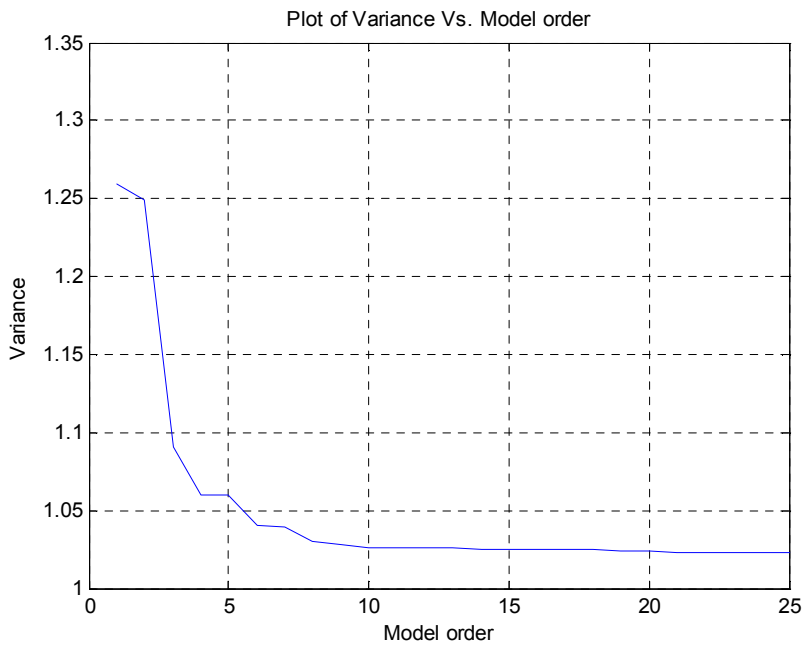

Figure 6.6 Plot of variance Vs model order

Estimated AR parameters for model order 10:

1.157715 -0.591488 0.630775 -0.271905 0.220209 -0.152097 0.090882 -0.065840 0.005977 -0.045113

Original AR parameters of the model through which the time series data is generated:

1.15620 -0.60582 0.69749 -0.32119 0.26957 -0.17467 0.09010 -0.09005 0.01647 -0.05458

The power spectral density (PSD) for different model orders is evaluated, and it is observed that the PSD for model orders greater than 10 is similar. Figures 6.7, 6.8, and 6.9, 6.10 give the power spectral density vs. frequency plot for model orders 9, 10, 11, and 12, respectively. The PSD for a particular model order is given by the equation 6.1.

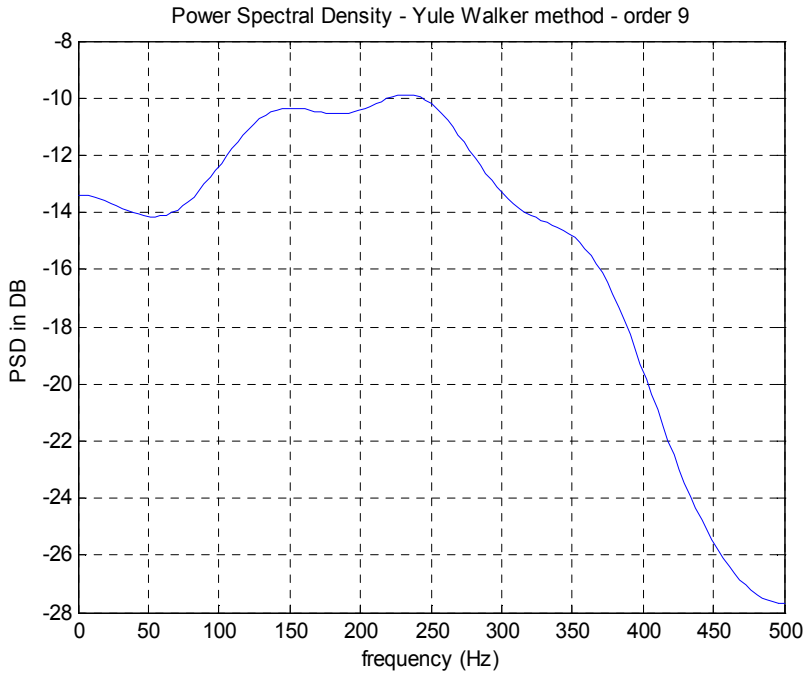

Figure 6.7: Power spectral density plot for model order 9

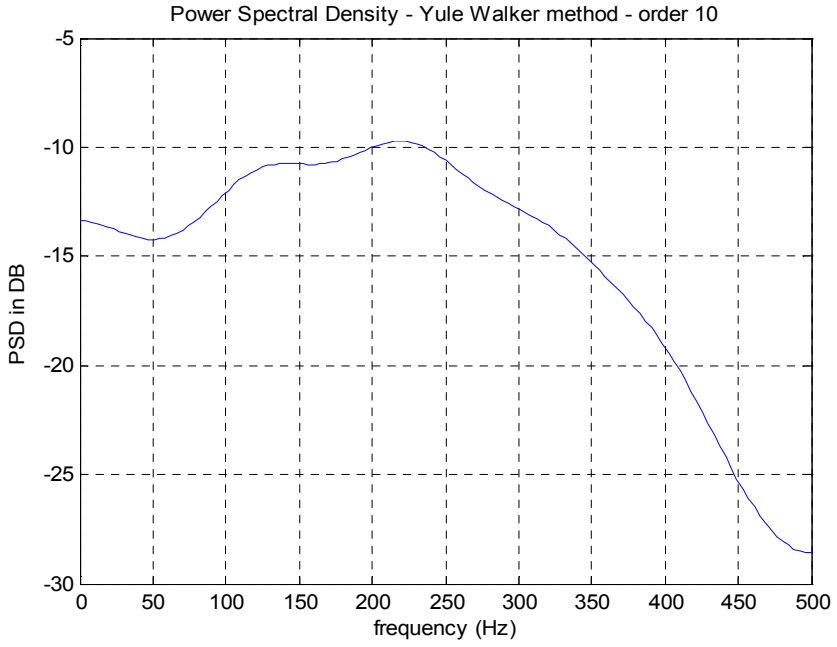

Figure 6.8 Power spectral density plot for model order 10

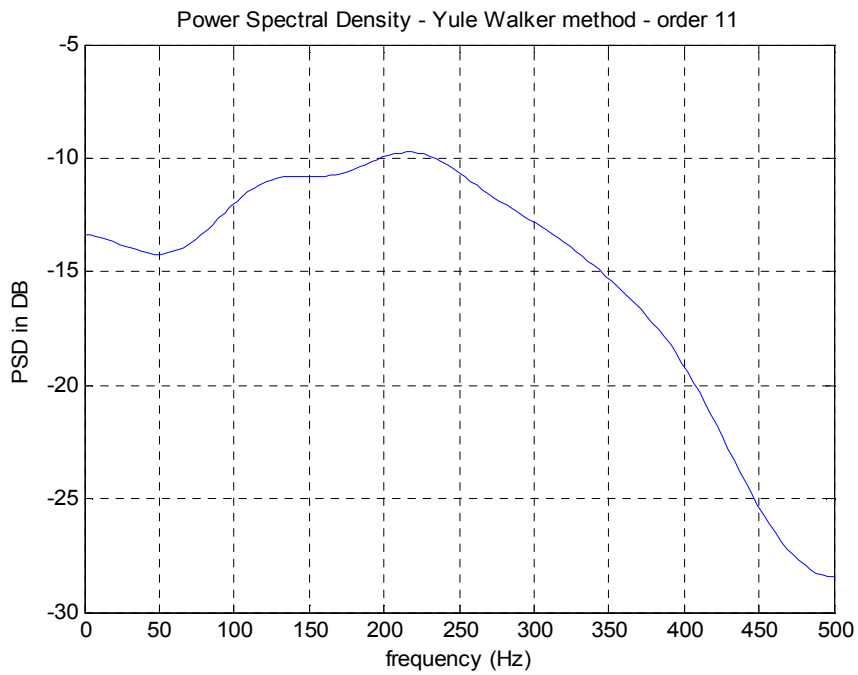

Figure 6.9 Power spectral density plot for model order 11

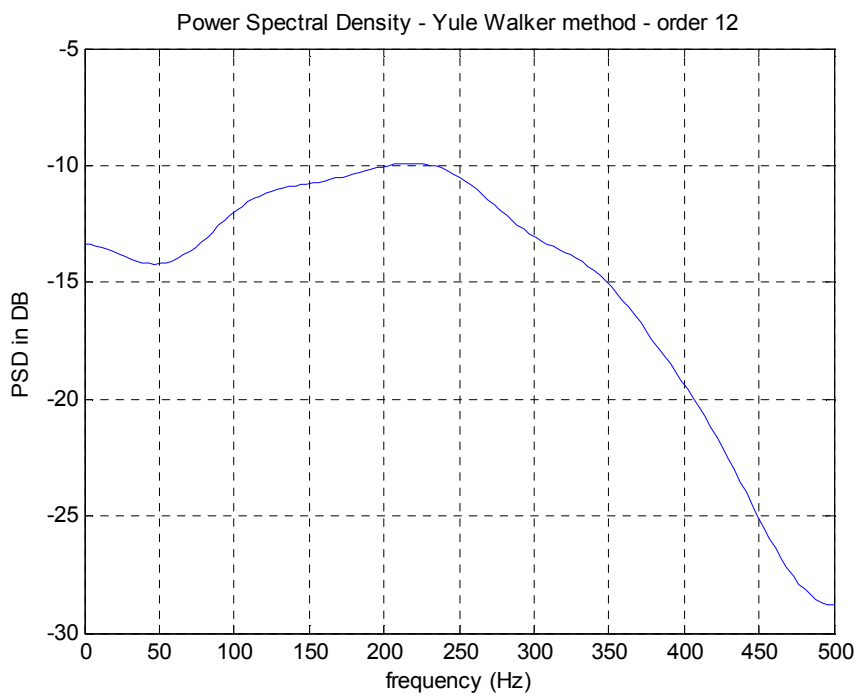

Figure 6.10 Power spectral density plot for model order 12

Using Test Data Set 2:

Specifications:

Total number of samples: 1024

A block of 1024 is considered because it is easy to perform block computations with fixed point representation of numbers.

Maximum order: 25.

The auto correlation sequence generated using the test data set 2 on a PC is:

0.981422	0.932467	0.860238	0.773216	0.678796	0.582349
0.487326	0.395781	0.308957	0.227691	0.152753	0.084818
0.024445	-0.027907	-0.071861	-0.107471	-0.135231	-0.155876
-0.170158	-0.178809	-0.182488	-0.181846	-0.177589	-0.170541
-0.161593					

Figure 6.11 gives the plot of autocorrelation sequence against lag. The model order is considered to be ‘8’ as the AIC function is minimum for that particular model order. The plot of the variance verse model order is given by the Figure 6.12. We observe from the variance plot that the variance decreases as the model order increases and around the optimal model order it tends to be nearly constant.

The power spectral density (PSD) for different model orders is evaluated, and it is observed that the PSD generated for model orders 8,9, and 10 is almost similar and there is little difference in the PSD for model orders greater than 8. Figures 6.13 and 6.14 give power spectral density vs. frequency plot for model orders 8, and 6 respectively. The power spectral density is calculated using Equation 6.1.

Estimated AR parameters for model order 8:

1.817602	-0.729018	-0.262774	0.067808	0.094208	0.019240
-0.011128	-0.027561				

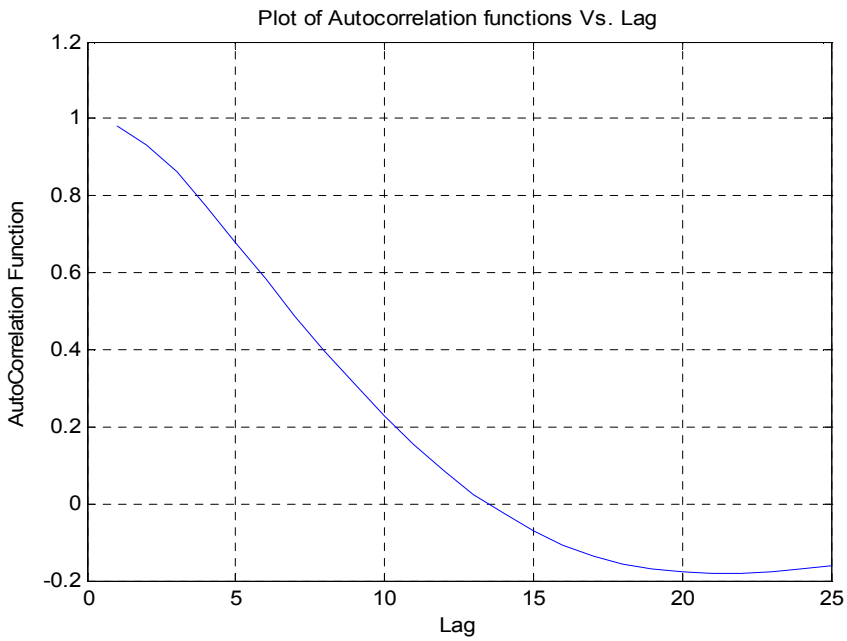

Figure 6.11: Plot of autocorrelation function Vs Lag

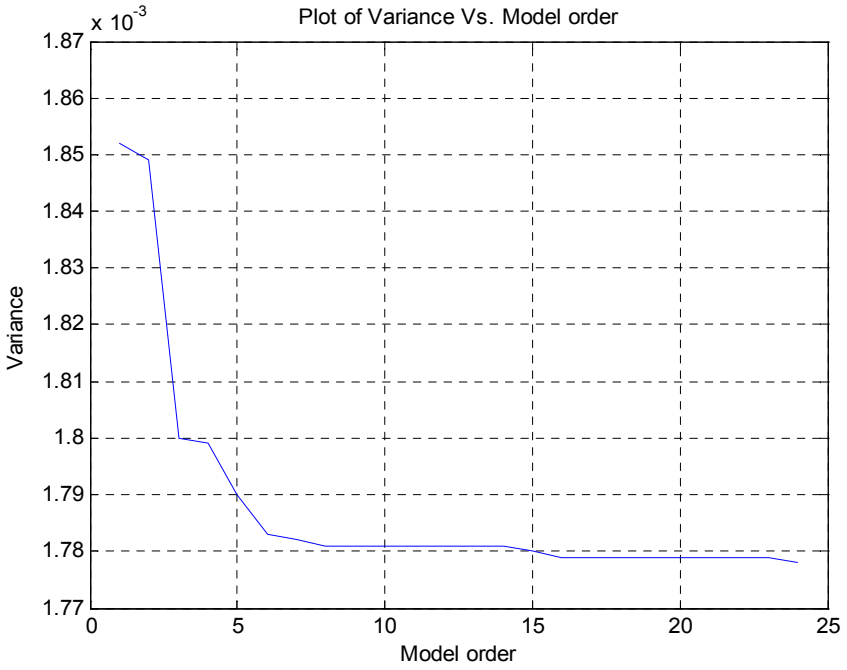

Figure 6.12 Plot of variance Vs model order

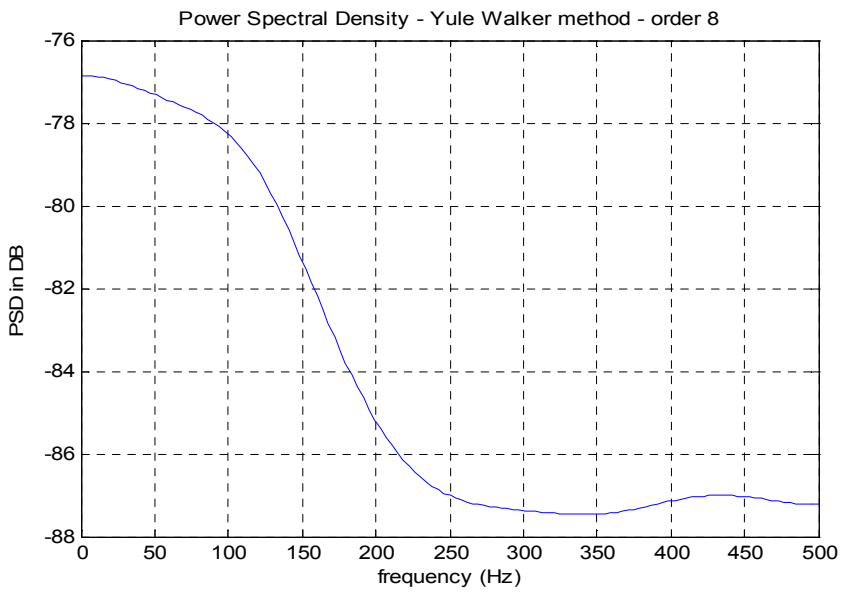

Figure 6.13 Power spectral density plot for model order 8

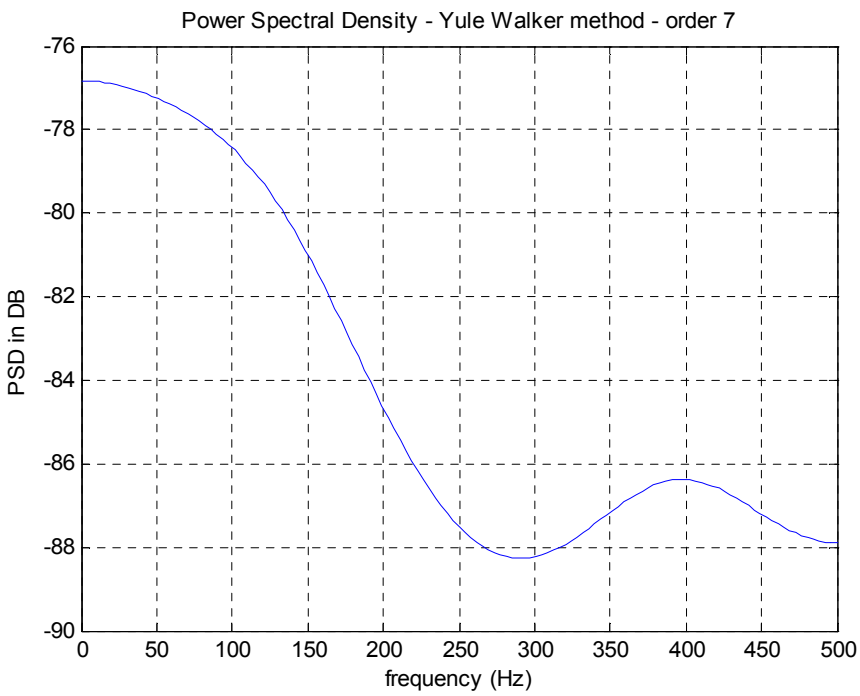

Figure 6.14 Power spectral density plot for model order 6

6.3 Estimated parameters using fixed point numbers on PC

The software generated for the AR parameter estimation consists of macros, which converts the floating-point numbers to fixed-point numbers. The complete data set, which is in the floating-point representation, is converted to fixed point using the macros and the autoregressive parameters are estimated on the PC. The new estimated autocorrelation sequence differs from autocorrelation sequence estimated using floating-point numbers by an error margin of 0.0001 due to the overflow conditions.

Using Test data-1

Specifications:

Total number of samples: 1024

Maximum order: 25. Generally up to 5 % of the total data set is considered to be the maximum model order for which the AR parameters are estimated recursively.

Optimal model order: 10

Estimated AR parameters for model order 10 with fixed point numbers:

(a) In fixed point representation:

18487870	-7961432	8775699	-2981324	2284315	-1747619
953901	-790422	64131	-691830		

(b) In floating point representation:

1.083820	-0.544189	0.645291	-0.274969	0.198357	-0.101958
0.052591	0.018342	-0.066263	-0.029109		

Estimated AR parameters for model order 10 with floating point numbers:

1.157715	-0.591488	0.630775	-0.271905	0.220209	-0.152097
0.090882	-0.065840	0.005977	-0.045113		

Original AR parameters of the model through which the time series data is generated:

1.15620	-0.60582	0.69749	-0.32119	0.26957	-0.17467
0.09010	-0.09005	0.01647	-0.05458		

The estimated autoregressive parameters vary from the original parameters with error margin of 0.001. The graphical plots generated for PSD, noise variance and AIC are almost similar to plots generated in Section 6.1.

Using Test Data Set 2:

Specifications:

Total number of samples: 1024

Maximum order: 25.

Optimal model order: 8

Estimated AR parameters for model order 8 with fixed point numbers:

(a) In fixed point representation:

30490661 -12183795 -4533045 1278319 1517592 291335 -126921 -487727

(b) In floating point representation:

1.817385 -0.726211 -0.270191 0.076194 0.090456 0.017365 -0.007565 -0.029071

Estimated AR parameters for model order 8 with floating point numbers:

1.817602 -0.729018 -0.262774 0.067808 0.094208 0.019240 -0.011128 -0.027561

The estimated autoregressive parameters vary from the original parameters with error margin of 0.001. The graphical plots generated for PSD, noise variance and AIC are similar to plots generated in the Section 6.1.

6.4 Estimated parameters using fixed-point numbers on PowerPC MPC 555

For the implementation of the algorithm on a PowerPC, the block size is restricted to 1024 because of the memory constraints. The internal flash memory of the PowerPC board is 448 Kbytes and the SRAM is 26 Kbytes. The analog sensor signals are converted to digital signals in QADC module of PowerPC and the time series data is generated at a sampling frequency of 10 Hz. A block of 1024 samples is considered for the block computations. Noise variance is calculated using recursive formula and also by calculating the residual sequence.

Using Test data 1

Specifications:

Total number of samples: 1024

Maximum order: 25

The auto correlation sequence generated using the test data set 1 on a Power PC MPC 555 with fixed-point representation of numbers is given in Figure 6.15 and Figure 6.16 gives the generated autoregressive parameters generated.

We observe that there is a slight variation between the autocorrelation sequence and AR parameters generated by implementing the algorithm on a Power PC and the autocorrelation sequence and AR parameters generated on the PC because of the redefined logarithmic function and the overflow conditions. The error is in the range of 0.0001. The noise variance values and AIC values for different model orders are found to be similar to the values generated in Section 6.2.

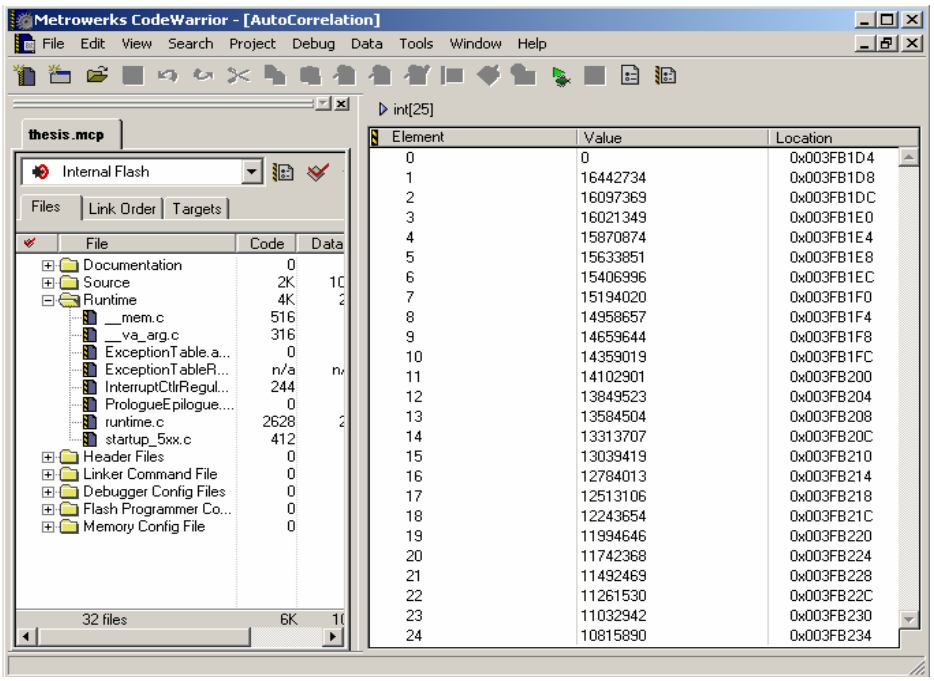

Figure 6.15 Screenshot showing the autocorrelation sequence for predefined AR model

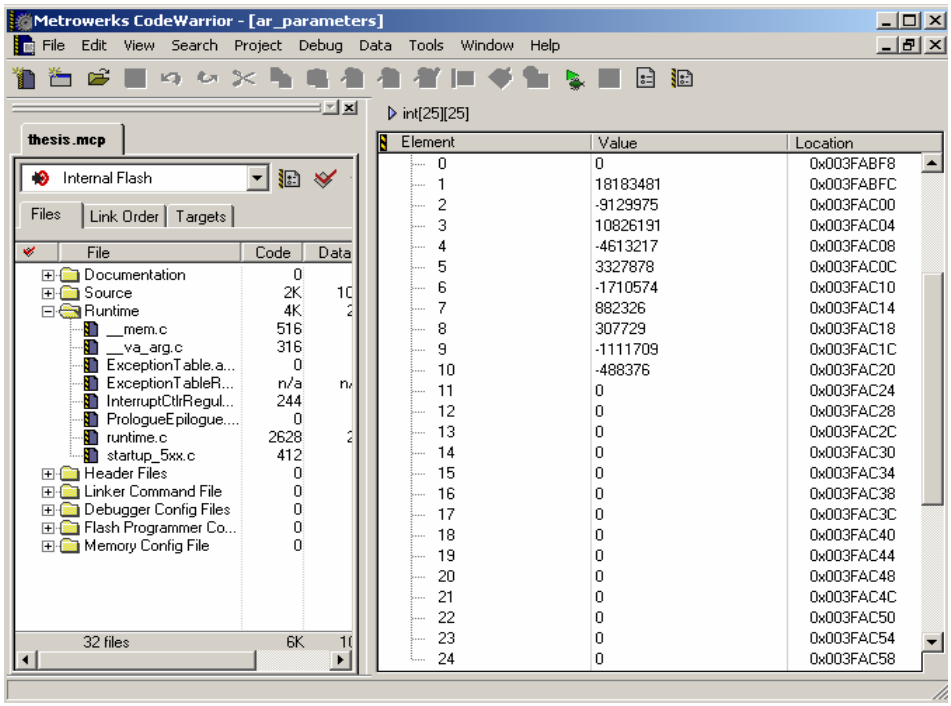

Figure 6.16 Screenshot showing the AR parameters of order ‘10 ’

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

In the previous chapter the results of the autoregressive implementation of the process sensor signals on a microprocessor are presented. The results produced are quite promising and are according to the given specifications. This chapter outlines the conclusions and presents future work of this research.

7.1 Conclusions

The following conclusions are drawn from the results of the research work.

1. Estimation of the autoregressive parameters and model order selection is done with a precision up to 4th decimal point with 8.24 fixed point representation.
2. Implementation of the Autoregressive analysis using fixed point implementation increases the speed of the block computations and reduces the power consumption.
3. The microprocessor implementation of the AR analysis is constrained by the memory limitation of the board and better precision of the results, multiple block computations can be achieved with increased memory.
4. Autoregressive analysis of process sensor signals involves fewer computations when compared to moving average analysis and mixed autoregressive and moving average analysis.

7.2 Future Work

Future scope of the research work includes

1. Implementation of the Autoregressive analysis on 8, 16 bit processors like ARM processors, etc. This can result in lesser power consumption than the 32 bit processor.
2. AR analysis can be extended to the multivariate noise signals which are vectors depending on multiple parameters.

3. Power spectral density and other dynamic behavior parameter estimation such as impulse response, step response etc can be done for the analysis noise signals.

REFERENCES

[1] S.L. Marple, Jr., *Digital Spectral Analysis with Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[2] G.E.P. Box and G.M. Jenkins, *Time Series Analysis: Forecasting and Control,* Holden-Day, San Francisco, 1970.

[3] B.R. Upadhyaya and T.W. Kerlin, "Estimation of Response Time Characteristics of Platinum Resistance Thermometers by the Noise Analysis Technique," ISA Transactions, Vol. 17, No. 4, pp. 21-38, 1978.

[4] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing – Principles, Algorithms, and Applications,* Third edition, Macmillan Coll Div, 1992.

[5] M. J. Roberts, Probability and Random Variables (course notes).

[6] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, N.J., 1975.

[7] H. Schildt, C: the complete Reference, 4th Edition, McGraw – Hill, 2000.

[8] E. Balaguruswamy, Programming in ANSI C, Second Edition, McGraw Hill, 1994.

[9] W. Stallings, Computer organization and Architecture, Designing for Performance, Sixth edition, Prentice Hall, 2003.

[10] J. L. Hennessy & D. A. Patterson, Computer Architecture: A Quantitative Approach, Third Edition, Morgan Kaufmann, 2003.

[11] Thesis and DOE report attached: B.R. Upadhyaya et al., "Autonomous Control of Space Reactor Systems", Annual Report, November 2006.

[12] http://www.freescale.com

APPENDICES

A – SOURCE CODE

```
/*********************************************************************
Name of code: Autoregressive Analysis

Purpose of code: To fit a model to process sensor signals and to
estimate the autoregressive parameters and optimal model order of the
model.

Author of code: Swetha Priyanka Pakala

Developed under the guidance of Gregory D. Peterson
at The University of Tennessee in the Tennessee Advanced
Computing Laboratory.

Copyright (C) 2006 Swetha Priyanka Pakala and Gregory D. Peterson

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to

Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA  02110-1301 USA.

You may also view the GNU Lesser General Public License at
http://www.gnu.org/licenses/lgpl.html

For additional information or queries, send email to gdp@utk.edu.
*************************************************************/
/* Variables:
   'Samples' is the two dimensional array consisting of the time
    series   data with white noise

   'blocksize' is the block size

   'number_blocks' is number of blocks

   'max_order' gives the maximum order possible for the model

   'mean' is a one dimensional array consisting of the avg mean value
    of each and every block

   'covariance' is the two dimensional array consisting of the
    covariance values
```

```
   'initial_autocor' is the intial Autocorrelation

   'AutoCorrelation' is the one dimensional array consisting of the
    Autocorrelations of lag varying from 1 to max_order

   'ar_parameters' is the two dimensional array consisting of AR
    parameters like a11,a21,a22,a31,a32,a33 etc

   'variance' is the one dimensional array consisting of variance value
    of each and every block

   'aic' is the one dimensional array consisting of the AIC values of
    different orders

*********************************************************************/

#include "mpc555.h"
#include <stdio.h>
#include <math.h>
#include "fdlibm.h"

// declaring and initializing global variables

#define blocksize 1024
#define radix 24
int sampleCount = 0;
int mySamples[blocksize];
int pitctr = 0; // declaring and initializing a global variable
int samples[1][1024],ar_parameters[25][25]={0},initial_autocor=0,
AutoCorrelation[25]={0},temp1=0,numerator=0,denominator=0,residualError
[1024]={0},residualVariance=0;

// Declaration of the Functions
int numsum(int n);
int densum(int n);
int mul(int a,int b);
int divide(int u,int v);
int natural_logarithm(int u);
int taylorlog(int z);
void compute_AR_parameters();


void init555()
{
  USIU.SYPCR.B.SWE = 0; // Disable Watchdog timer
  USIU.PLPRCR.B.MF = 0x009; // Set 40 MHz system clock for 4MHz crystal
  UIMB.UMCR.B.HSPEED = 0;     // Set the IMB to run at full clock speed
}

void initPIT()
{
// SPECIFIC INITIALIZATIONS FOR PIT:
  USIU.PITC.B.PITC = 0x619; // Load desired count value
  USIU.PISCR.B.PITF = 1; // Freeze enabled to stop PIT
  USIU.PISCR.B.PTE = 1; // Enable PIT to start counting
```

```
      USIU.PISCR.B.PIRQ = 0x80; // Assign PIT interrupt to Level 0
   USIU.PISCR.B.PIE = 1;      // Enable PIT interrupt
   USIU.SIMASK.R = 0x40000000; // Enable level 0 interrupt
}


initQADC_A()
{     // SPECIFIC INITIALIZATIONS FOR QADC-A:

   USIU.PDMCR.B.PRDS = 1;     // Disable pullups on QADC pins
   QADC_A.QADC64MCR.R = 0x0080;// Configure QADC module for defaults:
   QADC_A.QACR0.R = 0x0013;   // Stop mode disabled, SUPV mode on
   QADC_A.QACR1.B.MQ1 = 0;    // Module is master, int clk = IMBCLK
   return 0;
}

LoadCCW_A()
{

   QADC_A.CCW[0].R = 0x0032;        // Convert CH 50, 2 QCLKs
   QADC_A.CCW[1].R = 0xFF;          // End conversion command
   return 0;

}

// MAIN FUNCTION

void main()
{
      int i=0;
      init555(); // Perform minimal chip initialization
      initPIT();  // Initialize PIT
      asm(" mtspr EIE, r0");  // Enable interrupts
      initQADC_A();      // Initialize QADC A
      LoadCCW_A();
      while (1)
      {
            if(sampleCount >= 1024)
            {
                  compute_AR_parameters();
                  sampleCount=0;
            }

            i++;
      }       // loop forever
}

Convert_A()
{
 QADC_A.QACR1.R = 0x2100; // Trigger Queue 1 using s/w
                          // single scan mode
      return 0;
}
```

```
void level_0()
{
      int shiftedSample;
      USIU.PISCR.B.PS = 1; // clear the interrupt
      pitctr++; // update global variable
      Convert_A();  // Perform conversions
      mySamples[sampleCount] = QADC_A.RJURR[0].R;
      shiftedSample = mySamples[sampleCount]<<2;
      mySamples[sampleCount] = mySamples[sampleCount]+shiftedSample;
      mySamples[sampleCount] = mySamples[sampleCount]<<14;
      sampleCount++;
}
// DETERMINATION OF THE AUTO REGRESSION PARAMTERS AND THE MODEL ORDER
ESTIMATION

void compute_AR_parameters()
{
 // Declaration of the local variables
  int samples_count,max_order,number_blocks,blockCount=0,lag=0,
      order=0,k=0,shifted_1_byradix=0;
  int mean[25]={0},normalisationfactor[25]={0};
  int covariance[2][25]={0},variance[25]={0},aic[25]={0};

  for(samples_count=6;samples_count<1024;samples_count++)
  {
      for(k=1;k<=5;k++)
      {
      temp1=temp1+mul(ar_parameters[5][k],samples[0][samples_count+k]);
      }
      residualError[samples_count]=samples[0][samples_count]-temp1;
      temp1=0;
  }
  for(samples_count=6;samples_count<1024;samples_count++)
  {
  temp1=mul(residualError[samples_count],residualError[samples_count]);
  temp1=temp1/blocksize;
  residualVariance=residualVariance+temp1;
  }
  //reading the samples
  max_order=25;
  for(samples_count=0;samples_count<1024;samples_count++)
  {
      samples[blockCount][samples_count]=mySamples[samples_count];
  }
      /*Reading the time series data into the array 's[i][j]'
    where 'i' is the no. of blocks and 'j' is the block size*/

  for(samples_count=0;samples_count<1024;samples_count++)
  {
     temp1=samples[blockCount][samples_count]/blocksize;
     mean[blockCount]=mean[blockCount]+temp1;
  }
  number_blocks=1;
  shifted_1_byradix=1<<24;
```

```
  for(blockCount=0;blockCount<number_blocks;blockCount++)
 {
 //Calcuation of the mean of the block
 for(samples_count=0;samples_count<blocksize;samples_count++)
 {
samples[blockCount][samples_count]=samples[blockCount][samples_count]-
mean[blockCount];// Subtracting the mean from all the samples
temp1=samples[blockCount][samples_count]/blocksize;
temp1=mul(temp1,samples[blockCount][samples_count]);
// Calculation of c0

normalisationfactor[blockCount]=normalisationfactor[blockCount]+temp1;
}
// Calculation of the Covariance values of each block
   for(order=1;order<=max_order;order++)
   {
   for(lag=0;lag<=(blocksize-order-1);lag++)
   {
      temp1=samples[blockCount][lag]/blocksize;
      temp1=mul(temp1,samples[blockCount][lag+order]);
      covariance[blockCount][order]=covariance[blockCount][order]+temp;
   }
covariance[blockCount][order]=divide(covariance[blockCount][order],norm
alisationfactor[blockCount]);
// Calculation of normalised auto correlation functions from lag 1 to
max_order
   }
}
    // final normalised autocorrelation functions with lags from 1 to
max_order
for(lag=1;lag<=max_order;lag++)
{
for(blockCount=0;blockCount<=number_blocks;blockCount++)
{
AutoCorrelation[lag]=AutoCorrelation[lag]+covariance[blockCount][lag];
}
AutoCorrelation[lag] = AutoCorrelation[lag]/number_blocks;
}
// calculating the final  R0 value averaged of all the blocks
for(blockCount=0;blockCount<=number_blocks;blockCount++)
{
initial_autocor = initial_autocor
normalisationfactor[blockCount]/number_blocks;
}
// initial a11 parameter
ar_parameters[1][1]=AutoCorrelation[1];
/****AR paramemter calculation  for the orders from 1 to max_order****/
/************************MODEL RECURSIVE ALGORITHM********************/
/* FORMULA:

                 a[n+1][i]=a[n][i]-a[n+1][n+1]*a[n][n-i+1]
                             where i= 1,2,.......n
a[n+1][n+1]={R[n+1]- summation{a[n][i]*R[n+1-i]}}/{rin
summation{a[n][i]*R[i]}}
                  where i= 1,2,......n
```

```
          Where a[n+1][i] is the ith AR parameter of the model whose
order is n+1*/

    // Determination of the a[n+1][n+1] parameters of different orders
      for(order=1;order<max_order-1;order++)
      {
      numerator=AutoCorrelation[order+1]-numsum(order);
      denominator=shifted_1_byradix-densum(order);
      temp1=divide(shifted_1_byradix,denominator);
      ar_parameters[order+1][order+1] = mul(numerator,temp1);

      // Determination of a[n+1][i] parameters
      for(k=1;k<=order;k++)
      {
      ar_parameters[order+1][k] = ar_parameters[order][k] -
mul(ar_parameters[order+1][order+1],ar_parameters[order][order+1-k]);
      }
      }

    /******** Calculation of the initial Varience sigma2********/

    temp1=mul(ar_parameters[1][1],ar_parameters[1][1]);
      temp1=(shifted_1_byradix-temp1);
    variance[1]=mul(temp1,initial_autocor);

      /**** Calculation of the sigma2 values from 1 to max_order*****/
      for(order=1;order<max_order;order++)
      {

      temp1=mul(ar_parameters[order+1][order+1],ar_parameters[order+1][
            order+1]);
      temp1=shifted_1_byradix-temp1;
      variance[order+1]=mul(variance[order],temp1);
      }
      // calculation of the AIC values from 1 to max_order
      for(order=1;order<=max_order;order++)
      {
      temp1=natural_logarithm(variance[order]);
      aic[order]=((blocksize*temp1)+2*order*shifted_1_byradix);
    }
    temp1=0;
    sampleCount=0;
}

/* Defining the Function numsum which determines

       Summation{a[n][i]*R[n+1-i]}
                        where i=1,2....n.           */

int numsum(int n)
{
int i;         // i is the index for different order
int y = 0;      // y is the summation
for(i=1;i<=n;i++)
```

```
{
y = y + mul(ar_parameters[n][i],AutoCorrelation[n+1-i]);
}
return y;
}

/* Defining the Function densum which determines

       Summation{a[n][i]*R[i]}
                        where i=1,2....n.           */

int densum(int n)
{
  int i=0;                // i is the index for different orders
  int x = 0;              // x being the summation
  for(i=1;i<=n;i++)
  {
      x = x + mul(ar_parameters[n][i],AutoCorrelation[i]);
  }
  return x;
}

// Multiplication of two Fixed point numbers u and v
int mul(int u,int v)
{

     int result=0;
     long long answer=0;
     answer=(long long)u*(long long)v;
     result=(int)(answer>> radix);
     return(result);    // result is the product of u and v
}

// Division of two fixed point numbers x and y

int divide(int x,int y)

{
     int z=0;
     long long xl=0,zl=0;
     xl = (long long)x << radix; // x1 is shifted version of x by radix
     if (x < 0)
        xl = xl & (-1 << radix);
        zl = xl/(long long)y;
        z = (int)zl;            //z is the quotient in fixed point
        return(z);
}

// To determine natural logarithm

int natural_logarithm(int b)
{
   int y=0,z=0;
   y=1024*1024*16;
   z = (int)((taylorlog(b)-24*taylorlog(2))*y);
```

```
   return(z);
}

/*DETERMINATION OF THE NATURAL LOGARITHM OF A FIXED POINT NUMBER IN
8.24 FIXED POINT REPRESENTATION*/

int taylorlog(fixed_number)
{
int fixed_eps=0.001,fixed_result,count,y,square_y,factor,temp1,temp2,
    shifted1,number_terms;
if(number<=0)
  fixed_result=0;
  else
  {
     if(number==1)
     fixed_result=0;
     else
     {
         shifted1=1<<radix;
         /**** y= (x-1)/(x+1)*****/
         temp2=fixed_number+shifted1;
         y=divide(fixed_number,temp2);
         temp1=divide(shifted1,temp2);
         y=y-temp1;
         count=3;
  /******calculating multiplying factor (x-1)^2/(x+1)^2******/
         square_y=mul(y,y);
         factor=mul(y,square_y);
         temp1=count*shifted1;
         fixed_result=y+divide(factor,temp1);
         temp1=fixed_result-y;
         if (temp1<0)
         temp1=-temp1;
         number_terms=2;
         while(temp1>fixed_eps)
         {
              y=fixed_result;
              count=count+2;
              factor=mul(factor,square_y);
              temp2=count*shifted1;
              fixed_result=y+divide(factor,temp2);
              temp1=fixed_result-y;
              number_terms=number_terms+1;
          }
      }
      temp2=shifted1*2;
      fixed_result=mul(temp2,fixed_result);
      return(fixed_result);
  }
}
```

B – MATLAB CODE FOR TEST DATA GENERATION

```
% x(t)=1.15620*x(t-1)-0.60582*x(t-2)+0.69749*x(t-3)-0.32119*x(t-4)+0.26957*x(t-5)-
0.17467*x(t-6)+0.09010*x(t-7)-0.09005*x(t-8)+0.01647*x(t-9)-0.05458*x(t-10)+a(t)

clear all;

N=1024;
% Generation of Random Numbers
x(1) = randn(1,1);
x(2) = randn(1,1);
x(3) = randn(1,1);
x(4) = randn(1,1);
x(5) = randn(1,1);
x(6) = randn(1,1);
x(7) = randn(1,1);
x(8) = randn(1,1);
x(9) = randn(1,1);
x(10) = randn(1,1);

% Opening the files
fid = fopen('ar10(1).txt','w');
fid1 = fopen('ar10(2).txt','w');

% Test data Generation using the predefined autoregressive parameters

for k=11:(N+10)
   s=randn(1,1);
   x(k)=1.15620*x(k-1)-0.60582*x(k-2)+0.69749*x(k-3)-0.32119*x(k-4)+0.26957*x(k-
5)-0.17467*x(k-6)+0.09010*x(k-7)-0.09005*x(k-8)+0.01647*x(k-9)-0.05458*x(k-10)+s;

   fprintf(fid,'%f,',x(k));
   fprintf(fid1,'%f\t',x(k));
end;
status = fclose(fid);
```

C – RESULTS

Estimation of AR parameters using floating point numbers on PC

Lag	Autocorrelation Sequence	Lag	Autocorrelation Sequence
1	0.981312	1	0.98142
2	0.980456	2	0.93247
3	0.958669	3	0.86024
4	0.950477	4	0.77322
5	0.939282	5	0.6788
6	0.922813	6	0.58235
7	0.904767	7	0.48733
8	0.886751	8	0.39578
9	0.867483	9	0.30896
10	0.846133	10	0.22769
11	0.82417	11	0.15275
12	0.802691	12	0.08482
13	0.780979	13	0.02445
14	0.759306	14	-0.0279
15	0.738271	15	-0.0719
16	0.717586	16	-0.1075
17	0.696871	17	-0.1352
18	0.675628	18	-0.1559
19	0.655006	19	-0.1702
20	0.634519	20	-0.1788
21	0.61424	21	-0.1825
22	0.595468	22	-0.1818
23	0.577337	23	-0.1776
24	0.558824	24	-0.1705
25	0.54026	25	-0.1616

Estimation of AR parameters using fixed point numbers on PC

Lag	Autocorrelation Sequence	Lag	Autocorrelation Sequence
1	16442734	1	0.98006
2	16097369	2	0.95948
3	16021349	3	0.95495
4	15870874	4	0.94598
5	15633851	5	0.93185
6	15406996	6	0.91833
7	15194020	7	0.90563
8	14958657	8	0.89161
9	14659644	9	0.87378
10	14359019	10	0.85586
11	14102901	11	0.8406
12	13849523	12	0.8255
13	13584504	13	0.8097
14	13313707	14	0.79356
15	13039419	15	0.77721
16	12784013	16	0.76199
17	12513106	17	0.74584
18	12243654	18	0.72978
19	11994646	19	0.71494
20	11742368	20	0.6999
21	11492469	21	0.68501
22	11261530	22	0.67124
23	11032942	23	0.65762
24	10815890	24	0.64468
25	10578309	25	0.63052

Lag	Autocorrelation Sequence	Lag	Autocorrelation Sequence
1	16465517	1	0.981421
2	15644097	2	0.932461
3	14431955	3	0.860212
4	12971763	4	0.773177
5	11387334	5	0.678738
6	9768914	6	0.582273
7	8174474	7	0.487237
8	6638306	8	0.395674
9	5181312	9	0.30883
10	3817665	10	0.227551
11	2560153	11	0.152597
12	1420180	12	0.084649
13	407092	13	0.024265
14	-471367	14	-0.028096
15	-1208973	15	-0.07206
16	-1806660	16	-0.107685
17	-2272500	17	-0.135452
18	-2618877	18	-0.156097
19	-2858547	19	-0.170383
20	-3003764	20	-0.179038
21	-3065475	21	-0.182717
22	-3054638	22	-0.182071
23	-2983085	23	-0.177806
24	-2864810	24	-0.170756
25	-2714699	25	-0.161809

VITA

Swetha Priyanka Pakala was born on 10th December, 1983 in the city of Hyderabad, India. She went to Little Flower High School, Hyderabad, India for high school. Swetha did her undergraduate coursework at Sreenidhi Institute of Science and Technology, Hyderabad, India with Electronics and Communication Engineering as major. She developed a great interest for digital design during her undergraduate coursework and came to United States of America in 2005 to pursue higher studies. She joined University of Tennessee in 2005 and graduated with a Master of Science Degree in Electrical Engineering in December 2006.