THE DEVELOPMENT

OF A

PORTABLE DATA ACQUISITION SYSTEM

FOR USE IN

COMMERCIAL AND INDUSTRIAL ENERGY AUDITS

A Thesis

by

Michael Andrew Davis

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 1993

Major Subject: Mechanical Engineering

THE DEVELOPMENT OF A PORTABLE DATA ACQUISITION SYSTEM FOR USE IN COMMERCIAL AND INDUSTRIAL ENERGY AUDITS

A Thesis
by
Michael Andrew Davis

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of
MASTER OF SCIENCE

Approved as to style and content by:

Dennis O'Neal
(Chair of Committee)

B. Don Russell
(Member)

Warren Heffington
(Member)

for George Peterson
(Head of Department)

December 1993

Major Subject: Mechanical Engineering

ABSTRACT

The Development

of a

Portable Data Acquisition System

for use in

Commercial and Industrial Energy Audits.

(December 1993)

Michael Andrew Davis, B.S., Texas A&M University

Chair of Advisory Committee: Dr. Dennis O'Neal

The need for a functionally advanced instrumentation and data analysis system combined with the improvements in electronics technology has led to the design of a flexible and updateable data acquisition system. This system utilizes a distributed processing approach to allow for the real-time gathering and analysis of the types of data required during energy audits and long-term energy use studies. This system uses the traditional transducer-based approach for non-power systems data, but utilizes advanced direct digital sampling and processing for all power systems information. Using this approach, a single system is able to measure both electrical power data (demand, power factor, current, etc.) and non-power systems data (temperature, relative humidity, etc.) with a minimum of transducers and provide the data to evaluation algorithms for as-gathered processing.

The software has been designed, using a message-based approach, to allow the end user to build application specific analysis routines that can be added to the system and executed during the data acquisition process. Using library routines developed with the system, the user is able to add end-use specific code that is executed at run time. This capability allows the user to analyze the data in real-time, thus avoiding the storage and post processing of large quantities of data.

ACKNOWLEDGMENTS

I would like to thank my Heavenly Father for giving me the ability and the opportunity to further my education. I would like to thank my wife, Sally, for being willing to continue "living the life of a student" while I worked for something that I wanted but did not have to have. I would like to thank my two sons, Andrew and Collin, for being understanding about all of the times they needed me when I was not there.

I would also like to thank Dr. Dennis O'Neal and Dr. Don Russell for all of the patience, guidance, and support they gave me during this project.

The author gratefully acknowledges the financial support provided by the State of Texas through the ERAP program. Sharif Shahrier assembled, programmed, and tested the Digital Signal Processing (DSP) side of the Power Systems Data Acquisition Board (PSDAB). Steve Williams wrote and tested the "C" source code used for the Power Harmonics (PHA) algorithms.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION - THE NATURE OF THE PROBLEM

A. Introduction

Energy professionals have long expressed the need for an easy-to-use, yet functionally advanced, instrumentation and data analysis system for monitoring energy use and performance of energy related equipment and technologies. The easy to install equipment could be used during audits to gather much-needed use and performance data on process equipment, HVAC systems, high-demand machinery, or overall facility use. Through self-contained software, the data would be analyzed and presented to the auditor for immediate use and for subsequent analysis.

As new energy saving technologies have been devleloped, they have also introduced the need to measure new types of data. Some of the new technologies, such as variable frequency drives (VFD's) and electronic ballasts, have raised issues concerning power quality and harmonic noise generation. In electric power systems, there exist loads and devices such as static power converters, inverters and rectifiers that possess nonlinear voltage/current characterstics. When supplied with sinusoidal alternating voltage these loads draw nonsinusoidal alternating current containing associated harmonics and DC component besides the fundamental component. The flow of harmonic currents through the system produces corresponding voltage drops that in turn produce harmonic current in linear loads [1]. The level of harmonics in electrical delivery systems has been increasing over the past several years [2]; therefore, it is important to be able to measure active and

Journal Model is *IEEE Transactions on Automatic Control.*

reactive powers in both sinusoidal and nonsinusoidal conditions [1]. Quantities such as active and reactive power originally defined to describe certain properties of sinusoidal systems are to be measured in nonsinusoidal systems as well [1]. Measurement of active power is required for control, and revenue metering [3]. Measurement of reactive power is for control, revenue metering, and power factor compensation [4]. Currently available analog meters, designed and calibrated for sinusoidal systems, are unsuitable for nonsinusoidal systems [4]-[9]. For instance, errors of up to 15% were observed during the testing of particular kVA demand meters on waveforms representing actual field conditions [9]. Several new analog metering schemes for measuring power in nonsinusoidal systems have been proposed so far [10]-[13]. The decrease in cost associated with digital signal processor technology allows the digital power measurement techniques to be a viable alternative to analog metering. Various digital algorithms for power measurement have been proposed [14]-[16], and implemented [14]-[23]. However, the existing digital power meters are designed to either measure reactive power in sinusoidal systems or only active power in nonsinusoidal systems [1].

A new approach to the design of digital algorthms for electric power measurements was taken by Soljanin [1] and Kezunovic [26] which was modeled after the earlier analog-based digtal techniques. The new approach was shown to be appropriate for both sinusoidal systems and nonsinusoidal power systems [1][26][28]. Also, existing data acquisition systems are incapable of continuously monitoring at the sampling rates required for harmonic data analysis. Some systems allow for "snapshot" harmonic readings, but these were not intended for use as continuous monitoring devices.

In long-term installations, the device could be used to monitor the efficiency and performance of various energy technologies, including validation of energy savings and the documentation of performance claims and specifications. It could also be used to monitor the active and reactive power harmonics in sinusoidal and nonsinusoidal systems.

The following sections provide a short description of the data acquisition process, relevant background and market information concerning data acquisition systems, information on existing products, unmet end user needs, and a description of a "state-of-the-art" system. After this there are two final sections: objectives and a description of how the rest of the thesis is organized.

B. The Data Acquisition Process

The data acquisition process consists of making a record of some discrete data or event. This record keeping can be done as an isolated one-time activity, but usually consists of an organized attempt to gather a series of records that will be used to describe some event, process or system. The actual record-taking usually consists of using an appropriate transducer to output an electrical signal to a computer for translation and storage.

Current Data acquisition technology utilizes a transducer-based approach for acquiring all types of data, including power systems information [29]. The typical method of gathering power systems data is to use a transducer that inputs the voltage and current information and outputs an electrical signal that is proportional to the item of interest. Two examples of transducer applications in power systems are demand and power factor readings.

The current approach is to have a transducer that measures the particular power systems information that is of interest. Using the power factor of a load on a sixty hertz power supply as an example, the transducer could be designed to output a four-to-twenty milliamp DC signal that would be calibrated to be proportional to the power factor of the load. The demand of the same load would be measured by a transducer that outputs a DC signal proportional to the demand established by the load. If you wanted data on both demand and power factor, you would need to purchase two transducers. If you also needed to know RMS current, voltage, and information about power consumed in the harmonics you, need still more transducers or you would have to purchase a single transducer that outputs multiple signals where each signal represents each item of interest.

Transducers are also used to acquire non-power systems data such as temperature, relative humidity, wind speed, water flow, and other data that are of interest to the energy manager.

After data are acquired, it is generally stored for later evaluation by an energy manager or auditor. This will occur over time and be evaluated only when enough data have been gathered and stored such that the energy manager or auditor can define the characteristics of a process with the data.

C. Background

The market for energy monitoring is highly fragmented and consists of a wide variety of products focused at different segments of the market. There are few products designed specifically for use in energy audits. However, many products designed for related applications are often used by energy auditors to monitor and record energy use [25].

Energy auditors use equipment ranging from simple current and voltage meters to computer-based instrumentation systems because a single system does not exist that can be used to obtain all of the required data. The reason that a single system does not exist appears to be that a generalized approach has not been taken in the design of current data acquisition systems. All of the systems reviewed appear to have been designed with certain target markets in mind. The results of this approach are a loose conglomeration of energy- and non-energy-related data acquisition products being used in the energy monitoring process. Since a single system does not exist that can be used to gather all types of data, the energy manager - and any other researcher - must use a collection of components to obtain the required data.

The lack of a generalized design approach is a direct result of market influences on the companies that design and build data acquisition systems and components. Because of the direct influence of the market for data acquisition equipment on the design of the system, a description of the market and end users is appropriate.

D. Description of Industry and Product Markets

The market for this product includes industrial, commercial, and government facilities. These users may purchase energy monitoring systems for use in monitoring equipment, processes, or entire facilities. In addition to direct use by industrial, commercial and government facilities, markets can be found among service companies that perform energy audits for industrial, commercial, and government clients. These companies include utilities, HVAC (heating, ventilating, and air-conditioning) maintenance specialists, and

energy auditors. Customers may also include researchers interested in monitoring energy usage.

The market for energy monitoring equipment for use specifically in energy auditing reaches into the tens of millions of dollars. In addition, the total energy monitoring equipment market, including revenue meters, is in the billions of dollars. Potential customers, nationwide, include over ten million commercial facilities and over four hundred thousand industrial facilities [25].

E. Industry Trends and Developments

As energy costs have fallen in recent years, the pressure to cut back on energy use has been reduced. The result has been a decline in the number of energy audits that are being performed. The impact on the market for energy monitoring equipment has been to cause growth to level off and sales to stabilize. However, three factors should cause energy auditing activities and energy monitoring equipment sales to increase. These factors are a projected increase in energy costs, increased environmental awareness, and increased governmental regulation [25].

Energy monitoring technology has improved greatly over the last decade along with improvements in electronics technology. However, manufacturers of energy monitoring equipment have been slow to exploit new developments in computer technology. Despite this slow reaction, many companies in this market, such as Dranetz and Esterline Angus, have indicated that they are currently engaged in new product development to take advantage of modern computer capabilities. These new product efforts did not impact the

design of this system since these companies and their competitors were reluctant to divulge the details of their research and development efforts.

F. Key Success Factors

The key factors influencing the success of a data acquisition system in the energy monitoring market include: marketing efforts, servicability, and product attributes. The marketing efforts and servicability issues were not dealt with during this research and will not be addressed in this thesis. The product attributes were the key focus of this research.

A successful energy monitor must have several product attributes. These attributes will be discussed for existing systems and then the unmet needs will be highlighted. Meeting these unmet needs was the focus of this research. The following paragraphs will cover some of the key attributes in a data acquisition system.

The successful energy monitor must be easy to use. The technicians who will be installing and using the system usually will not have had extensive training in the use of the equipment. As a result, the equipment must be easy to use and simple to install.

Another key factor is flexibility and expandability. Flexibility requires a system to be able to measure a variety of variables from a range of sensors. Expandability requires a system to be able to use a large number of sensors, measuring inputs over a wide range.

Programmable means that a user can customize the software to acquire data for an end use application other than those planned for by the developers of the system.

Memory is important because it allows a system to record more information over a longer period of time. This memory could include both volatile and non-volatile forms of storage.

A system should also be rugged, compact and able to communicate over phone lines so that data can be downloaded to a computer for further processing.

Computer software is important in the success of the system. It should be user friendly and must process the information so that it can be presented to the user in an easily understood form. The software should also allow flexibility in its use.

G. Existing Data Acquisition Systems

A review was conducted of the current data acquisition market to assess the capabilities of the systems currently available [29]. It was found that a lot of variety existed in the current systems. A "typical" system could be described along with a list of additional capabilities. It should be noted that most manufacturers have at least one of the extra features built into their base system. Both the typical system and the list of additional features are discussed below.

The systems were also evaluated with regard to their capabilities for expansion. Most systems were only slightly expandable with the "options" being limited to dedicated products offered by the original equipment manufacturer. If the user wanted to add a capability that the manufacturer did not offer, his only choice for expansion was to purchase a system from another vendor that offered the desired function.

The "typical" system consists of basic analog inputs, memory for storage of "programs" and data, and a serial port for communications. The inputs are composed of eight channels which are divided between analog and digital inputs. The analog channels sample eight-bit data at a maximum interval of once a second with a maximum sample frequency of about 400hz per channel. The memory is a total of 32Kbytes or 64Kbytes of battery backed ram.

The memory is used for both data and storage. Unfortunately, the programs usually take about 10Kbytes, which is a big percentage of the total storage capacity. The typical system includes a standard IEEE-RS232 serial port, which connects to a modem or directly to a computer for transfer of data and "programs."

The list of additional features includes, but is not limited to, extra memory (up to 256Kbytes for one model), additional inputs, the ability to measure some level of harmonics, the ability to measure three-phase power, a 1.4Mbyte floppy drive, and an IEEE-RS485 serial port for networking.

When describing a typical system it should be noted that most manufacturers can claim that their system is not typical because they have certain extra features not included in the description of the typical system. In this research all systems reviewed were still classified as typical since none of the manufacturers included more than a small number of the extra features in their systems.

H. Unmet Needs

Another list that was assembled consisted of currently available digital and data acquisition technologies that were low cost, rugged, and dependable. A partial listing includes: IBM-PC compatible embedded controller boards, memory, hard disks and controllers, high-speed data acquisition boards with sixteen-bit internally multiplexed sample and hold, and multiple serial ports including RS-485, RS-422, and RS-232. Additional optional features included low cost digital signal processing chips and boards which make true real-time processing of data possible.

When the list of options was added to the "typical" system and compared to how a system could be used in the energy auditing, monitoring, and evaluation process, a list of unmet needs was constructed. When this list was compared to existing and available technologies, it was clear that a system could be designed that satisfied most of the "unmet needs."

I. A State-of-the-Art Data Acquisition System

A "state of the art" data acquisition system should have most or all of the options and available technologies listed above. A minimal configuration would be an i80286 based mpu with two megabytes or ram, a twenty megabyte hard drive, a one-point-four megabyte floppy, and RS-485 serial port for networking, and an RS-232 serial port for telephone communications. The data acquisition portion of the system should consist of analog input boards that sample data with a minimum resolution of twelve bits for data with a low range and sixteen bits for data that vary over a large range (such as large industrial types of power systems). Whenever appropriate, DSP technology should be incorporated to allow the data to be processed in real-time.

J. Objectives

This research is the first part in a series of proposed studies on different aspects of a computer-based commercial and industrial data acquisition and analysis system. The purpose of this research investigation was to take advantage of earlier work which had demonstrated the feasibility of using a computer-based monitoring system to document

efficiency improvements and fuel use reductions. There were two primary objectives of this research:

1. Develop a detailed set of commercialization design goal specifications for both hardware and software systems.
2. Design, build, and laboratory-test the first prototype of hardware and software systems using the design goal specifications as a guide.

With this in mind, an overall single system approach was taken with a goal of designing a product that could be used to gather energy-related data.

The design was to take advantage of as much new technology as possible and build into the system the ability to upgrade the design as technology improves. For example, consider the transducer approach to measuring electrical power consumption. Power systems data are unique in that it can be directly measured using analog-to-digital conversion techniques. In the previously given example, all of the required information such as power factor, current, harmonics, etc., could be calculated from a single set of readings. This approach would require fewer sensors and, at the same time, would provide a better resolution of the data.

K. Organization Of Thesis

Chapters two and three - Hardware System Design and Software System Design - provide descriptions of the objectives and procedure for each design as well as descriptions of the prototype system that was built using the design goal specifications as a guide. Included in this description is the measured system performance of the prototype as tested in the lab. A summary is provided at the end of both chapters.

Important Conclusions and Recommendations are provided in the last chapter. The focus of this chapter is on recommendations for development of the commercial system. Also covered are difficulties encountered during the design, prototyping, and testing phases of the project.

CHAPTER II

HARDWARE SYSTEM

A. Introduction

The goal behind the hardware design was to develop a data acquisition system that would meet the current needs of expected users, take full advantage of existing technology, and allow for easy updating of the system as technology improves. This goal placed several constraints on the system and required that it include state-of-the-art technology, have the ability to be upgraded, be expandable, portable, and easy to use.

The specific goals of this research as related to the hardware system were to develop a detailed set of commercialization design goal specifications. Using the design goal specifications as a guide, a prototype system was to be built and tested with simulated data. After the system was tested in the lab, field tests were also conducted.

B. Review of Existing Systems

The first step taken in conducting this research was to review existing products to determine what type of systems were on the market, the level of technology currently being used, what kinds of technology existed, and how much of the new technology could be incorporated into a system.

The types of equipment reviewed included hand held instrumentation, pc based systems that were semi-portable, and non-portable systems that were intended for use in long term studies.

In the review process a typical hardware structure was observed which consisted of a microprocessor/microcontroller, a small amount of memory, a serial port, a predetermined number of channels, and low resolution conversion from analog inputs to digital outputs (usually 8-bits). Figure 1 gives a block diagram representation of the typical system.

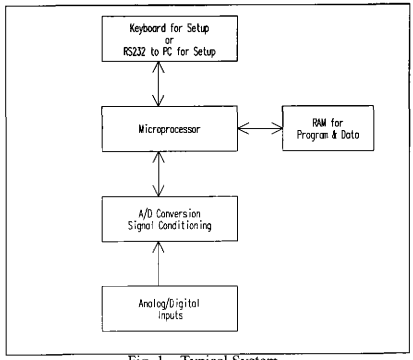

Fig. 1. Typical System.

The typical system was based on an Intel 8088 4Mhz CPUs or equivalent. Supporting this processor was a small amount of memory (64Kbytes or less) which was used to hold both program and data. Quite often about a third of the system memory was used for program storage. The system calibration constants were typically stored in Electrically Erasable Programmable Read Only Memory (EEPROM) while the system's bootstrap code was kept in Programmable Read Only Memory (PROM) or Electrically Programmable Read Only Memory (EPROM). The system had eight channels of 8-bit analog to digital conversion capabilities with fixed ranges of inputs which usually consisted of 4-20

milliamp signals. The typical system was strictly transducer-based with the transducers outputting a 4-20 milliamp signal. The typical system was not very flexible. It could not be networked or have additional capabilities added to it.

C. Designed ERAP System

After existing systems were reviewed and the typical system profile was determined, a new system design was developed. It was observed that the typical system was composed of three basic operations consisting of a user interface, digital recording of the data, and conversion of the input signal from analog to digital. These three processes are outlined in Figure 2.

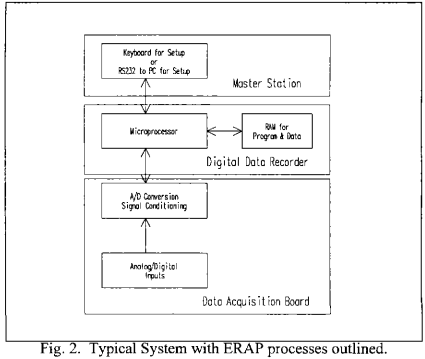

Fig. 2. Typical System with ERAP processes outlined.

The new system design was based upon the three processes listed above. This approach led to the ERAP system's being segmented into three major components: the Master Station (MS), the Digital Data Recorder (DDR), and the Data Acquisition Board (DAB). The three

components would be functionally similar to components of the existing systems except that each component would now be a stand-alone piece of hardware capable of functioning independent of, or in conjunction with, the others. The method of communication between the different parts of the system would be through serial I/O. From the MS to the DDR the serial communications would be via an RS232 serial port. The DDR would communicate with the DAB via an RS-485 serial port. Figure 3 is a block diagram of ERAP system.

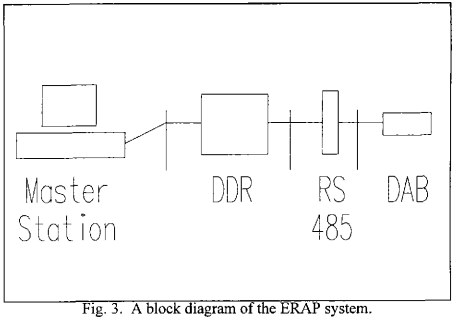

Fig. 3. A block diagram of the ERAP system.

The Master Station was determined to consist of a 80386 based microcomputer. This computer would include certain minimum hardware components such as 2Mbytes of Random Access Memory (RAM), VGA graphics, a 120 Mbyte hard drive, a mouse, a 3.5" high density floppy disk drive, an RS232 serial port, and a 2400 baud modem. The Master Station did not require any special design enhancements beyond what is included on a typical microcomputer.

The Digital Data Recorder (DDR) was based on the 80286 CPU running at either 12 MHz or 16 MHz. The DDR consists of a minimum of 2 Megabytes of RAM, an RS232 serial port, an RS-485 serial port, a 40 Mbyte hard drive, a high density 3.5" floppy drive and a modem compatible with the Master Station modem.

The Data Acquisition Board was designed to be a sub-system shell serving the acquisition of five classes of data which consists of 1) serial data, 2) status indicators, 3) digital data, 4) analog signals, and 5) power systems data. The DAB design consisted of a simple processor (80186, 80286, etc.), memory, an RS-485 serial port, eeprom for system constants, eprom for firmware, and input hardware depending on the type DAB being built. For this project two DAB's were selected for prototyping and testing. These were the analog and power systems DAB's.

The analog DAB was to consist of 8 channels of analog input with 8 bit a/d conversion. This data was to be transmitted back to the DDR via the RS-485 bus.

The power systems DAB was selected as the second board to prototype because of the high sample rate associated with power systems and the large number of floating point operations per second that have to be done to process the data in real-time. Since more computational speed was required, a more elaborate design had to be considered. The Power Systems DAB (PSDAB) was designed to consist of an 80386 CPU with a math coprocessor doing the calculations and managing the serial port. The data acquisition consisted of 8 channels of simultaneous sample and hold with 16 bit a/d data conversion and 2 Mbytes of ram. Figure 4 is a block diagram representation of the power systems DAB.

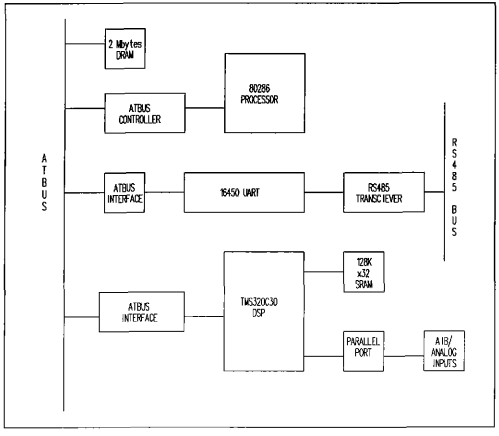

Fig. 4. Block diagram representation of the power systems DAB.

D. Developed Hardware Specifications

The commercialization specifications were developed to be a guide to the minimum functionality required for a fully developed commercial product based on the research done under ERAP project number 260. The specifications were not intended to be and cannot serve as a design of a properly engineered commercial product since that was not the goal of the ERAP project. The commercialization specifications describe the overall system structure, the major components of the system, and the minimum performance requirements of each of the major components. The specifications were developed after the initial system design and were submitted to the project sponsor in a quarterly report.

E. Construction of the Prototype

Major system components were identified and parts were obtained. One of the goals of the project was to develop the first prototype from existing commercially available products and an attempt was made to build the system from off the shelf components. Because of this decision, vendors were identified that could provide each of the major components. It became clear fairly early in the project that some of the system design features would have to be compromised because some of the components simply did not exist and other components could not be obtained at prices that would allow the system to be sold at a reasonable price.

Because of the lack of availability of certain key components, some compromises had to be made in the design and construction of the first prototype. In conjunction with initial hardware design some tests were being conducted on the software. The software was being developed in a parallel effort to the hardware and is explained in detail in the following chapter. The tests from the software showed that the 80386 processor was not capable of handling the power systems calculations in a real-time system. More test were conducted on the software to determine what kind of processing power was required and it was determined that a floating point DSP should be used. Because of this heavy computational requirement and the decision to build the prototype from off the shelf components, it was decided that an 80286 pc with an in-slot DSP board with a dedicated anolog-to-digital sampling board would be used to construct the power systems DAB. The DSP board was based on the TI TMS320C30 chip which was capable of 33 million floating point operations per second (MFLOPS).

F. Constructed Master Station

The Master Station hardware was constructed as per specifications. It consisted of an 80386-based microcomputer running at 20Mhz. Other components consisted of 7 Mbytes of RAM, 200 Mbytes of hard disk space, a 3.5" high density floppy, a 5.25" high density floppy, a VGA card and monitor, 2 RS232 serial ports, a modem, and a mouse.

G. Constructed Digital Data Recorder

The digital data recorder was based on a standard 80286 microcomputer running at 12 MHz. It had a 40 Mbyte hard drive, a 3.5" high density floppy drive, 2 Mbytes of RAM, a monochrome graphics controller card and monitor, an RS232 serial port, and an RS-485 serial port. The monitor was required for use during development and field tests. In the actual production model, a monitor and keyboard were not expected to be required.

The RS-485 serial port was added by installing an in-slot communications board to the PC-AT's ISA bus with a full size card.

H. Constructed the Power Systems DAB

The power systems DAB consisted of a 80286 based PC with a 40Mbyte hard drive, one megabyte of RAM, a high density floppy drive, a monitor, an in-slot DSP board with a dedicated analog sampling board, and an RS-485 serial port. The RS-485 serial port was provided by installing an in-slot communications card.

The DSP board was supplied by Sonitech and proved to be reliable. The board turned out to be very difficult to interface with until an analog interface board was purchased from

another vendor. The AIB was purchased from Spectrum Signal Processing and with their assistance proved to be an easy match with the Sonitech DSP board.

The as-constructed PSDAB consisted of the 80286 board managing the communications with the DDR and storage of the results from the DSP board until called for by the DDR. The DSP board managed gathering all of the samples, performing the required calculations, and uploading of the results from the DSP to the PC when requested by the 80286.

I. Construction of the General Purpose DAB

The slow speed general purpose DAB was a general purpose embedded controller board from Z-World Engineering. This board consisted of a Z180 processor running at 9Mhz, 64Kbytes of ram, an RS-485 serial port, and 8 channels of analog data acquired with 10 bits of A/D conversion resolution. This board was configured to gather RMS current and voltage on two of the channels.

J. Integration of the System

After all of the components had been obtained the system was put together and integration with software was done. After the software was developed and integrated with the hardware, tests were conducted. The tests consisted of connecting known loads to the DAB's and adjusting system parameters until the data could be gathered reliably. After initial calibration was done, an analog recording system was used to record power systems data at an electrical substation on the Texas A&M Campus. These signals were then fed to the PSDAB and the calculated values were compared to the recorded values. Additional

testing consisted of applying known variable loads to the system and verification of the recorded data. The results of the tests are included in the following section.

K. Results

With the DSP board the power systems were able to be done in real-time. Real-time for this system meant that the power system data was gathered, calculated and the information was available to the user once every twenty milliseconds. During tests the results of the calculations were integrated and averaged over a one second interval and up-loaded to the 286 once per second. The analog samples were gathered with a resolution of 12 bits. Because of sampling errors, distortion in the AIB's op-amps, and calculation round-off errors the accuracy of the data available to the end user was measured to be a reliable 10 bits. With this data acquisition system connected to a peak load of 25Kw, a load as small as 25 watts could reliably be detected. Since the data was available to the 286 processor once every second, the load could be detected within one second of being brought on-line.

Actual measurements showed that a .2 amp load could be measured with sensors calibrated to measure a 200 amp load. This is about a .1% error.

The RS-485 network was operated at a transmission rate of 57.6Kbaud. This resulted in a network throughput as high as 3,000 channels/sec with continuous monitoring. All communications were interrupt-driven transmissions.

The RS232 to master station throughput was considerably slower with only 150 channels per sec. The slower throughput was because the MS-to-DDR connection was at 2400 baud.

With the application of transient loads to the analog inputs it was observed that changes took three seconds to propagate through the system from the AIB to the master station. This was because of the polling procedure implemented on both the DDR and the MS. These are explained in detail in the software section included in the next chapter. The time delays of when an input was applied and when it was observed by each of the system components was three second for the Master Station, two seconds for the Digital Data Recorder, and one second for the Power Systems Data Acquisition Board.

L. Field Test Results

Field tests indicated that the system was able to handle the data throughput required for the Power Systems DAB support. The major problems encountered during field testing were memory allocation problems and failed sensors. The stability of the system was determined to be directly related to the memory allocation management. Initially the system would run a while and then fail due to memory errors. It was determined that memory was becomming fragmented and a memory cleanup routine was added. This routine helped but did not altogether solve the problem. A robust error handling routine had to be implemented before the DDR became stable. The accurracy of the field test data was always as reliable as the laboratory test data.

M. Summary

The users can see data as often as once a second, including power systems data. The software can see power systems data as often as every 20 milliseconds at the DSP board

level. For most applications, this system will provide multiple channels of power systems and non-power systems data in real-time.

CHAPTER III

SOFTWARE SYSTEM

A. Objectives

The design of the software to support the ERAP hardware required that the system be able to adapt to the flexible nature of the hardware, allow the user to access all of the features of the system, and provide for future growth without complete restructuring of the software. To accomplish this design, a message-based approach was taken in the design of the software system which consisted of treating all system activities as events occurring as the result of a message. The message could be either hardware or software generated.

With a message-based approach four types of operations take place: 1) the handling of incoming messages, 2) the handling of outgoing messages, 3) some level of intermediate processing, and 4) prioritization of system activities. Incoming messages are the results of interaction with an external device such as the master station or a DAB, and consist of the downloading of instructions or the uploading of data. Outgoing messages are determined by system parameters or the user's interacting with the master station, and consist of commands being sent to other nodes on the network or data requests by previously received commands. Prioritizing system activities is the process of determining what the system will do next after all of the system parameters have been reviewed. Intermediate processing is anything other than the handling of messages and the determination of system priorities.

B. Procedure

Major software system components were identified and included startup code, running code, and shutdown code. The function of the startup code was to setup the interrupts and serial ports, read in system defaults from disk or EEPROM, setup the function pointer array, and set the system flags to startup values. The shutdown code restored all of the interrupts to original values, saved system flags, and either exited to the operating system or started the execution of another software package.

The running code consisted of a loosely constructed set of operational rules that determined from system parameters what code would be executed at any given time. The decision to run a particular block of code was determined by the priorities associated with that code and the current settings of the system flags. In other words, there was no predetermined flow of the program. The running code was built on an array of function pointers with the main loop consisting of simply indexing into the array and running the indexed function. After the function had been executed, it would return a value that was the next index into the array. The program was exited when an index was returned which ran the shutdown code. Figure 4 gives an overview of the "structure" of the program and Figure 5 gives an example of the main loop being executed.

With the development of the basic operational code, a detailed message system was required. This system consisted of support routines for generating messages, placing them in the appropriate message queue(s), and reading the various message queues. In addition to the support routines for the messages, a detailed command structure was developed and was used as the basis for all messages.

The message-based approach was taken with all three hardware systems - MS, DAB, and DDR. Since a single command structure was used for all systems, the same high level support routines were used on all systems. Low-level routines, such as reading data from

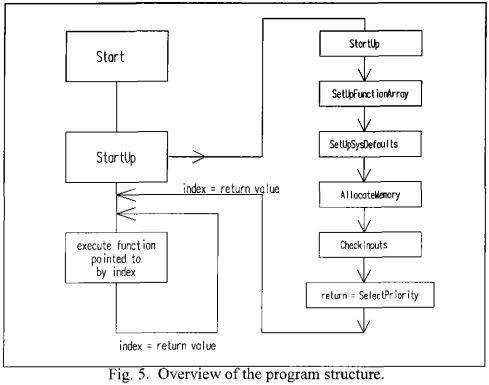

Fig. 5. Overview of the program structure.

the registers of an A/D converter, were programmed specifically for the platform being used. The names and calling conventions of the machine-dependent routines were kept the same from one platform to another, whenever possible, in order to reduce custom code. In theory, this means that any of the system components could control or be controlled by any of the other components and that the software system is the same from one platform to another. Figure 6 gives an example of the main program loop common to platforms.

Even though the same basic code exists from one platform to another, there are some application specific blocks of code that were included specifically for use only on certain hardware systems. For example, the GUI (graphical user interface) used on the master

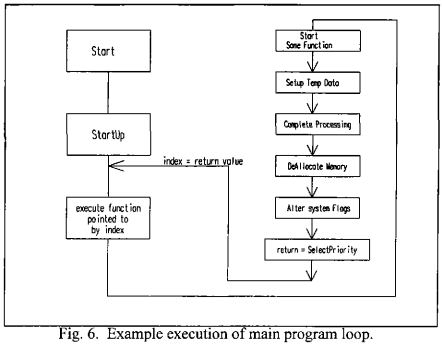

Fig. 6. Example execution of main program loop.

station was specific to the MS. None of the GUI code was needed or included for any of the other system components. If a GUI were needed on a DAB, it would be a small matter to include the code.

C. Power Systems Software - The PHA Algorithm

The calculation of the power systems data is a block of code specific to the power systems DAB and is based on the PHA algorithm [4]. The PHA is a digital algorithm for measurement of active and reactive powers in non-sinusoidal signals [3]. Many algorithms were proposed and used up to now with the common feature that they provide the same value if the voltage and currents are pure sinusoids, but the results differ if other harmonics are present. A possible reason for this is that there are multiple definitions of active and reactive powers for nonsinusoidal signals. For example, some algorithms define active power as the average of the instantaneous power taken over a period while others define it

as the average of the fundamental harmonic only. Even more numerous are the definitions of reactive power. The aim of the PHA algorithm is to provide a common theoretical base for calculation of real and reactive powers in nonsinusoidal periodic signals and to promote a new method for the design of new algorithms.

In digital algorithms it is useful to adopt a discrete time model of voltage and current wave forms. If we consider N discrete samples then the active power of the kth harmonic Pk is given by:

$$(1) \qquad P_k = \sum_{l=1}^{N} \sum_{m=1}^{N} v_l i_m h_{lm}$$

$$(2) \qquad v_i = \sum_{k=0}^{N-1} V_k \cos\left[\frac{2\pi lk}{N} + \phi_k\right], \phi_0 = 0$$

$$(3) \qquad i_m = \sum_{k=0}^{N-1} I_k \cos\left[\frac{2\pi mk}{N} + \psi_k\right], \psi_k = 0$$

$$(4) \qquad h_{lm} = \frac{2}{N^2} \cos\left[\frac{2\pi k(l-m)}{N}\right], 1 \le l \le N, 1 \le m \le N$$

Here, $H = h_{lm}$ is an N X N square weight matrix. In this case,

k : *Harmonic number (DC = 0)*
v_l : *Discrete voltage samples*
i_m : *Discrete current samples*
V_k : *Voltage amplitude of harmonic k*
I_k : *Current amplitude of harmonic k*
ϕ_k : *Voltage phase angle of harmonic k*
ψ_k : *Current phase angle of harmonic k*
N : *umber of discrete samples*
h_{lm} : *Weight values*
l, m : *Integers*

Similarly the reactive power of the k^{th} harmonic is given by:

$$(5) \qquad Q_k = \sum_{l=1}^{N} \sum_{m=1}^{N} v_l i_m hlm$$

where

$$(6) \qquad h_{lm} = \frac{2}{N^2} \sin\left[\frac{2\pi k(l-m)}{N}\right], 1 \le l \le N, 1 \le m \le N$$

and v_l and i_m are as before.

The equations 1 through 6 form the basis of the PHA algorithym[28]. It is noted that other parameters can be computed directly from the real and reactive power measurements. For example, the total power S_k and power factor PF_k are given below.

$$(7) \qquad S_k = \left[P_k^2 + Q_k^2\right]^{1/2}$$

$$(8) \qquad PF_k = \frac{P_k}{S_k}$$

Also, RMS voltages and currents can be computed as follows.

$$(9) \qquad V_{rms} = \left[\frac{1}{N}\sum_{l=1}^{N} v_l^2\right]^{\frac{1}{2}}$$

$$(10) \qquad I_{rms} = \left[\frac{1}{N}\sum_{m=1}^{N} i_m^2\right]^{\frac{1}{2}}$$

D. Results

Testing the system was a multiple step process which started with testing of the PHA algorithm using simulated data and ended with using the system to monitor known loads. Initially the PHA algorithm was tested with simulated signals. A program was written to generate equally spaced samples of three phase current and voltage waveforms over a single cycle according to models given by equations 2 and 3. The results of these tests are shown in Table 1.

After the simulated tests were concluded, the power systems DAB was integrated with the rest of the system, and general system characterization was done. When real loads were applied to the analog inputs, the measurement error was less than .1%. With this system a .2 amp load was reliably measured using 200 amp sensors. This translated into a reliable accuracy of 10 bits. With this system we were able to measure a 25 watt bulb turning on or off when using sensors calibrated for monitoring loads as high as 25 kw.

When a load was changed, it would be seen by the DAB within one second. The change would show up on the DDR within another second and be visible to the MS in another second. In other words, it took three seconds for a change to propagate through the system from the DAB to the Master Station. This was because of the polling used to move the data from one component to another.

Table 1 - Actual and PHA computed harmonic powers.

HN (Harmonic Number)	Actual Powers		PHA Powers	
	Pi	Qi	Pi	Qi
1	2,491.54	-1,163.88	2,491.54	-1,163.86
2	19.02	-6.94	19.02	-6.94
3	15.45	-4.16	15.45	-4.16
4	12.06	-2.15	12.06	-2.14
5	8.96	-0.8	8.96	-0.8
6	6.25	-0.01	6.25	-0.01
7	3.99	0.34	3.99	0.34
8	2.22	0.38	2.22	0.39
9	0.97	0.26	0.97	0.26
10	0.24	0.09	0.24	0.09

When moving the data around, the system was able to transmit 3,600 channels/second on the RS-485 bus. This was done using a four wire full-duplex communications link. When switched to a half-duplex (two wire) mode, the system was able to move 3,000 channels/second of data. To put this in perspective, this system would be able to monitor, on a continuous basis, power systems data for twenty, three-phase systems. Included in the information delivered for storage and/or processing would be rms current, rms voltage,

power factor, and the amount of real and reactive power consumed in the fundamental frequency and up to the 10th level harmonics.

Because of the hardware setup on the DSP side of the power systems DAB, the calculated values of real and reactive power were available at minimum intervals of 20 milliseconds. If the DSP had not had to manage the acquisition of the data, the calculations could have been done in real-time. "Real-time" means that the calculations for one cycle of data would be completed before the next cycle of data had been gathered. Which would mean the calculated data would be delivered to the user at a frequency of 60hz.

E.Field Tests

Field test data showed that harmonic information for a "real" system could be gathered and stored on a continuous basis. Included in the appendix is a detailed list of equipment, bus-level schematic diagrams of the communications system and the PSDAB, a listing of the software, and samples of a portion of the data gathered during the field studies along with some represntative graphs.

F. Summary

The software system was developed using a message based approach to control the flow of information. The PHA algorithm was successfully implemented and tested on the PSDAB. The entire system was integrated together and was able to gather and transmit power systems data, in real-time, while providing the end user with power systems data including 10th order harmonics and PF for a three phase system. The system was tested with a throughput of 3,000 channels/sec of continuous data.

CHAPTER IV

CONCLUSIONS - ISSUES AND RECOMMENDATIONS

A. Background

Some problems were encountered during the design and construction of the hardware prototype. Estimates of the number of calculations required to compute 10th order harmonics were about 12 million floating point computations per second for a single three phase system. Initially some software tests of the PHA Algorithm were conducted on an 80386 based machine and it was determined that the computations could not be done in real-time, even with a math coprocessor. Because of this, a high end dsp board was selected to insure that computations could be done in real-time.

A DSP board was selected that could eventually be used in a stand alone mode. With the DSP board three dedicated data acquisition boards were purchased and setup in a configuration recommended by manufacturer. Some problems were encountered with this setup and the system was unreliable for this application. After about four months of problems the setup was abandoned. A single data acquisition board was purchased from a competitor and was used to replace the manufacturer's data acquisition boards. Some minor calibration problems were encountered, but after calibration the system ran smoothly.

Problems encountered during the development of the software consisted of communications problems on the network and memory shortages on the Master Station.

Initially the system was setup for full duplexing on the network, but multiple nodes would try to access bus simultaneously. The problem was solved by going to a semi-full duplexing mode, which slowed down the system.

The memory shortage on the master station interface was resolved by adding memory and performing screen saves in extended memory/disk only.

Some data transmission errors were encountered. These were corrected with an encryption scheme. The problem was that when the system was networked, the DDR had a tendency to loose track of where the data came from and where it should be stored. This was corrected by encoding address information in the data structure.

Initially the software had a top-down structure which made networking and expanding the system very difficult. This was corrected by going to a message-based system with a priority-based message queue.

B. Conclusions

The PHA algorithms were thoroughly tested and proved to be reliable and accurate. With these algorithms a wealth of three-phase data can be measured with a minimum of sensors. With higher quality sensors and A/D conversion processes, even greater resolution of the data can be achieved.

By freeing up the DSP to do nothing but calculations, the system performance could be increased. To do this a custom-engineered power systems DAB would have to be designed.

To be able to manage more information, the transmission rates on the 485 bus would have to be increased. This would be a small task and is easily achieved with existing technology.

With the drop in price associated with all of the components used in this system and any system based on digital technology, building a low-cost, high-performance data acquisition system is an achievable goal, but it cannot be done by using off-the-shelf

components. The main reason a system cannot be built from existing components is that no one makes exactly the right components. Existing components have to be manipulated with extensive programming or hardware fixes and are less reliable than custom-engineered components. The performance of off-the-shelf boards is usually optimized for some other task and maximum benefit from the hardware is not achievable. Another reason to use custom-engineered components is reliability of sources. If you find a product that can be used and built it into your system, and then the manufacturer goes out of business, your ability to support your product has been compromised.

C. Recommendations

Based on the results of this research it is recommended that a custom-engineered system be built. The major components to be designed are the DDR, the DAB, and the power systems DAB. The DDR should have as a minimum an 80386 processor and should use either a 32-bit operating system or a 16/32-bit dos extender for developed applications. This would give the users access to as much ram as could be installed in the machine.

REFERENCES

[1] Emina Soljanin. "New Approach to the Design of Digital Algorithms for Electric Power Measurements," *Master's Thesis*, Department of Electrical Engineering, Texas A&M University. pp 1-2, December 1989.

[2] T. C. Shuter, H. T. Volkommer, Jr. and T. L. Kirkpatrick, "Survey of Harmonic Levels on the American Power Distribution System," *IEEE PES Winter Meeting*, Paper No. 89WM 111-9 PWRD, 1989.

[3] W. Shephard and P. Zand, *Energy Flow and Power Factor in Nonsinusoidal Circuits*. Cambridge University Press, Cambridge, Great Britain, 1979.

[4] A. J. Baggot, "The Effect of Waveshape Distortion on the Measurements of Energy Tariff Meters," *IEEE Metering, Apparatus and Tariffs for Electricity Supply*, Conference Publications No. 156. pp. 280-284, London, 1977.

[5] A. Spalti, "Apparent Energy Metering in Three Phase Networks," *Apparatus and Tariffs for Electricity Supply*, Conference Publication No. 156, pp. 285-286, London, 1977.

[6] Y. Baghzouz and O. T. Tan, "Harmonic Analysis of Induction Watthaour Meter Performance," *IEEE/PES Summer Meeting*, Paper No. 84SM 686-2, Seattle, Washington, July 1984.

[7] R. Arsenau and P. S. Filipski, "Effects of Harmonics on Watthour and Demand Meters," *CEA Meeting*, Calgary, October 1985.

[8] M. D. Cox and T. B. Willliams, "Induction Varhour and Solid State Varhour Meters Performances on Nonlinear Loads," *IEEE PES Winter Meeting*, Paper No. 89WM 049-8 PWRD, 1989.

[9] R. Arsnau et al., "Optimum Metering Systems for Loads With High Harmonics Distortion," *CEA Report* No. 043-D-610, Canada, May 1988.

[10] L. S. Czarnecki, "Measurement Principles of a Reactive Power Meter for Nonsinusoidal Systems," *IEEE Transactions on Instrumentation and Measurement*, vol. IM-30, pp. 209-212, September 1981.

[11] L. S. Czarnecki, "Measurement of the Individual Harmonics Reactive Power in Nonsinusoidal Systems," *IEEE Transactions on Instrumentation and Measurement*, vol. IM-32, pp. 383-384, June 1983.

[12] L. S. Czarnecki, "Measurement Principles of the Reactive Current RMS Value and the Load Susceptance for Harmonic Frequency Meters," *IEEE Transactions on Instrumentation and Measurement*, vol. IM34, pp. 14-17, March 1985.

[13] R. A. Lopez, J. C. M. Asquerno, and G. Rodriguez-Izquerno, "Reactive Power Meter for Nonsinusoidal Systems," *IEEE Transactions on Instrumentation and Measurement*, vol. IM-26, pp. 258-260, September 1977.

[14] G. S. Hope, O. P. Malik and J. Chang, "Microprocessor Based Active and Reactive Power Measurement," *International Journal of Electric Power and Energy Systems*, vol. 3, pp. 75-83, April, 1981.

[15] E. O. Schweitzer, III, "Digital Revenue Metering Algorithm: Development, Analysis, Implementation, Testing, and Evaluation," *EPRI Report*, EL-1601, November 1980.

[16] R. F. Turgel, "Digital Wattmeter Using a Sampling Method," *IEEE Transaction on Instrumentation and Measurement*, vol. IM-23, pp. 337-341, Dec. 1974.

[17] C. H. Dix, "Calculated Performance of a Digital Wattmeter Using Systematic Sampling," *IEEE Proceedings*, vol. 129, pp. 172-175, May 1982.

[18] F. J. J. Clarke and J. R. Stockton, "Principles and Theory of Wattmeters Operating on the Baseis of Regularly Spaced Sample Pairs," *Journal of Physics and Scientific Instruments* vol. 15, pp. 645-652, June 1982.

[19] M. F. Matouka, "A Wide-Range Digital Power/Energy Meter for Systems with Nonsinusoidal Waveforms," *IEEE Transactions on Industrial Eelctronics*, vol. IE-29, pp. 18-31, February 1982.

[20] G. N. Stanbakken, "A Wide-band Sampling Wattmeter," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-103, pp. 2919-2926, Oct. 1984.

[21] A. C. Corney and R. T. Pullman, "Digital Sampling Laboratory Wattmeter," *IEEE Transactions on Instrumentation and Measurement*, vol. im-36, pp. 54-59, March 1987.

[22] V. V. Athani, "Power Systems Measurements Usinf Mocroprocessors," *IETE Technical Review*, vol. 2, pp. 180-184, Aug. 1981.

[23] J. J. Hill and W. E. Alderson, "Design of a Microprocessor Based Digital Wattmeter," *IEEE Transactions on Industrial Electronics and Control Instrumentation*, vol. IECI-28, pp. 180-184, August 1981.

[24] C. I. Budeanu, "Puissances Reactives et Fictives," Institut de l'Energie, Publ. 2, Bucharest Romunia 1927.

[25] B. Don Russell, *Market and Technology Transfer Plan*, pp 2-3. Department of Electrical Engineering, Texas A&M University, September 1989.

[26] M. Kezunovic. "New Approach to the Design of Digital Algorithms for Electric Power Measurements," *IEEE Transactions on Power Delivery*, vol. 6, no. 2, pp. 518-522, April, 1991.

[27] Sharif M. Shahrier. "A TMS320C30 Based Real-time Data Acquisition System," *Internal Report to Don Russell*, Department of Electrical Engineering, Texas A&M University, College Station, Texas, September, 1992.

[28] Steve E. Williams. "Digital Algorithms for Nonsinusoidal Power Calculations," *Internal Report to Don Russell*, Department of Electrical Engineering, Texas A&M University, College Station, Texas, February 1990.

[29] R. E. Abbott, S. G. Hadden, and T. L. Gruber. "Survey of End-Use Metering Equipment and Sensors," *EPRI Report* No. RP 2568-21, February 1992.

APPENDIX A

EQUIPMENT LIST

The following is a detailed list of the hardware and software used in the development of the data acquisition system. This includes a separate list of the hardware used for each of the major system components including the Master Station, the Digital Data Recorder, and the Power Systems Data Acquisition Board.

A. Master Station

IBM compatible PC consiting of::

80386, 20MHz central processing unit

7 Mbytes DRAM

120 Mbyte Hard Disk

Quatech© RS485 AT-Bus serial port card

2-rs232 serial ports

1 parrallel port

1 - 1.2 Mbyte 5.25" floppy drive

1 - 1.44 Mbyte floppy drive

1 - VGA card with 1 Mbyte video ram

B. Digital Data Recorder

IBM compatible PC consisting of:

80286, 12 MHz central processing unit

2 Mbytes DRAM

40 Mbyte Hard Disk

Quatech© RS485 AT-Bus serial port card

1 RS232 serial port

1 parrallel port

1 - 1.2 Mbyte 5.25" floppy drive

1 - 1.44 Mbyte floppy drive

1 - VGA card with 256 Kbytes video ram

C. Power Systems Data Acquisition Board

IBM compatible PC consiting of:

80286, 12 MHz central processing unit

2 Mbytes DRAM

40 Mbyte Hard Disk

1 - Quatech© RS485 AT-Bus serial port card

1 - RS232 serial port

1 - parrallel port

1 - 1.2 Mbyte 5.25" floppy drive

1 - 1.44 Mbyte floppy drive

1 - Monochrome video card

1 - Spirit-30 AT/ISA board with 32 Kbytes, SPOX software

1 - Spectrum 32 channel, 12 bit A/D converter board with Quad S/H

APPENDIX B

SCHEMATIC DIAGRAMS

This section includes a communications-bus schematic diagram of the entire system, Figure 7, and a PC-AT bus level schematic diagram of the Power Systems Data Acquisition Board, Figure 8.

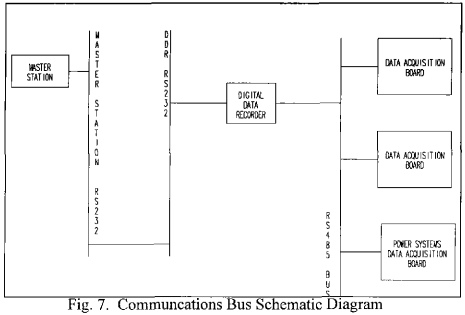

Fig. 7. Communcations Bus Schematic Diagram

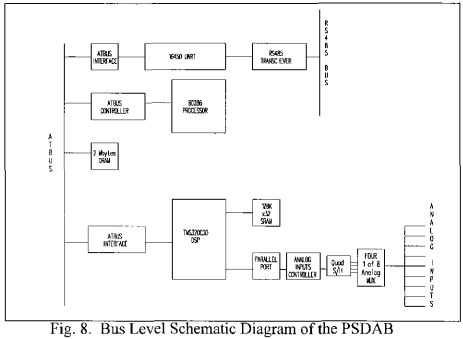

Fig. 8. Bus Level Schematic Diagram of the PSDAB

APPENDIX C

EXAMPLES OF ACQUIRED FIELD TEST DATA

This section includes samples of data acquired during field tests and a few representative graphs of the data. The graphs represent only a portion of the data available to the system's end user and not all of the data acquired is represented by these graphs or sample outputs. Since the system was capable of generating large quantities of data (over 40 Mbytes/day), the following graphs will be of samples covering a very small period of time, usually one minute total with the data points spaced at two second intervals. The first line of Figure 9 is the time stamp for the data. Lines 2-11 are the real and reactive power data for the harmonices 1-10. Columns 1-3 are real power for phases A-C respectively. Columns 4-6 are reactive power for phases A-C respectively. Line twelve has the RMS voltage and current values for each phase.

```
Mon Oct 25 13:12:45 1993
9.752770e+003  1.733573e+004  1.652579e+004  8.818174e+003 4.912281e+003 1.137377e+004
-2.813201e-001 -3.890681e-003 6.239249e-002  -1.052172e+000 -1.066294e-002 -2.321390e-001
1.727009e+001  -1.304336e+001 -5.899220e-001 -3.236081e+001 1.169166e+001 4.823206e+000
2.584696e-002  -6.382391e-005 -4.029695e-002 -3.955369e-001 1.249290e-002 -1.285078e-001
1.248106e+001  -1.958458e+000 8.026574e+000  1.617364e+001 1.697305e+001 2.189764e+001
3.368478e-003  7.862520e-003  9.303925e-003  -1.823849e-002 -2.610909e-003 -4.948118e-003
1.336248e-001  -1.783871e-003 1.180208e-001  1.767647e-001 9.909474e-003 1.151633e-002
1.104150e-002  2.608395e-003  -3.740027e-003 8.400273e-003 -1.427569e-005 9.446409e-003
1.313501e-002  -1.283231e-002 1.219227e-002  -2.284508e-003 -2.194631e-003 -4.144840e-002
2.381521e-003  3.056772e-004  -5.628111e-003 1.041170e-002 -5.704908e-004 2.115273e-003
```

Fig. 9. Sample data gathered during field tests.

Figure 10 is a graph of real power as a function of time for the fundamental harmonic of a three phase system operating at 60hz. Figure 11 is a graph of the power factor for the same three phase system over the same time period.

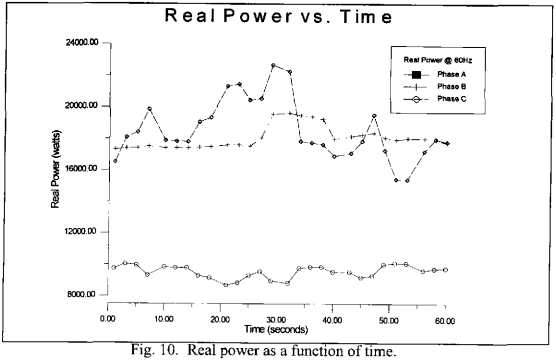

Fig. 10. Real power as a function of time.

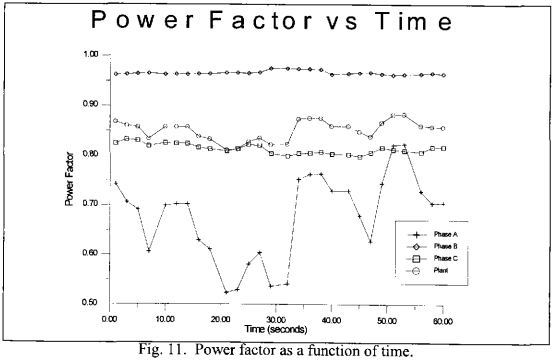

Fig. 11. Power factor as a function of time.

Figure 12 is a graph of the real power as a function of time for the 2nd through the 10th order harmonics for phase A at the field test site. Figure 13 is a graph of the reactive power as a function of time for the 2nd through the 10th order harmonics for phase A at the field test site.

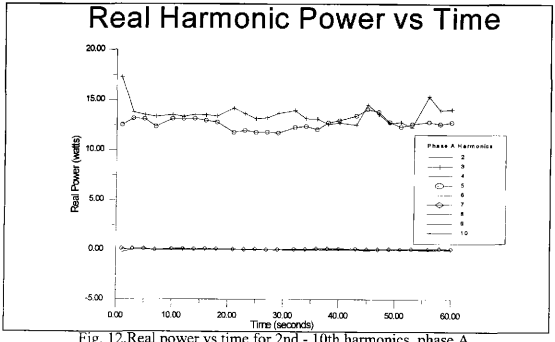

Fig. 12.Real power vs time for 2nd - 10th harmonics, phase A.

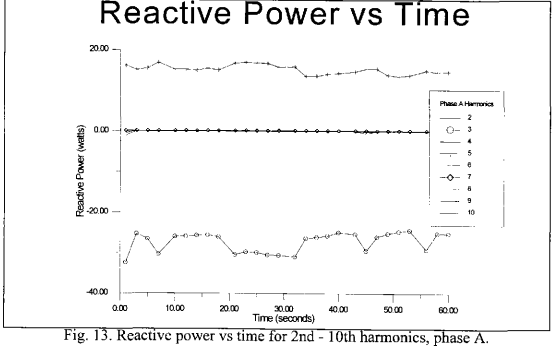

Fig. 13. Reactive power vs time for 2nd - 10th harmonics, phase A.

APPENDIX D

SOURCE CODE LISTINGS

Source code listings for each of the systems major components except for the graphical user interface (GUI). The GUI was based on proprietary code from a third party vendor and cannot be listed here. All other major software systems are listed included both the PC-side and DSP-side of the power systems data acquisition board (PSDAB). The PSDAB code includes the PHA algorithms described in chapter 3.

```
// ********************** PC SIDE CODE **********************
#include <stdio.h>
#include <time.h>
#include "dsptypes.h"
#include "eclproto.h"
#include "xc.h"
#include "daberror.h"
#include "dabcom.h"
#define   ESC 0x001B
#define TIMER_TICK 0x1C
unsigned long tickcount = 0, totalcount;
// void (_interrupt _far *our_handler)(void);
void _interrupt _far our_handler(void);
int port = COM3;
void main(int argc, char *argv[])
{
  int dspstatus;
  unsigned int ch,portstatus, irq, nodeid;
  hARMONICpOWERS *harmonic_data;
  time_t thistime, lasttime;
  int stat, error, i;
  int portstat;
  void (_interrupt _far *old_handler)(void);
  unsigned intno = TIMER_TICK;
  old_handler = _dos_getvect(intno);
  printf("the address of the old handler is : %Fp\n",old_handler);
  _dos_setvect(intno, our_handler);
  printf("\ninstalled new handler: %p\n",our_handler);
  thistime = time(&thistime);
  lasttime = thistime;
  printf("DAB program for crap project #260 \n \nTest print.\n");
```

```
DSPSaveDataOff();
DABSaveDataOff();
DSPSetupFileNumber();
dspstatus = DSPOFF;
for (i=0;i<argc;i++){
        if (strncmp(argv[i],"-nid",4)==0){
                nodeid = argv[i][4];
        }
        if (strncmp(argv[i],"-irq",4)==0){
                irq = argv[i][4];
        }
        if (strncmp(argv[i],"-dsp",4)==0){
                dspstatus = DSPON;
        }
        if (strncmp(argv[i],"-sav",4)==0){
                DSPSaveData();
                DABSaveData();
        }
}
/* A '1' is downloaded to SPIRIT-30 at address 'flag' which      */
/* signals the C30 that the data is ready for execution.         */
if (dspstatus==DSPON){
        error = input_parameters();
        if (error==BAD_FILE_POINTER) exit(1);
        dsp_setup();
        read_samples();
        send_dsp_samples();
        DSPSetupDSPBoard();
}
portstatus = DABComInit(irq, (char)nodeid);
harmonic_data = DABGetCurrentDSPBuffer();
DABRegCurrentSendId(1);
xc_dtr(COM3,1);
xc_cts(COM3);
for (;;) {
        if (xck_keyt()) {       /* A key is struck */
                if ((ch = xck_getc() & 0xFF) == ESC) break;  /* Exit */
                if (ch== 'B') DABTransmitDSPResultsBuffer();  /* Exit */
                error = xc_putc(COM3, (char)ch);
        }
        if (tickcount>=40){
                tickcount = 0;
                printf("another second has gone by!\n");
                if (totalcount>=5700){
//                      DSPNextFileName();
```

```
                totalcount = 0;
            }
            if (dspstatus==DSPON){
                get_dsp_results(harmonic_data);
            }
            else{
                DABRequestDSPData();
            }
        }
        DABCheckComPorts();
    }
    DSPCloseFile();
    if ( portstatus == COMMUNICATIONS_ON ) xc_unlk(port);
    xc_exit();
    _dos_setvect(intno,old_handler);
}
void _interrupt our_handler(void){
    tickcount++;
    totalcount++;
    if (tickcount>40) tickcount = 0;
}
/*********************************************************************/
/************************* The PC side program ***********************/
/*********************************************************************/
#include "sys\types.h"
#include "sys\stat.h"
#include "io.h"
#include "errno.h"
#include "fcntl.h"
#include "dsptypes.h"
#include "dspdata.h"
#include "eclproto.h"
#include "xc.h"
#include "daberror.h"
#include "dabcom.h"
#include "malloc.h"
//
// This function reads the calibration parameters from a disk file
//
#define SAVEOFF     0
#define SAVEON      1
int storedata = 0;
void DSPSaveData(void){
    storedata = SAVEON;
    DSPSetupSaveBuffer();
```

```
}
void DSPSaveDataOff(void){
  storedata = SAVEOFF;
}
int input_parameters()
{
FILE *cal_par_file;
cal_par_file = fopen("CAL_PAR.DAT","r");
if (cal_par_file==NULL){
  printf("Calibration parameter file missing.  Exiting program\n");
  return BAD_FILE_POINTER;
}
fscanf(cal_par_file,"%d %d %d",&hours, &minutes, &seconds);
for (j=0;j<PHASES;j++)
  fscanf(cal_par_file,"%f %f",&calib_params[j][0], &calib_params[j][1]);
return 0;
}
dsp_setup(void)
{
/*************************************************************/
/* This function performs the following:-                    */
/*                                                           */
/* 1. Downloads C30 exectable code                           */
/* 2. Resets the  C30                                        */
/* 3. Obtains SPIRIT-30 addresses from the global symbol     */
/*    table                                                  */
/* 4. Downloads demand window parameters onto SPIRIT-30      */
/* 5. Downloads sensor calibration parameters onto the       */
/*    SPIRIT-30                                              */
/*************************************************************/
  /* Download DSP executable file 'dpsa.out' which performs */
  /* the power harmonic analysis                            */
  if (dsp_dl_exec ("dpsa_db.out") == -1)  {
            printf("error in downloading the file \n");
            exit(1);
    }
  dsp_reset();      /* RESET C30 to give control to dpsa.out  */
  printf("\nWaiting for DSP to reset...\n");
  /* Here is a small delay to ensure that the C30 enters its  */
  /* 'waiting_for_flag_to_set' loop before the PC side program */
  /* initiates communication.                                 */
  for (j=0;j<1000;j++)
    for (l=0;l<300;l++);
  /* To find SPIRIT-30 addresses of labels. These addresses   */
  /* are obtained from the symbol table prepared by dsp_dl_exec() */
```

```
  /* library routine while downloading "dpsa_db.out"          */
  vtemp_addr      = get_laddr("_vtemp");
  ctemp_addr      = get_laddr("_ctemp");
  v_addr          = get_laddr("_v");
  c_addr          = get_laddr("_c");
  p0_addr         = get_laddr("_p0");
  p1_addr         = get_laddr("_p1");
  q0_addr         = get_laddr("_q0");
  q1_addr         = get_laddr("_q1");
  rms0_addr       = get_laddr("_rms0");
  rms1_addr       = get_laddr("_rms1");
  flag_addr       = get_laddr("_flag");
  buffer_addr     = get_laddr("_buffer");
  c2u_addr        = get_laddr("_c2u");
  hrs_addr        = get_laddr("_hrs");
  min_addr        = get_laddr("_min");
  sec_addr        = get_laddr("_sec");
  calib_params_addr = get_laddr("_calib_params");
  dsp_dl_int_array(hrs_addr, 1, &hours);
  dsp_dl_int_array(min_addr, 1, &minutes);
  dsp_dl_int_array(sec_addr, 1, &seconds);
  dsp_dl_long_array(calib_params_addr,SENSORS,(long *)calib_params);
}
send_dsp_samples(void)
{
/*************************************************************/
/* This function downloads three phase current and voltage*/
/* samples onto a buffer on the SPIRIT-30. It is used for */
/* test purposes only.                                       */
/*************************************************************/
    array_size1 = PHASES*SIZE;
    dsp_dl_long_array(vtemp_addr, array_size1, (long *)v);
    dsp_dl_long_array(ctemp_addr, array_size1, (long *)c);
 }

get_dsp_results(hARMONICpOWERS *harmonic_data)
{
/*************************************************************/
/* This function uploads results from the SPIRIT-30. A   */
/* double buffering scheme is maintained on the DSP side */
/*************************************************************/
    array_size2 = (PHASES*(NH+1));
        array_size3 = PHASES*2;
        status=0;
        while (status != 1)
```

```
            dsp_up_int_array(c2u_addr, 1, &status);
            status=0;
            dsp_dl_int_array(c2u_addr, 1, &status);
            dsp_up_int_array(buffer_addr, 1, &buffer); /* read the flag      */
          if (buffer == 0)
            {
              dsp_up_long_array(p0_addr,array_size2,(long *)p);
              dsp_up_long_array(q0_addr,array_size2,(long *)q);
              dsp_up_long_array(rms0_addr,array_size3,(long *)rms);
              printf("\t\t\t\t\t---------ZERO---------\n");
              print2screen(harmonic_data);
            }
            else
            if (buffer == 1)
              {
                   dsp_up_long_array(p1_addr,array_size2,(long *)p);
                   dsp_up_long_array(q1_addr,array_size2,(long *)q);
                   dsp_up_long_array(rms1_addr,array_size3,(long *)rms);
                   printf("\t\t\t\t\t---------ONE---------\n");
                   print2screen(harmonic_data);
              }
}
void read_samples(void)
{
/*****************************************************/
/* This function reads in three phase current and   */
/* voltage samples from data files. It is used for  */
/* test purposes only.                    */
/*****************************************************/
/* Read in v[] from voltage.dat                */
/* Read in c[] from current.dat                */
   for (j=0;j<PHASES;j++)
   {
     vin=fopen("voltage.dat","r");
     cin=fopen("current.dat","r");
     for (k=0;k<SIZE;k++)
     { fscanf(vin,"%f",&v[j][k]);
       fscanf(cin,"%f",&c[j][k]);
     };
     fclose(vin);
     fclose(cin);
   };
}
void print2screen(hARMONICpOWERS *harmonic_data)
{
```

```
/*************************************************************/
/*This function outputs the DSP calculation results onto  */
/*the screen. It is used for test purposes only          */
/*************************************************************/
    int k,j;
    time_t datatime;
    datatime = time(&datatime);
    harmonic_data->datatime = datatime;
    printf("\n RMS v (1) = %5.3f",rms[0][0]);
    printf("\t RMS v (2) = %5.3f",rms[1][0]);
    printf("\t RMS v (3) = %5.3f",rms[2][0]);
    printf("\n RMS c (1) = %5.3f",rms[0][1]);
    printf("\t RMS c (2) = %5.3f",rms[1][1]);
    printf("\t RMS c (3) = %5.3f",rms[2][1]);
    printf("\nThe time is %s",ctime(&harmonic_data->datatime));
    printf("\n");
//    for (k=1;k<=NH;k++)
//      {
//        printf("\n");
//        for (j=0;j<PHASES;j++){
//          printf("%6.5e ",p[j][k]);
//        }
//        for (j=0;j<PHASES;j++){
//          printf("%6.5e ",q[j][k]);
//        }
//      }
   for (j=0; j<PHASES;j++)
     {
        for (k=1;k<=NH;k++){
          harmonic_data->p[j][k] = p[j][k];
          harmonic_data->q[j][k] = q[j][k];
          p[j][k] = 0;
          q[j][k] = 0;
          }
     }
    harmonic_data->rms[0][0] = rms[0][0];
    harmonic_data->rms[1][0] = rms[1][0];
    harmonic_data->rms[2][0] = rms[2][0];
    harmonic_data->rms[0][1] = rms[0][1];
    harmonic_data->rms[1][1] = rms[1][1];
    harmonic_data->rms[2][1] = rms[2][1];
  for (j=0; j<PHASES; j++){
        for (k=0;k<2;k++)
        rms[j][k] = 0;
  }
```

```
  if (storedata==SAVEON) DSPWriteToDisk(harmonic_data);
  }
void DSPSetupDSPBoard(void){
  status = 1;
  dsp_dl_int_array(flag_addr, 1, &status);  /* set the flag */
}
FILE *filehandle;
int filenumber, count, totalwriten, cycles;
char currentfilename[17], alternate[17];
char *outputbuffer;
void DSPSetupFileNumber(void){
  char buffer[17];
  int i;
  filenumber = 0;
  filehandle = NULL;
  cycles = -1;
  count = sizeof(hARMONICpOWERS);
  DSPNextFileName();
}
void DSPNextFileName(void){
  char buffer[17];
  int i;
  filenumber++;
  for (i=0;i<17;i++){
        buffer[i] = 0;
        currentfilename[i] = 0;
        alternate[i] = 0;
  }
  itoa(filenumber, buffer,10);
  sprintf(currentfilename,"dsp%s.dat",buffer);
  sprintf(alternate,"time%s.dat",buffer);
  totalwriten = 0;
//    filehandle = fopen(currentfilename,"aw+b");
//    if (filehandle!=NULL) fclose(filehandle);
//    filehandle = fopen(currentfilename,"aw+b");
}
#define CYCLES      200
#define BYTESTOWRITE   sizeof(hARMONICpOWERS)
void DSPWriteToDisk(hARMONICpOWERS *dspbuffer){
  char *buffer, hourbuffer[3], minutebuffer[3], secondsbuffer[3];
  char *timebuffer;
  int i,j, hour, minutes, seconds;
  hARMONICpOWERS *storedspdata;
  cycles++;
//    storedspdata = (hARMONICpOWERS *) outputbuffer;
```

```
//      storedspdata[cycles] = *dspbuffer;
//      filehandle = fopen(currentfilename,"awb");
//      fwrite(dspbuffer,BYTESTOWRITE,1,filehandle);
//      fclose(filehandle);
  timebuffer = ctime(&dspbuffer->datatime);
  hourbuffer[0] = timebuffer[11];
  hourbuffer[1] = timebuffer[12];
  hourbuffer[2] = 0;
  minutebuffer[0] = timebuffer[14];
  minutebuffer[1] = timebuffer[15];
  minutebuffer[2] = 0;
  secondsbuffer[0] = timebuffer[17];
  secondsbuffer[1] = timebuffer[18];
  secondsbuffer[2] = 0;
  hour = atoi(hourbuffer);
  minutes = atoi(minutebuffer);
  seconds = atoi(secondsbuffer);
  if ((hour<7) || (hour>17)){
        return;
  }
  filehandle = fopen(currentfilename,"aw");
  fprintf(filehandle,"%s",timebuffer);
  for (j=1;j<(NH+1);j++){
        for (i=0;i<PHASES;i++){
              fprintf(filehandle,"%10.6e ",dspbuffer->p[i][j]);
        }
        for (i=0;i<PHASES;i++){
              fprintf(filehandle,"%10.6e ",dspbuffer->q[i][j]);
        }
        fprintf(filehandle,"\n");
  }
//      fprintf(filehandle,"\n");
//      for (i=0;i<PHASES;i++){
//          for (j=1;j<(NH+1);j++){
//              fprintf(filehandle,"%6.3f ",dspbuffer->q[i][j]);
//          }
//      }
//      fprintf(filehandle,"\n");
  for (i=0;i<PHASES;i++){
        for (j=0;j<2;j++){
              fprintf(filehandle,"%5.3f ",dspbuffer->rms[i][j]);
        }
  }
  fprintf(filehandle,"\n");
  fclose(filehandle);
```

```
  if (cycles==CYCLES){
        cycles = 0;
        DSPNextFileName();
        totalwriten = 0;
  }
  totalwriten = totalwriten + 716;
  printf("\ncycle = %u, count = %u, totalwriten = %u\n",cycles,count, totalwriten);
  seconds = seconds + ((hour -7) * 3600) + minutes*60;
  filehandle = fopen("graph1.dat","aw");
  for (phase=0;phase<PHASES;phase++){
        fprintf(filehandle,"%u %u %10.6e",seconds,phase+1,dspbuffer->p[phase][1]);
  }
  close(filehandle);
//    filehandle = fopen("graph2.dat","aw");
//    for (phase=0;phase<PHASES;phase++){
//    fprintf(filehandle,"%u%u%10.6e",seconds,phase+1,dspbuffer->p[phase][1]);
//    }
//    close(filehandle);
}
void DSPCloseFile(void){
  if (filehandle!=NULL) fclose(filehandle);
}
DSPSetupSaveBuffer(void){
  int i;
  outputbuffer = malloc(100 * sizeof(hARMONICpOWERS));
  for (i=0;i<(100*sizeof(hARMONICpOWERS));i++){
        outputbuffer[i] = 0;
  }
}
void DSPSetupDummyHarmonicData(void){
    int k,j;
   for (j=0; j<PHASES;j++)
     {
        for (k=1;k<=NH;k++){
          p[j][k] = 1000020100+k+j;
          q[j][k] = 20200+k+j;
          }
     }
    rms[0][0] = 469.1;
    rms[1][0] = 469.2;
    rms[2][0] = 469.3;
    rms[0][1] = 120.1;
    rms[1][1] = 120.2;
    rms[2][1] = 120.3;
    for (j=0;j<PHASES;j++)
```

```
    {
     for (k=1;k<=NH;k++)
       printf("\n p[%d][%d]= %6.3f \t q[%d][%d]= %6.3f",
       j+1,k,p[j][k],j+1,k,q[j][k]);
       printf("\n");
    }
}
//
//          Communications side to DAB code
//
#include "eclproto.h"
#include "xc.h"
#include "malloc.h"
#include "stdio.h"
#include "sys\types.h"
#include "sys\stat.h"
#include "io.h"
#include <time.h>
#include "errno.h"
#include "fcntl.h"
#include <dos.h>
#include "dsptypes.h"
#include "dabcom.h"
#include "daberror.h"
#define ESC 0x001B
struct {
  char nodeid;
  int  irq;
  char port;
  char status;
  int  bytestoget;
  char *getbuffer;
  int  bytestosend;
  char *sendbuffer;
  } port;
#define SAVEDATAOFF    0
#define SAVEDATAON     1
char hdu_buffer[6], hdu_id[4];
hARMONICpOWERS _far *currentHPbuffer;
char currentsendid;
int savestatus;
void DABSaveData(void){
  savestatus = SAVEDATAON;
}
void DABSaveDataOff(void){
```

```
  savestatus = SAVEDATAOFF;
}
void DABRegCurrentSendId(char newport){
  currentsendid = newport;
}
void DABRegDSPBuffer(hARMONICpOWERS _far *newbuffer){
  currentHPbuffer = newbuffer;
}
hARMONICpOWERS _far *DABGetCurrentDSPBuffer(void){
  return currentHPbuffer;
}
int DummyRecieveRoutine(char *dummy, int dummycount)
{
  printf("Recieved stuff.\n");
  return 0;
}
void DummyDSPSend(void){
}
void DummyMessageHandler(char stuff){
}
int DABGetPortStatus(void){
  return port.status;
}
void DABSetPortStatus(char status){
  port.status = status;
}
int DABGetDSPData(void){
  int bytestoget, bytes_read, i;
  char *buffer;
  bytestoget = sizeof(struct hARMONICpOWERS);
  bytes_read =bytestoget;
  buffer = (char *) currentHPbuffer;
  while (xc_test(COM3)<bytestoget);
  bytes_read = xc_get(COM3,buffer,&bytestoget);
  port.status = STANDBY;
  printf("Data In.\n");
  for (i=0;i<3;i++){
        printf("rms[%u][0]=%f, rms[%u][1]=%f\n",i, currentHPbuffer->rms[i][0],
        i,currentHPbuffer->rms[i][1]);
  }
  printf("\nThe time is %s\n",ctime(&currentHPbuffer->datatime));
  if (savestatus==SAVEDATAON) DSPWriteToDisk(currentHPbuffer);
  return 0;
}
int DABCheckPortStatus(void){
```

```
    return port.status;
}
int DABComInit(int irq,char nodeid)
{
    int port_data;
    int error;
    error = xc_entr(8);
    port.status = STANDBY;
    hdu_buffer[5] = '\0';
    hdu_id[0] = 'H';
    hdu_id[1] = 'D';
    hdu_id[2] = 'U';
    port.nodeid = nodeid - 48;
    port.irq = irq - 48;
    if (port.irq<3 || port.irq>5) port.irq = 3;
    hdu_id[3] = port.nodeid;
    port.port = COM3;
    setport(0x3E8,1,port.irq,COM3,0);
    xc_link(COM3,0);
    xc_init(COM3,BAUD57K,NOPAR,DATA8,STOP1);
    xc_dtr(COM3,1);
    currentHPbuffer = (hARMONICpOWERS *) malloc(sizeof(struct
            hARMONICpOWERS));
    return COMMUNICATIONS_ON;
}
int DABCheckComPorts(void){
    int return_value,i;
    char *bytestoget;
    return_value = 0;
    switch(port.status){
    default: if (xc_test(port.port)){
            for (i=0;i<4;i++){
                hdu_buffer[i] = hdu_buffer[i+1];
            }
            hdu_buffer[4] = xc_getc(port.port);
            printf("%c",hdu_buffer[4]);
            if (strncmp(hdu_buffer,hdu_id,4)==0){
                return_value = hdu_buffer[4];
                switch(hdu_buffer[4]){
                    case SENDDSPDATA: port.status=STANDBY;
                        DABTransmitDSPResults();
                        break;
                    case RECEIVEDSPDATA: port.status=STANDBY;
                        DABGetDSPData();
                        break;
```

```
                        default :
                                break;
                }
        }
        }
    }
    return return_value;
}
void DABTransmitDSPResults(void){
    int numofbytes, count, i;
    char *sendbuffer,*copypointer;
    char buffer[] = { 'H','D','U' };
    count = 3;
    xc_put(COM3,buffer,&count);
    xc_putc(COM3,currentsendid);
    xc_putc(COM3,(char) RECEIVEDSPDATA);
    numofbytes = sizeof(hARMONICpOWERS);
    printf("transmitting!\n");
    xc_put(COM3,(char *)currentHPbuffer,&numofbytes);
}
void DABTransmitDSPResultsBuffer(void){
    int numofbytes,  i;
    char *sendbuffer;
    numofbytes = sizeof(hARMONICpOWERS);
    printf("transmitting!\n");
    sendbuffer = malloc(numofbytes*sizeof(char));
    for (i=0;i<numofbytes;i++){
            sendbuffer[i] = 'A';
    }
    xc_put(COM3,sendbuffer,&numofbytes);
    free(sendbuffer);
}
void DABRequestDSPData(void){
    int count;
    char buffer[] = { 'H','D','U' };
    count = 3;
    xc_put(COM3,buffer,&count);
    xc_putc(COM3,currentsendid);
    xc_putc(COM3,(char) SENDDSPDATA);
}
//
//    Communications side to DAB code - Header File Only
//
#define COMMUNICATIONS_ON      1
#define COMMUNICATIONS_OFF     -1
```

```
#define STILL_SENDING        -1
#define BUFFER_IN            5
// communications status constants
#define STANDBY      0
#define SENDING      1
#define RECIEVING    2
#define SENDDSPDATA  3
#define RECEIVEDSPDATA 4
int DABComInit(int, char);
int DABCheckComPorts(void);
void DABSendStuff(char,char,int,char*);
void RegTransmittComplete(void (*)(void));
void RegRecieveComplete(void (*)(char *,int));
void RegDABDftMsgHandler(void (*)(char));
void RegDABSendDSPData(void (*)(void));
void (*NotifyRecieveComplete)(char *, int);
void (*NotifyTransmittComplete)(void);
void (*DABSendDSPData)(void);
void (*DABDefaultMessageHandler)(char);
void DABRegDSPBuffer(hARMONICpOWERS *);
void DABTransmitDSPResults(void);
void DABRegCurrentSendId(char);
void DABTransmitDSPResultsBuffer(void);
hARMONICpOWERS _far *DABGetCurrentDSPBuffer(void);
void DABRequestDSPData(void);
void DABSaveData(void);
void DABSaveDataOff(void);
//
//          DAB error codes
//
#define BAD_FILE_POINTER -1
#define BAD_POINTER      -1
#define OUT_OF_MEMORY    -2
#include <stdio.h>
#include "s30tools.h"
#include "dspconst.h"
#include <time.h>
/* SOME GLOBALS */
//typedef struct dostime_t {
//      unsigned char hour;    /* 0-23 */
//      unsigned char minute;  /* 0-59 */
//      unsigned char second;  /* 0-59 */
//      unsigned char hsecond; /* 0-99 */
//      }dostime_t;
typedef struct hARMONICpOWERS{
```

```
    time_t datatime;
    float  p[PHASES][NH+1];
    float  q[PHASES][NH+1];
    float  rms[PHASES][2];
}hARMONICpOWERS;
//typedef struct hARMONICpOWERS{
//float rms[PHASES][2];
//}hARMONICpOWERS;
void print2screen(hARMONICpOWERS *);
void DSPSaveData(void);
void DSPWriteToDisk(hARMONICpOWERS *);
void DSPSaveDataOff(void);
void DSPSetupFileNumber(void);
void DSPNextFileName(void);
void DSPCloseFile(void);
DSPSetupSaveBuffer(void);
#include <stdio.h>
#include "s30tools.h"
/* SOME GLOBALS */
#define SIZE 22
#define NH  10
#define PHASES 3
#define RMS2 2
#define SENSORS 6
FILE *vin, *cin;
int buffer;
int c2u;
int j, k;
int l;
int status;
int hours;
int minutes;
int seconds;
int sample_flag=0;
int array_size1;
int array_size2;
int array_size3;
float calib_params[PHASES][RMS2];
float phase[PHASES];
float v[PHASES][SIZE], c[PHASES][SIZE];
float p[PHASES][NH+1];
float q[PHASES][NH+1];
float rms[PHASES][2];
/* The following are set equal to the corresponding absolute      */
/* address of the label in the DSP program by calling 'get_laddr()' */
```

```
/* with the label's name                                    */
long v_addr  = 0;
long c_addr  = 0;
long vtemp_addr = 0;
long ctemp_addr = 0;
long p0_addr = 0;
long p1_addr = 0;
long q0_addr = 0;
long q1_addr = 0;
long rms0_addr=0;
long rms1_addr=0;
long flag_addr=0;
long hrs_addr=0;
long min_addr=0;
long sec_addr=0;
long buffer_addr = 0;
long c2u_addr = 0;
long calib_params_addr = 0;
long phase_addr = 0;
/*************************************************************************/
/*********************** The DSP side program ************************/
/*************************************************************************/
#include<float.h>
#include<math.h>
#include<stdlib.h>
#include "dpsa_db.h"
main()
 {
 long window_size, ctr;
 int a,s,i,k,j,h;
 int hours;
 int minutes;
 int seconds;
 float vstemp[PHASES], cstemp[PHASES];
 float temp_p,temp_q;
 float proto(int j);
 float qh(int h);
 float ph(int h);
 float active_power(int j);
 float reactive_power(int j);
 /* Configure primary bus for zero wait states */
 waitst();
 /* Set expansion bus wait states */
 i = *exp_bus_ctl_reg;       /* Get current status */
 i &= 0xffffff07;            /* Mask appropriate bits */
```

```
 i |= 0xc8;                /* Set to 7 wait states */
 *exp_bus_ctl_reg = i;     /* Write back */
while (flag != 1);
for (h=1;h<=NH;h++)
  { ph(h);
    qh(h);
  }
dsp30(calib_params, RMS3);
dsp30(phase, PHASES);
while (1)
{
if (buffer == 1)
{
  for (i=0;i<PHASES;i++)
    {
      for (h=0;h<NH+1;h++)
        {p0[i][h] = 0;
         q0[i][h] = 0;
        };
      for (h=0;h<2;h++)
        rms0[i][h] = 0;
    };
window_size = (hrs*3600) + (min*60) + sec;
for (ctr=0;ctr<window_size;ctr++)
{
for (a=0;a<ONE_SEC_WINDOW;a++)
/*This is a one second demand window*/
{
            daq ();
            for (j=0;j<PHASES;j++)
              {
                for (k=0;k<SIZE;k++)
                  {
                    v[j][k] = v[j][k]*calib_params[j][0];
                    c[j][k] = c[j][k]*calib_params[j][1];
                  };
                  vstemp[j]=0;
                  cstemp[j]=0;
              };
            for (j=0;j<PHASES;j++)
              {
                 for (s=0;s<SIZE;s++)
                   {vstemp[j]  += v[j][s]*v[j][s];
                    cstemp[j]  += c[j][s]*c[j][s];
                   };
```

```
                };
                for (j=0;j<PHASES;j++)
                  {
                    rms0[j][0] += sqrt(vstemp[j]/SIZE);
                    rms0[j][1] += sqrt(cstemp[j]/SIZE);
                  };
                  for (j=0; j<PHASES;j++)
                    {
                      for (k=0;k<SIZE;k++)
                        {v_temp[k] = v[j][k];
                         c_temp[k] = c[j][k];
                        };
                        for (h=1;h<=NH;h++)
                           {temp_p = active_power(h);
                            temp_q = reactive_power(h);
                            p0[j][h] += temp_p;
                            q0[j][h] += temp_q;
                            };
                    };
                }
              }
/* To compute the integral average of the line parameters      */
for (j=0;j<PHASES;j++)
   {
     rms0[j][0] = rms0[j][0]/(ONE_SEC_WINDOW*window_size);
     rms0[j][1] = rms0[j][1]/(ONE_SEC_WINDOW*window_size);
     for (h=1;h<=NH;h++)
       {p0[j][h] = p0[j][h]/(ONE_SEC_WINDOW*window_size);
        q0[j][h] = q0[j][h]/(ONE_SEC_WINDOW*window_size);
        /* Apply phase correction to p and q */
        temp_p = p0[j][h];
        temp_q = q0[j][h];
        p0[j][h] = temp_p*cos(phase[j]) + temp_q*sin(phase[j]);
        q0[j][h] = temp_q*cos(phase[j]) - temp_p*sin(phase[j]);
       };
   };
ieee30(p0,NH3);
ieee30(q0,NH3);
ieee30(rms0, RMS3);
buffer = 0;
c2u = 1;
}
else if (buffer == 0)
   {
    for (i=0;i<PHASES;i++)
```

```
    {
      for (h=0;h<NH+1;h++)
        {p1[i][h] = 0;
         q1[i][h] = 0;
        };
        for (h=0;h<2;h++)
          rms1[i][h] = 0;
    };
window_size = (hrs*3600) + (min*60) + sec;
for (ctr=0;ctr<window_size;ctr++)
{
for (a=0;a<ONE_SEC_WINDOW;a++)
/*This is a one second demand window*/
{
            daq ();
            for (j=0;j<PHASES;j++)
              {
              for (k=0;k<SIZE;k++)
                  {
                    v[j][k] = v[j][k]*calib_params[j][0];
                    c[j][k] = c[j][k]*calib_params[j][1];
                  };
                 vstemp[j]=0;
                 cstemp[j]=0;
              };
              for (j=0;j<PHASES;j++)
                {
                  for (s=0;s<SIZE;s++)
                     {vstemp[j]  += v[j][s]*v[j][s];
                      cstemp[j]  += c[j][s]*c[j][s];
                     };
                };
                for (j=0;j<PHASES;j++)
                  {
                    rms1[j][0] += sqrt(vstemp[j]/SIZE);
                    rms1[j][1] += sqrt(cstemp[j]/SIZE);
                  };
                  for (j=0;j<PHASES;j++)
                    {
                      for (k=0;k<SIZE;k++)
                        {
                          v_temp[k] = v[j][k];
                          c_temp[k] = c[j][k];
                        };
                        for (h=1;h<=NH;h++)
```

```
                        {temp_p = active_power(h);
                         temp_q = reactive_power(h);
                         p1[j][h] += temp_p;
                         q1[j][h] += temp_q;
                        };
                    };
                  }
                }
  /* To compute the integral average of the line parameters      */
  for (j=0;j<PHASES;j++)
    {
      rms1[j][0] = rms1[j][0]/(ONE_SEC_WINDOW*window_size);
      rms1[j][1] = rms1[j][1]/(ONE_SEC_WINDOW*window_size);
      for (h=1;h<=NH;h++)
        {p1[j][h] = p1[j][h]/(ONE_SEC_WINDOW*window_size);
         q1[j][h] = q1[j][h]/(ONE_SEC_WINDOW*window_size);
         /* Apply phase correction to p and q */
         temp_p = p1[j][h];
         temp_q = q1[j][h];
         p1[j][h] = temp_p*cos(phase[j]) + temp_q*sin(phase[j]);
         q1[j][h] = temp_q*cos(phase[j]) - temp_p*sin(phase[j]);
        };
    };
  ieee30(p1,NH3);
  ieee30(q1,NH3);
  ieee30(rms1, RMS3);
  buffer = 1;
  c2u = 1;
}
}
}
/*********************************************************************/
/* matrix multiplier */
float active_power(int j)
{
 int k,m;
 float pc=0;
 for (k=0;k<SIZE;k++)
 { for (m=0;m<SIZE;m++)
   pc += c_temp[k]*v_temp[m]*hp[j][k][m];
 }
 return(pc);
}
/*********************************************************************/
/* matrix multiplier */
```

```
float reactive_power(int j)
{
  int k,m;
  float pc=0;
  for (k=0;k<SIZE;k++)
  { for (m=0;m<SIZE;m++)
    pc += c_temp[k]*v_temp[m]*hq[j][k][m];
  }
  return(pc);
}
/*********************************************************************/
/* Reactive power weight matrix builder */
float qh(int l)
{
  int k,m;
  for (k=0;k<SIZE;k++)
  {  for (m=0;m<SIZE;m++)
     hq[l][k][m]=2*sin(2*PI*l*(k-m)/SIZE)/(SIZE*SIZE);
  }
  return;
}
/*********************************************************************/
/* Active power weight matrix builder */
float ph(int l)
{
  int k,m;
  for (k=0;k<SIZE;k++)
  {  for (m=0;m<SIZE;m++)
     hp[l][k][m]=2*cos(2*PI*l*(k-m)/SIZE)/(SIZE*SIZE);
  }
  return;
}
/*********************************************************************/
daq ()
{
/*********************************************************************/
/* This function performs data acquisation. It acquires 25 samples from  */
/* each current and voltage channel in a three phase power system. The   */
/* samples are required by the main C30 caculation routine to determine  */
/* the fundamental RMS currents, voltages and real and reactive powers of */
/* the first ten harmonics.                                              */
/*********************************************************************/
int temp;
int i,k;
  /* Clear all buffers */
```

```
for (k=0;k<PHASES;k++)
    for (i = 0; i < SAMPLES; i++)
      {
        v[k][i] = 0;
        c[k][i] = 0;
      };
/* Apply positive calibration pulse */
*control_reg_ptr &= !CALD;
*control_reg_ptr |= CALD;
*control_reg_ptr &= !CALD;
/* Loop until calibration complete */
while (!(*control_reg_ptr & CALD)) ;
/* Set sample rate timer */
*timer_reg_ptr = SAMPLE_RATE;
/* Acquire current and voltage samples from phase 1 */
for (i=0; i<SAMPLES; i++)
  {
    /* Set track bit */
    *control_reg_ptr = 0x40;
    /* Clear status flags with dummy read on data */
    temp = *timer_reg_ptr;
    /* Poll status register AIB until S/H is triggered */
    while (!(*control_reg_ptr & EOC)) ;
    /* Read result from ADC */
    temp = *timer_reg_ptr;
    /* Set hold bit */
    *control_reg_ptr = 0x01;
    /* Poll status register of AIB until S/H is in HOLD mode */
    while (!(*control_reg_ptr & EOC));
    /* Read result from ADC */
    temp = *timer_reg_ptr;
    /* Set hold bit and channel 0*/
    *control_reg_ptr = 0x00;
    /* Poll status register of AIB until EOC is set */
    while (!(*control_reg_ptr & EOC));
    /* Read result from ADC */
    temp = *timer_reg_ptr;
    temp &= MASK;   /*Mask appropriate bits */
    if (temp>32767)
        temp = temp - WORD17;
    temp = temp/16; /*Scale voltage*/
    v[0][i] = temp;
    /* Set hold bit and channel 1*/
    *control_reg_ptr = 0x01;
    /* Poll status register of AIB until EOC is set */
```

```
    while (!(*control_reg_ptr & EOC));
    /* Read result from ADC */
    temp = *timer_reg_ptr;
    temp &= MASK; /*Mask appropriate bits */
    if (temp>32767)
        temp = temp - WORD17;
    temp = temp/16; /* scale current */
    c[0][i] = temp;
  }
/* Acquire current and voltage samples from phase 2 */
for (i=0; i<SAMPLES; i++)
  {
    /* Set track bit */
    *control_reg_ptr = 0x44;
    /* Clear status flags with dummy read on data */
    temp = *timer_reg_ptr;
    /* Poll status register of AIB until S/H is triggered */
    while (!(*control_reg_ptr & EOC)) ;
    /* Read result from ADC */
    temp = *timer_reg_ptr;
    /* Set hold bit */
    *control_reg_ptr = 0x05;
    /* Poll status register of AIB until S/H is in HOLD mode */
    while (!(*control_reg_ptr & EOC)) ;
    /* Read result from ADC */
    temp = *timer_reg_ptr;
    /* Set hold bit and channel 4*/
    *control_reg_ptr = 0x04;
    /* Poll status register of AIB until EOC is set */
    while (!(*control_reg_ptr & EOC)) ;
    /* Read result from ADC */
    temp = *timer_reg_ptr;
    temp &= MASK; /* Mask appropriate bits */
    if (temp>32767)
        temp = temp - WORD17;
    temp = temp/16; /* Scale voltage */
    v[1][i] = temp;
    /* Set hold bit and channel 5 */
    *control_reg_ptr = 0x05;
    /* Poll status register of AIB until EOC set */
    while (!(*control_reg_ptr & EOC));
    /* Read result from */
    temp = *timer_reg_ptr;
    temp &= MASK; /* Mask appropriate bits */
    if (temp>32767)
```

```
            temp = temp - WORD17;
        temp = temp/16; /* Scale current */
        c[1][i] = temp;
    }
    /* Acquire current and voltage samples from phase 3 */
    for (i=0; i<SAMPLES; i++)
      {
        /* Set track bit */
        *control_reg_ptr = 0x48;
        /* Clear status flags with dummy read on data */
        temp = *timer_reg_ptr;
        /* Poll status register of AIB until S/H is triggered */
        while (!(*control_reg_ptr & EOC)) ;
        /* Read result from ADC */
        temp = *timer_reg_ptr;
       /* Set hold bit */
        *control_reg_ptr = 0x09;
        /* Poll status register of AIB until S/H is in HOLD mode */
        while (!(*control_reg_ptr & EOC)) ;
        /* Read result from ADC */
        temp = *timer_reg_ptr;
       /* Set hold bit and channel 8*/
        *control_reg_ptr = 0x08;
        /* Poll status register of AIB until S/H EOC is set */
        while (!(*control_reg_ptr & EOC)) ;
        /* Read result from ADC */
        temp = *timer_reg_ptr;
        temp &= MASK; /* Mask approriate bits */
        if (temp>32767)
            temp = temp - WORD17;
        temp = temp/16; /* Scale voltage */
        v[2][i] = temp;
        /* Set hold bit and channel 9 */
        *control_reg_ptr = 0x09;
        /* Poll status register of AIB until EOC is set */
        while (!(*control_reg_ptr & EOC));
        /* Read result from ADC */
        temp = *timer_reg_ptr;
        temp &= MASK; /* Mask appropriate bits */
        if (temp>32767)
            temp = temp - WORD17;
        temp = temp/16; /* Scale current */
        c[2][i] = temp;
}
}
```

```
// ******************************************************
// ************  DPSA_DB.H  ***********************
// ******************************************************
/* SOME CONSTANTS */
#define PI 4*atan(1.0)
#define NH 10
#define SIZE 22
#define SIZE3 66
#define ONE_SEC_WINDOW 10
#define PHASES 3
#define NH3 33
#define RMS3 6
#define RMS2 2
#define SAMPLE_RATE 0xe854
#define SAMPLES 22
#define MASK 0xfff0
#define WORD16 32768
#define WORD17 65536
#define EOC  1 << 7
#define CALD 1 << 5
/*  SOME GLOBALS */
int flag = 0;    /* semaphor for PC-DSP communication     */
          /* -initialize to zero (reset)          */
int buffer = 0;  /*Active high*/
int *control_reg_ptr = (int *)0x804002; /*AIB control and status registers*/
int *timer_reg_ptr  = (int *)0x804003; /*AIB timer and ADC result registers*/
int *exp_bus_ctl_reg = (int *)0x808060; /*C30 expansion bus control register*/
int c2u = 0;
int hrs, min, sec;/*Demand window parameters              */
float calib_params[PHASES][RMS2];     /*amplitude calibration parameters*/
float phase[PHASES];                  /*phase calibration parameters*/
float v[PHASES][SIZE];   /* voltage matrix []                  */
float vtemp[PHASES][SIZE];
float c[PHASES][SIZE];   /* current matrix []                  */
float ctemp[PHASES][SIZE];
float v_temp[SIZE];      /* voltage matrix [] for temporary storage */
float c_temp[SIZE];      /* current matrix [] for temporary storage */
float hp[NH+1][SIZE][SIZE]; /* H matrix - real power of all harmonics */
float hq[NH+1][SIZE][SIZE]; /* H matrix -reactive power of all harmonics */
float p0[PHASES][NH+1], q0[PHASES][NH+1];
float p1[PHASES][NH+1], q1[PHASES][NH+1];
float rms0[PHASES][RMS2], rms1[PHASES][RMS2];
```

VITA

Michael Andrew Davis received his B.S. in Mechanical Engineering from Texas A&M University in August of 1986. After receiving his degree, Mr. Davis was employed as a technical support engineer to Central Power & Light Company's Home Office Marketing Department. He left CP&L in August of 1989 to pursue his M.S. in Mechanical Engineering at Texas A&M. While at A&M he worked for the Department of Electrical Engineering as a Research Associate and served as the project engineer for ERAP project #260. After the ERAP project he was employed by Texas Digital Systems in College Station, Texas as the Manager of Software Development. He left TDS in February 1993 to work as a consultant and is currently developing software for conducting energy audits with pen-based computers. He is currently living at 305 Pershing, College Station, TX, 7840, and can be reached at (409) 764-7639.