DISTURBANCE REJECTION AND EMBEDDED NETWORKING FOR A CLASS OF COMMERCIAL DC- DC POWER CONVERTERS

MADHURA SHALIGRAM

Bachelor of Engineering in Electronics and Telecommunication

University of Pune, India

August, 2001

submitted in partial fulfillment of requirements for the degree

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

at the

CLEVELAND STATE UNIVERSITY

May, 2006

This dissertation has been approved
for the Department of Electrical and Computer Engineering
and the College of Graduate Studies by

_____

Dissertation Committee Chairperson, Dr. Zhiqiang Gao

_____

Department/Date

_____

Dr. Yongjian Fu

_____

Department/Date

_____

Dr. Nigamanth Sridhar

_____

Department/Date

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my advisor, Dr. Zhiqiang Gao, for his guidance, patience, inspiration and never-ending support during my research work. Working in Advanced Engineering Research Lab (AERL) has been a great learning experience for me. I would also like to thank Mr. Jack Zeller and Mr. Jack Redilla for their help throughout my research work.

I appreciate Dr. Yongjian Fu and Dr. Nigamanth Sridhar for being on my committee and for taking time to evaluate this thesis.

Special thanks go to Aaron Radke, Michael Gray and Bosheng Sun for their valuable help during this research work. I would also like acknowledge my lab mates and friends, Zhan Ping, Ivan Jurcic, Robert Miklosovic, Frank Goforth, Radhika Kotina and Classica Jain, for their support during my dissertation. I would like to express my gratitude to the electrical engineering department, especially Adrienne Fox and Jan Basch.

Finally, I would especially like to thank my husband, Shantanu, our families and our loco parents Professor Brody and Mrs. Brody for their encouragement and support throughout these years.

DISTURBANCE REJECTION AND EMBEDDED NETWORKING FOR A CLASS OF COMMERCIAL DC- DC POWER CONVERTERS

MADHURA SHALIGRAM

ABSTRACT

In this research, advanced control and communication technique is investigated for the purpose of reliable, efficient and well-regulated DC to DC power conversion. The concept of active disturbance rejection is applied to a commercial DC-DC power converter regulation problem which results in significant improvement in voltage regulation over the existing proportional-integral (PI) control method. An embedded web technology has been utilized to develop a control area network (CAN) to Ethernet gateway which enables real time parametric configuration, online tuning, and fault diagnosis and accommodation of a power converter on a local CAN network. Control and communication hardware and software were successfully implemented using Motorola 56F8323 and Motorola MCF5282 microcontroller. Remote maintenance, control and configuration of a DC-DC power converter on a CAN network is facilitated with the development of Internet enabled interactive graphical user interface using a Web browser.

TABLE OF CONTENTS

Page

NOMENCLATURE

GUI: Graphical user interface

PMAD: Power management and distribution system

CAN: Control area network

HTTP: Hyper text transfer protocol

HTML: Hyper text markup language

IP: Internet protocol

TCP: Transmission control protocol

UDP: User datagram protocol

RTOS: Real time operating system

API: Application program interface

MIME: Multipurpose Internet mail extensions

CGI: Command gateway interface

MCU: Micro controller unit

ISR: Interrupt service routine

CSMA/CD: Carrier sense multiple access/ Collision detection

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER I

INTRODUCTION

With increasing use of DC-DC power converters in modern applications, there has been a significant improvement in their performance over the years. Next generation products from most of the converter manufacturers are almost certain to feature a wide input voltage range, extensive output trim capabilities, better dynamic performance, a degree of built-in intelligence and communications facilities to address a wide range of power management issues. Communication, command and control advancements in the development of power converters are byproducts of the increasing system complexity. In this chapter, current trends in the power converter market are presented followed by a discussion of motivation for this research, problem formulation, and thesis organization.

1.1 Motivation for Research

Over the last decade, several salient technology drivers have caused a transformation in the design and manufacturing of the DC-DC power converters. A principal aim of most of the leading DC-DC converter manufacturers is to achieve accurate output voltage, excellent load regulation, fast disturbance rejection, limited short circuit current, very low noise, and very low electromagnetic interference (EMI). Another emergent trend for next generation power converters is to incorporate in the power converter embedded advanced power management functions such as built-in intelligence, communication capabilities, and remote configuration [1].

Control of DC-DC power converters has always been a challenging problem for control engineers. In the past, commercial DC-DC converters have been predominantly controlled by analog integrated circuit technology and linear system design techniques. But due to the time-varying and switching nature of the converter, its dynamics are highly nonlinear and difficult to deal with. Because of the susceptibility to noise, EMI, and the parasitic part of the converter components, performance of a practical converter deviates from the theoretical prediction making the disturbance rejection extremely important. It is difficult to obtain satisfactory control results using classical control theories or model-based modern control theory. Researchers have been investigating applications of several nonlinear control laws to control DC-DC power converters, but most of these control schemes require either accurate mathematical model or are too complex to develop and implement [5-17]. That is why very few of these control schemes have been implemented on commercially available power converters. Complexity and

cost involved in the implementation of these schemes are important consideration factors in case of commercial power converters. In recent years, with the rapid development of advanced high-speed digital circuits, digital controllers are gradually replacing the analog ones [2,3]. Digital control offers several possibilities such as low power dissipation, immunity to analog component variations, and most importantly compatibility with the other digital systems. Communication functions can be embedded quite easily in a digital system and control structures and system parameters can be changed by modifying only the software. These features of digital control make modern control systems easily realizable and implementation of the complex control strategies becomes readily possible. Digital communication capability embedded in an individual power converter allows not only "reconfigurable control" but also "distributed intelligence" at the system level.

With the embedded digital communication channel developed among power converters and a supervisory probe, system information such as voltage, current, temperature, etc. can be exchanged at modular as well as system level. For supervisory mode of communication, Internet is seen as the most cost-effective way for remotely monitoring, controlling, and updating firmware of the power devices [19]. With the availability of embedded Internet enabling technology at low cost, the idea of Internet-ready power converters can be made commercially viable.

1.2 Literature Review

The proportional-integral-derivative (PID) control algorithm has been widely used in many industrial power devices due to its simple structure and easy but effective tuning. Despite of its popularity, long term practical experience has shown that the PID technology itself has certain limitations and shortages. Application of several nonlinear control laws including fuzzy logic, neural networks, genetic algorithms, digital signal processing (DSP) algorithms, and adaptive inverse controllers to control the DC-DC power converters has been investigated in the past [5-17]. Few publications in this area have examined the feasibility of a digital nonlinear controller for a DC-DC converter, either with a simple low-power demonstration system or with simulation. These control schemes have proven to be superior to the conventional PID controller [5]. These methods can deal with highly nonlinear and time-variant systems where the mathematical models are difficult to be obtained. A recent interesting paper also presents the results of an experimental comparison of five nonlinear control algorithms simulated for a DC-DC power converter [17]. The increasing performance and falling price of the microcontrollers and digital signal processors allow the possibility of cost effective digital implementation of a nonlinear controller which can adapt quickly to the changes in the converter dynamics.

The commercial-off-the-shelf (COTS) movement has introduced a trend of producing standard converter modules which can be connected in parallel to cover a wide power range. An issue that arises with a system designed from modular components is that it is desirable to have a means to communicate among the individual modules and

with any supervisory system that may be present [19]. The ability to monitor and control converters via any standard communication protocol allows a power system to be assembled in a highly reliable, fault tolerant and autonomous manner. Several standard protocols such as CAN, high speed serial bus, RS232, I2C, Ethernet have been employed as communication methods among power devices in the distributed power systems [19, 20]. The worldwide familiarity, standardization, and availability of Ethernet, along with its current and potential performance levels, has prompted increased consideration of Ethernet as a viable communications technology for distributed power systems. For linking individual power devices together, CAN bus is extensively used in industrial automation systems mainly because of its robust nature [20].

1.3 Problem Formulation

The main objective of this research work is to improve disturbance rejection and fault diagnosis capability of a commercial DC-DC converter. An existing commercial power converter manufactured by Core Technology, Inc. was chosen as a test bench to investigate advanced control and communication technique in a cost effective and commercially viable manner. In the first version of this compact converter, digital proportional-integral (PI) control and CAN communication connectivity were combined in an industry standard full brick package. A Motorola 56F8323 microcontroller was used to implement this control communication hardware.

The focus of this research is to implement a robust cost effective controller for this commercial converter to achieve: 1) Rapid disturbance rejection, 2) Good dynamic performance (i.e. rise time, overshoot, settling time and limited output ripple) in the presence of input voltage variations (and load changes), and 3) Robustness around the operating point (e.g. in the case of a load change). A novel control scheme, active disturbance rejection controller, is designed and programmed in the existing microcontroller MCF56F8323 and compared to the existing PI control method.

To enhance communication capability at the system level, a CAN to Ethernet gateway is developed using Motorola MCF5282 microcontroller. With the development of this gateway, individual converters on local CAN network can be remotely monitored, controlled and tuned over the Internet virtually from anywhere in the world using a web browser based user interface. This digitally controlled "Smart" converter with advanced control algorithm, CAN bus and Internet communication capability is the first of its kind in the industry.

1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 discusses the first version of Core Technology DC-DC power converter, presenting the existing control and communication method as well as the proposed control and communication scheme for this converter. The design, simulation, implementation and hardware results of the application of ADRC to the commercial power converter are provided in Chapter 3. Chapter 4 presents the

microcontroller based implementation of the CAN to Ethernet gateway and development of graphical user interface to facilitate remote connectivity. Concluding remarks and the future research work are discussed in Chapter 5.

CHAPTER II

RESEARCH MOTIVATION FOR DC-DC POWER CONVERTER

With combined expertise of Advanced Engineering Research Lab and Core Technology, Inc., a digitally controlled compact DC-DC converter with embedded communication interface has been developed. A commercial DC-DC power converter designed and manufactured by Core Technology was used in the experimental setup for this research work. In this chapter, a detailed overview of the half bridge isolated 200 watt Core Technology DC-DC converter is presented. Existing control and communication capabilities as well as proposed advanced control and communication schemes are discussed.

2.1 Recent Trends in DC-DC Power Converters

In electrical engineering, a DC to DC converter is a circuit which converts DC power from one voltage to another. It is a special class of power converter, a kind of switched-mode converter, which generally performs the conversion by following these steps:

1. Convert DC to AC, for example with an oscillator or a chopper.
2. Change voltage at AC, using a transformer or inductor or diode/capacitor voltage multiplier.
3. Convert back to DC, using a rectifier (and usually provide voltage regulation).

Some basic topologies of switching power converters are buck, boost, buck-boost, SEPIC and Cuk converter [25]. DC-DC converters are a rich source of EMI. Their frequency spectrum begins at the switching frequency and often extends over 100MHz. The first rule in optimizing a layout of such a power system is to isolate the converter. In majority of the applications, it is desired to incorporate a transformer into the switching converter to obtain DC isolation between the converter input and output. There are several ways of incorporating transformer isolation into any DC-DC converter. The full-bridge, half-bridge, forward, and push-pull converters are the commonly used isolated versions of the buck converter. Incorporating state-of-the-art packaging techniques in the design of converters has resulted in significant advancements in the development of light weight, compact, reliable, and cost-effective converters. A review of “brick” package offerings in the market shows a continuous effort to halve the package sizes every five to seven years, with new packages delivering about 65-75% of the power of the previous packages [26].

The power converter chosen for this research is an isolated half bridge converter in an industry standard full brick package.

Today in the market, DC-DC power converters are employed in several applications, varying from power supplies for distributed power systems, personal computers, spacecraft power systems, laptop computers, telecommunications equipments to DC motor drives. The power industry has responded to the needs of its customers with the improvement in the performance of DC-DC products and by incorporating new features. With increased number of applications requiring high power in a compact space, higher power density is a continual demand with considerable implications on the converter's efficiency and thermal performance. Plus, there is a requirement for fast dynamic performance to accommodate high slew rate supply demands of advanced silicon loads. Modern applications demand exceptionally fast transient response capabilities from the power source. In addition to compact size, fast transient response and rapid dynamics are the main technical drivers behind the development of the next generation power converters. Such needs are stimulating factors innovations in the design of the commercial DC-DC converters.

With any DC-DC converter, the goal is to have a predictable, controlled, and monotonic output voltage ramp-up at turn on and no negative voltage spike at turn off. With conventional analog control, even slight changes to the operating characteristics might necessitate changes to the component values and PCB layout. With digital control, significant changes are now possible in the firmware, ranging from the provision of negative or positive logic for remote on/off to changes in current limit settings to handle unusual loads. Growing use of microcontrollers, plus the general design trend towards

digital signal processing, means that advanced control methodologies can now be incorporated into the power devices for better performance. The programmability of microcontrollers adds flexibility for both manufacturer as well as end users. Enhanced dynamic performance is the key target for the control strategy.

A recent trend in the field of power systems is distributed power architecture. The key advantages of distributed power systems are that they are inherently upgradeable, flexible and expandable. Distributed architecture enables off-the-shelf products to be used to help shorten time-to-market. It also provides electronics systems designers a better technical solution for many of the issues they face when using the latest high-speed semiconductor devices such as electromagnetic coupling (EMC), voltage regulation, transient current-handling capabilities for dynamic loads, and thermal management. The power system which is used as the test bench in this research work is a distributed power system with four modular DC-DC power converters connected in parallel as shown in Figure 1.

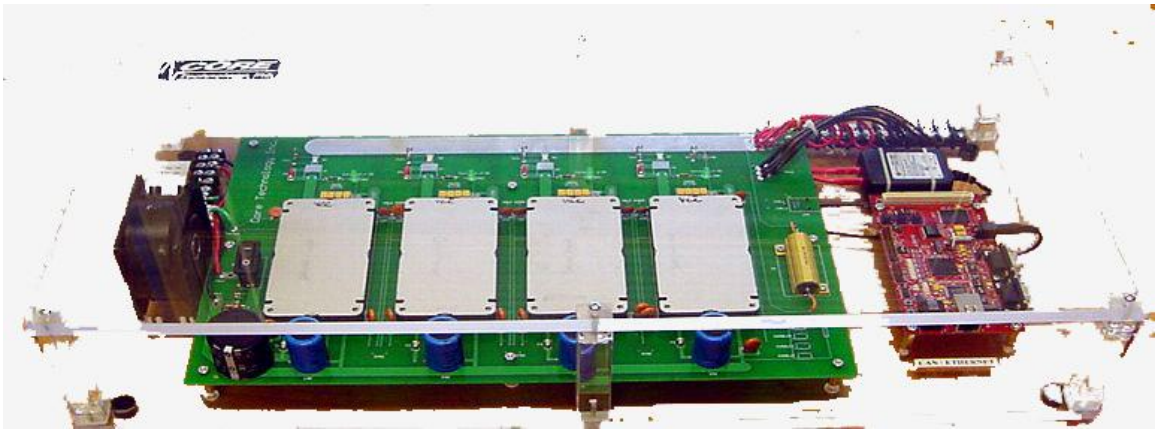


Figure 1: Experimental Set up of Power System

In a distributed control system, to have system level control distributed among an array of power converters, each converter must have the ability to make system level decisions. This implies that every converter must have knowledge of itself as well as of all the other converters in the system. All power converter modules must therefore share the knowledge of their operating conditions with all the other modules via a communication network. This makes it necessary to have a modular power converter which has the preliminary communication and system level power management functions embedded in it. Additionally, a power converter with remote configuration capability will also provide powerful control flexibility and will help accelerate a new generation of DC-DC converters, featuring power management, health monitoring, and remote fault diagnosis over a communication network.

2.2 A Commercial Half Bridge Isolated DC-DC Power Converter

An existing commercially available DC-DC power converter was chosen to implement advanced digital control and embedded networking technology. This power converter is a part of Core Technology's family of high density power modules designed to reduce product development time while achieving maximum power performance. These modules are regulated, isolated, and have been targeted for distributed power system level designs utilizing modular power converters. Utilization of active load sharing allows sharing the total output current accurately. In addition, they feature input enable, high efficiency, and 135 watts per cubic inch power density. Core Technology's regulated power modules can operate up to a base plate temperature of 100 ℃ and

deliver up to 600 Watts of output power in a 2.35” x 4.6” x 0.5” form factor. The selected C300 series modular power converter is shown in Figure 2. This power converter was previously controlled by analog means and current sharing was provided via a Texas Instruments load share chip.

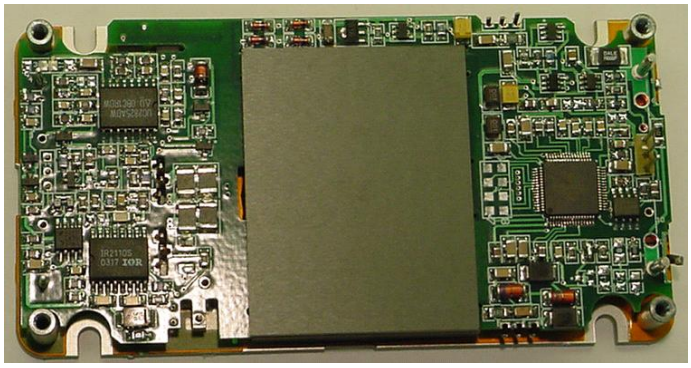


Figure 2: Core Technology Compact DC-DC Converter

2.3 Control Methodology for the Power Converter

A diagram of the major functional units of the digitally controlled Core Technology power converter is shown in Figure 3. This converter utilizes an industry standard PWM control chip on the primary to drive the upper and lower FETS for the half bridge topology. This portion of the feedback loop is driven in the traditional manner. Analog PWM on the primary side is driven with the transistor side of an opto-coupler. The opto-coupler bridges the primary and the secondary and provides isolation. The secondary of the converter contains a DSP based microcontroller, Motorola 56F8323, which is used to implement digital control. The voltage and current are sensed at the output of the converter and are fed back to the micro-controller as analog inputs.

The micro-controller performs an analog to digital conversion with its on chip analog to digital converter. It then performs the proportional-integral (PI) control and provides the digital values to be converted back to an analog signal to drive the opto-coupler. Once these samples have been calculated, they are sent out to the serial port of the micro-controller and converted into an analog signal by a digital to analog converter chip (DAC7512N). The analog signal from the DAC is sent to a precision current mirror driver circuit to drive the opto-coupler.

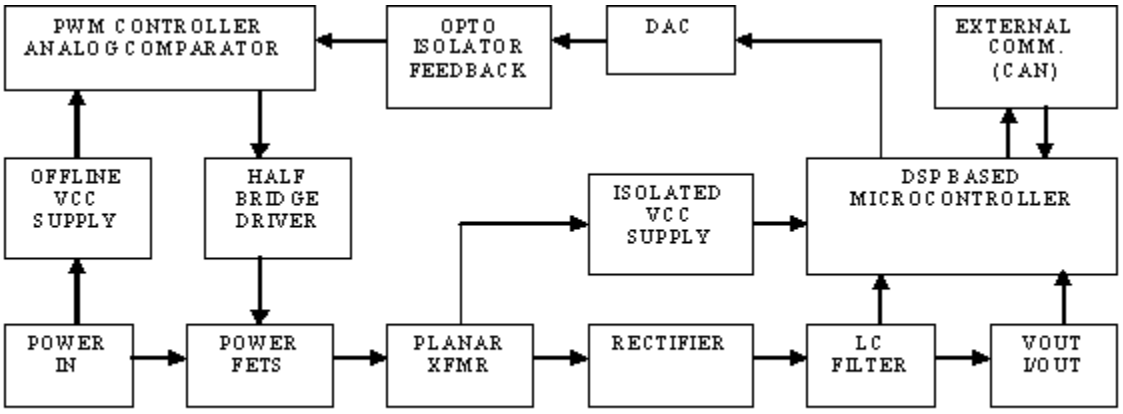


Figure 3: Digitally Controlled Smart Converter Block Diagram

In the first digital version of this converter, PI controller is implemented on the output side of the transformer and control update is performed with a frequency of 50 KHz. This PI controller is a benchmark set by Core Technology for its commercial converter [26]. It was tuned to obtain the best possible dynamic response and this was experimentally verified at Cleveland State University. Although PI controller proved to be satisfactory, its performance was limited and the response of the controller was slow. In simulations and also in hardware tests, it was observed that at high PI gains, system was tending to instability. It was observed that classical PI controller results into slow

dynamics and require a lengthy integration interval to compensate disturbances for a DC-DC power converter.

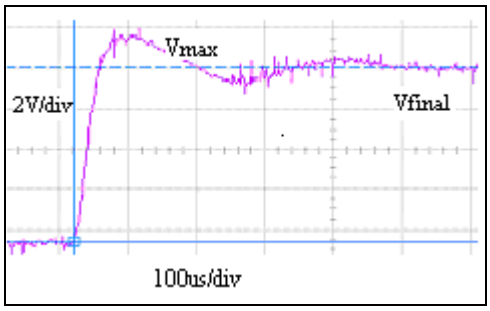
The purpose of this research work was to develop a better digital advanced controller which can be implemented with commercial converters such as Core Technology power converter in a cost effective way. The proposed active disturbance rejection controller can handle highly nonlinear dynamics and external disturbances with a stable output, small steady error, and fast disturbance rejection. It is easy to tune and it is implemented in the existing hardware platform of MC56F8323 microcontroller chip.

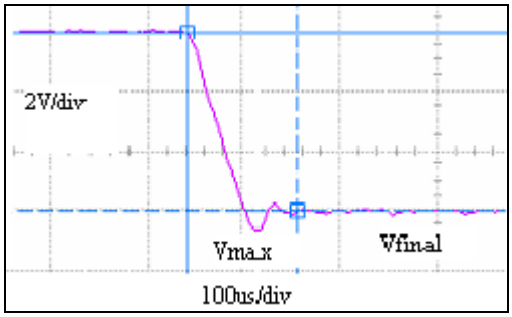
2.4 The Converter Model Approximation

For simulation purposes, approximate model of this converter was derived by examining the response of the converter to a step input using a digital oscilloscope. The output curves obtained were used to determine the time constant, steady state gain, and transfer function for the plant. Table 1 shows results of the open loop tests carried out at various loads to establish an approximate mathematical model of the converter. After analyzing the transient response plots, as shown is Figure 4, and using transient response equations, approximate converter transfer function was derived.

TABLE I: OPEN LOOP RESPONSE DATA

Vstart (volts)	**Vfinal (volts)**	**Vmax (volts)**	**Load (Amp)**	**Overshoot *OS*%**	**Zeta** ξ	**Ts**	**ω**
12	7.3	6	4	27.66	0.45	$1.00*10^{-3}$	$8.89*10^{3}$
12.79	8.17	6.5	4	36.15	0.4	$9.00*10^{-4}$	$1.11*10^{4}$
12.95	19.59	20.5	4	13.7	0.5	$8.50*10^{-4}$	$9.41*10^{3}$
19.68	13.75	12.75	4	16.86	0.3638	$9.63*10^{-4}$	$1.14*10^{4}$
9.76	14.39	15	5	13.17	0.3357	$4.81*10^{-4}$	$2.48*10^{4}$
13.97	9.5	8	5	33.56	0.4369	$6.09*10^{-4}$	$1.50*10^{4}$
14.45	19.62	20.7	5	20.89	0.3873	$4.45*10^{-4}$	$2.32*10^{4}$
19.68	13.75	12.75	5	16.86	0.3638	$9.63*10^{-4}$	$1.14*10^{4}$
11.34	15.54	16.5	6	22.86	0.397	$3.59*10^{-4}$	$2.81*10^{4}$
14.8	19.39	20.3	6	19.83	0.3817	$3.20*10^{-4}$	$3.28*10^{4}$
19.53	15.3	13.7	6	37.83	0.4488	$6.43*10^{-4}$	$1.39*10^{4}$
6.14	11.01	12	7	20.33	0.3844	$4.73*10^{-4}$	$2.20*10^{4}$
14.02	9.45	7.45	7	43.76	0.463	$4.46*10^{-4}$	$1.94*10^{4}$
12.63	19.17	19.75	7	8.869	0.2888	$3.06*10^{-4}$	$4.53*10^{4}$
19.45	13.41	11.41	7	33.11	0.4355	$6.21*10^{-4}$	$1.48*10^{4}$





Load Step Down – 8A-3A

Load Step Up – 3A-8A

Figure 4: Open Loop Transient Response of the Converter

The standard form of the transfer function for a second order system is:

$$T(s) = \frac{K\omega_n{}^2}{s^2 + 2\xi\omega_n s + \omega_n{}^2} = \frac{b}{s^2 + as + b} \tag{2.1}$$

where K is the steady state gain of the plant, ω_n is the natural frequency, and ξ is the damping ratio. The values of overshoot and settling time are related to the damping ratio and undamped natural frequency. Percent overshoot can be calculated theoretically using:

$$\%OS = e^{-(\xi\pi/\sqrt{1-\zeta^2})} * 100 \tag{2.2}$$

Experimentally, $\%OS$ can be found using the transient response plot as:

$$\%OS = \frac{V_{\max} - V_{final}}{V_{final}} * 100 \tag{2.3}$$

where $V_{\max}$ is highest oscillation voltage and V_{final} is the final oscillation voltage. Settling time T_s is given as:

$$T_s = \frac{4}{\xi * \omega_n} \tag{2.4}$$

Equations (2.2) and (2.4) can be solved to provide unique solutions for the two parameters. First, the overshoot equation will be solved for the damping ratio, and then the settling time equation can be solved for the undamped natural frequency.

$$\xi = \frac{-\ln(\%OS/100)}{\sqrt{\pi^2 + \ln^2(\%OS/100)}} \tag{2.5}$$

$$\omega_n = \frac{4}{\xi * T_s} \tag{2.6}$$

By solving equation (2.5) and (2.6) for a number of test condition results, we can get averaged values of the damping ratio and undamped natural frequency for the given converter as:

$$\begin{aligned} \xi &= 0.4 \\ \omega_n &= 19{,}400 \end{aligned} \tag{2.7}$$

By comparing the given form of the open-loop transfer function to the corresponding standard form for a second-order system as shown in equation (2.1), with the averaged values of percent overshoot and settling time and with steady state gain K as 1, the second order transfer function for the given converter is established as:

$$T(s) = \frac{(19{,}400)^2}{s^2 + 2(0.4)(19{,}400)s + (19{,}400)^2} \tag{2.5}$$

From these values, we can see that the converter dynamics are fast and the plant is highly unstable. The software simulation, discussed in the next chapter, uses this transfer function as the plant model of the DC-DC power converter. The characteristics of the simulated transient output voltage waveforms were compared to experimental transient output voltage waveforms to ensure that the responses are correlated.

2.4 Embedded Communication Interface

The key aspects of a highly reliable, fault tolerant and autonomous electrical power distribution and management (PMAD) system are [27]:

- Distributed control and
- Remote control and Health monitoring

In order to develop a distributed control strategy with parallel operating DC-DC power converters, it is necessary to set up an embedded data network which allows individual modular DC-DC power converters to communicate and share their operating conditions. For Core Technology power converter, this was accomplished through the use of a Control Area Network (CAN). In the first phase of this research [20], on-board CAN controller of Motorola 56F8323 hybrid DSP/microcontroller was used to embed CAN communication interface in the individual power converter. With individual digital controller on each converter and with an established communications network, these converters were able to collaborate and act as one intelligent power converter.

In order to provide a means to remotely monitor and control this power system as a whole, a CAN to Ethernet gateway is developed in this research work. It collects data on local CAN network and transfers data and control to and from an external entity using Internet. The worldwide familiarity, standardization, and availability of Ethernet, along with its current and potential performance levels, prompted to use Ethernet as the choice for monitoring and controlling the system remotely. This gateway is developed using Motorola's Coldfire architecture MCF5282 microcontroller chip and embedded networking technologies. It offers the ability to remotely manage a power system at the

modular as well as system level. A fully bi-directional graphical user interface is developed that lets remote users control and monitor system parameters and reconfigure individual parameters of each converter. Figure 5 shows topology of the power system built using Core Technology commercial power converters for Internet connectivity.

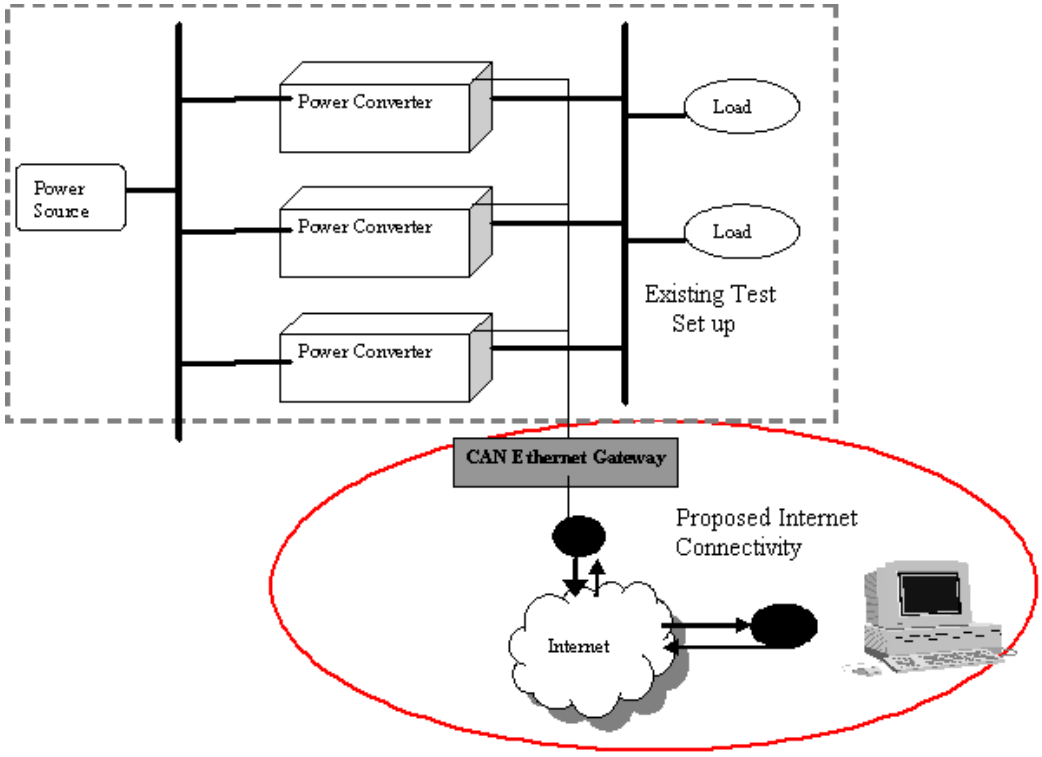


Figure 5: Communication Interface Topology

CHAPTER III

ADVANCED CONTROL DESIGN FOR DC-DC POWER CONVERTER

Fast switching and highly complex nonlinear dynamics makes a DC-DC power converter a difficult plant to control. A novel concept of active disturbance rejection control (ADRC) fits the nature of this problem well. In this chapter, the basic concept of ADRC is described in brief. Application of this controller to the commercial power converter is explained in detail with software simulation, microcontroller based hardware implementation and ADRC hardware test results compared to digital PI control.

3.1 Active Disturbance Rejection Control

The proposed active disturbance rejection controller has been shown superior to conventional PID controller for applications in several fields such as motion control, web tension regulation, DC-DC power converters, modern aircraft control, and multi-input multi-output systems with uncertain time delays. This controller naturally provides the ability to deal with nonlinear time-variant systems where explicit mathematical models are difficult to be obtained or control variables are too hard to measure.

In the conventional feedback control schemes, an accurate mathematical plant model is the basis to achieve desired closed-loop response. But a new control technique, active disturbance rejection control [18], estimates plant disturbances in real time instead of relying on an explicit mathematical expression of the plant. The primary reason of using feedback control is to deal with variations and uncertainties of the plant dynamics and unknown disturbances. Most of the existing control design methods require detailed knowledge of unknown disturbances before the control design is carried out. Instead, ADRC observes these internal and external disturbances in real time and actively compensates for them in the control signal without an explicit mathematical expression. ADRC was originally proposed by Han [18, 21] and simplified by Gao using bandwidth parameterization tuning technique [22].

As mentioned earlier, DC-DC power converter is a complex nonlinear plant with fast dynamics. Disturbances are common since it is a commercial power converter in industrial environments. The objective of this controller design is not only to have a small steady state error, but also to provide good disturbance rejection. ADRC seems to be an

ideal choice for such kind of control problem. The active disturbance rejection control algorithm is applied to Core Technology DC-DC power converter as shown below. The approximated second order transfer function of the converter in (2.5) is first converted to differential equation form as:

$$\ddot{y}(t) = -2\xi\omega_p \dot{y}(t) - \omega_p^2 y(t) + b_0 \omega_p^2 u(t) \tag{3.1}$$

Assuming $a_1 = -2\xi\omega_p$ and $a_2 = \omega_p^2$, in general a second order plant takes this form:

$$\ddot{y} = -a_1 \dot{y} - a_2 y + w + bu \tag{3.2}$$

where y is the output, u is the input, and w is the input disturbance. Rewriting it as:

$$\begin{aligned} \ddot{y} &= -a_1 \dot{y} - a_2 y + w + (b - b_0)u + b_0 u \\ &= f(t, y, \dot{y}, w) + b_0 u \end{aligned} \tag{3.3}$$

where $f(t, y, \dot{y}, w)$ or simply f includes both external disturbances, w, along with all known and unknown internal dynamics, $-a_1 \dot{y} - a_2 y + (b - b_0)u$, to form the generalized disturbance, f. It represents true dynamic behavior of the system. b_0 is the approximated value of b.

The key idea behind ADRC is to treat $f(t, y, \dot{y}, w)$ as a state variable, estimate it in real time, and then actively compensate it without necessity of any mathematical expression. A new type of observer, extended state observer (ESO), satisfies this need. It observes and estimates the value of f without complete knowledge of the plant model[28]. A brief description of the ESO for (3.3) is presented here; details of its digital implementation and generalization are presented in [29]. The main idea is to use an

augmented state space model of (3.3) that includes $f(y, \dot{y}, w, t)$, as an additional state. The augmented state space form of (3.3) with $x_1 = y$, $x_2 = \dot{y}$ and $x_3 = f$ is

$$\begin{aligned} \dot{x} &= Ax + Bu + Eh \\ y &= Cx \end{aligned} \tag{3.4}$$

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ b_0 \\ 0 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, E = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The state observer is:

$$\begin{aligned} \dot{z} &= Az + Bu + L(y - \hat{y}) \\ \hat{y} &= Cz \end{aligned} \tag{3.5}$$

If observer gains $L = [\beta_1 \quad \beta_2 \quad \beta_3]$ are selected appropriately, this observer will provide an estimate of the state (3.4). The success of ADRC depends on the close estimation of the third state of the observer, z_3, which approximates generalized disturbances, f. The ESO in its original form employs nonlinear observer gains. Recently, a new tuning method was proposed [22], which simplified its implementation. With bandwidth parameterization for ESO, observer design and tuning is greatly simplified. All observer gains are a function of the observer bandwidth, ω_o. The selection of the observer gain is calculated as:

$$\beta_1 = 3\omega_o, \beta_2 = 3\omega_{o,}^2 \beta_3 = \omega_o^3 \tag{3.6}$$

Considering (3.6), (3.5) can be written for a second order system as:

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \end{bmatrix} = \begin{bmatrix} -3\omega_o & 1 & 0 \\ -3\omega_o^2 & 0 & 1 \\ -\omega_o^3 & 0 & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} + \begin{bmatrix} 0 & 3\omega_o \\ b_o & 3\omega_o^2 \\ 0 & \omega_o^3 \end{bmatrix} \begin{bmatrix} u \\ y \end{bmatrix} \tag{3.7}$$

The main advantage of the ESO is that as long as the order of the plant remains the same, changes in the plant dynamics will be accounted for by the ESO and ultimately in the controller design. This will modularize the controller to be able to handle a wider range of plants and account for fluctuations in the actual plant. With a well-tuned observer, the observer state z_3 will closely track $z_3 = f(y, \dot{y}, w, t)$. Once the observer is built and well tuned, the control law can be realized as:

$$u = \frac{(u_0 - z_3)}{b_0} \tag{3.8}$$

Equation (3.3) then reduces to

$$\ddot{y} = \left(f - z_3\right) + u_0 \approx u_0 \tag{3.9}$$

which can be controlled with a simple PD controller:

$$u_0 = k_p (r - z_1) - k_d z_2 \tag{3.10}$$

where r is a constant set point. The controller tuning is further simplified with $k_d = 2\omega_c$ and $k_p = \omega_c^2$, where ω_c is the closed loop control bandwidth. Obviously, the bigger ω_c is, the faster the response. Both the bandwidths, ω_o and ω_c, are also limited by hardware constraints such as actuator saturation and sensor noise. Figure 6 shows the block diagram of the active disturbance rejection control method.

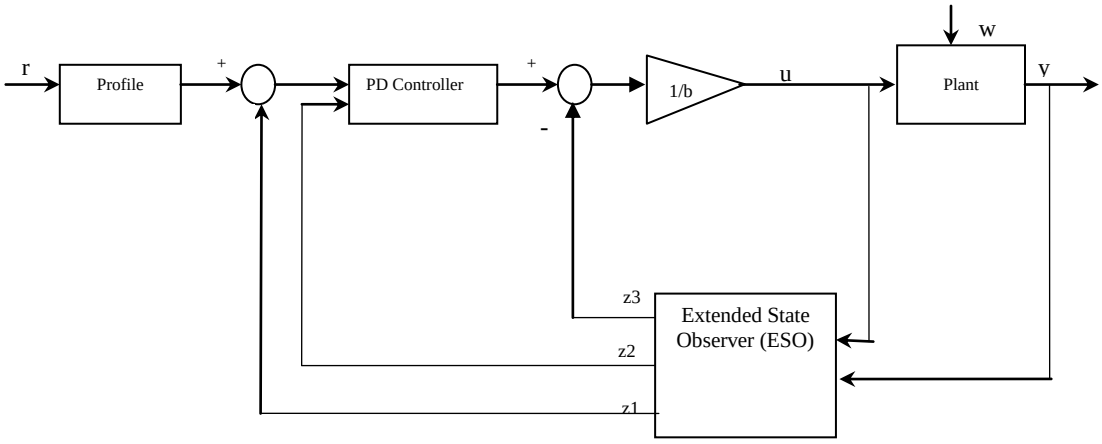


Figure 6: ADRC Configuration

There are three main blocks: Profile, PD controller, and ESO. The converter is treated as a "black box," which has only the terminals corresponding to the output voltage and the control input (duty cycle). The plant is observed by the ESO. The change in its dynamics and external disturbances $f(t, y, \dot{y}, w)$ are tracked by z_3 and then canceled by the control signal. The plant becomes a double integrator, which can be controlled by a simple PD controller.

DC-DC Converter Complexities

Throughout this discussion, we have taken into consideration an approximate second order model of the converter as developed in section 2.4. But we also need to consider the fact that in the real world, actual plants are usually different from the models, irrespective of the methods used to obtain the models. Since ADRC uses minimal information about the plant, it is robust against the model uncertainties and it can effectively reject the disturbances. Moreover, based on the plant parameter values derived for the converter as presented in equation (2.7), it can be observed that the converter dynamics are very fast. The high operating frequency of the converter places a severe time constraint on the control system's feedback and control calculation. The

dynamics of the system behave faster than the available sampling rate. This means that inevitably there will be responses that behave in erratic manners before the control has time to catch up. Additionally, the half bridge circuit of the converter operates in three different topologies during one duty cycle, making it highly complex, discontinuous and nonlinear. The control problem is further aggravated by the sensitivity of the converter characteristics in the steady-state operating conditions. As we are working in digital domain for the controller implementation, we have additional constraints to consider. Sampling frequency of the controller is limited by the hardware in the actual implementation. All these factors make the converter a difficult plant to control. In the following subsections, application of ADRC to this control problem is demonstrated along with simulation as well as hardware implementation results.

3.2 Design, Tuning and Response of ADRC

A MATLAB Simulink version 7.0 is used for the control simulation of the converter. Figure 7 shows ADRC block diagram simulated using MATLAB Simulink.

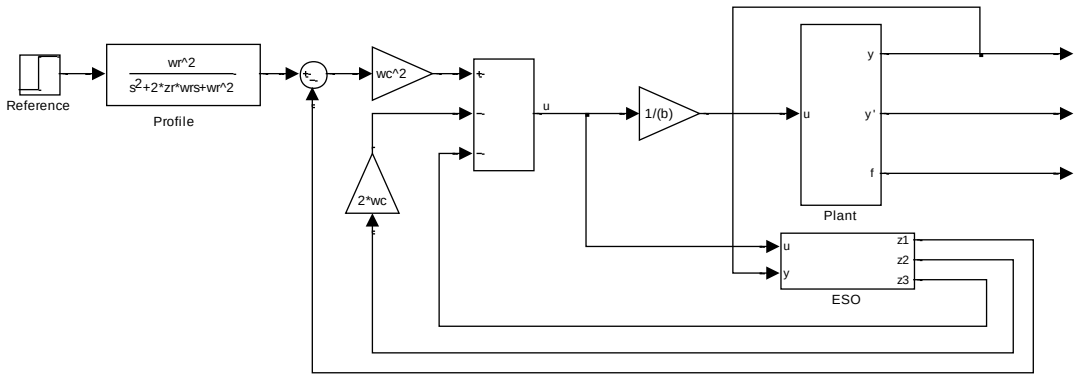


Figure 7: ADRC Design for the Plant

3.2.1 Software Simulation

In order to compare $y, \dot{y}$ and $f(t, y, \dot{y}, w)$ with ESO output z_1, z_2 and z_3 respectively, $y, \dot{y}$ and $f(t, y, \dot{y}, w)$ need to be extracted from the plant for testing purposes. Rewriting (2.5) as:

$$\ddot{y} = -15.52 \times 10^3 \dot{y} - 3.7636 \times 10^8 y + 737665600u \tag{3.11}$$

Figure 8 shows a Simulink block diagram of the power converter as the plant. Although not available in a real plant, the generalized disturbance effect of f is modeled as an output of the plant for testing purposes.

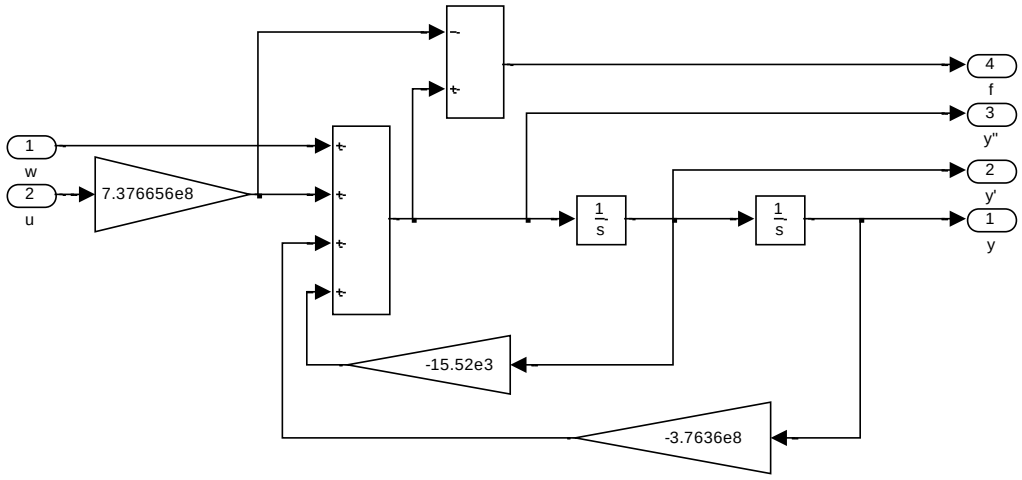


Figure 8: Simulation of the Plant

Figure 9 shows subcomponents of ADRC scheme including the controller and the extended state observer. The zero order hold block available in Simulink block library is used to simulate the on chip A/D converter on the microcontroller which is used in the implementation of the secondary of the power converter. It samples converter output and converts it into digital data. The analog input ranges from 0 to 5V, and a 12 bit register

stores the data. The quantization interval is equal to 0.00122 volts/bit. The sampling frequency is 50 KHz (every 20 microseconds the A/D samples the output data). This digitized data is then sent back to the controller and a new control signal is calculated and sent to the plant.

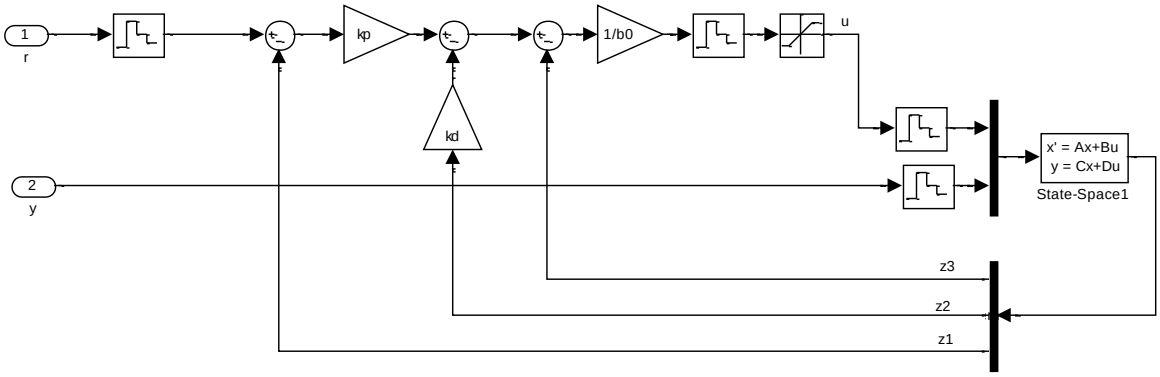


Figure 9: Simulation of Controller and Observer

Nonlinear Saturation

Almost all physical plants have known nonlinear limitations to the control signal. In this plant, analog control signal is generated with a 12 bit buffered digital to analog converter DAC7512. This imposes saturation limits on the control signal u. Including these limits in the ESO greatly improves its performance for large disturbances and errors. A common second order plant in (3.3) with saturation limits on control signal can be described as:

$$\ddot{y} = f + b_0 sat(u, u_{\min}, u_{\max}) \tag{3.12}$$

where $u_{\min}$ is the smallest possible value of the control signal u and $u_{\max}$ is the largest possible value. In state space form this becomes:

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_2 + b_0 sat(u, u_{\min}, u_{\max}) \\ \dot{x}_3 &= \dot{f}\end{aligned} \tag{3.13}$$

This small modification produces a simple nonlinear disturbance observer which has a much better performance over the linear observer for this system:

$$\begin{aligned}\dot{z}_1 &= z_2 - \beta_1(y - z_1) \\ \dot{z}_2 &= z_3 + b_0 sat(u, u_{\min}, u_{\max}) + \beta_2(y - z_1) \\ \dot{z}_3 &= \beta_3(y - z_1)\end{aligned} \tag{3.14}$$

where z_3 tracks f.

In the final version of the simulations, the controller and the observer are implemented as a single S-Function, with command reference *r* and plant output y are inputs, and control signal u, and observer states $z_1, z_2,$ and z_3 are outputs. Three programmable parameters include observer bandwidth ω_o, controller sampling frequency, ω_s and b_0. Details of this S function can be found in Appendix A.

3.2.2 Tuning

According to the tuning technique shown in [18], the controller gain was calculated as $\omega_c = 2{,}500$ and observer gain as $\omega_0 = 1.885 \times 10^4$. The verification of this simulation is shown in Figure 10. These values were established with the software simulation as a starting point to implement ADRC in hardware as discussed in the next section. ADRC was tuned to reduce the settling time to 2 milliseconds. This settling time

is ten times less compared to the industrial version of the converter. The hardware implementation and results are discussed in the next section 3.4.

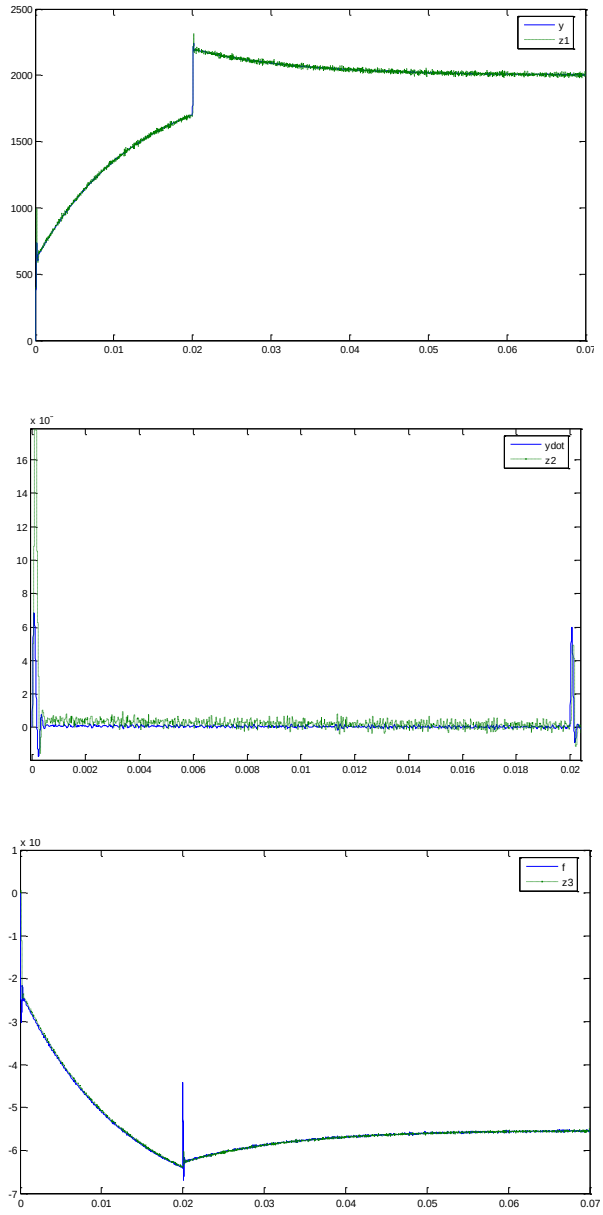


Figure 10: Comparison of $z_1, z_2,$ and z_3 to $y, \dot{y},$ and f

3.3 Hardware Implementation of ADRC

In this section, hardware implementation of the proposed active disturbance rejection controller for Core Technology power converter is presented. It demonstrates the possibility of applying this control method in a commercial power converter in a cost effective manner. The Motorola 56F8323 hybrid digital signal processor (DSP)/microcontroller was chosen to implement the controller. The 56F8323 is a member of the 56800E core-based family of Motorola controllers. It combines on a single chip processing power of a DSP and functionality of a microcontroller with a flexible set of peripherals resulting into an extremely cost-effective solution. Because of its low cost with DSP and MCU combined functionality in a unified, C-efficient architecture, 56F8323 is well suited for implementation of this advanced controller for the power converter. It includes a number of useful peripherals such as two twelve bit ADCs, a six channel PWM module, a flex CAN module, etc. These peripherals are especially useful for industrial control applications. This choice of microcontroller was made based on the microcontroller speed and the convenience of reusing the component in the existing hardware to utilize its power to the fullest.

In this digital controller implementation, the sampling frequency is 45 KHz. Every 22 μs, the converter output voltage is sampled and digitized, a new duty ratio is calculated based on the sampled data and the control algorithm, a new control signal is then generated, and sent to the MOSFETs through a DAC and analog PWM converter to regulate the output voltage. The physical tuning process of ADRC follows a pattern: estimate b_0, set initial values of ω_0 and ω_c and gradually increase until the noise level

and oscillation in the control signal and output exceed the tolerance. The values of these tuning parameters estimated with simulations were used as a reference for the hardware tuning. Two key issues in programming the control algorithm in the microcontroller were 1) unsigned integer arithmetic and 2) saturation bounds on the control signal.

Control function was originally coded using floating point variables but it was too slow to support any other functionality than control. To manage this constraint the controller was modified to use only 32 bit fixed point variables which greatly improved the speed of the control calculations. With the use of fixed point integer arithmetic, computational delays are reduced but it also required major modifications in the way that the control algorithm was coded. The unsigned integer arithmetic means that all the arithmetic calculations and results are between 0 and 2^{32} and no negative numbers exist in the system. If an arithmetic operation results in a number out of this range (positive or negative), it will set the carry flag and will cause the result to wrap around. Now even if this range of integers for positive numbers seems to be large, in actual controller implementation, intermediate iterations of the control loop have state values larger than this range. To handle this problem, transformed states were defined which would fit into the integer bound limits.

The second implementation issue was caused by the bounds on control signal [23]. Almost all physical plants have nonlinear limitations to the control signal which are practically known. Including these known saturation limits in the observer can greatly enhance estimation performance. A simple interesting modification to the ESO was done to allow the power converter to work with ADRC with a bounded control signal. As shown in Figure 11, a saturation enhancement to disturbance estimation is a simple

modification to the standard disturbance estimation which enhanced its performance and reduced estimation error by more closely matching the plant model.

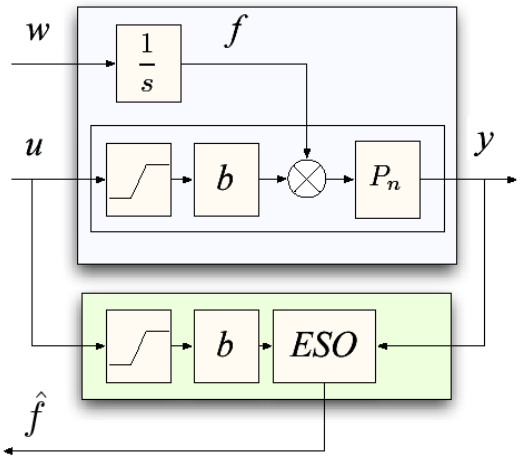


Figure 11: Simple Saturation Modification to Disturbance Estimation

3.4 Test Results

Two categories of tests, which cover the basic performance areas, i.e., load regulation and line regulation, were carried out to evaluate the performance of the controller with the given plant. The system performance was assessed by considering the response of the converter output voltage to the step changes in the reference and step changes in the load current for ADRC as well as the PI controller. The PI controller used for comparison during the controller evaluation is the controller that is currently implemented in the commercially marketed version of this DC-DC power. The dynamic response to the change in the output load was examined for various test cases.

The results for a load step down condition are shown in Figure 12 for a load change from 8A to 3 A for input voltage of 90 volts and output voltage as 12 volts.

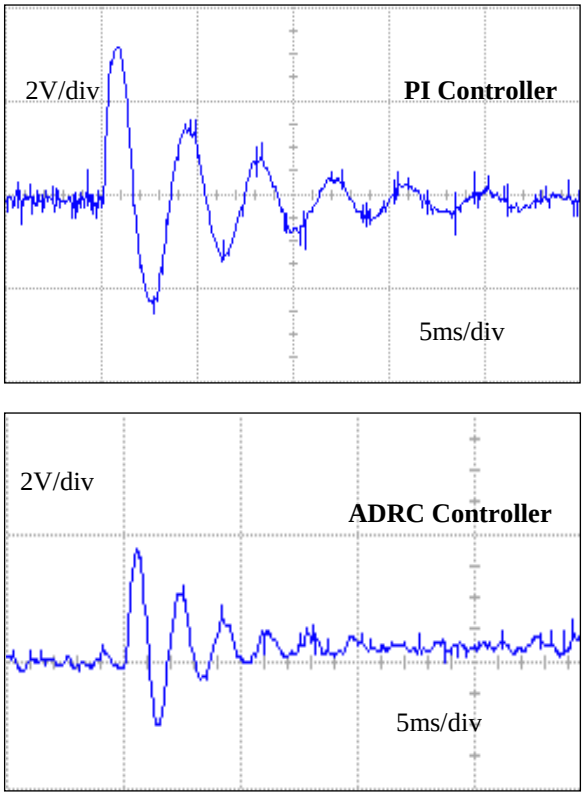


Figure 12: Load Step Down Disturbance Rejection

It can be observed that for the PI controller, the voltage deviation is 2.5 V and the recovery time is 20 milliseconds, where as for the ADRC, the voltage deviation is 1.8 V with the recovery time of 6 milliseconds.

The results for a load step up condition are shown in Figure 13 for a load change from 3A to 8A for input voltage as 90 volts and output voltage as 12 volts.

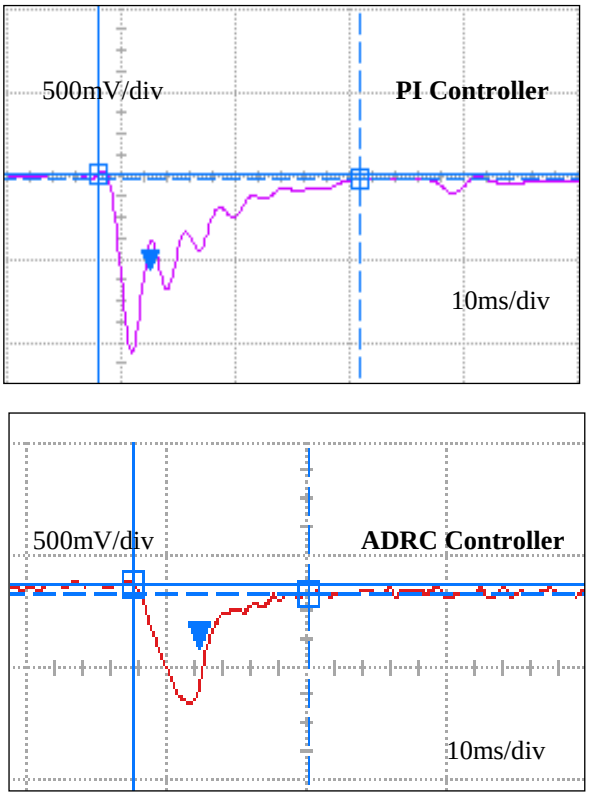


Figure 13: Load Step Up Disturbance Rejection

It can be observed that for the PI controller, the voltage deviation is 1.1 V with the recovery time of 18 milliseconds, and for the ADRC, the voltage deviation is 0.5 V with the recovery time of 6 milliseconds.

A step change in the reference voltage set point is applied from 8 volts to 12 volts. Figure 14 shows the response of the converter for both controllers at 5A load with input voltage as 90 volts and initial output voltage as 8 volts.

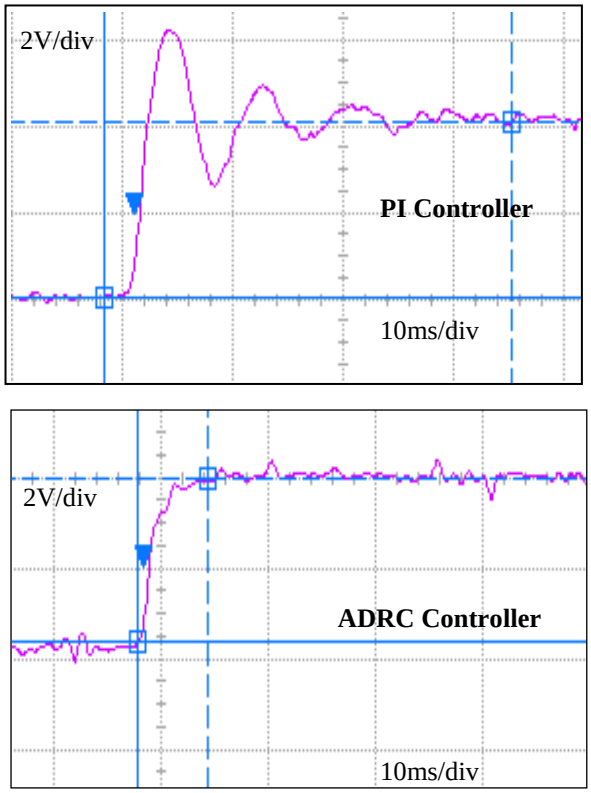


Figure 14: Input Voltage Step Change Disturbance Rejection

For the PI controller, the voltage deviation is 2 V with the recovery time of 30 milliseconds, and for the ADRC, there is no voltage deviation with the recovery time of 6 milliseconds.

Conclusions

As mentioned earlier, the steady-state error settles down within the error limit for both controllers. However, the dynamic response is relatively slow for PI controller, with the controller taking about 18 milliseconds to settle in case of load increase

and 22 milliseconds in case of load reduction. ADRC settles at a much faster time base taking 6 milliseconds to settle in case of load increase as well as reduction. Based on the hardware results as presented in this section, ADRC provides better rejection to the input supply variations and the load transients and it also provides a simple method for tuning.

CHAPTER IV

INTERNET CONNECTIVITY WITH CAN TO ETHERNET GATEWAY

Integration of an advanced communication topology in a power converter for its parametric modification and remote health monitoring was achieved with the development of the CAN to Ethernet gateway. This gateway links the Internet world with the local CAN bus network of DC-DC power converters. A secure and real-time access to the power system data is made possible from anywhere in the world. A fully bi-directional graphical user interface is developed, that lets remote users adjust controllable parameters of the converter and monitor observable parameters. This gateway and the graphical interface together made the local network of the power converters global, introducing the ability to communicate with an upper level probing interface and enhancing reconfiguration and monitoring capability of the converter and the power system as a whole.

4.1 Embedded Internet Connectivity

Now a day, Internet is seen as the most cost-effective way of remotely monitoring and controlling any embedded system because of the availability of low cost embedded Internet enabling technology. From a practical standpoint, Internet communication currently lacks the performance required to support real-time communication. Even in an Intranet environment that circumvents Internet bottlenecks, Internet Protocol (IP) lacks the required determinism and the capability to allocate the dedicated bandwidth, which is essential for real time communication. Even with these constraints, the Internet and intranets are still suitable for performing functions such as remote diagnostics, configuration and management which do not require hard real time behavior. The designers looking to provide Internet connectivity in their products have to choose one of the three implementation options of UDP or TCP as listed below:

1. Do it yourself solution – the development of a proprietary UDP or TCP/IP stack.
2. Software solution – UDP or TCP/IP stack is implemented in software. This approach requires a microcontroller unit (MCU) with sufficient functionality to implement both the UDP or TCP/IP stack and the application program.
3. Hardware solution – UDP or TCP/IP stack is implemented in hardwired logic.

The approach chosen in this research work for the design of the CAN to Ethernet gateway is UDP and TCP/IP stack implemented in software. On the monitoring side,

most of the computers have an Ethernet connection with a UDP and TCP/IP stack built into the operating system. Since the transport layer framework is already in place for the higher level communication side, two different communication topologies were implemented in this research work, one using UDP/IP and the second one using TCP/IP as the basic scheme.

4.1.1 UDP Client – Server Approach

The first approach is software driven implementation of an UDP client and server using network programming in Java. UDP is a connectionless protocol, which means that there is no need for a connection to be started. The server on the gateway side can broadcast data to a large number of clients without being aware of the number of clients listening to it. The number of UDP open sockets at a time depends on the available free ports. On the monitoring side, the UDP client is developed with a graphical interface using Java. The communication topology of this approach is presented in Figure 15.

Figure 15: UDP Client – Server Approach

The main advantage of this approach is all flow control, transaction logging, etc is up to the user programs. So it is possible to employ and implement only the features necessary for a given application. This increases the transmission speed up to three times as compared to the TCP because of the lower amount of processing at the source network

leads. Also, there is a much less overhead on packet handling and operating system of the server as well as the client. The main disadvantage of this approach is that since the client software is developed using Java, it is necessary to have Java virtual machine on the computer to be used as the remote monitoring station and also each computer needs to have the UDP client software program running on it to communicate with the UDP server on the gateway.

4.1.2 Embedded Web server Approach

The second approach implemented for achieving the Internet connectivity is using embedded web server. This choice is more advanced and direct means of providing a WWW interface to an embedded system by integrating a complete web server along with its content into the embedded system itself. With the concept of the embedded web server, a user sitting at any standard browser sees a web enabled device as a web site. The device presents itself and its state graphically and responds to the buttons, hot links, and the entire array of familiar browser controls. The potential for this kind of interface is essentially unlimited. The HTML interface makes the device look like a web page, enabling it to be remotely queried, configured and managed via common web browsers such as Microsoft Internet Explorer and Netscape Navigator. Real time operating system (RTOS) vendors are currently working on taking a similar approach, adding browsers and HTML interfaces that will enable embedded systems based on their RTOS to manage or be managed by the other Internet ready systems. The topology of this approach is shown in Figure 16.

Figure 16: Embedded Web Server Approach

By embedding a tiny web site into the firmware, an intuitive, easy-to-use, and inexpensive graphical user interface is established for remote control, configuration, monitoring, and diagnostics. Furthermore, this embedded web based interface supports multiple users, and can be run from browsers running on anything from a PC to a pager. The most important part about this approach is although the word "server" is often associated with the software component that is heavyweight, embedded web servers can actually be implemented efficiently in a small footprint.

4.2 Hardware and Software Development Components

In this subsection, hardware and software components used in the development of the CAN to Ethernet gateway for both design approaches are discussed. Employing an embedded real time operating system was essential to develop the application software for this gateway. Therefore, a brief overview of the software design principles using a real time operating system is also presented in this section.

4.2.1 Hardware Components

Motorola's 32 bit Coldfire architecture MCF5282 microcontroller is chosen for the implementation of this gateway. It has on-chip FlexCAN support as well as on-chip Ethernet controller. The FlexCAN module is a communication controller, which implements the CAN protocol. The integrated Fast Ethernet Controller (FEC) of MCF5282 performs the full set of IEEE 802.3/Ethernet CSMA/CD media access control and channel interface functions. The FEC supports connection and functionality of 10/100 Mbps 802.3 media independent interface (MII). Figure 17 shows M5282LITE development board used in the development of this CAN-Ethernet gateway. It provides Ethernet transceiver (PHY) to complete interface to the on-chip Ethernet controller. By combining 32-bit processing power with Ethernet capabilities, MCF5282 can operate as a web server on any Ethernet network running UDP or TCP/IP.

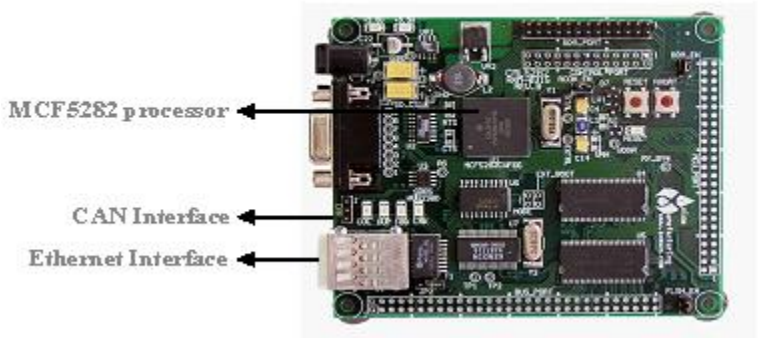


Figure 17: MCF 5282 Development Board

4.2.2 Real Time Operating System

Most of the programmers are familiar with the traditional systems which employ a looping construct for the main part of the application and user interrupts to handle any time critical events. These are so called foreground/background (or superloop) systems, where interrupts run in foreground (because they take priority over everything else) and the main loop runs in the background when no interrupts are active. Foreground/background systems are relatively simple from a programming standpoint as long as there is a little interaction amongst the functions in the main loop and interrupt service routines (ISRs) in the foreground. But they have several drawbacks such as loop timing is affected by any changes in the loop and/or in the ISR code. Also, the response of the system to the inputs is poor because information made available by an ISR to a function in the loop can not be processed by the function until it gets its turn to execute. Rigidly sequential nature of the program execution in the superloop affords very little flexibility to the programmer, and complicates the time critical operations. As applications grow in size and complexity, this approach loses its appeal because it becomes increasingly difficult to characterize the interaction between the foreground and the background. For an application which has complex design and limited processing resources, use of super loop approach can be tedious and time consuming. Many advantages can be realized by splitting a foreground/background application into an application with multiple independent tasks. This method of structuring applications employs a software framework that manages over all program execution according to a set of clearly defined rules. With these rules in place, the performance of an application can be characterized in a relatively straightforward manner, regardless of its size and

complexity. Many embedded system applications such as this gateway can benefit from using an approach involving the use of multiple, concurrent tasks communicating amongst themselves, and all managed by a kernel with clearly defined run time behavior. With the use of readily available RTOS software foundations which are optimized for a particular family of microprocessors, time to market for a product can be drastically reduced. Details of the basic terminology related to RTOS are provided in Appendix E.

4.2.3 Software Components

An integrated software suite called RTXC Quadros is chosen as the basic software framework for the development of this application. It is well suited to support a wide range of Internet enabled applications using MCF5282. This software package includes two separate components - RTXC Quadros, and Quadnet. RTXC Quadros is a multitasking real time operating system which is optimized for fitting complex and real time functionality into MCF5282 in much less time and memory. The RTXC Quadnet is a standard networking protocol stack, which consists of Ethernet drivers, core Internet protocols such as IP, UDP, TCP, ICMP, ARP, and DHCP, and application layer protocols such as HTTP (a small web server). All of the system software, the RTXC Quadros operating system and the RTXC Quadnet networking stack, fits into less than half of the on-chip flash memory and uses only half of the on-chip RAM. The other halves of the Flash and RAM are available for the user applications.

Metrowerk's CodeWarrior Development Studio is used as an integrated development environment (IDE). It consists of all the necessary tools required to complete an embedded development project including compiler, source level debugger,

editor, code navigation system, and project manager. Figure 18 shows organization of the several software components employed in the firmware development of the CAN to Ethernet gateway.

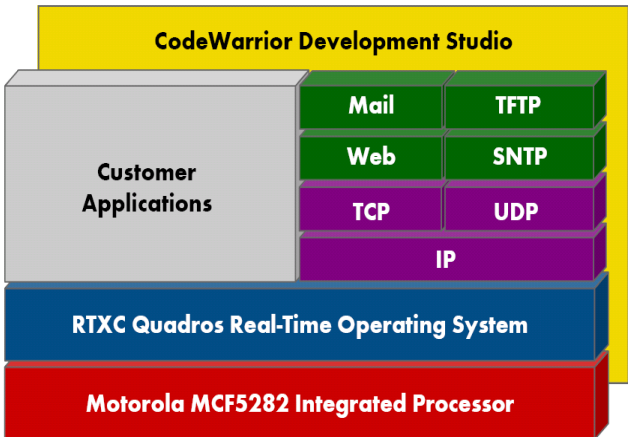


Figure 18: Software Components used in Firmware Development

4.2.4 Flash and RAM Memory Map

Processor memory requirement of a real time operating system is an important criterion while selecting a particular operating system for an application. An application developed using RTXC Quadros consists of multiple elements which contain the operating system elements, including the RTXC Quadros kernel and its API library, the Quadnet Internet communication protocol stack and its API libraries, the device drivers for UART, timers and Ethernet, certain application configuration data, system startup and initialization code. All of this is contained in one file (approximately 192K bytes) and is programmed in on-chip flash memory of MCF5282, which is 512 K bytes. The rest of the flash program memory can be utilized by the user application program. The total RAM available in MCF5282 is 64 Kbytes, half of which is used by the RTXC Quadros

operating system and the related components and the remainder is reserved for the user applications.

4.3 Software Design Approaches

An RTOS enabled application or program is the end product of combining application specific tasks, ISRs, data structures etc. with a RTOS to form a single program. In order to take full advantage of the multitasking abilities of a RTOS, it is necessary to divide an application into a number of independent tasks such that at any particular time, the processor is making the best use of its processing power by running the task is most important. From point of view of modular approach for the software development, functional requirements of an application are decomposed into a suite of functional entities called threads. RTXC Quadros real time operating system is used as a framework for executing multiple threads in the order of their priorities and for managing the system resources among these independent tasks. The system timings and inter-thread communication is established by calling various kernel services through a comprehensive application program interface (API) to achieve the desired system behavior. The RTXC kernel software is used as any other software library. The utilization of this RTOS allowed the gateway functionality to be designed and expanded easily, since functions or tasks can be added to each module without requiring major changes to the software. The RTXC/ss component of the RTXC kernel features a single stack model with a low-latency thread scheduler, which makes it ideally suited for the applications requiring high frequency interrupt processing.

Figure 19 shows organization of the application software using Quadros RTXC.

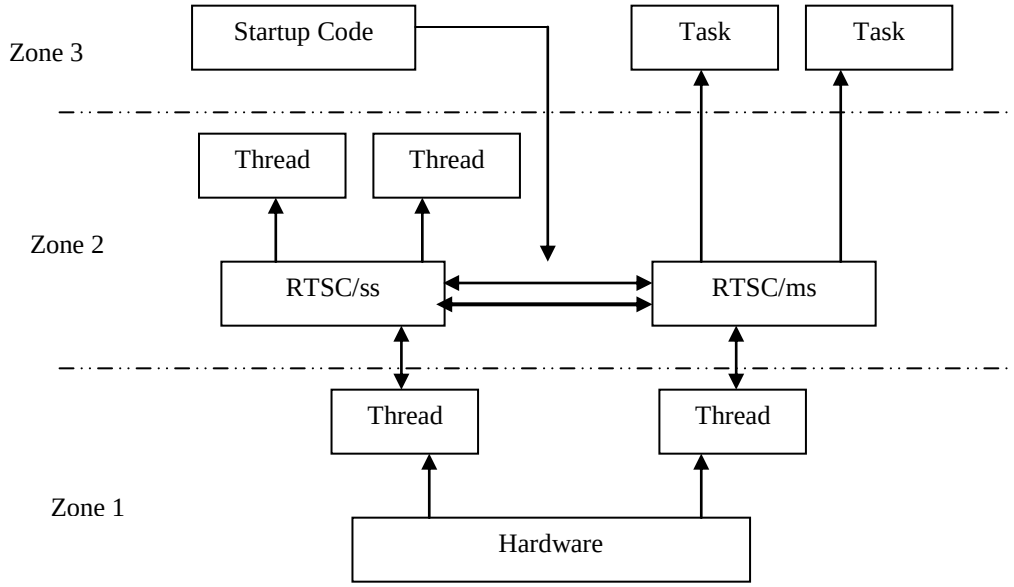


Figure 19: Organization of Application Software using Quadros RTXC

The functional specifications of this CAN to Ethernet gateway can be described as:

1. Sense and store power converter data received on the CAN network.

2. Transfer this data to the higher network layers over the Ethernet.

3. Accept and interpret remote user commands.

4. Support a graphical user interface to display data and accept user inputs on a remote terminal.

The user commands need to be interpreted into appropriate CAN messages for communication over the local CAN network, so that each converter on the network can read these commands and change its state accordingly.

In the process of software design and development for this gateway, two different approaches were taken into consideration, namely UDP client-server approach and embedded web server approach. Both the approaches are based on the client-server software architecture, as it is versatile, message-based, and modular. The main aim of both the approaches is to achieve the functionality stated in the functional specifications of the gateway. With further survey and considering future requirements, the embedded web server approach clearly seemed to be advantageous over the UDP client-server approach and it was a clear choice for the final implementation. But in the following subsection, a brief overview of the UDP client-server approach is presented which will help to understand the disadvantages of this approach and the benefits of developing the gateway with embedded web server approach.

4.3.1 UDP Client-Server Approach

In client-server architecture, a client is defined as a requester of services and a server is defined as the provider of services. In the CAN to Ethernet gateway application, the gateway is treated as a server which would provide its clients updated system data over the Internet. On each remote terminal, UDP client software would run to request the access to the power system data. The UDP server and client software is implemented using socket and network programming in Java. When a UDP client running on a remote terminal requests the data from the server, the server running on the gateway parses the most recent converter data received over the CAN bus into a UDP packet, this datagram is then sent to the client requesting the data and it is routed with the destination address as the IP address of the requesting client. For the implementation of the user commands,

when the server receives a UDP packet reserved for a user command, the user command is interpreted to an equivalent CAN command using a command interpretation table, as explained in the next section, and the required command action (such as voltage set point change or current weight change) is taken by the hardware. Figure 20 shows the algorithm used to implement UDP server and client on the gateway and the remote terminal respectively.

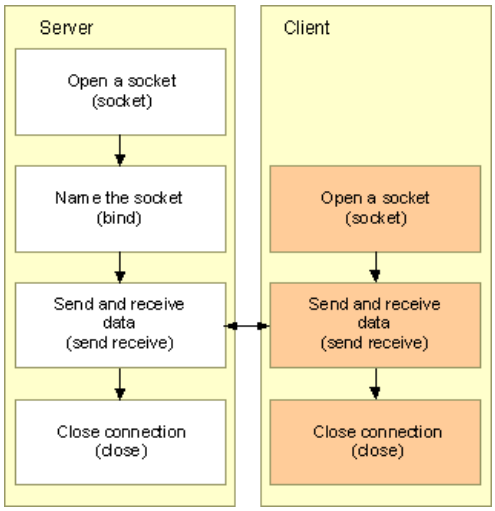


Figure 20: UDP Client- Server Algorithm

The main advantage of this approach is enhanced transmission speed due to the flexibility in UDP protocol implementation. But some of the main disadvantages are: 1) Remote terminal must have the client Java program to run as a remote monitoring station, 2) Remote machine needs to have Java virtual environment to run the UDP client program, 3) UDP is strictly a request-send protocol and the client has to explicitly request the information to the server. With embedded web server approach as presented in the next subsection, all these shortcomings were overcome. Figure 21 shows snapshot of the graphical user interface running on the client side which is developed using graphical

toolkit (AWT) available in Java. The software code for this interface is included in Appendix F.

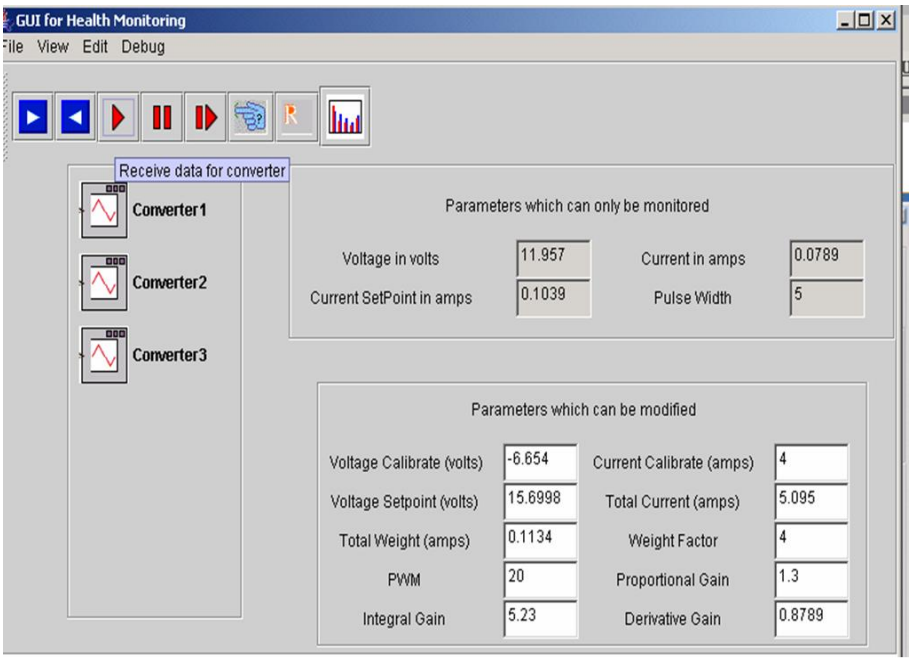


Figure 21: UDP Client Graphical User Interface

4.3.2 Embedded Web Server Approach

This approach is based on the client server architecture as well. But instead of developing custom application layer software, it is developed using HTTP server. RTXC Quadnet protocol stack has an embedded web server built in its application layer. It is not required to develop a HTTP client separately, since all the web browsers have a built in HTTP client. The human interface to the gateway is developed mainly using HTML. The power system information is delivered via the hypertext transfer protocol (HTTP) of the TCP/IP protocol suite, when requested by a remote user's web browser. With this approach, data over the CAN network can be accessed practically from anywhere within

the world via Internet using any standard web browser. This approach is implemented in the final version of the gateway and its software design and development details are discussed in the next three sections.

4.4 Software Modules in Embedded Web Server Architecture

In the embedded web server approach, communication between the remote users and the local CAN bus is achieved with a client server approach in which the server is running on the gateway firmware and Micro-web component of the server downloads a Java applet to a browser when a connection is requested and then continuously sends converter data such as voltage, current, pulse width modulation values and controller gains to the client applet. Figure 22 shows architectural components of the embedded web server built in RTXC Quadnet protocol stack.

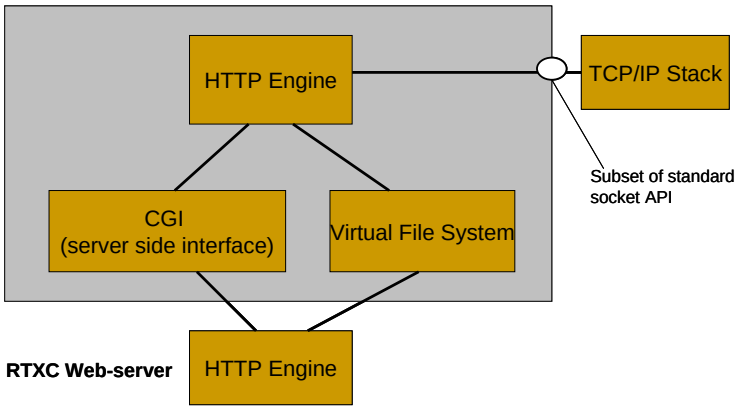


Figure 22: Web Server Architecture

In this approach of implementation, the functionality of the gateway can be divided into three main modules: embedded web server or servlet module, command interpretation

table (a database that stores the user network commands and equivalent CAN message formats), and applet module (to implement remote user interface).

4.4.1 Servlet

The functionality of the servlet module can further be divided into three individual tasks: CAN receive – Ethernet transmit, Ethernet receive – CAN transmit, and interrupt service routines to handle real time critical events in the application. The interactive remote user interface is developed with the GUI components of HTTP web server included in Quadnet networking protocol stack. This embedded web server relies on a set of user defined web pages (written in HTML and Java language) to provide HTML network output. It communicates with a remote web browser using HTTP protocol over TCP/IP. This application uses a number of components supported by Micro- web server such as virtual file systems, security, server side includes (CGI), Java support, server push, and forms.

Inclusion of real time data such as voltage and current for each power converter directly into a HTML/Java document is possible with server side includes which are written using CGI script. The CGI script allows HTML pages to interact with the programming applications. Unlike HTML pages, a CGI program is executed in real-time and can output dynamic information. It allows key words to be included in a web page so that when a particular web page is requested, it will cause the web server to execute a function specified in the web page. The function call sends data directly to the browser as a part of the requested document.

To implement user commands for changing the voltage set point and current weight from the remote terminal, form action posts are implemented. Forms allow a browser to send input back to the server. This feature is useful if there are commands or specific settings which need to be sent to the server. A form action post consists of keywords in the web page which instruct the browser to send a POST command back to the specified IP address (usually the web server's address). The command sent back is based upon the input from the browser side. Upon receipt of a POST command, the web server will call a function that parses input from the browser and performs an action based on what is found in the input. Thus, server side includes accept the forms containing voltage set point or current weight values submitted by the user and process them accordingly, for example, if a command is received to change the voltage set point, a CAN packet is built with the appropriate command identifier and the value of the desired voltage set point and then this CAN packet is transmitted over the CAN bus.

A feature called server push enables the web server to continuously send data to a remote client without explicit refresh requests from the client. With this feature, a web server can send complete chunks of data while keeping the HTTP connection open indefinitely. The server push relies on a variant of the MIME message format called "multipart/mixed". It is used for updating the field data such as voltage and current of an individual power converter dynamically.

Server side includes, form action posts and server push are implemented via a table of names and pointers to the functions. Finally, authentication and security is of critical importance while developing a web server based gateway to ensure that only the authorized users have access to the network of DC-DC power converters. Two methods

of security are provided in this application. The first method allows the server to authenticate web requests based upon the IP address of the requestor. If this feature is turned on, for any request that arrives with an IP address not specified in the IP address list, the connection is reset. The second method, basic authentication, as specified in RFC 1945, provides the ability to force the browser to authenticate itself for specific web pages. The authentication consists of a user name and a password which is 64 bit encoded.

4.4.2 Command Interpretation Table

The local CAN network of the power converter uses an extended CAN frame format for all the message types. The 29 bit message identifier field is broken down into a 3 bit message type field and a 26 bit address field. Three message types are currently defined. Current share messages as shown in Figure 23 are transmitted by each power converter in the power system every 10 milliseconds. This complete frame is embedded in the data field of a TCP/IP packet and then sent to the client side every 2 seconds. On the remote terminal side, voltage and current values for each power converter are parsed from the HTTP packet and are interpreted with the CAN message format specified in Figure 23.

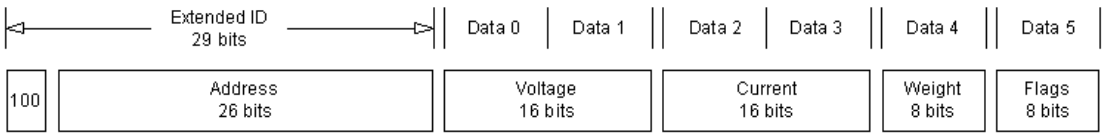


Figure 23: Current Share Message

For changing the current weight and voltage set point from the remote terminal, current or voltage parameter values set by the user for a particular converter are

submitted as Forms to the server via a web browser. On the server side, by referring to a look up table, equivalent CAN message is built with the values sent by the browser. This message is then transmitted over the local CAN network. The CAN message formats for voltage set point and current weight changes are shown in Figure 24.

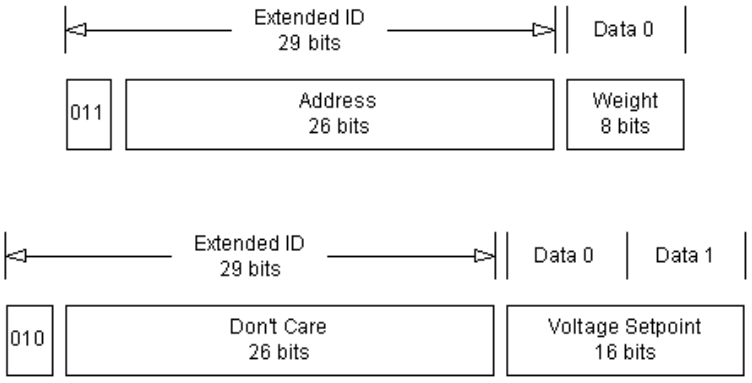


Figure 24: Current Weight and Voltage Set point Change

4.4.3 The Applet Module

On the client side, the graphical user interface (GUI) is developed using a Java applet embedded in a HTML page. This interface provides the ability to display the voltage and current values for each converter on the local CAN network in a graphical manner and also allows the users on the remote terminal to modify the real time parameters of the power system over the Internet. This applet displays the converter data in the form of advanced dynamic graphics for quick and better visual indication of the current state of the power system. A Java graphics toolkit called “EspressChart” is used to build these charts programmatically. EspressChart is a pure Java software tool, which has the flexibility to run on nearly any Java platform. Chart designer API classes of EspressChart provide an easy to use Java package which is used in GUI development for

the application. The use of this API makes it possible to include dynamic graphical charts in the user interface of this application. A detailed documentation of dial chart EspressChart API is provided in Appendix D.

4.5 Software Organization

In the previous section, several features of the web server utilized in this application were discussed. The overall organization of the application software can be descried as follows: The system initialization mainly includes the configuration of the web server parameters, timers and interrupts. An interrupt occurs every time a CAN packet is received on the CAN interface of the gateway. The data in this CAN packet is saved in a file. Every 2 seconds, the periodic system timer interrupt occurs and this saved CAN data is sent over to the remote terminal as a HTTP packet using the server push feature. Whenever data is received from the remote terminal, it is interpreted by the command table as a user command and an equivalent CAN command is then sent over the CAN bus. In terms of real time operating system, there are three independent tasks, each with unique priorities. These tasks wait for different events to occur. Task_CAN_Write is a task which is signaled every time a CAN receive interrupt occurs. This task writes the data received on the CAN bus into a file to be read later by the task which sends the data over the HTTP. Task_Senddata_HTTP is signaled by a periodic timer interrupt which occurs every 2 seconds. Whenever this task is ready to run, the most recent data received on the CAN bus is read from the history file and sent to the remote web browser. The highest priority task is called Task_User_Command and it is

signaled if a form is submitted by the user from the remote terminal via a web browser. It takes action depending on the command submitted by the user. It can be either current weight change for individual power converter or voltage set point change for the entire power system. The values submitted by the user via web browser need to be parsed and then passed to the Task_User_Command by the CGI script function.

4.6 Remote User Interface implemented in a Java Applet

The main functions of the HTTP server are to allow the remote user access to the gateway database and to provide a dynamic user interface with reliable two-way communication for requesting and viewing the data. Development of an interactive graphical user interface is an important design component in the software development of the CAN to Ethernet gateway. Hypertext Markup Language (HTML) is the basic language used to write web pages. HTML is parsed by any standard web browser when a web page downloads. It consists of tags (commands to tell the browser how to render the text, where to load in graphics, etc. on the web page) and the actual text. It is not a full-blown programming language and is essentially static in nature. A good choice to design dynamic and interactive web pages is the use of Java applets. Java applets are fully self-contained programs which can be embedded in a HTML page. The applets embedded in a web page can then be downloaded to a web browser from a web server and executed locally on the browser. As Java has better GUI tools and classes which are useful for developing graphics and network enabled applications, it is much easier to design and develop an interactive and user friendly graphical use interface using Java applets.

In this application, Micro-web, the web server running on RTXC Quadnet downloads a Java applet to a browser and then continuously sends data such as voltage and current of an individual power converter to the applet. The applet then graphically displays the data and controls such as dial scale, sliders, text box and submits buttons which are used to control hardware managed by the embedded system running Micro-web. Java can be easily used with Micro-web and RTXC Quadnet by developing a Java applet and then referencing it in a Micro-web HTML page via <applet> tag. In this application, the applet is compiled into a virtual file for a fully diskless Java enabled application. For a diskless application, the web page source files are first converted to C data structures using HTML to C compiler, and then references to each web page name, C data structure name, and C data structure size are added to the virtual file table of the web server. As shown in Figure 25, the main control panel is designed with a Java applet embedded in a HTML. This panel displays measurement results in the form of a dial scale for each power converter over the CAN network. Remote users can change current weight for each converter and voltage set point for the entire power system with this interface.

At run-time, the converter data transmitted by the web server is parsed by the applet running on the browser side. This data is then passed directly into the chart function of the applet and the dynamic charts are created on the fly. The Jviewer component loads these charts directly, allowing the users to interact with the chart in real time, and showing live up-to-the-minute data.

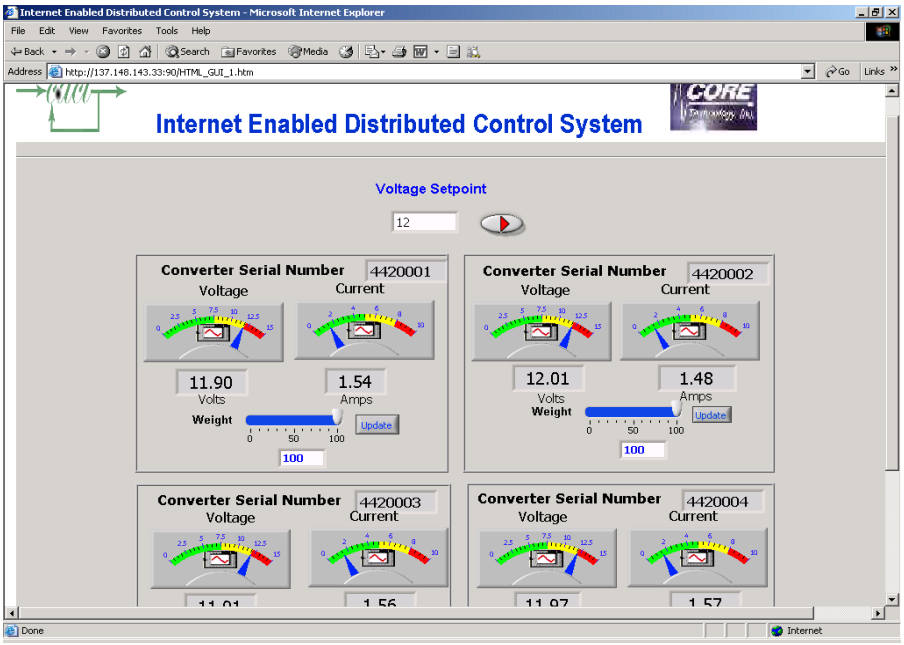


Figure 25: Graphical User Interface using Java, HTML and JavaScript

In this graphical interface, depending on the data received for each power converter of the power system, dial charts are updated. Slider and other graphical components such as text box and buttons are implemented using Java Swing class. With the help of this user interface, the concept of remote monitoring and external control of the power converter system setup over the Internet was successfully verified.

CHAPTER V

CONCLUSIONS AND FUTURE RESEARCH

The results of this research work demonstrate that the ADRC provides a promising alternative to the classical control theories for controlling a nonlinear plant like a DC-DC power converter. It has good disturbance rejection, small steady state error, and fast dynamic response. The cost of implementation is an important factor for any commercial product. This requirement is fulfilled with low cost microcontroller based hardware implementation of the controller. An additional effort has been made in this research work to enable online tuning of the converter controller over the Internet. The communication capability of the local CAN network of power converters is enhanced with the development of CAN to Ethernet gateway development. This gateway is implemented with the microcontroller based embedded web server using C language. A graphical user interface is developed using Java and HTML to facilitate online tuning and remote monitoring over the Internet for each converter connected to the local CAN network.

This research work is done as an attempt to build a framework for intelligent PMAD health monitoring problem. System level diagnostic functions such as fault detection and remediation require intelligent power subsystems, which can communicate relevant information to the other power systems. With embedded Internet connectivity, it is much easier to monitor and share the information related to each individual power converter in a power system. For other system functions such as energy and load management, the digital nature of the advanced control algorithm allows the control loop to determine the type of loads that are present and how their behavior is changing. To further improve this framework for the health monitoring system, following steps are recommended.

1. The single most important information to determine the health of an individual converter is its temperature. There is little information concerning the health of the physical power converter that can be obtained from the control loop itself. On-chip temperature sensor available on the microcontroller 56F8323 can be used to monitor temperature of the individual power converter and thermal limits can be programmed to add in thermal shutdown feature to the converter.

2. Analyzing load connected to the power system can provide useful information regarding dynamic behavior of the system. An artificial step can be introduced in the power system once in a while and response of the system can be recorded and analyzed for determining the health of the overall power system.

3. Recording the history of the converter would provide an effective tool for finding out symptoms of the converter before it fails. Currently data from the

converter (voltage and current) is sampled every 2 seconds. This data can be saved on a remote terminal over a period of time and can be used for system analysis.

4. At present, only secondary part of the power converter is digitally controlled. A fully digitally controlled power converter, with digital secondary as well as primary, would allow to add in features such as auto calibrate for the converters in the power system.

REFERENCES

1. Perry Schugart, “Reduce Time to Market with Rapid prototyping of high power converters,” Power Electronic Systems, American Superconductor Corporation, Middleton, WI 53562.

2. Artesyn Technologies, “Power Conversion Trends,” A mid update from Artesyn Technologies, 2004.

3. Robert M. Button, Center Peter, E. Kascak, and Ramon Lebron-Velilla, “Digital Control Technologies for Modular DC-DC Converters,” NASA TM-211370, 2002.

4. Guoshun Yang, Jie Wu, Yuanrui Chen, and Norbert Cheung, “An automatic disturbance rejection controller for matrix converter,” Proc. of Power Electronics Systems and Applications Conference, November 2004, pp. 119-124.

5. Petri Vallittu, Teuvo Suntio, Seppo J. Ovaska , Oyj, “Digital Control of Power Supplies Opportunities and Constraints,” Helsinki University of Technology, Helsinki, Finland.

6. Idris Gadoura, Kai Zenger, Teuvo Suntio, and Petri Vallittu, “New Methodology for Design, Analysis, and Validation of DOC Converters Based on Advanced Controllers,” Telecommunications Energy Conference, 1999, pp. 454.

7. Kurokawa, F., Sato, T., Matsuo, H., and Eto, H., “Output characteristics of DC-DC converter with DSP control,” Telecommunications Energy Conference, 2002., pp. 421 – 426.

8. Tarun Gupta, R. R. Boudreaux, R. M. Nelms, and John Y. Hung, “Implementation of a Fuzzy Controller for DC–DC Converters Using an Inexpensive 8-b Microcontroller,” Industrial Electronics, IEEE Transactions onVolume 44, Issue 5, Oct. 1997, pp. 661 – 669.

9. Ram ́on Leyva, Luis Mart ́ýnez-Salamero, Bruno Jammes, Jean Claude Marpinard, and Francisco Guinjoan, “Identification and Control of Power Converters by Means of Neural Networks,” IEEE Transactions, Volume 44, Issue 8, Aug. 1997, pp. 735 – 742.

10. Schutten, M.J. and Torrey, D.A , “Genetic Algorithms for Control of Power,” Power Electronics Specialists Conference, 1995,pp. 1321 - 1326.

11. Gupta, T., Boudreax, R.R., Nelms, R.M., and Hung, J.Y. , “Digital control of a Z V S full-bridge DC-DC converter,” Proceedings of the IEEE Applied Power Electronics conference, 1995, pp. 687493.

12. Guang Feng, Wanfeng Zhang, and Yan-Fei Liu, “An adaptive digital controller for switching DC-DC converters,” Proceedings of the IEEE international Energy conversion conference, 1991, pp. 507-511.

13. Hau-Chuen Chan', K.T. Chau', and C.C. Chant, “ A neural network controller for switching power converters,” Power Electronics Specialists Conference,. PESC '93 Record, 24th Annual IEEE, 20-24 June 1993, pp. 887 – 892.

14. Guang Feng, Wanfeng Zhang, and Yan-Fei Liu, “A New Current Mode Fuzzy Logic Controller with Extended State Observer for DC-to-DC Converters,” Applied Power Electronics Conference and Exposition, 2004, Nineteenth Annual IEEE Volume 3, 2004, pp. 1771 – 1777.

15. A. Rubaai and A. Ofoli, “A Hardware implementation of an adaptive network-based fuzzy controller for DC-DC converters,” Industry Applications Conference, 2004. Record of the 2004, IEEE Volume 4, 3-7 Oct. 2004 pp. 2623 - 2629.

16. G.Escobar, R Ortega, H.Sira-Ramirez, J-P. Vilain, and I.Zein, “An experimental comparison of several nonlinear controllers for power converters,” Proceedings of the IEEE Industrial Electronics, 1997, pp. 5589-5594.

17. Bosheng Sun, “DSP based Advanced Control Algorithms for a DC-DC Power Converter,” Master’s Thesis, Cleveland State University, Cleveland, OH, June 2003.

18. Zhiqiang Gao, Yi Huang, and Jingqing Han, "An Alternative Paradigm for Control System Design," Proc. of the 40th IEEE Conference on Decision and Control, Orlando,FL, December 2001, pp. 4578-4585.

19. WungHoa Park, KiWoong Seong, ManWoo Lee, and JongMan Yang, “A DC/DC converter test board and control system,” Proc. of ICALEPCS, 2003, Gyeongju, Korea.

20. Michael Gray, Zhiqiang Gao, and Robert Button, “Distributed, Master-less Control of Modular DC-DC Converters,” 2nd International Energy Conversion Engineering Conference, 2004, pp. 1-9.

21. Zhiqiang Gao, “ On Practical Applications of the Active Disturbance Rejection Control Design Strategy,” Department of Electrical and Computer Engineering,Cleveland State University,Cleveland, OH 44115.

22. Zhiqiang Gao, "Scaling and Bandwidth-Parameterization Based Controller Tuning," Proc. of the 2003 American Control Conference, June 2003, pp.4989-4996.

23. Radke Aaron, “A nonlinear enhancement to disturbance estimation,” Cleveland State Univeristy, Cleveland,OH 44114.

24. B. Sun and Z. Gao, “A DSP-Based Active Disturbance Rejection Control Design for a 1KW H-Bridge DC-DC Power Converter,” IEEE Transactions on Industrial Electronics, Volume 52, No. 5,October 2005.

25. L. Wuidart, “Topologies for switched mode power supplies,” STMicroelectronics, Application note.

26. Bosheng Sun, “Cleveland State University-Core Technlogy Project Final Report”, Cleveland State University, Cleveland OH 44114.

27. Robert Button, “Intelligent Systems for Power Management and Distribution,” NASA TM-211370, 2002.

28. J. Han, “A Class of Extended State Observers for Uncertain Systems”, Control and Decision, Vol.10, No.1, 1995, pp.85-88, (In Chinese).

29. R. Miklosovic, A. Radke, and Z. Gao, "Discrete Implementation and Generalization of the Extended State Observer,” Proc. of American Control Conference, June 14-16, 2006.

APPENDICES

A. S- Function for Discrete Linear ESO using fixed point mathematics

```
/* title:LADRC For a non-unity gain and control signal saturation 2nd
Order Plant Using Backward Euler Integration
 * author:Rob Miklosovic modified by Aaron Radke and Madhura Shaligram
 * date:2/23/05
 *   rewrite the lardc24bs sfunction to include b0 and saturations and
outputs
 * abstract:There are 2 inputs, R and Y.  There are 4 outputs, u, z1,
z2,and z3.  There is 1 parameter vector, [wc,wo,T,b0,uMin,uMax], where T
is the step size.
 */
#define S_FUNCTION_NAME ladrc24bs
#include "simstruc.h"
#include "math.h"
#include "fixedpoint.h"
double wc,wo,T,b0,uMin,uMax,B3;
long B1,B2,Kp,Kd,ec;
float eo;
int uOut,lx1;
```

```
long fs,lx2,lx3fs,lx3b0,lec,leo,lB1,lB2,lB3,lB3_1,lu0;

static void mdlInitializeSizes(SimStruct *S)

{

   ssSetNumContStates(    S, 0);                /* number of continuous states */

   ssSetNumDiscStates(    S, 4);                /* number of discrete states */

   ssSetNumInputs(        S, 2);                /* number of inputs */

   ssSetNumOutputs(       S, 4);                /* number of outputs */

   ssSetDirectFeedThrough(S, 0);                /* direct feedthrough flag */

   ssSetNumSampleTimes(   S, 1);                /* number of sample times */

   ssSetNumInputArgs(     S, 1);                /* number of input arguments */

   ssSetNumRWork(         S, 0);        /* number of real work vector elements */

   ssSetNumIWork(         S, 0);                /* number of integer work vector

elements */

   ssSetNumPWork(         S, 0);                /* number of pointer work vector

elements */

}

#define params           ssGetArg(S,0)                  /* Input Arguments */

static void mdlInitializeSampleTimes(SimStruct *S)

{
```

```
    T=mxGetPr(params)[2];
    ssSetSampleTimeEvent(S, 0, T);
    ssSetOffsetTimeEvent(S, 0, 0.0);
}
static void mdlInitializeConditions(double *x, SimStruct *S)
{
    x[0]=0.0;
    x[1]=0.0;
    x[2]=0.0;
    x[3]=0.0;
wc=mxGetPr(params)[0];
wo=mxGetPr(params)[1];
/*T is the [2] parameter*/
b0=mxGetPr(params)[3];
uMin=mxGetPr(params)[4];
uMax=mxGetPr(params)[5];
uOut=0.0;
B1=3*wo;
B2=3*wo*wo;
```

```
B3=wo*wo*wo;

Kp=wc*wc;

Kd=2*wc;

fs=1/T;

lB1=B1;

lB2=B2;

lB3=B3/fs;

lB3_1=B3/(fs*fs);

}

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)

{

        if(ssIsSampleHitEvent(S,0,tid)) {

                leo=u[1]-lx1;

                eo= leo;


        lx1=lx1+lx2/fs+(16*leo)/10;

        lx2=lx2+ lx3fs +41666*leo+16392*x[0];

        lx3fs=lx3fs+7716*leo;  //B3*e0/(fs*fs) i.e. B3/(fs*fs)=25000^3/(50000)^2

        lx3b0=(lx3b0 + 12784*leo);
```

```
            x[1]=lx1;

            x[2]=lx2;

            x[3]=x[3]+B3*leo/fs;

        //Controller PArt

            lec=u[0]-lx1;       /*r-z1*/

             lu0=5753*lec - lx2;       /* PD control law *///u0 sqroot b0=(Kp*lec -
Kd*lx2)/sq root b0;

            uOut=(lu0 -  lx3b0)/27160;

          /*saturation*/

            if(uOut > uMax)

            uOut = uMax;

            if(uOut < uMin)

            uOut = uMin;

            x[0]=uOut;


        }

    }

    static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,

    int tid)

    {
```

```
    y[0]=x[0];  /*u*/                    /* ADRC Extended State Feedback */

    y[1]=x[1];  /*z1*/

    y[2]=x[2];  /*z2*/

    y[3]=x[3];  /*z3  f */

}

static void mdlDerivatives(double *dx, double *x, double *u, SimStruct

*S, int tid)

{}

static void mdlTerminate(SimStruct *S)

{}

#ifdef  MATLAB_MEX_FILE

#include "simulink.c"

#include "fixedpoint.c"

#else

#include "cg_sfun.h"

#endif
```

B. C Code of ADRC for 56F8323

```
**  Filename  : digital_control.C
**  Project   : digital_control
**  Processor : 56F8323
**  Version   : Driver 01.03
**  Compiler  : Metrowerks DSP C Compiler
**  Date/Time : 3/2/2005, 11:03 AM
**  Abstract  : Active Disturbance Rejection Control with CAN bus communication, no secondary current limit
**      Main module.
/* MODULE digital_control */
/* Including used modules for compilling procedure */
#include "Cpu.h"
#include "Events.h"
#include "AD1.h"
#include "Bit1.h"
#include "SM1.h"
#include "Control_Timer.h"
#include "Transmit_Timer.h"
```

```
#include "PI1.h"

/* Include shared modules, which are used for whole project */

#include "PE_Types.h"

#include "PE_Error.h"

#include "PE_Const.h"

#include "IO_Map.h"

/* Include CAN modules */

#include "CAN_Driver.h"

#include "digital_control.h"

//Global variables required for ADRC//

long wc=12500,wo=25000,b0=737665600,b0sqrt=27160,uMin=0,uMax=2500;

long B1,B2,Kp,Kd;

double B3;

long lx1;

long fs=45000,lx2,lx3fs,lx3b0,leo,lB1,lB2,lB3,lB3_1,lu0;

long lec=0,r,y,uOut;

int u;

//Global variables for ADRC end here
```

```
// the following values need to be assigned
// address
// address range: 0 - 67,108,863 [0 - 3FFFFFF]
const unsigned long addr = 4420001;
// assumed values for max_ntwk size and max_V_setpoint
// may change as necessary
// 4096 = max 12 bit A-D value
// corresponds to roughly 19.15 V
const int max_V_setpoint = 2850; //Vout=14V
const int min_V_setpoint = 1050;
int V_setpoint = 2450, Vsetpoint_adj=0, Vsetpoint_final=0;
int I_setpoint = 0;
int I_max = 1800;
int Kidt=100,iKp=10;
int ADC_result[2];
int V_out=0, I_out=0;
int V_Error=0, I_Error1=0, I_Error2=0;
int count = 0;
int weight = 100, weight_total = 0;
```

```
int pos,P,PID;

long Sum=0;

int I;

long I_accum = 0;

long I_avg, I_tot;

// boolean variable used to turn unit on/off remotely

// via CAN bus.  this variable is connected to CAN

// protocol, but not to control function yet.

unsigned int status = 0x8000;

void main()

{

PE_low_level_init();

/*** End of Processor Expert internal initialization.                    ***/

// CAN initialization

// Control_Timer events disabled to ensure initialization completes

Control_Timer_DisableEvent();

canInit();

//ADRC Initial Calculations

B1=3*wo;
```

```
B2=3*wo*wo;

B3=wo*wo*wo;

Kp=wc*wc;

Kd=2*wc;

lB1=B1;

lB2=B2;

lB3=B3/fs;

//ADRC Initial Calculations end here

Control_Timer_EnableEvent();

for(;;)

 {

        // infinite empty for loop

        // all functionality driven

        // by interrupts

 }

}
```

```
        /*****************************************************************/
        // ADRC function
        /*****************************************************************/
        #pragma interrupt called
        void ADRC_Control()
        {
              // ADC
AD1_Measure(1);
                AD1_GetValue(ADC_result);
                V_out = ADC_result[0]>>2;
                V_out = V_out-4050;
                y=(long)V_out;
                r=(long)V_setpoint;
                leo=y-lx1;
                lx1=lx1+lx2/fs+((long)16*leo)/(long)10;
                lx2=lx2+ lx3fs +(long)41666*leo+(long)16392*(long)u;
;//B3*e0/(fs*fs)i.e.    B3/(fs*fs)=25000^3/(50000)^2
lx3fs=lx3fs+(long)7716*leo
```

```
                lx3b0=lx3b0 + (long)12784*leo;

                lec=(r-lx1);        /*r-z1*///lec is int=2000 initially

                //Sum =(long)100*(long)lec; // Sum = 200000; //Sum is long

                //I = Sum/Kidt; //I is int

               lu0=(long)5753*lec - lx2;

                /* PD control law *///u0 sqroot b0=(Kp*lec - Kd*lx2)/sq root b0;

                uOut=(lu0 -  lx3b0)/(long)27160;

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

                 /*saturation*/

                if(uOut > uMax)

                uOut = uMax;

                if(uOut < uMin)

uOut = uMin;

                u=uOut;

                /* DAC */

                Bit1_SetVal();

                asm(nop);

                Bit1_ClrVal();

                SM1_SendChar(4095-u);
```

```
}
/*************************************************************/
// Transmit voltage and current data
/*************************************************************/
// transmits voltage and current across CAN network.  used for current
// sharing and performance monitoring.
#pragma interrupt called
void transmitData() {
        // calculate average current
        I_avg = I_tot / count;
        // zero the total variables, count variable
        I_tot = 0;
        count = 0;
        // calculate average current for network
        // average current is I_setpoint
        weight_total = weight_total + weight;
        I_accum = I_accum + I_avg;
```

```
        I_setpoint = (I_accum * weight) / weight_total;
        I_Error2 = I_setpoint - I_avg;
        if (I_Error2 > 10) Vsetpoint_adj = Vsetpoint_adj + 1;
        if (I_Error2 <-10) Vsetpoint_adj = Vsetpoint_adj - 1;
        // zero accumulators
        I_accum = 0;
        weight_total = 0;
        pos = 0;
        // set weight in status
        status = weight;
        //status = Kidt;
        //pwm= PID_temp ;
        // transmit voltage & current values
        canTX(V_out, I_avg, addr,u,lec);
        }
/*****************************************************************/
// Update current function
/*****************************************************************/
// accumulates total current and counts other power converters on network
```

```
// used for current sharing

#pragma interrupt called

void update_I_total(int current, int volts, unsigned long address,

                                    unsigned int rx_status)

{

        long rx_weight;

        rx_weight = rx_status & 0xff;

        if (address != addr) {

                I_accum = I_accum + current;

                weight_total = weight_total + rx_weight;

                if (address < addr)

                        pos ++;

        }

}

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

// Voltage setpoint update function

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

// update voltage setpoint remotely via CAN network

#pragma interrupt called
```

```
void update_V_setpoint(int vSetpoint) {
        if (vSetpoint >= min_V_setpoint && vSetpoint <= max_V_setpoint) {
                V_setpoint = vSetpoint;
        }
}
/*******************************************************/
// Weight update function
/*******************************************************/
// update weight remotely via CAN network
#pragma interrupt called
void update_weight(int new_weight, unsigned long address) {
        if (address == addr) {
                if (new_weight <= 255)
                        weight = new_weight;
        }
}
/*************************************************************/
// Kidt update function
/*************************************************************/
```

```
// update Wo / Kidt remotely via CAN network

#pragma interrupt called

void update_Kidt(int new_Kidt, unsigned long address) {

if (address == addr)

        {

                        //Inverse Part

                        //IWo = new_Kidt;

                        //IB1 = IWo/3;

                        //IB2 = IWo*IWo/3;

                        //IB3 = IWo*IWo*IWo;


                        Kdadrc=new_Kidt;

                        z1=0;

                        z2=0;

                        z3=0;

                        //Non Inverse Part

                        //Wo=new_Kidt;

                        //B1 = Wo*3;

                        //B2 = 3*Wo*Wo;
```

```
                        //B3 = Wo*Wo*Wo;

                        //z1=0;

                        //z2=0;

                        //z3=0;

        }

}

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

// Kp update function

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

// update Wc remotely via CAN network

#pragma interrupt called

void update_Kp(int new_Kp, unsigned long address) {

if (address == addr)

        {

        //Inverse Part

                        //IWc=new_Kp;

                        Kpadrc=new_Kp;

                        //IKpadrc = IWc*IWc;

                        //IKdadrc= IWc/2;
```

```
                    //Non Inverse Part
                    //Wc=new_Kp;
                    //Kpadrc = Wc*Wc;
                    //Kdadrc = Wc*2;
                    //Kp=new_Kp;
                    z1=0;
                    z2=0;
                    z3=0;
            }
    }
    /* END digital_control */
```

C. C Code for Embedded Web Serer

```
/* ************************************************/
*  CAN_Ethernet.c
*Original Code Copy Righted by Quadros Systems, Inc. 2003
*Code modified for this application by M. Shaligram
#include "webapi.h"
#include "CAN_Ethernet.h"
/* ************************************************** */
; /* IP address of the microcontroller board to be used as web server */
byte KS_FAR my_ip_address[IP_ALEN]   =    {137,148, 143,74}
/* gateway's ip address - all 255's means no gateway   */
byte KS_FAR ip_gw_address[IP_ALEN]   =    {137,148,142,1;
/* IP mask address */
byte KS_FAR ip_mask_address[IP_ALEN] =    {255,255,254,0};
RTIP_BOOLEAN KS_FAR use_dhcp  = FALSE;
void app_entry(void);
int ks_kernel_init(void);
#if (INCLUDE_WEB_SECURITY)
extern browser browser_info;
```

```
        #endif


        #if (INCLUDE_WEB_AUTHENTICATION)
        extern struct web_auth_entry auth_info;
        #endif
        #if (INCLUDE_WEB_EXTRA_MIME)
        extern MIME_FOR_PAGES mime_page_info;
        #endif
        /*
**********************************************************************  */
        /* This task runs the application. It is responsible for
        1. Sense and store power converter data on CAN network,
        2. Transfer this data to higher network layers on Ethernet,
        3. Accept and translate user network commands,
**********************************************************************  */
        void app_entry(void) /* __fn__ */
        {
          char io_buf[80];
          int app_status = 0;
          if (ks_kernel_init())
```

```
        {
          for(;;)
            asm(NOP);
        }
        /* register the dhcp components */
      #if (INCLUDE_DHCP_CLI)
        XN_REGISTER_DHCP_CLI();
      #endif
        /* register the web server */
      #if (INCLUDE_WEB)
        XN_REGISTER_WEB_SRV();
      #endif
      tm_puts("\n");
tm_puts("  *  Web Server Initialized  *");
      /* register callbacks before initializing rtip in case an error occurs */
      register_callbacks();
      /*
**************************************************************************  */
        if (xn_rtip_init() != 0)
          exit_clisrv();
```

```
/* Initialize the virtual file system */
if (vf_init())
{
tm_puts("Initialization of virtual filesystem failed. Exiting program.");
exit_clisrv();
}
/* Get IP address info */
get_ip_address_source();
/* Open the interface - i.e. do xn_interface_open or xn_attach */
while ((app_status = app_interface_open()) <= 0)
{
   if (app_status != -2)
   {
      tm_puts("Could not open the Ethernet interface. Exiting program.");
      exit_clisrv();
   }
}
 /* print out my IP address */
```

```
    display_my_ip();


    /* set the system time */
    set_time();
#if (INCLUDE_WEB)
    /* Add virtual files to the memory file system */
    populate_ram_disk(&virtual_file_table[0]);


#if (INCLUDE_WEB_SECURITY)
    /* first configure the security settings for the server */
    if (http_set_browser_list((PFBROWSER)&browser_info, 2) < 0)
    {
        DEBUG_ERROR("http_set_browser_list failed", NOVAR, 0, 0);
    }
#endif


#if (INCLUDE_WEB_AUTHENTICATION)
    if (http_set_auth(&auth_info, sizeof(auth_info)/sizeof(web_auth_entry)))
    {
```

```
            DEBUG_ERROR("http_set_auth failed", NOVAR, 0, 0);
        }
    #endif
    #if (INCLUDE_WEB_EXTRA_MIME)
        http_mime_fields((PFMIME_FOR_PAGES)&mime_page_info);
    #endif
        http_set_post_function_list(&post_function_table[0]);
        http_set_get_function_list(&get_function_table[0]);
        /* start the web server */
        start_web_server();
        tm_puts("The WEB task is running in the background.");
        tm_puts("You can now send data requests to CAN to Ethernet gateway from a
browser.");
    #endif
        /* Beginning of main shell */
        while (1)
        {
            /* print out my IP address */
            display_my_ip();
```

```
        display_tests();
        store_CAN_data();
    }
}
void exit_clisrv(void)
{
    int error;
    PFCCHAR errorstring;
    error = xn_getlasterror();
    errorstring = xn_geterror_string(error);
    tm_printf("\nQuadnet returned the following error:\n");
    tm_printf("%s\n", errorstring);
    tm_puts("Exiting program");
    app_interface_close();
    xn_rtip_exit();
    for (;;)
    {
      asm { nop }; /* spin here - resolve error */
    }
```

```
}


/* end of file – CAN_Ethernet.c */

/*                                                                  */
```

D. Java code for EspressChart API

```
// No DrillDown, Time-Based Zooming, or Histogram
// available for Dial Charts
// set dial clock area color
IAxis hYAxis = chart.gethYAxis();
hYAxis.setColor(Color.white);
// set ticker marks invisible
hYAxis.setTickersVisible(false);
// set labels out side of the dial
hYAxis.setLabelOutsidePlotArea(true);
// show labels
hYAxis.gethLabel().setVisible(true);
// set scale
hYAxis.setScaleAutomatic(false);
hYAxis.setMinScale(new Integer(10));
hYAxis.setMaxScale(new Integer(110));
hYAxis.setScaleStep(new Integer(20));
// set dial chart starting & ending angles (in degrees)
chart.gethDialProperties().setStartAngle(-90.0);
chart.gethDialProperties().setEndAngle(90.0);
```

```
// set Needle Options (1.0 == needle radius)
// setNeedleLength (index of needle, ratio)
chart.gethDialProperties().setNeedleLength(0, 0.8);

// set Line thickness
chart.gethDataPoints().setLineThickness(2); // pixels
chart.gethYAxis().setArrowhead(true);

// set center point,
// setCenterPointRadius( ratio of point radius : dial radius)
chart.gethDialProperties().setCenterPointRadius(0.1);
chart.gethDialProperties().setCenterPointColor(Color.black);

// set control ranges
// ControlRange(double startScale, double endScale,
//  java.awt.Color color, java.lang.String title,
//  int thickness, int offset, java.awt.Point center,
//  boolean drawBorder, boolean showInLegend)
IControlRangeSet hCRanges = chart.gethControlRanges();

// draw the background of the dial chart
hCRanges.addElement( new ControlRange(10, 110, Color.black, "", 5, 0, null, false,
false));

// block out the bottom half of the dial with black color
hCRanges.addElement( new ControlRange(110, 10, Color.black, "", 100, 0, null, false,
false));
```

```
// range of 10 to 45 is cyan color
hCRanges.addElement( new ControlRange(10, 45, Color.cyan, "", 90, 10, (new Point(0,
2)), false, false));

// range of 45 to 70 is yellow
hCRanges.addElement( new ControlRange(45, 70, Color.yellow, "", 90, 10, null, false,
false));

// range of 70 to 90 is orange
hCRanges.addElement( new ControlRange(70, 90, Color.orange, "", 90, 10, null, false,
false));

// range of 90 to 110 is red
hCRanges.addElement( new ControlRange(90, 110, Color.red,"", 90, 10, null, false,
false));

// put a white color half pie on the dial chart, smaller radius
// than the control areas, to create the gradual slimming effect
// note the center is shifted.
hCRanges.addElement( new ControlRange(10, 110, Color.white, "", 70, 30, (new
Point(15, 0)), false, false));

// No DrillDown, Time-Based Zooming, or Histogram
// available for Dial Charts

// set dial clock area color
IAxis hYAxis = chart.gethYAxis();
hYAxis.setColor(Color.white);
```

```
// set ticker marks invisible
hYAxis.setTickersVisible(false);

// set labels out side of the dial
hYAxis.setLabelOutsidePlotArea(true);

// show labels
hYAxis.gethLabel().setVisible(true);

// set scale
hYAxis.setScaleAutomatic(false);
hYAxis.setMinScale(new Integer(10));
hYAxis.setMaxScale(new Integer(110));
hYAxis.setScaleStep(new Integer(20));

// set dial chart starting & ending angles (in degrees)
chart.gethDialProperties().setStartAngle(-90.0);
chart.gethDialProperties().setEndAngle(90.0);

// set Needle Options (1.0 == needle radius)
// setNeedleLength (index of needle, ratio)
chart.gethDialProperties().setNeedleLength(0, 0.8);

// set Line thickness
chart.gethDataPoints().setLineThickness(2); // pixels
chart.gethYAxis().setArrowhead(true);

// set center point,
// setCenterPointRadius( ratio of point radius : dial radius)
```

```
chart.gethDialProperties().setCenterPointRadius(0.1);
chart.gethDialProperties().setCenterPointColor(Color.black);

// set control ranges
// ControlRange(double startScale, double endScale,
//  java.awt.Color color, java.lang.String title,
//  int thickness, int offset, java.awt.Point center,
//  boolean drawBorder, boolean showInLegend)
IControlRangeSet hCRanges = chart.gethControlRanges();

// draw the background of the dial chart
hCRanges.addElement( new ControlRange(10, 110, Color.black, "", 5, 0, null, false,
false));

// block out the bottom half of the dial with black color
hCRanges.addElement( new ControlRange(110, 10, Color.black, "", 100, 0, null, false,
false));

// range of 10 to 45 is cyan color
hCRanges.addElement( new ControlRange(10, 45, Color.cyan, "", 90, 10, (new Point(0,
2)), false, false));

// range of 45 to 70 is yellow
hCRanges.addElement( new ControlRange(45, 70, Color.yellow, "", 90, 10, null, false,
false));
```

```
// range of 70 to 90 is orange
hCRanges.addElement( new ControlRange(70, 90, Color.orange, "", 90, 10, null, false,
false));

// range of 90 to 110 is red
hCRanges.addElement( new ControlRange(90, 110, Color.red,"", 90, 10, null, false,
false));

// put a white color half pie on the dial chart, smaller radius
// than the control areas, to create the gradual slimming effect
// note the center is shifted.
hCRanges.addElement( new ControlRange(10, 110, Color.white, "", 70, 30, (new
Point(15, 0)), false, false));
```

E. RTOS Fundamentals

Basic Terms

A *task* is a sequence of instructions, sometimes done repetitively, to perform an action (For example: read a keypad, display a message on an LCD).In other words, it is usually a small program inside a bigger one.

A task's *priority* suggests the task's importance relative to other task. It may be fixed or variable, unique or shared with other tasks.

An event is an occurrence of something (e.g. a key was pressed, an error occurred) that a task can wait for.

In order to multitask, such that all tasks appear to run concurrently, some mechanism must exist to pass control of processor and its resources from one task to another. This is called *task switching* . When one task suspends running, this task's context (generally the complete contents of the stack and the values of the registers) is usually saved for reuse during later execution.

Task switching is the job of *scheduler*. It suspends one task and resumes another when certain conditions are met. The faster the scheduler is able to switch tasks, the better the performance of the overall application, since time spent in switching task is the time spent without any tasks running.

In *Preemptive approach* for task switching a running task is interrupted by scheduler whenever another high priority task is ready to run. In *co-operative approach*, a task must voluntarily relinquish control of the processor to scheduler before another

task may run. This means even if a high priority task is made ready to run, it has to wait till current running task gives control back to scheduler so that scheduler can look for highest priority ready to run task and execute it.

Intertask communication is an orderly means of passing information from one task to another following some well established programming concepts. Semaphores, messages, message queues and event flags can be used to pass information in one form or another between tasks and, in some cases, ISRs.A *task's state* describes what the task is currently doing. Tasks change from one state to another via clearly defined rules. Common task states might be ready/eligible, running, delayed, waiting, stopped and destroyed/ uninitialised.*Kernel* software consists of scheduler and services to handle intertask communication.The *timer* is another piece of software that keeps track of elapsed time and/or real time for delays , timeouts and another time-related services. The timer is only as accurate as the timer clock provided by your system.The *operating system* contains the kernel, the timer and remaining software (called services) to handle tasks and events.

F. GUI code for UDP Client

```
package gui;

import javax.swing.*;

import javax.swing.border.*;

import java.awt.*;

import java.util.*;

import java.awt.event.*;

import java.applet.*;

import java.lang.*;

import java.io.*;

import java.net.*;

import java.math.*;

public class GUIMain{

TextField
voltage,curretnt,voltage_calibrate,current_calibrate,voltage_setpoint,total_current,
total_weight,weight_factor,current_setpoint,pwnm,pulse_width,prop_gain,integra
l_gain,der_gain;

Label
lvoltage,lcurretnt,lvoltage_calibrate,lcurrent_calibrate,lvoltage_setpoint,ltotal_cur
```

```
rent,ltotal_weight,lweight_factor,lcurrent_setpoint,lpwnm,lpulse_width,lprop_gai
n,lintegral_gain,lder_gain;
Label setpara_label,monitorpara_label;
public String [] output_matrix = new String [14];
 //CheckboxGroup converter;
 ButtonGroup converter;
 JPanel main_panel,converter_panel,set_panel,monitor_panel;
 //Menu Items
 MenuBar menuBar;
 Menu menu;
 MenuItem menuItem;
 //Tool Bar
 JToolBar toolBar;
 JButton next,previous,sendt,listen,pause,resume,graph,reset;
 String imgLocation,imgLocation1,imgLocation2,imgLocation3,imgLocation4,
imgLocation5,imgLocation6,imgLocation7,imgLocation8;
Frame f = new Frame("GUI for Health Monitoring");
 //For reading data from File
 public File read_data;
 public Properties properties ;
```

```
 public FileInputStream fis;
 public String token;
//Packet Related parameters
 DatagramPacket send_packet;
 byte [] mainbuf= {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 byte []
buf1,buf2,buf3,buf4,buf5,buf6,buf7,buf8,buf9,buf10,buf11,buf12,buf13,buf14,buf
15,buf16;
 //File realted functions for send and receive operations
String filenamewrite;
 File fwrite;
 FileInputStream fisread;
 FileOutputStream foswrite;
byte [] writebyte;
byte [] readbyte;
 public GUIMain() {
        super();
        init();
  }
```

```
// For Setting Gridbag constraints in Grid bag layout
 public void init()
 {
  try
{
  //Adding GUI
main_panel = new JPanel();
main_panel.setBorder(new javax.swing.border.TitledBorder(new
javax.swing.border.EtchedBorder()));
GridBagLayout gridbag = new GridBagLayout();
//   setFont(new Font("Helvetica", Font.PLAIN, 14));
 //Menu
menuBar = new MenuBar();
//Build the first menu.
menu = new Menu("File");
menuBar.add(menu);
//a group of JMenuItems
menuItem = new MenuItem("Open File");
//menuItem.getAccessibleContext().setAccessibleDescription("Open a File");
menu.add(menuItem);
```

```
menuItem = new MenuItem("Save");

menu.add(menuItem);

menuItem = new MenuItem("Print");

menu.add(menuItem);

menuItem = new MenuItem("Exit");

menu.add(menuItem);

menu = new Menu("View");

menuBar.add(menu);

//a group of JMenuItems

menuItem = new MenuItem("Graph Editor");

menu.add(menuItem);

 menuItem = new MenuItem("Analyse");

menu.add(menuItem);

menu = new Menu("Edit");

menuBar.add(menu);

//a group of JMenuItems

menuItem = new MenuItem("Undo");

menu.add(menuItem);
```

```
 menuItem = new MenuItem("Redo");

menu.add(menuItem);

menuItem = new MenuItem("Cut");

menu.add(menuItem);

menuItem = new MenuItem("Copy");

menu.add(menuItem);

menuItem = new MenuItem("Paste");

menu.add(menuItem);

menu = new Menu("Debug");

menuBar.add(menu);

//a group of JMenuItems

menuItem = new MenuItem("Send");

menu.add(menuItem);

menuItem = new MenuItem("Listen");

menu.add(menuItem);

//menuItem = new CheckBoxMenuItem("Stop Listening");

//menu.add(menuItem);

//main_panel.add(menuBar);

main_panel.setLayout(gridbag);
```

```
 //Tool Bar
toolBar=new JToolBar();
imgLocation = "next_g.gif";
next=new JButton ("");
next.setIcon(new ImageIcon(imgLocation,"Next Converter"));
next.setToolTipText("Go to next Converter");
toolBar.add(next);
imgLocation1 = "prev.gif";
previous=new JButton ("");
previous.setIcon(new ImageIcon(imgLocation1,"Previous Converter"));
previous.setToolTipText("Go to previous converter");
toolBar.add(previous);
imgLocation3 = "run.gif";
listen=new JButton ("");
listen.setIcon(new ImageIcon(imgLocation3,"Listen"));
listen.setToolTipText("Receive data for converter");
toolBar.add(listen);
imgLocation4 = "pause.gif";
pause=new JButton ("");
```

```
pause.setIcon(new ImageIcon(imgLocation4,"Stop Listening to Port"));

pause.setToolTipText("Stop listening to port");

toolBar.add(pause);

imgLocation5 = "resume.gif";

resume=new JButton ("");

resume.setIcon(new ImageIcon(imgLocation5,"Resume Listening to Port"));

resume.setToolTipText("Resume listening to port");

toolBar.add(resume);

imgLocation7 = "finger.gif";

sendt=new JButton ("");

sendt.setIcon(new ImageIcon(imgLocation7,"Sendt"));

sendt.setToolTipText("Send to Converter");

 toolBar.add(sendt);

imgLocation8 = "reset.gif";

 reset=new JButton ("");

 reset.setIcon(new ImageIcon(imgLocation8,"reset"));

 reset.setToolTipText("Clear all parameters");

 toolBar.add(reset);
```

```
imgLocation6 = "histogram32.gif";
graph=new JButton ("");
graph.setIcon(new ImageIcon(imgLocation6,"Graph Anaylsis"));
graph.setToolTipText("Show graph analysis");
toolBar.add(graph);
//Adding toolbar to MainPanel
//toolBar.addSeparator();
gridbag.setConstraints(toolBar,new GridBagConstraints(0, 0, 2, 1, 0.0,0.0,GridBagConstraints.WEST, GridBagConstraints.NONE,new Insets(0,0, 0,500), 20, 20));
main_panel.add(toolBar);
//Adding converter panel to main panel
converter_panel = new JPanel(new GridLayout(6, 1));
converter_panel.setBorder(new javax.swing.border.TitledBorder(new javax.swing.border.EtchedBorder()));
gridbag.setConstraints(converter_panel,new GridBagConstraints(0, 1, 1, 2, 0.0, 0.0,GridBagConstraints.NORTH, GridBagConstraints.NONE,new Insets(0, 100, 0, 0), 20, 20));
main_panel.add(converter_panel);
```

```
monitor_panel = new JPanel(new GridBagLayout());
monitor_panel.setBorder(new javax.swing.border.TitledBorder(new
javax.swing.border.EtchedBorder()));
gridbag.setConstraints(monitor_panel,new GridBagConstraints(1, 1, 1, 1, 0.0,
0.0,GridBagConstraints.CENTER, GridBagConstraints.NONE,new Insets(0, 0,
30, 0), 20, 20));
main_panel.add(monitor_panel);
set_panel = new JPanel(new GridBagLayout());
set_panel.setBorder(new javax.swing.border.TitledBorder(new
javax.swing.border.EtchedBorder()));
gridbag.setConstraints(set_panel,
new GridBagConstraints(1, 2, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 0, 0));
main_panel.add(set_panel);
converter = new ButtonGroup();
JRadioButton conv1= new JRadioButton("Converter1",new
ImageIcon("converter.gif"), true);
conv1.setActionCommand("Converter1");
```

```
  JRadioButton conv2= new JRadioButton("Converter2",new
ImageIcon("converter.gif"), false);
  conv2.setActionCommand("Converter2");
  JRadioButton conv3= new JRadioButton("Converter3",new
ImageIcon("converter.gif"), false);
  conv3.setActionCommand("Converter3");
  converter.add(conv1);
  converter.add(conv2);
  converter.add(conv3);
  converter_panel.add(conv1);
  converter_panel.add(conv2);
  converter_panel.add(conv3);
  lvoltage = new Label("Voltqage in volts");
  lcurretnt = new Label("Current in amps");
  lcurrent_setpoint = new Label("Current SetPoint");
  lpulse_width = new Label("Pulse Width");
  voltage = new TextField("", 5);
  voltage.setEditable(false);
  curretnt = new TextField("", 5);
  curretnt.setEditable(false);
```

```
current_setpoint = new TextField("", 5);
current_setpoint.setEditable(false);
pulse_width = new TextField("", 5);
pulse_width.setEditable(false);
monitorpara_label=new Label("Parameters which can only be monitored");
monitor_panel.add(monitorpara_label, new GridBagConstraints(0, 0, 4, 1, 0.0,
0.0,   GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0,10, 0), 5, 5));
monitor_panel.add(lvoltage,new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
monitor_panel.add(voltage,new GridBagConstraints(1, 1, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
monitor_panel.add(lcurretnt,new GridBagConstraints(2, 1, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
monitor_panel.add(curretnt,new GridBagConstraints(3, 1, 1, 1, 0.0, 0.0,
 GridBagConstraints.CENTER, GridBagConstraints.NONE,
 new Insets(0, 0, 0, 0), 5, 5));
```

```
monitor_panel.add(lcurrent_setpoint,new  GridBagConstraints(0,  2,  1,  1,  0.0,
0.0, GridBagConstraints.CENTER, GridBagConstraints.NONE,
 new Insets(0, 0, 0, 0), 5, 5));
monitor_panel.add(current_setpoint,new GridBagConstraints(1, 2, 1, 1, 0.0, 0.0,
 GridBagConstraints.CENTER, GridBagConstraints.NONE,
 new Insets(0, 0, 0, 0), 5, 5));
 monitor_panel.add(lpulse_width,new GridBagConstraints(2, 2, 1, 1, 0.0, 0.0,
 GridBagConstraints.CENTER, GridBagConstraints.NONE,
 new Insets(0, 0, 0, 0), 5, 5));
monitor_panel.add(pulse_width,new GridBagConstraints(3, 2, 1, 1, 0.0, 0.0,
  GridBagConstraints.CENTER, GridBagConstraints.NONE,
  new Insets(0, 0, 0, 0), 5, 5));
// Row1
voltage_calibrate = new TextField("", 5);
current_calibrate = new TextField("", 5);
voltage_setpoint = new TextField("", 5);
total_current = new TextField("", 5);
total_weight = new TextField("", 5);
lvoltage_calibrate = new Label(" Voltage Calibrate");
lcurrent_calibrate = new Label("Current Calibrate");
```

```
lvoltage_setpoint = new Label("Voltage Setpoint");

ltotal_current = new Label("Total Current");

ltotal_weight = new Label("Total Weight");

//Row2

weight_factor = new TextField("", 5);

pwnm = new TextField("", 5);

prop_gain = new TextField("", 5);

integral_gain = new TextField("", 5);

der_gain = new TextField("", 5);

lweight_factor = new Label("Weight Factor");

lpwnm = new Label("PWM");

lprop_gain = new Label("Proportional Gain");

lintegral_gain = new Label("Integral Gain");

lder_gain = new Label("Derivative Gain");

//set_panel.add(new Button("test"));

setpara_label=new Label("Parameters which can be modified");

set_panel.add(setpara_label,new GridBagConstraints(0, 0, 4, 1, 0.0, 0.0,

GridBagConstraints.CENTER, GridBagConstraints.NONE,

new Insets(0, 0, 10, 0), 5, 5));
```

```
   set_panel.add(lvoltage_calibrate,new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0,
   GridBagConstraints.CENTER, GridBagConstraints.NONE,
   new Insets(0, 0, 0, 0), 5,5));
set_panel.add(voltage_calibrate,new GridBagConstraints(1, 1, 1, 1, 0.0,
0.0,GridBagConstraints.CENTER, GridBagConstraints.NONE,
          new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(lcurrent_calibrate,new GridBagConstraints(2, 1, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(current_calibrate,new GridBagConstraints(3, 1, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(lvoltage_setpoint,new GridBagConstraints(0, 2, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(voltage_setpoint,new GridBagConstraints(1, 2, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
```

```
 set_panel.add(ltotal_current,new GridBagConstraints(2, 2, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(total_current,new GridBagConstraints(3, 2, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(ltotal_weight,new GridBagConstraints(0, 3, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(total_weight,new GridBagConstraints(1, 3, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(lweight_factor,new GridBagConstraints(2, 3, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(weight_factor,new GridBagConstraints(3, 3, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
```

```
 set_panel.add(lpwnm,new GridBagConstraints(0, 4, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(pwnm,new GridBagConstraints(1, 4, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(lprop_gain,new GridBagConstraints(2, 4, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(prop_gain,new GridBagConstraints(3, 4, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(lintegral_gain,new GridBagConstraints(0, 5, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(integral_gain,new GridBagConstraints(1, 5, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
```

```
 set_panel.add(lder_gain,new GridBagConstraints(2, 5, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
 set_panel.add(der_gain,new GridBagConstraints(3, 5, 1, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE,
new Insets(0, 0, 0, 0), 5, 5));
  //Defining File part
 token = "";
  //Creating properties object
  properties = new Properties();
  //creating the object of FileInputStream
  fis = new FileInputStream(new File("prop.properties.txt"));
  //loading the properties object from the values of FileInputStream
  properties.load(fis);
  listen.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
   try{
      String read;
```

```
ButtonModel bm=converter.getSelection();
read=properties.getProperty(bm.getActionCommand() );
//voltage.setText(read);
StringTokenizer st = new StringTokenizer(read);
for (int i=0; i<14;i++)
{
 token = st.nextToken();
 output_matrix[i] = token;
 }
 voltage.setText(output_matrix[0]);
 curretnt.setText(output_matrix[1]);
 voltage_calibrate.setText(output_matrix[2]);
 current_calibrate.setText(output_matrix[3]);
 voltage_setpoint.setText(output_matrix[4]);
 total_current.setText(output_matrix[5]);
 total_weight.setText(output_matrix[6]);
 weight_factor.setText(output_matrix[7]);
 current_setpoint.setText(output_matrix[8]);
```

```
        pwnm.setText(output_matrix[9]);
        pulse_width.setText(output_matrix[10]);
        prop_gain.setText(output_matrix[11]);
        integral_gain.setText(output_matrix[12]);
        der_gain.setText(output_matrix[13]);
    }
     catch(Exception ex)
     {
       ex.printStackTrace();
     }
}
  });

 reset.addActionListener(new ActionListener() {
 public void actionPerformed(ActionEvent event) {
        voltage.setText("");
        curretnt.setText("");
        voltage_calibrate.setText("");
        current_calibrate.setText("");
```

```
        voltage_setpoint.setText("");
        total_current.setText("");
        total_weight.setText("");
        weight_factor.setText("");
        current_setpoint.setText("");
        pwnm.setText("");
        pulse_width.setText("");
        prop_gain.setText("");
        integral_gain.setText("");
        der_gain.setText("");
  }
}
);

sendt.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent event) {

try{
     buf1 = new byte[];
```

```
buf2 = new byte[];

buf3 = new byte[];

buf4 = new byte[];

buf5 = new byte[2];

buf6 = new byte[2];

buf7 = new byte[2];

buf8 = new byte[2];

buf9 = new byte[2];

buf10 = new byte[2];

buf11 = new byte[2];

buf12 = new byte[2];

buf13 = new byte[2];

buf14 = new byte[2];

buf15 = new byte[2];

buf2 = voltage.getText().getBytes();

mainbuf[0] = buf2[0];

if(buf2.length>1)

mainbuf[1] = buf2[1];
```

```
buf3 = curretnt.getText().getBytes();

mainbuf[2] = buf3[0];

if(buf3.length>1)

mainbuf[3] = buf3[1];

buf4 = voltage_calibrate.getText().getBytes();

mainbuf[4] = buf4[0];

if(buf4.length>1)

mainbuf[5] = buf4[1];

buf5 = current_calibrate.getText().getBytes();

mainbuf[6] = buf5[0];

if(buf5.length>1)

mainbuf[7] = buf5[1];


buf6 = voltage_setpoint.getText().getBytes();

mainbuf[8] = buf6[0];

if(buf6.length>1)

mainbuf[9] = buf6[1];


buf7 = total_current.getText().getBytes();
```

```
mainbuf[10] = buf7[0];

if(buf7.length>1)

mainbuf[11] = buf7[1];

buf8 = total_weight.getText().getBytes();

mainbuf[12] = buf8[0];

if(buf8.length>1)

mainbuf[13] = buf8[1];

buf9 = weight_factor.getText().getBytes();

mainbuf[14] = buf9[0];

if(buf9.length>1)

mainbuf[15] = buf9[1];

buf10 = current_setpoint.getText().getBytes();

mainbuf[16] = buf10[0];

if(buf10.length>1)

mainbuf[17] = buf10[1];

buf11 = pwnm.getText().getBytes();

mainbuf[18] = buf11[0];

if(buf11.length>1)

mainbuf[19] = buf12[1];
```

```
buf12 = pulse_width.getText().getBytes();

mainbuf[20] = buf12[0];

if(buf12.length>1)

mainbuf[21] = buf12[1];

buf13 = prop_gain.getText().getBytes();

mainbuf[22] = buf13[0];

if(buf13.length>1)

mainbuf[23] = buf13[1];

buf14 = integral_gain.getText().getBytes();

mainbuf[24] = buf14[0];

if(buf14.length>1)

mainbuf[25] = buf14[1];

buf15 = der_gain.getText().getBytes();

mainbuf[26] = buf15[0];

if(buf15.length>1)

mainbuf[27] = buf15[1];

String s = new String(buf15, 0, buf15.length);

// voltage.setText(s);

//System.out.println(s);
```

```
ButtonModel bm=converter.getSelection();
String convid=bm.getActionCommand();
convid=convid.substring(9);
 buf1 = convid.getBytes();
 //System.out.print("Converterid"+convid);
 mainbuf[28] = buf1[0];
// mainbuf[29] = buf1[1];
 // Action to be taken for sending the data
 //1. Take values from text field.
 // 2.Convert values to bytes
 //3.Form a packet with selected converter id
 //4.Send the packet.

 byte[] buffer = new byte[30];
 DatagramPacket incoming = new DatagramPacket(buffer, buffer.length);
 DatagramSocket ds = new DatagramSocket(2134);
 InetAddress ipaddress = InetAddress.getLocalHost();
 DatagramPacket send_packet = new
DatagramPacket(mainbuf,mainbuf.length, ipaddress, 2134);
```

```
DatagramSocket sender = new DatagramSocket();
sender.send(send_packet);
ds.receive(incoming);
  byte[] data = incoming.getData();
  //System.out.println(data.length);
  String localid = new String(data, 28, 1);
  filenamewrite="Converter"+ localid+ ".txt";
  fwrite= new File (filenamewrite);
  System.out.print(filenamewrite);
  foswrite = new FileOutputStream(fwrite,true);
  String nlf = System.getProperty("line.separator");
  foswrite.write(nlf.getBytes());
  foswrite.write(data);
  String temp=new String(data,0,data.length);
  System.out.print(temp);;
//    Write To File
      //    Define different strings for all parameters
}
```

```
catch(Exception ex)
   {
     ex.printStackTrace();
   }
  }
}
);


   f.addWindowListener(new WindowListener() {
     public void windowClosing(WindowEvent e) {
       f.dispose();
     }
     public void windowActivated(WindowEvent e) {}
     public void windowClosed(WindowEvent e) {}
     public void windowDeactivated(WindowEvent e) {}
     public void windowDeiconified(WindowEvent e) {}
     public void windowIconified(WindowEvent e) {}
     public void windowOpened(WindowEvent e) {}
   });
```

```
  f.setMenuBar(menuBar);
  f.add("North", main_panel);
  //f.pack();
  f.setSize(700, 472);
  f.setResizable(false);
  f.show();
 }
 catch(Exception ex)
  {
    ex.printStackTrace();
  }
 }


  public static void main(String args[]) {
        //Frame f = new Frame("GUI for Health Monitoring");
        GUIMain g1 = new GUIMain();
            }
}
```