

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY
CAMPUS MONTERREY

DIVISION DE INGENIERIA Y ARQUITECTURA
PROGRAMA DE GRADUADOS EN INGENIERIA



GO-BACK-N ARQ MULTIPLE STATION
DIGITAL WIRELESS POWER MONITORING SYSTEM DESIGN

T E S I S

PRESENTADA COMO REQUISITO PARCIAL PARA
OBTENER EL GRADO ACADÉMICO DE
MAESTRO EN CIENCIAS
CON ESPECIALIDAD EN INGENIERIA ELECTRICA

ING. ADRIAN ARNOLDO PINEDA DAVILA

MAYO DE 1999

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY
CAMPUS MONTERREY

DIVISION DE INGENIERIA Y ARQUITECTURA
PROGRAMA DE GRADUADOS EN INGENIERIA



GO-BACK-N ARQ MULTIPLE STATION
DIGITAL WIRELESS POWER MONITORING SYSTEM DESIGN

T E S I S

PRESENTADA COMO REQUISITO PARCIAL PARA
OBTENER EL GRADO ACADÉMICO DE
MAESTRO EN CIENCIAS
CON ESPECIALIDAD EN INGENIERIA ELECTRICA

ING. ADRIAN ARNOLDO PINEDA DAVILA

MAYO DE 1999

Master's Thesis

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at ITESM, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Any other reproduction for any purposes or by any means shall not be allowed without my written permission.

Signature Al A P. J.

Date May 31 1999

ABSTRACT

This thesis describes the design of a multiple station digital wireless power monitoring system for industrial plants. The objective of this thesis is the design of a wireless power monitoring system, complete with transducer, and the implementation of a four-station prototype. The prototype system implemented does not integrate the target power transducer. Instead, the remote stations are programmed to transmit token data in place of the power measurements. The base station is designed to call the remote stations 3 or 10 times (samples) on a system reset or communications recheck, respectively. Also, the base station is designed to call the remote stations 288 times when acquiring the power measurements. This number of samples would represent one full day of power monitoring with samples taken every five minutes.

The communications protocol functions although improvements can be made. The tested prototype system can conduct a reset of the entire system, recheck of the communication, and acquisition of data.

TABLE OF CONTENTS

List of TABLES	v
List of FIGURES	vi
1 INTRODUCTION	9
2 JUSTIFICATION OF WIRELESS POWER MONITORING SYSTEM	14
Hard-Wired Demand Metering	14
Efficient Use of Electrical Energy	20
Other Benefits	23
3 SYSTEM DESIGN	24
Base Laptop	24
Network Topology	27
Communications Protocol.....	27
Network Control Access Scheme	27
Retransmission Technique	28
Frame Structure.....	29
Base Station	35
Hardware.....	35
Embedded Controller Software.....	43
Remote Stations	46
Hardware.....	46
Embedded Controller Software.....	50
4 TESTING AND IMPLEMENTATION	53
5 CONCLUSION.....	57
REFERENCES	62
APPENDIX A: SCHEMATICS AND PROGRAMMING	63

LIST OF TABLES

<i>Number</i>	<i>Page</i>
Table 1. Microchip PIC16C77/JW features.....	11
Table 2. EIA/TIA-232-E and EIA/TIA-485 standards.	20
Table 3. Visual Basic interface screen command function descriptions.	24
Table 4. Number of bytes uploaded to data.txt.	26
Table 5. Format of one line in data.txt.....	26
Table 6. Serial EEPROM connection summary.	37
Table 7. SAMES 9604A features.....	47
Table 8. SAMES 9604A register contents and addresses.....	48
Table 9. Total project costs.....	57
Table 10. Base station cost summary.....	58
Table 11. Remote station cost summary.	59

LIST OF FIGURES

<i>Number</i>	<i>Page</i>
Figure 1. Network topology for power monitoring system.	9
Figure 2. Power monitoring prototype system block diagram.....	10
Figure 3. ITESM multiple point hard-wired power monitoring system.	15
Figure 4. Transmission line equivalent circuits.	17
Figure 5. RC circuit step response.	18
Figure 6. Pulse train on a transmission line.	18
Figure 7. Load profile of two transformers, ITESM.	21
Figure 8. Generalized go-back-N ARQ. Frame 3 was received in error.	29
Figure 9. Implemented go-back-N ARQ. Station 3 frame A received in error.	29
Figure 10. Frame structure.	31
Figure 11. Information field structure, paket type definitions.	32
Figure 12. Frame sequence on RESET, RECHECK, and TDM DATA.....	35
Figure 13. Controller to RPC chip byte transfer timing diagram.	41
Figure 14. RPC chip to Controller byte transfer timing diagram.	41
Figure 15. TX2 transmitter block diagram.	42
Figure 16. RX2 receiver block diagram.....	43
Figure 17. Communications timing diagrams.....	44
Figure 18. Results of test 1 RESET.	53
Figure 19. Results of test 2 RESET.	54
Figure 20. Results of test 3 TDM DATA.	55
Figure 21. Remote station 1 simulated results.	56
Figure 22. Three-Phase Energy Station 1.	56

ACKNOWLEDGMENTS

The author wishes to thank the Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM) and Texas Tech University (TTU) for the challenging opportunity of obtaining the Master of Science in Electrical Engineering degrees.

Special thanks to the Pineda and Dávila families, in the USA and México, for the help and support throughout this unique opportunity.

To Dr. Armando Llamas (ITESM) and Dr. Michael Parten (TTU): thanks for serving as thesis advisors for this project.

To the Hutton and Crawford families: special thanks for the unconditional support and friendship.

To the many friends I have made while in graduate school in the USA and México. To all, thanks for the memorable moments, support, and friendship. May we all take our limitless potential and achieve great things.

“Say to yourself, ‘My place is at the top.’ Be king in your dreams.”

– Andrew Carnegie 1889

DEDICATION

The author wishes to dedicate this thesis to his mother and father.

Through their courage and dedication I learned to envision great things and reach towards the impossible.

This thesis is dedicated to them for their unconditional sacrifices to our family that have blessed us with many opportunities.

1 INTRODUCTION

Power metering in large industrial plants in the Monterrey, Nuevo León México area is a promising area of opportunity. Monterrey, known as the industrial capital of México, is home to manufacturing plants of many international companies. By metering electrical power consumption within a complex, many of these industrial sites can take measures to save money by efficiently using electrical energy.

Figure 1 illustrates the network topology of the prototype power monitoring system. The bus is a wireless 433.92 MHz channel. All nodes are equipped with transmitter/receiver pairs operating at the bus frequency. A command packet from the base station is broadcast to all remote stations. Upon receiving a valid command, all stations execute immediately and wait for their token to be placed on the bus. After broadcasting a command, the base station will place a token on the bus, with station destination and source address information, to be received by station 1. Station 1 responds to the token by broadcasting its message while all other stations await their token. The token will circle clockwise in Figure 1 until reaching its final destination, the base station.

Network Topology : Token Bus

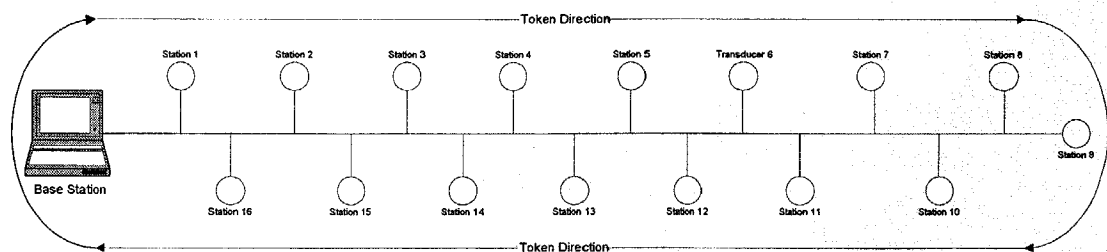


Figure 1. Network topology for power monitoring system.

Figure 2 illustrates the block diagram of the monitoring system.

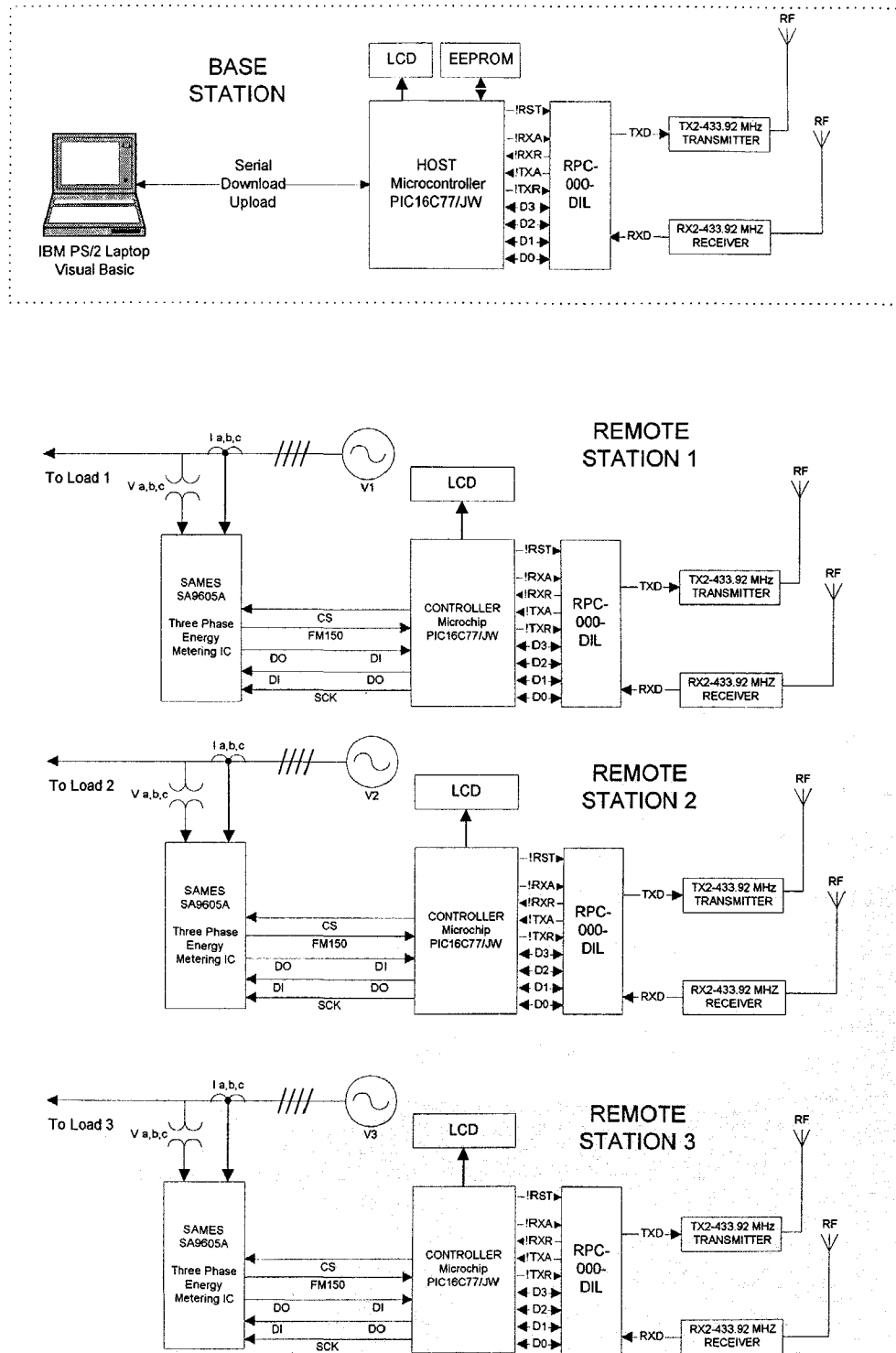


Figure 2. Wireless power monitoring system block diagram.

The central consists of one IBM PS/2 laptop, one Microchip PIC16C77/JW embedded controller, one Radiometrix 433.92 MHz TX2/RX2 transmitter/receiver pair, one Radiometrix RPC 000-DIL, one Optrex DMC 20171 liquid crystal display, one Texas Instruments MAX232 EIA/TIA-232-E transceiver, and two ATMEL AT24C256 32 Kbyte EEPROMs. A system control and data acquisition program developed in Visual Basic is resident in the laptop.

The remote stations consist of a Microchip PIC16C77/JW embedded controller, one Radiometrix 433.92 MHz TX2/RX2 transmitter/receiver pair, one Radiometrix RPC 000-DIL, one Optrex DMC 20171 liquid crystal display, and one SAMES SA9604A three phase energy metering IC. The power monitoring is done on the secondary side of the substation transformers. A substation is defined as a distribution point where a step down transformer sources a load.

The embedded controllers, all Microchip PIC16C77/JW, are 8-Bit CMOS micro-controllers. These devices employ a RISC architecture. "The separate instruction and data buses of the Harvard architecture allow a 14-bit wide instruction word with the separate 8-bit wide data. The two stage instruction pipeline allows all instructions to execute in a single cycle, except for program branches which require two cycles." [1] Table 1 illustrates the important features considered for this project.

Table 1. Microchip PIC16C77/JW features.

FEATURE	Microchip PIC16C77/JW
Oscillator Speed	20 MHz
Instruction Cycle Time	200 ns
Program Memory (Bytes) x 14	8K

Data Memory (Bytes) x 8	368
I/O Pins	33
Parallel Slave Port	Yes
Timer Modules	3
A/D Channels	8
Serial Communication	SSP(SPI/I ² C), USART
In-Circuit Serial Programming	Yes
Interrupt Sources	12
Watch Dog Timer	Yes
Assembly Instructions to Learn	35

The Microchip PIC16C77/JW was also chosen due to its compatibility with Micro Engineering Labs, Inc. PicBasic Pro Compiler. The programs are written in a derivative basic code making prototyping very rapid. Assembly language code can be intermixed with the basic program, using “@” for single line insertion or “ASM” followed by an assembly code block then “ENDASM”, for a very powerful means of executing detailed tasks. Microchip’s MPLAB development and simulation software, available online, was configured with PicBasic Pro to compile programs within its environment. MPLAB provided a complete editor, compiler, and programmer package. Microchip’s PicStart Plus programmer was used to program the micro-controller eeproms.

Figure 2 depicts the block diagram of a 3 remote station system in which all but the transducer IC has been prototyped and is the subject of this thesis. Figure 1 shows that

increasing the number of stations can easily be done with the same arrangement. In this case increasing EEPROM memory would be required as well as modifications to the base station controller firmware. For industrial plants 16 stations is an appropriate number while a college campus, for example, may require many more. It should be noted that the EEPROM memory area of the base station is the limiting factor but can easily and affordably be increased.

For this project importance was placed on the justification based on need and cost. Chapter 2 spells out the project justification in detail. Specific design details are covered in Chapter 3 while Chapter 4 deals with the testing and implementation of the prototype. Finally, Chapter 5 concludes the text with an analysis of the results and any recommendations.

2 JUSTIFICATION OF WIRELESS POWER MONITORING SYSTEM

The justification of a wireless power metering system must meet the dual requirement of feasibility and practicality of application. To this end, it will be shown that such a system can be quantifiably justified by considering the practical limits and costs of a hard-wired demand metering system, the efficient use of electrical energy, and other benefits.

HARD-WIRED DEMAND METERING

Hard-wired communication systems are problematic in two key areas: installation costs and transmission line losses. As the number of metering points and the distance from the data logging computer increase these problems may circumvent the installation of the system.

In 1994 a hard-wired monitoring system was developed and implemented on the campus of ITESM located in Monterrey, Nuevo León, México. The hard-wired monitoring system, Figure 3, is comprised of many subsystems: current transformers, instrumentation and power transformers, transducer/transmitter, wire pairs, receiver, data acquisition card, presentation and results program.

The substation transformer of Figure 3 steps down 13.8 kV of line voltage to a distribution level of 220 V. The transformer is Δ -Y connected. The secondary side line currents vary from 400 Amps for a 140 kVA substation and up to 2600 Amps for a 1000 kVA substation. Current transformers provide a proportional current signal no greater than 5 Amps to instrumentation transformers. Instrumentation transformers are used to lower the signal levels of the CTs and PTs to mA and mV levels for the electronics, respectively. The transducer/transmitter calculates three phase real and reactive power and provides a proportional frequency square wave output to the receiver.

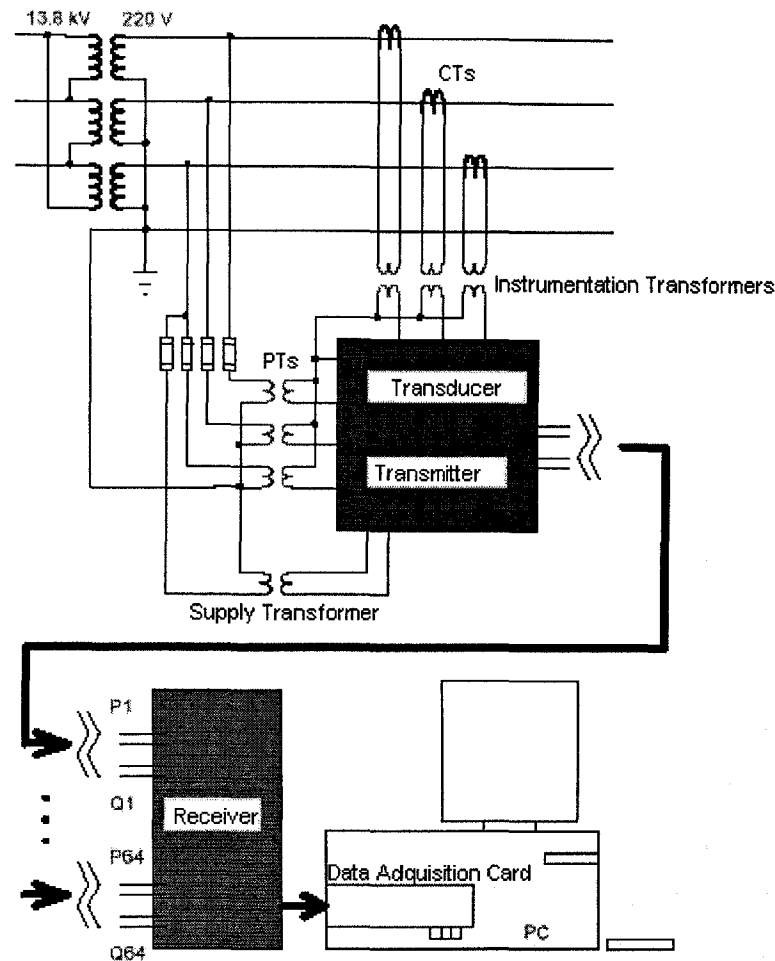


Figure 3. ITESM multiple point hard-wired power monitoring system.

Two two-wire pairs provide the physical link between the transducer / transmitter and the receiver. The receiver contains 8 multiplexors to time-division multiplex the square wave signals. Each multiplexor on the receiver can multiplex 8 signals. After multiplexing, the proportional frequency signals are demodulated to DC voltages which are fed to a DAS-1402 data acquisition card by Keithley MetraByte. A DAQ program developed in Visual Basic controls the system.

Where possible, local vendors in Monterrey were used in the design allowing the system to be attractive cost-wise. Although low-cost, this system is feature limited: it only

provides 3 phase real and reactive powers, and difficult point to point installation and cable damage could occur. [2]

INSTALLATION CONSIDERATIONS

In a given industrial plant a direct route may not exist between metering points and the data logging computer or system. The cost of making the physical circuit connection may exceed the proposed benefit of installing a power transducer. An alternative to costly installations exists in proposing a wireless power metering system that will link two points without the need of a physical circuit connection.

As an example of such a problem, the department of Electrical Engineering at ITESM Monterrey Campus experienced difficulties in connecting a hard-wired demand transducer in a student residence building located off campus. “Estudiantes XV” provides housing for both graduate and undergraduate students of ITESM and is considered an important load. Although a hard-wired demand metering system has been in place throughout the majority of the campus since 1994, “Estudiantes XV” was only metered until the summer of 1997. At that time it was decided to bridge the distance and make the physical circuit connection. An entire week elapsed, while workers attempted to gain access to underground conduits under a major avenue, before the physical link was made. Workers chiseled through concrete in an attempt to find any route to pass the four wires required to connect the transducer.

COMMUNICATION STANDARDS AND TRANSMISSION LIMITS

Designing a communications system is a complex task. Factors to consider include: data rate, data format, cable length, mode of transmission, termination, bus common mode range, connector type, and system configuration. Of special consideration are RC line losses and the type of communications standards to apply to the design.

RC Line Losses

Every physical medium has capacitance and resistance. Figure 4 illustrates the equivalent circuits for a transmission cable. The lumped parameter circuit is valid when the circuit length L and signal wave length λ relationship is such that $L \ll \lambda$. [3]

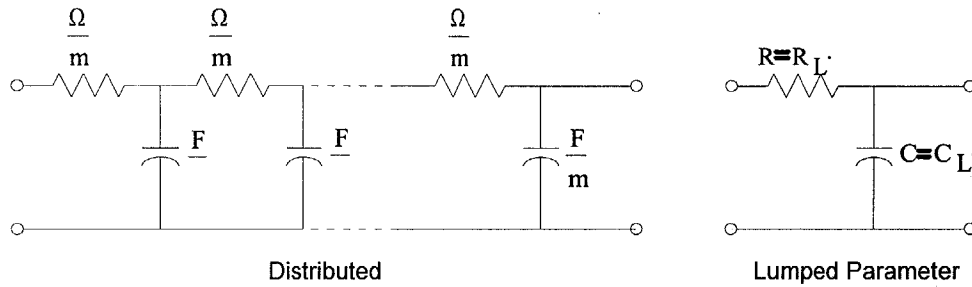


Figure 4. Transmission line equivalent circuits.

Figure 5 illustrates the transient response of an RC circuit both with switch SW closed and opened for a very long time then opened and closed, respectively. The capacitor voltage – time relationships for the circuits are as follows:

$$\text{Circuit 5.A: } V_c(t) = V_o \cdot e^{-\frac{t}{RC}}$$

$$\text{Circuit 5.B: } V_c(t) = V_o \cdot \left(1 - e^{-\frac{t}{RC}}\right)$$

Figure 6 illustrates an inherent delay of a RC circuit to a stepped response, as would be the case with a digital pulse. The rise time, often defined as the time between signal threshold levels of 10% and 90% of the input, is dictated by the R and C parameters of the circuit. A greater capacitance or resistance would increase the time constant RC and hence increase the rise time. An increase in rise time means a decrease in signal speed. Figure 6 illustrates a digital pulse train input V_{in} , sent down a transmission line, and the received output V_{out} .

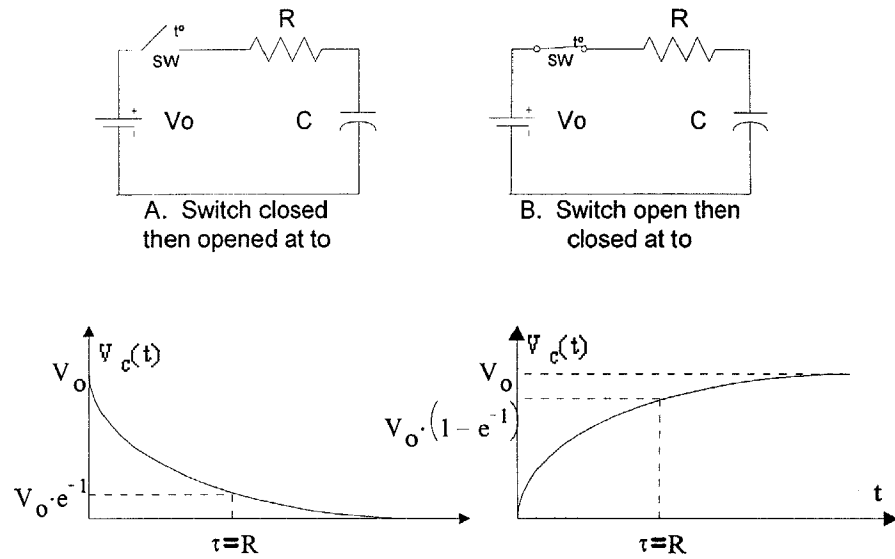


Figure 5. RC circuit step response.

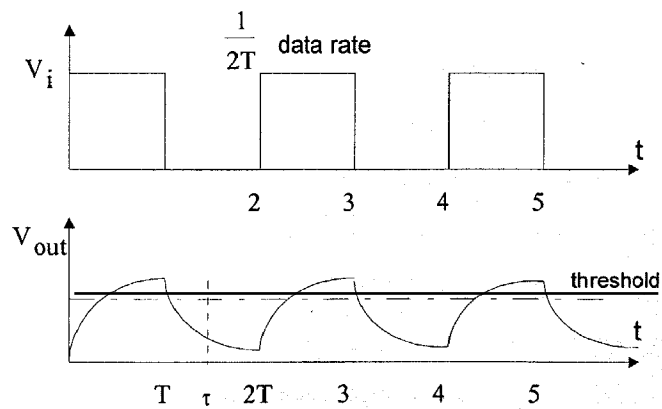


Figure 6. Pulse train on a transmission line.

Figure 6 illustrates that RC delays cause distortion when high data rates are used. The signal no longer has the semblance of a pulse train. The RC parameters of the circuit are

fixed therefore by increasing τ , the width of a pulse, the receiver is allowed sufficient time to recognize a strong signal. As an example, using an RS 232 cable with the following cable specifications:

Specification 1: 40-50 pF/ft, 50-ft length

Specification 2: Resistance not less than 3000 Ω if voltage not greater than 25 V

Resistance not greater than 7000 Ω if voltage between 3-25 V

By adjusting the width of the pulse to $\tau = 2RC$ (for $V_{out} = 0.866V_{in}$) the data rate becomes $DR = 1/2\tau = 1/4RC$ (bits/second). For this example using Recommended Standard (RS) 232:

$$DR = \frac{1}{4 \cdot 5000\Omega \cdot 50 \cdot ft \cdot 50 \cdot pF/ft} = 20,000 \frac{sample}{sec}$$

Clearly, the RC parameters of the circuit when increased limit the data rate.

Communication Standards Limits

There are two modes of operation for line drivers (generators) and receivers: unbalanced (single-ended) and balanced (differential). The advantage of unbalanced data transmission is when multiple channels are required; a common can be used. The disadvantage of unbalanced data transmission is in its inability to reliably send data in noisy environments. This is due to very limited noise margins. The sources of system noise can include externally induced noise, cross talk, and ground potential differences. Balanced data transmission requires two conductors per signal. In balanced data transmission the logical states are referenced by the difference of potential between the lines, not with respect to ground. Differential data transmission nullifies the effects of coupled noise and ground potential differences.

Data Terminal Equipment/Data Circuit terminating Equipment (DTE/DCE) interface standards are set by two industry trade associations: Electronic Industry Association (EIA) and the Telecommunications Industry Association (TIA). Although the EIA and the TIA have many communications standards two common standards EIA/TIA-232-E (formerly RS 232) and EIA/TIA-485 are compared in the following table. [4]

Table 2. EIA/TIA-232-E and EIA/TIA-485 standards.

Parameter	EIA/TIA-232-E	EIA/TIA-485
Interface type	Point to point	Multiple, bi-directional
Driver loaded Output Voltage	$\geq 5.0V $	$\geq 1.5V $
Driver Open Circuit Voltage	$\leq 25V $	$\leq 6.0V $
Max Data Rate	20 kbps (2500 pF load max)	10 Mbps @ 40 ft. 100 kbps @ 4000 ft.
Max no. transceivers	—	32

EIA/TIA-232-E is a poor choice for high data rate requirements. EIA/TIA-485 is a poor choice for applications where a high number of transceivers are required or distances are large. Both are poor choices in installations where a point to point physical link is not possible.

EFFICIENT USE OF ELECTRICAL ENERGY

Efficient use of electrical energy is possible from a monitoring system whether wireless or hard-wired. The efficient use of electrical energy is the act of using the metering data

from the monitoring system and based on the demand profiles make informed decisions about control or better use of the supplied power.

POWER FACTOR CAPACITOR LOCATION/RELOCATION

Power profiles of all substations throughout an industrial plant permit the correction of the power factor (PF) of the utility supply by clear knowledge of the exact amount of VARs needed. Power metering also allows the connecting of power factor correcting capacitors in a distributed manner making the entire system less susceptible to harmonic resonance in the presence of harmonics, reduces the effects of high transient turn on currents, and allows for the detection of unbalanced voltages. Power factor correction is advantageous for plants in the Monterrey area as this avoids costly power factor penalties from the Mexican Federal Electricity Commission. Improved power factor also reduces I^2R losses in the distribution system.

REDISTRIBUTION OF LOADS

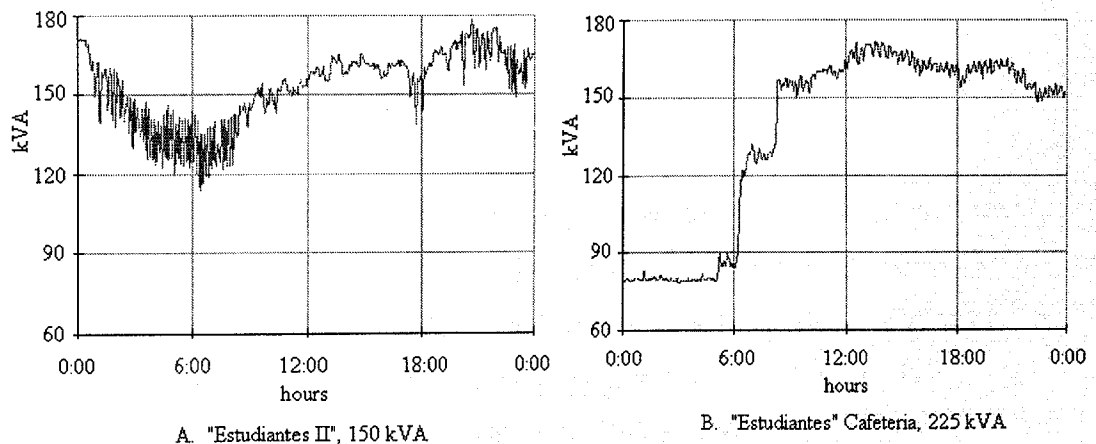


Figure 7. Load profile of two transformers, ITESM.

A power monitoring system also allows for the detection of abnormal or unexpected operation in the distribution transformers. As an example, the hard-wired monitoring

system developed and implemented at ITESM Monterrey Campus detected transformers that were operating above their nominal ratings and transformers that operated much below rated value, see Figure 7. The 150 kVA transformer of a student residence building, known as “Estudiantes II”, operated above its rated value while the 225 kVA transformer of cafeteria “Estudiantes” operated well below its rated value. Swapping these transformers stopped overheating in the 150 kVA transformer, heating which deteriorates winding insulation and reduces transformer life.

DETECTION OF ABNORMAL OPERATION

The monitoring system at ITESM also detected an abnormal condition. Two 750 kVA, 13.8 kV/220 V Δ -Y transformers connected in parallel, on both primary and secondary, source a two tower (North and South) building known as “CETEC”. Tap changes were needed on the transformers and hence were disconnected from the main supply. Upon re-energizing the transformers, workers unwittingly only connected one, the south tower transformer. The north tower transformer, energized through its secondary, appeared as a reactive load requiring only magnetization current. The load profile for that day indicated the problem.

ENERGY SAVINGS

Through a demand profile analysis, load factor can be easily determined during any period of time. Improvement in load factor can also be achieved by redistributing loads where possible and changing the time of day some loads operate. Reducing the load factor implies a decrease in the average demand billed thereby lowering energy consumption. Off-peak power is cheaper than during peak hours. Redistributing the operation of loads not needed during peak hours also reduces energy bills. With the objective of the efficient use of electrical energy, other uses of a monitoring system include consideration of the installation of high efficiency motors, installing variable frequency drives, and detection of motors that operate with little load. In general a

monitoring system is practical as it provides a means for the better use of electrical energy. [5]

OTHER BENEFITS

AUTOMATIC DEMAND CONTROL

Once fixed power factor correcting capacitors are in place the next step is the automatic connection of power factor correcting capacitors based on feedback information from the monitoring system. At the present time, ITESM has developed a single substation prototype automatically switched capacitor bank integrated with the current hard-wired transducer. A more attractive approach would be the switching of capacitors from the central logging station. This control could easily be programmed into a wireless monitoring system.

The justification of a wireless power monitoring system is complete. The advantages in terms of the efficient use of electrical energy and practicality have been spelled out. Next, a detailed description of the design of a prototype multiple station power monitoring system is offered and a look at a tested system consisting of a base station plus three remote stations.

3 SYSTEM DESIGN

The design of this digital wireless power monitoring system prototype can be separated into five separate modules: base laptop software control algorithm, network topology, communications protocol, base station hardware and embedded controller firmware, and remote station hardware and embedded controller firmware.

BASE LAPTOP

A Visual Basic program was written to provide the basic command and control functions of the monitoring system. Commands are sent via the Visual Basic user interface software to the base station embedded controller's hardware universal asynchronous receiver transmitter (USART). A Texas Instruments MAX232 is used to interface between the chip USART and the laptop's serial port.

Visual Basic Control and DAQ Program

A sample of the program's interface screen is included in the appendix for reference. Four large buttons indicate valid commands to be sent to the base controller. On power up or a power reset the embedded controller's program waits for a command from the user. Table 2 describes the functions of all buttons on the interface screen.

Table 3. Visual Basic interface screen command function descriptions.

BUTTON	DESCRIPTION
RESET SYSTEM	Required on all power ups to communicate with remote stations. Sends ASCII "A" to base controller USART. Will reset the base station and send "RESET" command to remote stations, logs the

	communications in EEPROM. 3 samples.
RECHECK UNITS	Sends ASCII "B" to base controller USART. Sends "RECHECK" command to remote stations and logs communications in EEPROM. 10 samples.
TDM DATA	Sends ASCII "B" to base controller USART. Sends "TDM Data" (Time Division Multiplex Data) command to remote stations and logs data in EEPROM. 288 samples
UPLOAD DATA	Sends ASCII "D", "E", or "F" to base controller depending on the number of samples selected in "Type of Upload (Samples)". Base controller then will upload that many samples to the laptop and save to file c:\data.txt
YES	Sends ASCII "y" to base controller USART.
NO	Sends ASCII "n" to base controller USART.
OK	Sends ASCII "g" to base controller USART.

The base station, via the LCD display module, at the appropriate times prompts the user for input. On a RESET SYSTEM, RECHECK UNITS, or TDM DATA command the base station, when it has completed the communication and data storing to EEPROM, will prompt the user if he desires to upload the results. At that time the user may choose to do so or come back to it later although initiating another command will write over the previous results in EEPROM. On any command execution, the base station, by default, begins storing in location 0 of the first EEPROM addressed 00.

If a data upload is initiated the file c:\data.txt will be created. This file contains, in tab delimited format, the number of samples of data as indicated by the user. Three, 10, or 288 samples of data may be uploaded to the laptop. Each sample consists of 120 bytes of data plus tabs and carriage returns. The following table illustrates the number of bytes uploaded for each case of 3, 10, or 288 samples. These totals include the data, tab delimiters, and carriage returns.

Table 4. Number of bytes uploaded to data.txt.

CASE	BYTES UPLOADED
(RESET) 3 samples	717
(RECHECK) 10 samples	2,390
(TDM DATA) 288 samples	68,832

Table 4 illustrates the format of one line in data.txt when loaded into Microsoft Excel as well as the number of bytes per field.

Table 5. Format of one line in data.txt.

Sample	station	cmd	energy	volt/f	sample	station	cmd	energy	volt/f	Sample	station	cmd	Energy	volt/f
2	1	1	18	18	2	1	1	18	18	2	1	1	18	18

The first line of the data file represents the first sample taken. Subsequent lines represent additional samples.

NETWORK TOPOLOGY

“The manner in which nodes are geometrically arranged and connected is known as the topology of the network.”[6] There exist, for LANs, two major classes of topologies: unconstrained and constrained. The first class, unconstrained topologies (also known as hybrid or mesh networks), as the name implies do not have a defined configuration. The nodes in the system are unconstrained, connected in an arbitrary manner. Routing problems exist in this class of topology. The second class, constrained topologies, has a predetermined interconnection scheme. The basic constrained topologies are known as bus, ring, and star configurations. Nodal connections for these networks are implied by the names.

Bus topology was chosen for this design.

In a bus topology all the network nodes are connected to a common transmission medium. As a result of this, only one pair of users on the network can communicate at the same time. Each network node thus has a unique address which is used when information is transmitted. When a data packet is sent out, it propagates throughout the medium and is received by all stations. To receive messages, each station continuously monitors the medium and copies those messages that are addressed to itself as the data packets go by.[6]

COMMUNICATIONS PROTOCOL

NETWORK CONTROL ACCESS SCHEME

A deterministic or non-contention access scheme is implemented in this design. Deterministic implies controlled access to the network. One station at a time can access the channel provided that they have received the digital signal authorizing them control. A token packet is used as the digital signal that allows each station access to the bus. Figure 1 illustrates the network topology for this design. Sixteen remote stations are shown although only 3 were implemented in the prototype. Although there exists a base station, the passing of this token throughout the bus is not vested in the base station, or

centralized, rather it is the responsibility of each station to place the token back on the bus once it has finished transmitting its message. This type of network control is known as distributed control.

The stations are ordered with identities and are offered access to the bus one at a time. Operation is similar to a token ring topology. In this case, the token contains specific destination addressing information. The result is the ordering of the stations resembles a logical ring as shown in Figure 1. Tokens may flow in only one direction making this a unidirectional token bus network.

RETRANSMISSION TECHNIQUE

Go-Back-N Automatic-Repeat-Request (ARQ) , also known as continuous ARQ, was chosen as the retransmission technique.

In this scheme a series of data frames is sent continuously without waiting for an acknowledgement. This will improve the throughput of the link, especially if the propagation delay is not negligible compared to the frame transmission time.

When the receiving station detects an error in a frame, it sends a NAK to the transmitter for that frame. All further incoming frames at the receiver are then discarded until the frame in error is correctly received. Thus, when the transmitting station receives a NAK, it must retransmit the frame in question plus all succeeding frames. Hence the name go-back-N, since the last N previously transmitted frames must be resent when an error occurs. [6]

The above describes the communication between two stations, A and B, as shown in Figure 8. In this case one station transmits N frames for a total of N frames. Analogously the power monitoring system design consists of N stations transmitting one frame for a total of N frames.

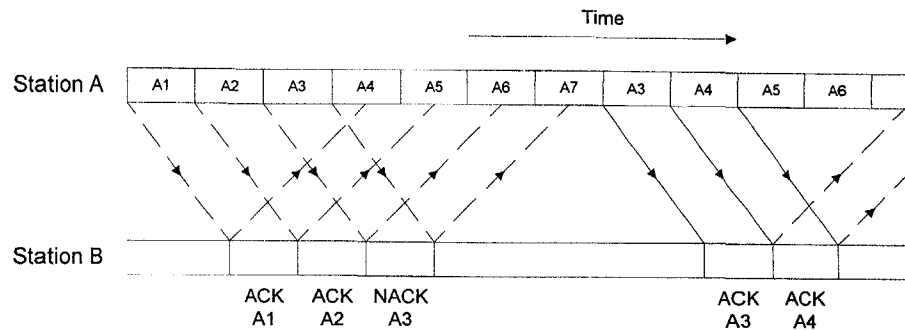


Figure 8. Generalized go-back-N ARQ. Frame 3 was received in error.

Figure 9 illustrates the go-back-N technique as applied to the design of the wireless power monitoring system. There are two important differences to note between Figure 8 and Figure 9. First instead of a single station transmitting N frames, in Figure 9 there are N stations transmitting a single frame. Also the base station, while internally acknowledging the reception of valid frames, does not transmit an ACK. It will only transmit a NACK frame if required.

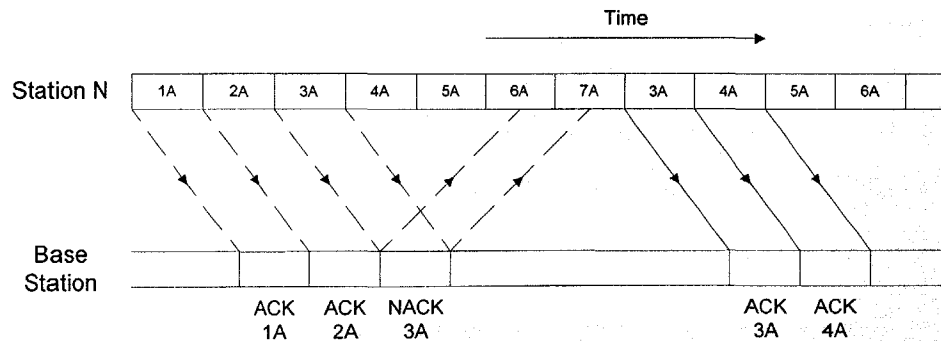


Figure 9. Implemented go-back-N ARQ. Station 3 frame A received in error.

FRAME STRUCTURE

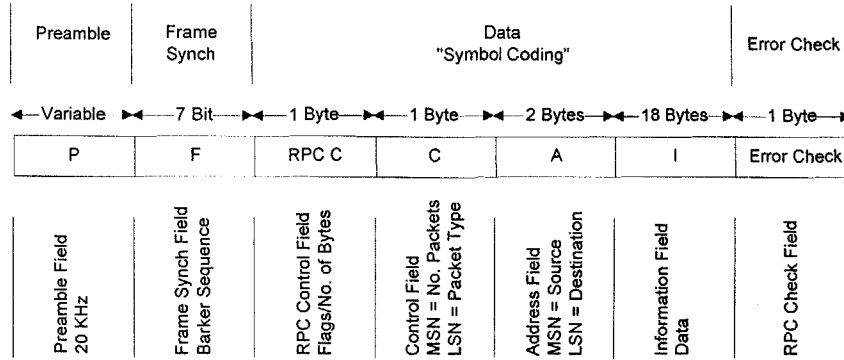
With the network access control scheme and retransmission technique defined, the next step is to define the structure of the frames being transmitted through the token bus network.

Figure 10 depicts the frame structure used in this design. A data frame in general is made up of four parts: preamble, frame synch, data, and error checking. The preamble is a 3.2 ms 20 kHz pulse train that allows other units to identify the incoming data frame. The frame synch is a special 7-bit barker sequence used to synchronize the data slicer of the receiver to the incoming data. The data portion of the frame is the packet we wish to transmit. Finally, the last part of the data frame is a checksum byte that checks for overall packet integrity.

Four fields define the data portion of the frame structure as follows. First is the RPC_C (RPC Control field). This is a one-byte control field whose purpose is to tell the RPC chip how many total bytes are in the data packet. More specifically, the lower 5 bits of the byte provide a 5-bit byte count. The upper three bits are used for preamble control and to designate the byte as a data byte or a memory byte. A memory byte is used to change the configuration of the RPC chip. Second, is the C (Control Field). The upper 4 bits of C indicate the number of packets in the message while the lower 4 bits of C indicate the type of data packet. Third, is the A (Address Field). This is a two byte wide field with the MSB indicating the source station address and the MSB indicating the destination station address. Lastly, is the 18 byte wide I (Information Field). For this power monitoring system there are 6 different packet types that may be placed on the bus. Figure 11 illustrates the different types of packets as defined by the different types of contents in the Information Field.

(Type 0) Command Packet. A command packet (type 0) is transmitted with a one-byte wide flag in the information field. The MSNibble of the flag represents the actual command (\$A for RESET, \$B for RECHECK, and \$C for TDM DATA). The LSNibble of the flag represents the number of times that command has been retransmitted to a particular station. The base station is the only station that may transmit a command packet.

FRAME STRUCTURE



FIELD STRUCTURES

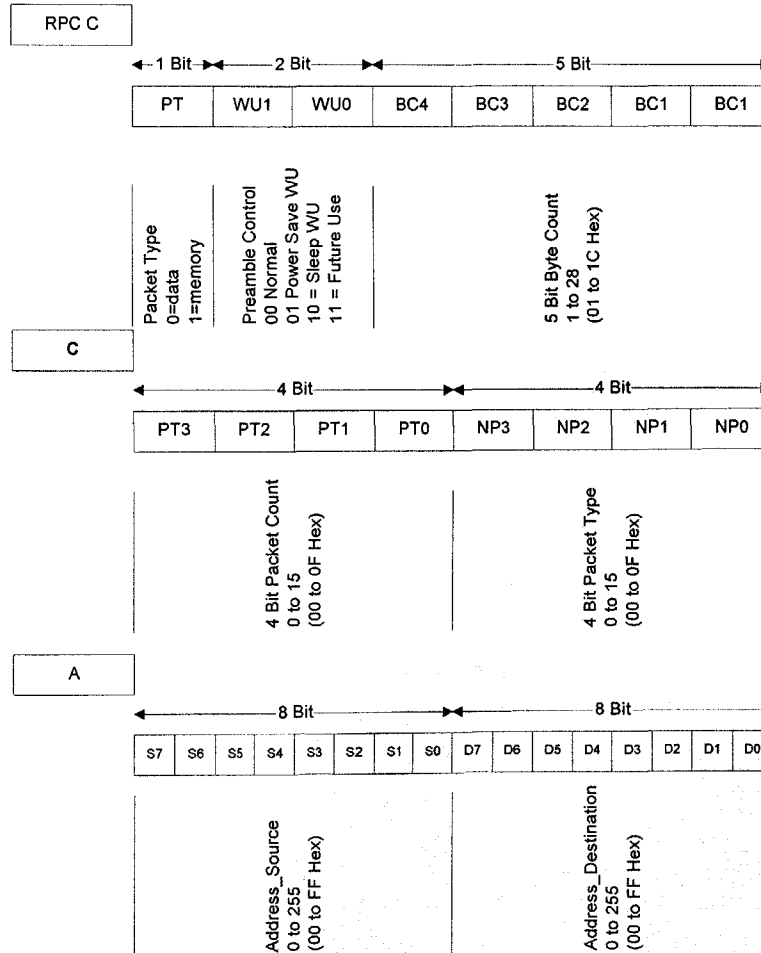


Figure 10. Frame structure.

INFORMATION FIELD STRUCTURE

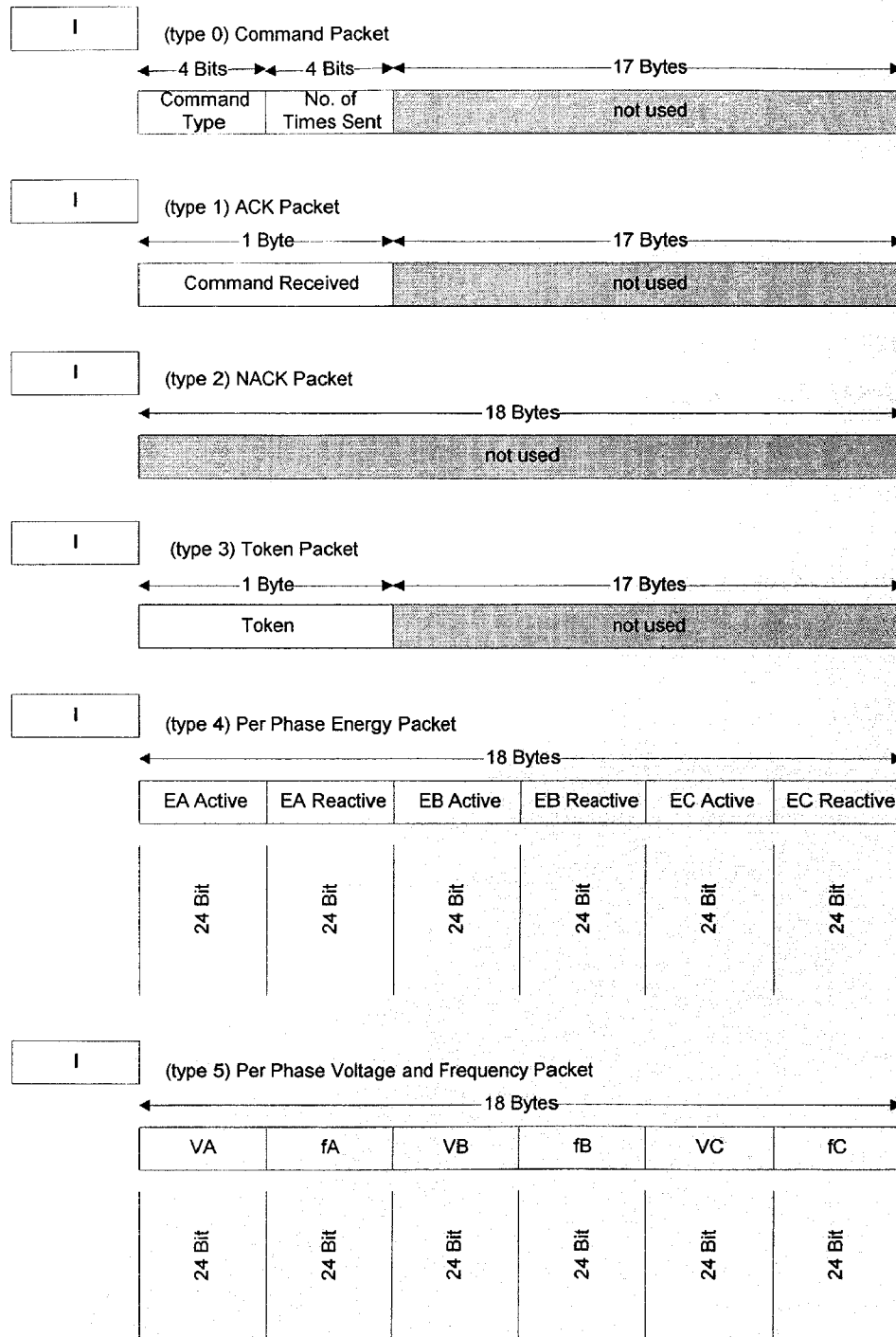


Figure 11. Information field structure, packet type definitions.

(Type 1) ACK Packet. An ACK packet (type 1) is transmitted with a one-byte wide acked command in the information field. This is the command that the remote station received from the base station and therefore acknowledges having received and acted on.

(Type 2) NACK Packet. A NACK packet (type 2) is transmitted with no data bytes in the Information Field. The base station is the only station authorized to send NACKs where the address destination portion of A (Address Field) is the important factor.

(Type 3) Token Packet. A Token packet (type 3) is transmitted with a one-byte token, \$AA, in the information field. Remote stations and the base station may transmit token packets. The address destination portion of A (Address Field) dictates the token destination.

(Type 4) Per-Phase Energy Packet. A per-phase energy packet (type 4) is transmitted with 18 bytes of energy data per phase. Remote stations are the only ones who transmit energy packets. All energies are 3 bytes wide.

(Type 5) Per-Phase Voltage and Frequency Packet. A per-phase voltage and frequency packet (type 5) is transmitted with 18 bytes of voltage/frequency data per phase. Remote stations are the only ones who transmit voltage/frequency packets. All voltages and frequencies are 3 bytes wide.

Figure 12 illustrates the packets placed on the bus and by which station for a RESET (\$Ax), RECHECK (\$Bx), and TDM DATA (\$Cx) command. All data bytes within the packets are given in hexadecimal.

(byte)		0	1	2	3	4	to	21
		RPC_C	C	A		I		
base	command	05	20	00	01	A1	not used	
	token	05	23	00	01	AA	not used	
station1	ack1	05	41	01	00	A1	not used	
	data1	16	44	01	00	44	---	44
	data1	16	45	01	00	45	---	45
	token1	05	43	01	02	AA	not used	
station2	ack2	05	41	02	00	A1	not used	
	data2	16	44	02	00	44	---	44
	data2	16	45	02	00	45	---	45
	token2	05	43	02	03	AA	not used	
station3	ack3	05	41	03	00	A1	not used	
	data3	16	44	03	00	44	---	44
	data3	16	45	03	00	45	---	45
	token3	05	43	03	04	AA	not used	
base	ack0	05	11	00	FF	A1	not used	

base	* nack0	5	22	00	01	\$45
	token1	5	23	0	1	AA

* base sends nack if error occurred plus token to station that was acked.

(byte)		0	1	2	3	4	to	21
		RPC_C	C	A		I		
base	command	05	20	00	01	B1	not used	
	token	05	23	00	01	AA	not used	
station1	ack1	05	41	01	00	B1	not used	
	data1	16	44	01	00	44	---	44
	data1	16	45	01	00	45	---	45
	token1	05	43	01	02	AA	not used	
station2	ack2	05	41	02	00	B1	not used	
	data2	16	44	02	00	44	---	44
	data2	16	45	02	00	45	---	45
	token2	05	43	02	03	AA	not used	
station3	ack3	05	41	03	00	B1	not used	
	data3	16	44	03	00	44	---	44
	data3	16	45	03	00	45	---	45
	token3	05	43	03	04	AA	not used	
base	ack0	05	11	00	FF	B1	not used	

base	* nack0	5	22	00	01	\$45
	token1	5	23	0	1	AA

* base sends nack if error occurred plus token to station that was acked.

(byte)		0	1	2	3	4	to	21
		RPC_C	C	A		I		
base	command	05	20	00	01	C1	not used	
	token	05	23	00	01	AA	not used	
station1	ack1	05	41	01	00	C1	not used	
	data1	16	44	01	00	energy data		
	data1	16	45	01	00	voltage/ frequency data		
	token1	05	43	01	02	AA	not used	
station2	ack2	05	41	02	00	C1	not used	
	data2	16	44	02	00	energy data		
	data2	16	45	02	00	voltage/ frequency data		
	token2	05	43	02	03	AA	not used	
station3	ack3	05	41	03	00	C1	not used	
	data3	16	44	03	00	energy data		
	data3	16	45	03	00	voltage/ frequency data		
	token3	05	43	03	04	AA	not used	
base	ack0	05	11	00	FF	C1	not used	

base	* nack0	5	22	00	01	\$45
	token1	5	23	0	1	AA

* base sends nack if error occurred plus token to station that was acked.

Figure 12. Frame sequence on RESET, RECHECK, and TDM DATA.

BASE STATION

As mentioned earlier, the base station controller receives commands from the laptop through a serial interface. Once a command is received the base controller may only be interrupted, prior to completing with its task, on a manual reset. In order to describe the base station in more detail a closer look is needed at the hardware and software of the controller.

HARDWARE

The hardware of the base station consists of one Microchip PIC16C77/JW embedded controller, a Radiometrix 433.92 MHz TX2/RX2 transmitter/receiver pair, one Radiometrix RPC 000-DIL, one Optrex DMC 20171 liquid crystal display, one Texas

Instruments MAX232 EIA/TIA-232-E transceiver, and two ATMEL AT24C256 32 Kbyte EEPROMs.

TI MAX232

The laptop communicates serially to the USART of the controller. A Texas Instruments MAX232 EIA/TIA-232-E transceiver is required to convert TTL level signals to EIA/TIA-232-E and vice versa. Three connections are required to the serial port: TX, RX, and Vss. These are connected to pins 2, 3, and 5 respectively of a DB-9 connector. The laptop serial port is then connected to the DB-9 connector. The base controller's hardware USART is located on PORTC.6 (TX) and PORTC.7 (RX). These pins are connected to the TI MAX232 pin 10 (T2IN) and pin 9 (R2OUT). The baud rate is set within the base station controller's software (HSER_BAUD) and within the Visual Basic interface program's Microsoft Comm Control module. Both baud rates are set to 9600. This yields relatively fast uploads while reducing errors.

OPTREX LCD MODULE

An Optrex DMC 20171 LCD module, in 4 bit mode, is used to show the current status of the base station as well as to notify the user of needed input. The LCD module is the basis for the user interface. PORTA.0 to PORTA.4 and PORTE.0 are used to interface with the LCD module. The Optrex LCD module was chosen as it uses a Hitachi HD44780 driver which is compatible with the PBASIC compiler used to program the Microchip PIC16C77/JW controller. For example, in order to display "hello" using a PBASIC program and a compatible LCD display the instruction would be LCDOUT "hello".

ATMEL EEPROMS

Two I²C compatible ATMEL AT24C256 serial EEPROMs are connected on a common bus to the controller for data storage. PORTC.4 (DPIN) and PORTC.5 (CPIN) are

connected to the SDA and SCL lines of the EEPROM common bus. A control byte in the PBASIC instruction to read and write, I2CREAD and I2CWRITE, determines the EEPROM to write to. The address of the first EEPROM is set to 0:0 (A1: A0) and the second to 0:1 (A1: A0). This being the case, a maximum of 4 serial EEPROMs could be addressed on the same two-wire bus from the controller. The following table illustrates the connections to the ATMEL serial EEPROMs.

Table 6. Serial EEPROM connection summary.

Function	Pin Name	EEPROM #1	EEPROM #2
Address Input 0	A0	Vss	+5 V
Address Input 1	A1	Vss	+5 V
Serial Data	SDA	PORTC.4 (DPIN)	PORTC.4 (DPIN)
Serial Clock Input	SCL	PORTC.5 (CPIN)	PORTC.5 (CPIN)
Write Protect	WP	Vss	Vss
No Connect	NC	-	-

Since the bus is shared two 4.7k pull-up resistors are connected to the SDA and SCL lines. The lines are also buffered to the controller using 1k resistances.

RADIOMETRIX RADIO PACKET FORMATTER IC

A Radiometrix RPC-000-DIL packet formatter IC (RPC chip) is used to receive and transmit data packets. "A data packet of 1 to 27 bytes downloaded by the Host micro-controller into the RPC's packet buffer is transmitted by the RPC's transceiver and will

'appear' in the receive buffer of all the RPC's within radio range. A data packet received by the RPC's transceiver is decoded, stored in a packet buffer and the Host microcontroller signaled that a valid packet is waiting to be uploaded." [7] The host controller may transfer up to 28 bytes to the RPC chip for transmission. The first byte of the 28 byte packet maximum is a control byte. The control byte provides control information in the form of 3 flag bits and a 5 bit byte count which indicates the number of bytes, itself included, that are to be transferred to the RPC chip by the controller for transmission. Thus the data portion of the packet contains 1 control byte and a maximum of 27 user data bytes. The data rate is given by the following formula for a 10 MHz oscillator:

$$DR = \frac{f_{oscillator}}{256} \text{ bit / sec} = 39.0625 \text{ kbit / sec}$$

Once the data portion of the packet is transferred to the RPC chip the actual packet that is sent to the transmitter is made up of four parts.

Preamble. A 20 kHz square wave has the function of allowing the data slicer in a remote receiver to establish the correct slicing point, then once the receiver settles, the remaining portion is used to positively identify and phase lock on to the incoming signal. The preamble time is set at the default 3.2 ms.

Frame sync. Synchronization of the data is achieved with a Barker sequence of 7 bits. The synch sequence allows the receiving RPC chips to positively identify incoming data.

Data. Upon receipt of a data packet to transmit the RPC chip codes each byte into a 12 bit symbol. Provides 50:50 bit balance, i.e. 6 ones and 6 zeros. The RPC chip allows no more than 4 consecutive ones or zeros minimizing the low frequency components in the code. Provides a minimum Hamming distance of 2 meaning each code generated is different from any other code by a minimum of 2 bits. Only 256 of 4096 possible codes are valid. This is 6.25%, which implies a 93.75% probability of trapping a byte error.

Check Sum. The receiver checks each symbol for integrity as such an 8-bit checksum is used to test for overall packet integrity. The check sum is also coded into a 12-bit symbol.

On the receiving side, decoding of a transmitted packet is also done in 4 stages.

Search. The RPC chip decoder searches the radio noise on the RXD line of the receiver for the preamble of 20 kHz. A 16 times over-sampling detector computes the spectral level of 20 kHz in 240 samples of the RXD signal. If the level exceeds a pre-set threshold the RPC chip will attempt to decode a packet.

Lock-in. The 240 samples from the over-sampling detector are used to compute the phase of the incoming preamble and synchronize the RPC chip's internal recovery clock to an accuracy of $\pm 2 \mu\text{s}$. The incoming data is sampled at the bit midpoint by the recovery clock and serially shifts the data through an 8 bit comparator. The comparator searches the sampled data for a frame synch byte and will abort this process if the packet fails to maintain a certain level of integrity. When the frame synch byte is detected the RPC chip will attempt to decode the data packet.

Decode. Twelve bit symbols are now read, decoded, and placed in the receive buffer. If an invalid symbol is received the RPC chip will abort the current process.

Check Sum. This last byte is decoded and verified against a sum of all received bytes by the receiving RPC chip. If this matches then the data packet is valid and !RXR is pulled low to inform the host controller that a valid packet has been received and awaits in the receive buffer. !RXR in this case is connected to PORTB.0 of the PIC16C77/JW controller and configured as an external interrupt.

The RPC chip is connected in parallel with the PIC16C77 as depicted in the appendix base station and remote station schematics. PORTC.0 is used as the reset line, active

low, from the controller to the RPC chip. PORTB is used to pass and received data to and from the RPC chip.

Table 6 details the pin functions of the RPC chip.

Table 6. Pin functions of RPC-000-DIL.

Name	Pin	Pin Function	I/O	Description
TXR	6	TX Request	I/P	Data transfer request from HOST to RPC
TXA	7	TX Accept	O/P	Data accept handshake back to host
RXR	8	RX Request	O/P	Data transfer request from RPC to HOST
RXA	9	RX Accept	I/P	Data accept handshake back to RPC
D0	2	Data 0	Bi-dir	4 bit bi-directional data bus. Tri-state between packet transfers. Driven on receipt for Accept signal until packet transfer is complete.
D1	3	Data 1	Bi-dir	
D2	4	Data 2	Bi-dir	
D3	5	Data 3	Bi-dir	

Figure 13 and 14 illustrate the handshaking signals and timing involved in byte transfers to and from the RPC chip respectively. To transfer a byte to the RPC chip from the controller PORTB.2 (!TXR) is brought low by the PIC16C77/JW controller. The RPC chip will accept this request by setting !TXA (PORTB.3) low. Now the controller proceeds to place the least significant nibble of the data byte on the bi-directional bus (D0–D3) and signals the readiness of this data to the RPC chip by asserting PORTB.2 (!TXR) high. Once the four bits have been accepted the RPC chip will assert !TXA

(PORTB.3) high. The cycle is repeated for the most significant nibble as well as any subsequent data bytes.

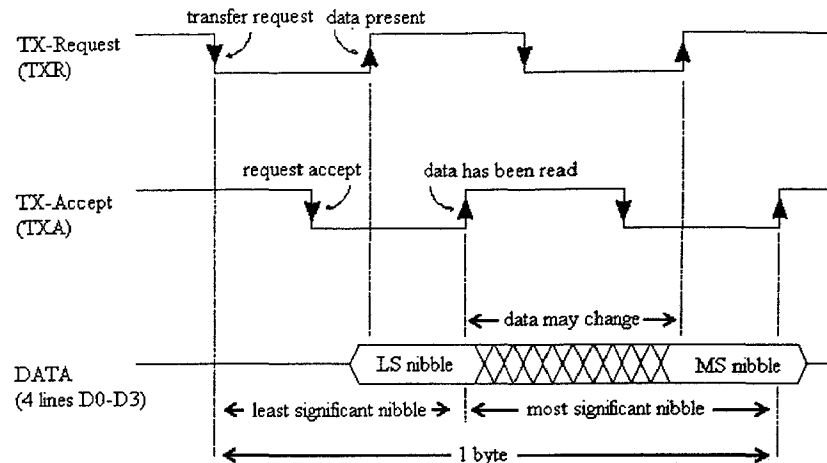


Figure 13. Controller to RPC chip byte transfer timing diagram.

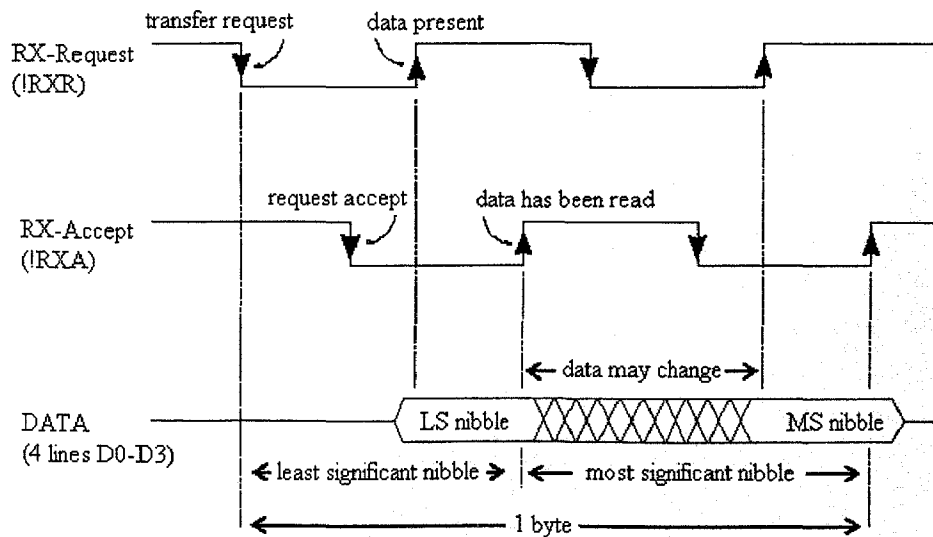


Figure 14. RPC chip to Controller byte transfer timing diagram.

Figure 14 shows that receiving a data byte from the RPC chip is similar. Two main differences are that the RPC chip is now driving the data bus and the hand shaking lines

are PORTB.0 (!RXR) and PORTB.1 (!RXA), receive request and receive accept respectively. The base station controller's PORTB.0 is interrupt enabled on the falling edge of the signal. When a valid data packet is ready in the RPC chip packet buffer, the chip will assert !RXR (PORTB.0) low and interrupt the controller. The controller must then service this interrupt and upload the packet waiting in the RPC chip packet buffer.

If the RPC chip receives a valid packet it will no longer enable the receiver, hence it will not be able to receive any data packets until the controller has uploaded the awaiting packet.

RADIOMETRIX TX2-F-5 433.92 MHz UHF TRANSMITTER

A 433.92 MHz UHF transmitter is used to transmit the data packets. Figure 15 illustrates the block diagram of the TX2 transmitter.

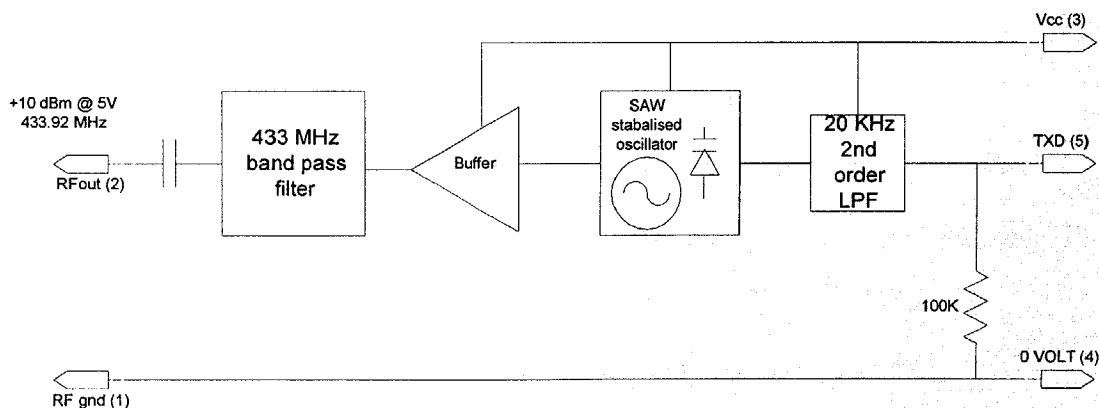


Figure 15. TX2 transmitter block diagram.

An input 20 KHz 2nd order low-pass filter is used to smooth the bit stream and eliminate high frequency components. The output band-pass filter is used to eliminate unwanted frequency spread upon transmitting. The oscillator is a surface acoustic wave (SAW) device. Interdigital transducers (IDTs) are used to excite SAWs. IDTs are electrode stripes of alternating polarity whose stripe widths and the gaps between stripes are one-

quarter of an acoustic wavelength at resonance. The resonance frequency of SAWs is determined by the electrode finger spacing. [8]

The transmitter has an open range of 300 m, +10 dBm, and can transmit up to 40 kbps.

RADIOMETRIX RX2 433.92 MHz UHF RECEIVER

A 433.92 MHz UHF receiver is used to receive the data packets. Figure 16 illustrates the block diagram of the RX2 receiver.

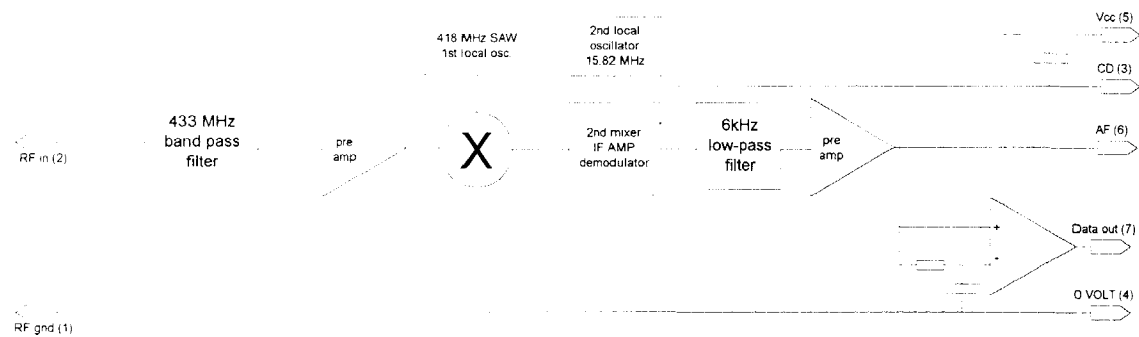


Figure 16. RX2 receiver block diagram.

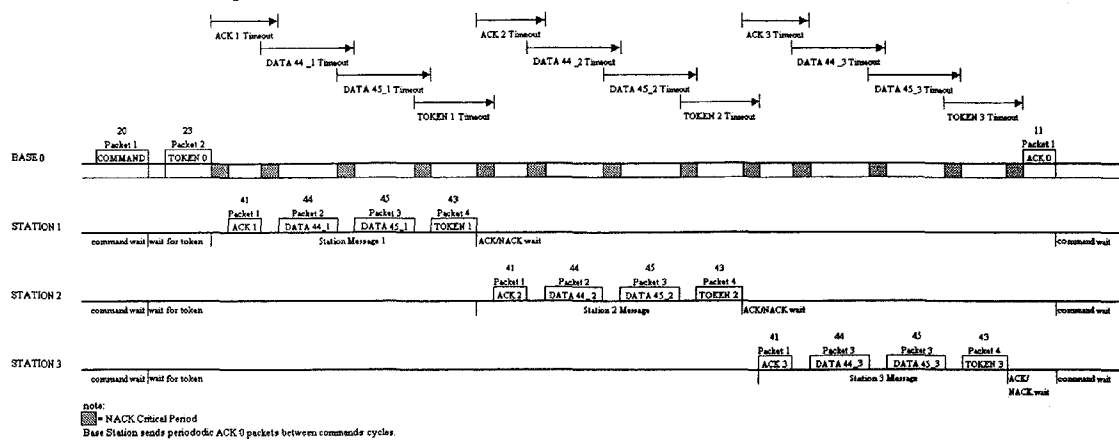
The receiver is a double conversion FM superhet. It contains a SAW front-end filter with image rejection of 50 dBm. The data rate of the receiver is up to 40 kbps.

Both the transmitter and receiver require a 15.5 cm rod whip antenna. The total distance from pin 2 of the transmitter/receiver and the antenna should not be greater than 15.5 cm.

EMBEDDED CONTROLLER SOFTWARE

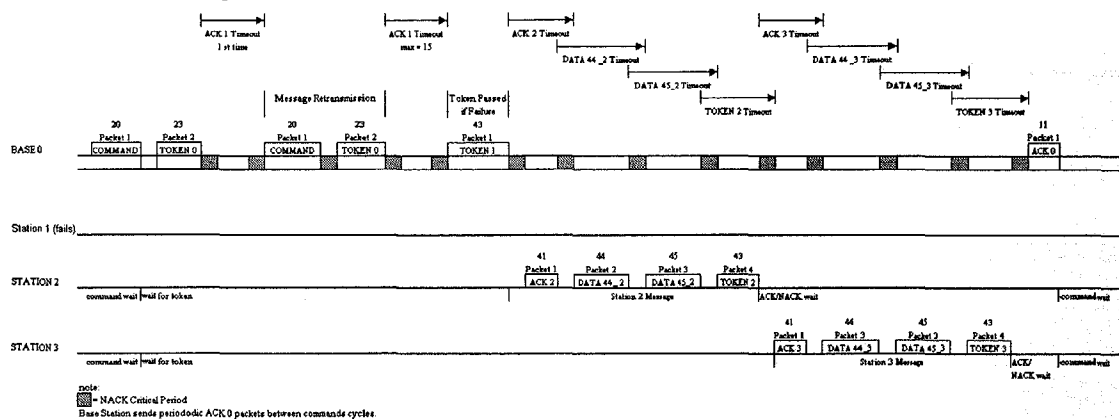
The general purpose of the base station controller firmware is to be able to receive commands via the laptop serial port, communicate with the remote stations given the protocol, store data to EEPROM, upload data to the laptop, and back again.

COMMAND PACKET = RESET, RECHECK or TDM_DATA



Station Failure

COMMAND PACKET = RESET, RECHECK or TDM_DATA



NAKED STATION

COMMAND PACKET = RESET, RECHECK or TDM_DATA

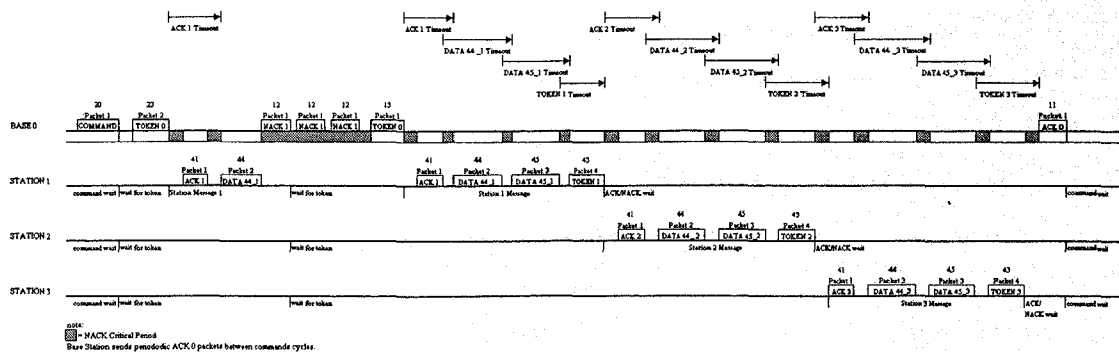


Figure 17. Communications timing diagrams.

Figure 17 illustrates how the base station and the remote stations implement the communications protocol.

First case is a sample taken with no communications problems, second a station fails, and third the go-back-N algorithm sends a NACK frame.

First, if the base controller received a command of either ASCII “A”, “B” or “C” from the laptop’s user interface software this would mean it would need to communicate with the remote stations. The base will send a two-packet message, command plus token, then wait for all stations to transmit a four-packet message each. The base station then transmits an acknowledgement (ACK) packet upon receiving the last station’s last packet.

Depending on the command sent to the stations the packet contents within their messages will vary. In summary, under trouble free communication, the base station will do the following:

- | |
|---|
| First. Transmit a command packet |
| Second. Pass the token to station 1 |
| Third. Record all data type packets in RAM for each station |
| Fourth Transmit an ACK to all and record the saved data packets to EEPROM |

This will conclude one sample interval. The ACK from the base stations signifies that the base has received all packets from all stations and recorded any data type packets sent by the remote stations.

Second, a retransmission scheme, in the event a station does not respond to a token passed or fails to receive the base station command, is needed. In the diagram the base station transmits its two-packet message, command plus token. Station one never

received the command packet therefore the base station, after transmitting the token, enables both a PORTB.0 (!RXXR) and TIMER0 interrupt. The timer is programmed to allow sufficient time for station one to transmit its ACK packet. Should the station transmit an ACK packet and the base station RPC chip receive it, an interrupt will be generated of type PORTB.0, the interrupts disabled, the packet uploaded, and the TIMER0 and PORTB.0 interrupts enabled again. If the TIMER0 interrupt is generated then the base station assumes the station did not receive the command or did not respond to the token passed and will therefore retransmit the original message. A maximum of 15 retransmissions is allowed before a station is logged as having failed.

Third, in the event the central begins to receive packets and/or messages from stations out of sequence then a Go-Back-N algorithm is implemented. In the diagram the inter-packet times are considered the not-acknowledged (NACK) critical periods. During these time periods the base station may transmit a NACK. This NACK is addressed to the remote station the base was expecting. Therefore all stations that follow the nacked station in sequence must retransmit their messages. In order to jam the channel, the base station sends the same NACK packet three times consecutively to increase the probability that all stations will received the NACK appropriately. The stations then, upon receiving a NACK, await their token.

REMOTE STATIONS

Three remote stations were prototyped and tested along with the base station previously described. Although a SAMES SA9604A is included in the design as the target transducer IC, it was not included in the prototype.

HARDWARE

Each remote station includes a simpler design than the base station. Like the base station, a Microchip PIC16C77/JW is the embedded controller. Also included is one Optrex

DMC-2017 LCD module, one Radiometrix RPC-000-DIL, and one Radiometrix 433.92 MHz TX2/RX2 transmitter/receiver pair.

A SAMES 9604A Three Phase Bi-directional Power/Energy Metering IC with serial SPI interface is included in the design. The SAMES 9604A IC has the following features illustrated in Table 7.

Table 7. SAMES 9604A features.

FEATURES
Bi-directional active and reactive power/energy measurement
Voltage and frequency measurement
Individual phase information accessible
SPI communication bus
Protected against ESD
Operates over a wide temperature range

The SAMES IC is connected to the SPI bus on the station controller. The communication between the transducer chip and the station controller is synchronous and would require the PicBasic Pro instruction SHIFTIN and SHIFTOUT. To access data in the SAMES 9604A registers the address of the register must be clocked out of the DO pin on the controller. After this, beginning the next clock cycle on DI of the controller will appear the contents of that register in 24-bit quantification. The registers of the 9604A are accessible when CS is high. Each phase of the power system corresponds to four

registers in the 9604A. Table 8 depicts the contents of each of the registers and their corresponding addresses.

Two address locations A4 and A5 are included for compatibility with future SAMES products. With CS high, data on the 9604A data in pin (DI) is clocked into the device on the rising edge of the clock (SCK). The registers must be enabled to be read. This is accomplished by sending 6_{HEX} (110) preceding the six-bit register address. The data clocked into DI will comprise of 110 A5 A4 A3 A2 A1 A0, in this order. Each register is 3 bytes wide (24-bit).

Table 8. SAMES 9604A register contents and addresses.

ID	REGISTER		A5	A4	A3	A2	A1	A0
1	Active	Phase 1	X	X	0	0	0	0
2	Reactive	Phase 1	X	X	0	0	0	1
3	Voltage	Phase 1	X	X	0	0	1	0
4	Frequency	Phase 1	X	X	0	0	1	1
5	Active	Phase 2	X	X	0	1	0	0
6	Reactive	Phase 2	X	X	0	1	0	1
7	Voltage	Phase 2	X	X	0	1	1	0
8	Frequency	Phase 2	X	X	0	1	1	1
9	Active	Phase 3	X	X	1	0	0	0
10	Reactive	Phase 3	X	X	1	0	0	1

11	Voltage	Phase 3	X	X	1	0	1	0
12	Frequency	Phase 3	X	X	1	0	1	1

Since the SAMES 9604A updates the registers on a continual basis they can only be accessed after the measurement cycle is complete. Pin 7 (FM150) provides a 280nsec pulse on voltage zero crossings. Each measurement cycle is 8 line frequency periods long.

Active and reactive energies are given in terms of a counter. The energy per count may be determined by the following formulas:

$$\text{Energy per count} = \frac{V \cdot I}{K} \text{ Watt} \cdot \text{sec}$$

Where V is the rated voltage, I the maximum current, and K is a factor defined as follows:

$$\begin{array}{ll} 640,000 & \text{for Active Energy} \\ \frac{640,000 \cdot 2}{\pi} & \text{for Reactive Energy} \end{array}$$

The measured voltage is given by:

$$V_{\text{measured}} = \frac{V \cdot \Delta n}{940,000 \cdot \Delta t}$$

Where V is the rated voltage, Δn is the difference in register values between successive reads, and Δt is the time difference between successive reads. [9]

The hardware of the remote stations functions very much the same as that of the base station. The difference is in the nature of the firmware of the remote stations.

EMBEDDED CONTROLLER SOFTWARE

The address of each of the remote stations is assigned upon programming. Upon initializing on power up the remote stations await a packet of command type (type 0) from the base station. The remote stations do not use an interrupt driven wait on PORTB.0 as the base station does. Rather they poll PORTB.0 and upload packets when PORTB.0 is asserted low (meaning a packet has arrived and awaits uploading). The base station message is two packets long. The first packet's "Control Field" is \$20 (2 packets in message, packet type 0) and the second packet in the message has a "Control Field" of \$23 (2 packets in message, packet type 3). Each station will receive the command packet, \$20, and execute the command in the packet "Information Field". The command sent to the stations is one-byte wide. As mentioned previously, the command byte's MSnibble contains the actual command of either ASCII "A" (RESET), "B" (RECHECK), or "C" (Acquire data). The LSnibble indicates the number of times the stations message had to be retransmitted in the event that the station did not receive the command on the first broadcast. Upon receiving the command packet each station saves the received command, executes that command, then awaits a token addressed to them. At this point the remote stations will ignore all incoming packets until receiving their token. Upon reception of the token the remote station will back off before transmitting it's message. In all cases a station's transmitted message consists of four packets, ACK, two DATA packets, and TOKEN. ACK, "Control Field" \$41 (four packets in message, packet type 1), packets contain the copied command that the station received. This command is logged at the central and is an indicator of the reliability of the communication between base station and remote station. Two DATA packets are required for transmitting all data pulled from the SAMES 9604A IC. The first DATA packet contains the energy readings at that substation. This energy packet is assigned a

“Control Field” of \$44 signifying 4 packets in the message with this particular packet being energy readings, type 4. The second DATA packet contains voltage and frequency information for the substation and is assigned a “Control Field” of \$45 with 5 indicating that it is a packet carrying per-phase voltage and frequencies. Finally, the fourth packet in the message is a token signifying that transmission of the message is complete. The token is addressed to the following station. After transmitting a token, the remote station awaits an “all clear” ACK message or a NACK message from the base station. If a NACK is received, “Control Field” \$11, then the remote station will test to see if the NACKed station is equal to or precedes it and if so will immediately await a token packet addressed to it. It is in this way that if messages need to be retransmitted they can be, the data to be placed in the “Information Field” of the data packets will always be saved in RAM buffers locally by the remote stations. If the remote station does not need to retransmit its message then it simply keeps waiting for either an ACK or NACK message from the base station.

As previously mentioned, remote stations may receive up to 3 different commands from the base station: RESET, RECHECK, and TDM DATA. In each case, after receiving a token, each station will transmit a four-packet message back to the base station. The difference lies in the contents of the data packets’ “Information Field” for each command. The contents of the messages for each command are considered next:

RESET. Upon receiving a reset command, \$Ax, the station will copy the received command to the variable *temp*, and fill the received and transmit packet buffers with \$FF. The 18-byte data buffer for the first DATA packet in the message, buf1, is filled with \$44 hex. The 18-byte data buffer for the second DATA packet in the message, buf2, is filled with \$45 hex. Next the remote station resets the RPC chip and awaits its token. The base station on start up uses this command to establish communication with the remote stations and ensure that they are online.

REHECK. Upon receiving a recheck command, \$Bx, the remote station performs the same tasks as if it had received a reset command. The difference is that the base station performs this command 10 times where as a reset is performed 3 times only.

TDM DATA. Upon receiving an acquire data command, \$Cx, the remote station will save the command to the variable *temp*, acquire data and place it in the appropriate data buffers (buf1 for energy data, buf2 for voltage/frequency data), and wait for a token. It should be noted that for the system that was prototyped, since the SAMES 9604A was not implemented, TDM DATA command places simulated data into the energy and voltage/frequency data buffers. Each remote station has a variable known as *sample*. *Sample* is incremented every time a TDM DATA command is received. The per-phase energy being simulated is the cosine of the sample while the per-phase voltage/frequency is the sine of the sample. The cosine and sine instruction in PicBasic Pro returns an 8-bit answer between ± 127 with the most significant bit being the sign bit.

The design of the monitoring system includes the SAMES 9604A Three-Phase Energy Metering IC. The communications system that was prototyped does not implement the SAMES 9604A but does provide the packet structure to transmit the data that the SAMES 9604A would provide. All that is required is the software subroutine to pull data from the 9604A registers.

An important feature of the 9604A is that its interface bus is tri-state able meaning that more than one 9604A IC could be connected to the same bus to the Microchip PIC16C77/JW. The prototyped remote stations currently have 18 bi-directional pins available for expansion. With three pins, (DI, D0, and SCK) being the common bus pins and two other pins required per transducer (FM150 and CS), a total of 7 separate transducer could possibly be connected to the same remote station. This is useful where a bank of transformers is located and only one transmitter/receiver would be needed for all.

4 TESTING AND IMPLEMENTATION

In order to test the communications protocol of the prototype monitoring system all but the SAMES 9604A metering IC was implemented in a base station plus three station hardware setup. The Visual Basic software was developed to send commands to the base controller, receive data and store to a file named data.txt. During these test the inter-packet and message times were not optimized, as this was not necessary until the communications algorithm functionality received a passing grade. Three tests were conducted: RESET, RECHECK, and TDM DATA.

RESET TEST

The command RESET (ASCII "A") was sent to the base controller and the following data was uploaded to the data file data.txt.

5/1/99 3:45		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39			
1	1	A1	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45		
2	1	A1	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	
3	1	A1	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	
1	2	A1	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	
2	2	A1	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	
3	2	A1	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	45	
1	3	FF	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB
2	3	FF	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB
3	3	FF	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB

Figure 18. Results of test 1 RESET.

Column 1 in data.txt represents the sample, column 2 the station, column 3 the command that was received in the ACK packet for each station, columns 4 to 21 represent the contents of the Energy Data packet, and columns 22 to 39 the contents of the Voltage/Frequency Data packet. As can be seen each station appropriately transmitted the data buffers for the appropriate data packets. Also, column 3 shows that stations 1 and 2, each sample, executed the command the first time that it was received and no retransmissions were necessary. For this test station 3 was turned off as such did not

The command RESET (ASCII “B”) was sent to the base controller and the following data was uploaded to the data file data.txt.

[illegible]

During this test station 3 was turned off. Station 1 required the command to be retransmitted 3 times on sample 5, 8, and 9. Station two did not require any retransmissions.

The command TDM DATA (ASCII “C”) was sent to the base controller and the following data was uploaded to the data file data.txt.

Figure 20. Results of test 3 TDM DATA.

It can be seen in each of the data packets a value is placed in the MSB of the 3 byte per-phase energies, voltages, and frequencies. Those values are the simulated data values of the sample's cosine, for \$44 DATA packets, and sine, for \$45 DATA packets.

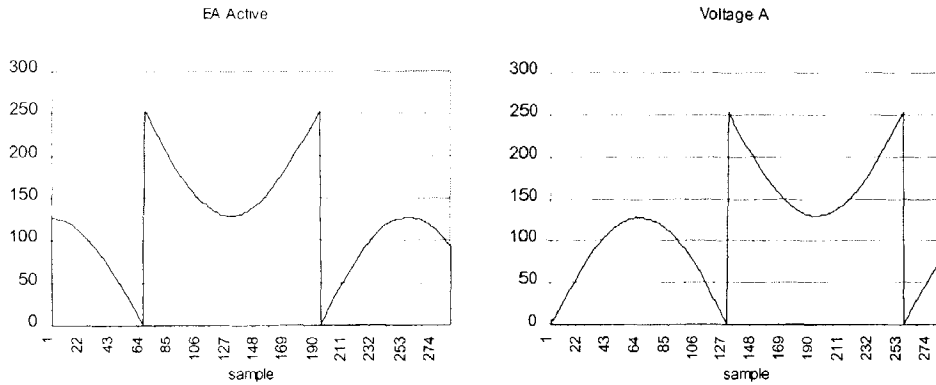


Figure 21. Remote station 1 simulated results.

As can be seen from Figure 21, packet \$44 did indeed return a perfect cosine function while packet \$45 a perfect sine function. Since the Microchip controller calculates an 8-bit sine/cosine the most significant bit carries the sign. Upon uploading the data was uploaded unsigned hence the resultant graphs. It should be pointed out that each station's energy was calculated as a cosine of the sample while the voltage and frequencies were calculated as the sine of the sample. In order to graph the total three-phase energy a summation of the per-phase data was done in Excel and the result is as follows.

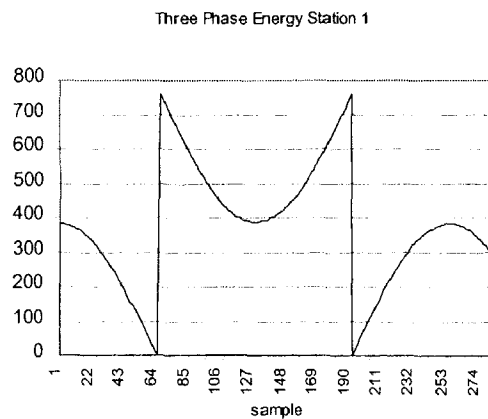


Figure 22. Three-Phase Energy Station 1.

5 CONCLUSION

To conclude this thesis the costs of the project, problems associated with prototyping and implementation, and the project's immediate future plans will be discussed.

PROJECT COST SUMMARY

Table 9 illustrates the costs of the proposed prototype. These costs do not reflect actual expenses as some devices were acquired free of charge. Table 9 also does not consider the cost of the Microchip PicStart Plus eeprom programmer or a UV eeprom lamp.

Table 9. Total project costs.

ITEM	Qty	Unit Price	Total Unit	Subtotal
Base Station	1	\$139.47	\$139.47	\$139.47
Remote Stations w/ transducer	3	\$155.13	\$ 465.39	\$ 604.86
TOTAL PROJECT COSTS	-	-	-	\$ 604.86

Table 10 considers the cost involved for the base station. The Radiometrix transmitter/receiver pair comprises 49.9% of the base station total cost. An alternative would be the design of a transmitter/receiver to lower the cost of the unit. Performance however might be sacrificed. Also the display used is an Optrex DMC-20171 20 character x 1 line LCD. These were chosen for their availability. A cheaper substitute would be a welcome option as the Optrex LCD makes up 22.2% of the base station cost.

Table 10. Base station cost summary.

ITEM	Qty	Unit Price	Total Unit	Subtotal
PIC16C77/JW	1	\$15.70	\$15.70	\$15.70
Radiometrix TX2-433-5V	1	\$26.32	\$26.32	\$42.02
Radiometrix RX2-433-F-5V	1	\$43.30	\$43.30	\$85.32
Whip antennas 433.92 MHz	2	\$0.00	\$0.00	\$85.32
Radiometrix RPC-DIL-000	1	\$8.00	\$8.00	\$93.32
Optrex DMC-20171 (LCD)	1	\$30.93	\$30.93	\$124.25
Texas Instruments MAX232	1	\$1.20	\$1.20	\$125.45
ATMEL AT24C256	2	\$4.00	\$8.00	\$133.45
FOX200 20.0 MHz	1	\$0.86	\$0.86	\$134.31
FOX100 10.0 MHz	1	\$0.76	\$0.76	135.07
Light Emitting Diodes	4	\$0.60	\$2.40	137.47
Res./Caps Various	-	-	\$2.00	\$139.47
TOTAL BASE STATION	-	-	-	\$139.47

Table 11 illustrates the cost of one remote station. The remote stations require neither EEPROMs nor MAX232. However the costs rises above the base station cost due to the inclusion of the SAMES transducer.

Table 11. Remote station cost summary.

ITEM	Qty	Unit Price	Total Unit	Subtotal
PIC16C77/JW	1	\$15.70	\$15.70	\$15.70
Radiometrix TX2-433-5V	1	\$26.32	\$26.32	\$42.02
Radiometrix RX2-433-F-5V	1	\$43.30	\$43.30	\$85.32
Whip antennas 433.92 MHz	2	-	-	\$85.32
Radiometrix RPC-DIL-000	1	\$8.00	\$8.00	\$93.32
Optrex DMC-20171 (LCD)	1	\$30.93	\$30.93	\$124.25
FOX200 20.0 MHz	1	\$0.86	\$0.86	\$125.11
FOX100 10.0 MHz	1	\$0.76	\$0.76	\$125.87
Light Emitting Diodes	4	\$0.60	\$2.40	\$128.27
Res./Caps Various	-	-	\$4.00	\$132.27
ECS-100A 3.579545 MHz	1	\$2.80	\$2.80	\$135.07
SAMES 9604A	1	\$20.06	\$20.06	\$155.13
TOTAL REMOTE STATION	-	-	-	\$155.13

The design of the remote stations includes a LCD just as with the base station. The display allows site monitoring of the remote station but more importantly provided a means to debug the programming. Currently the software functions and implements the

communications protocol well without failures. As such an option exists in using cheaper LED's to indicate the status of the remote stations instead of an expensive LCD. The communications is so rapid from station to station that the remote stations either are waiting for a command and display "Command Wait" or are waiting for an ACK/NACK from the base and display "ACK/NACK wait". As such the freed pins and program memory required by the display could be used to expand the number of SAMES transducers that could possibly be connected to one remote station controller.

PROTOTYPING AND IMPLEMENTATION

All four units, base plus remotes, were constructed on prototyping boards. Poor connections or moved components caused problems where they should not have existed. With many critical data transfers being demanded from the ICs used a more practical approach is to design PCBs for all units.

One of the most important ICs in the design is the Radiometrix RPC chip. These chips are actually preprogrammed Microchip PIC16F84 micro-controllers. Since the RPC chip contains configuration bytes in eeprom that may be changed by the user via the interface bus extreme care must be taken to prevent accidental writing to the RPC eeprom. Figure 10 illustrates that the most significant bit of the RPC C field of the frame structure dictates whether the next byte being passed to the RPC chip is data that needs to be transmitted or a change to the configuration in eeprom. The RPC chip provides many useful user changeable configurations but this system uses the default values. An unsolvable problem also occurred with two RPC chips. To be more specific, the RPC chips in question were able to transmit data packets but not receive. In their default power on state the RPC !RX pin should be at 0 volts while the !TX pin should be at 5 volts. In two of the RPC chips purchased from Radiometrix, on power up 5 volts signals were present on both !RX and !TX. As such due to the pnp transistor, on power up both the transmitter and receiver were turned off. The RPC chips were able to receive a data

packet from the controller, turn on the transmitter (!TX low), and transmit the data packet but not receive from the receiver. Accidental changes in the configuration bytes are suspected to be the source of this problem.

The communications algorithm is not optimized. The inter-packet times of the messages from the remote stations were set to allow the base station to receive the packets and process them. No optimization of this has been done here. Each ACK, DATA, and TOKEN packet is 7 ms, 12 ms, and 7 ms long. The inter-packet times are currently set for 10 ms between all packets. As such each message from the remote stations is 79 ms long. Further testing to increase the channel efficiency could optimize this.

PROJECT PLANS

This project is part of a two stage project to first prototype the design of a wireless power monitoring system and second to reach a full working implementation of the design. As such during the summer of 1999 at Texas Tech University (USA) work will continue on the project to complete the second stage. Plans include the design and construction of printed circuit boards (PCBs), SAMES transducer integration into the system, a redesign of the protocol to improve the algorithm efficiency, and testing the final product in a typical expected environment. Range testing will also be conducted after the PCBs are constructed. A thesis will be written for this project as well and available at the Texas Tech University library by author name, Adrian Pineda.

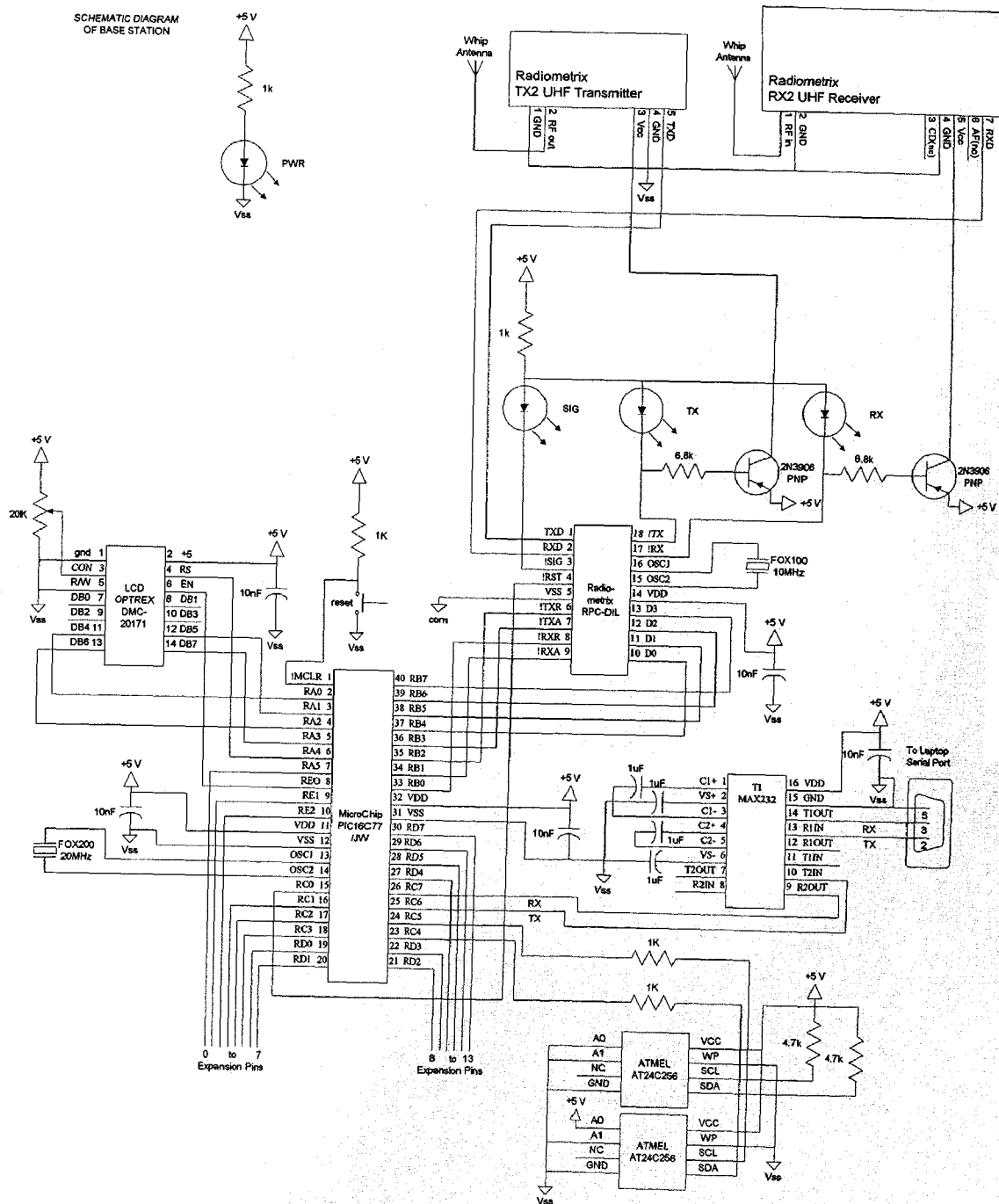
The author wishes to thank all those who contributed to this project and is available for comments or suggestions via email at the following address:

pineda@icec.org

REFERENCES

- [1] Microchip Literature. DS30390. Microchip Technology Inc. Chandler, Arizona 1997.
- [2] Llamas, Armando Ph.D., ET. Al. Sistema de Monitoreo de Potencias Real y Reactiva en Localidades Múltiples. Memorias de la XXV Reunión de Investigación y Desarrollo Tecnológico del Sistema ITESM, Tomo II, pp. 33-343. Jan. 1995.
- [3] Stutzman, Warren L. and Gary A. Thiele. Antenna Theory and Design. Second Edition. New York: John Wiley & Sons, Inc. 1998.
- [4] Goldie, John. Summary of well known interface standards. National Semiconductor Application Note 216. Jan. 1995.
- [5] Llamas, Armando Ph.D. and A Tejada. Aplicaciones del Sistema de Monitoreo de Potencias Real y Reactiva en el ITESM. I Simposio de Ciencia y Tecnología, Monterrey 400. Jan. 1995.
- [6] Keiser, Gerd E. Local Area Networks. New York: McGraw-Hill Book Company 1989.
- [7] Radiometrix Literature. RPC Data Sheet. Radiometrix Ltd. Hertfordshire, England. Feb. 1998.
- [8] Parzen, Benjamin. Design of Crystal and Other Harmonic Oscillators. New York: John Wiley & Sons. 1983.
- [9] SAMES Literature. 7190. PDS039-SA9604A-001 REV. C. SAMES (Pty) Ltd. 1998.


APPENDIX A: SCHEMATICS AND PROGRAMMING





GO BACK N ARO

MULTIPLE STATION DIGITAL WIRELESS
POWER MONITORING SYSTEM PROTOTYPE



ITESM
Campus Monterrey

RESET
SYSTEM

RECHECK
UNITS

TDM
DATA

UPLOAD
DATA

OK

YES

NO

TYPE OF UPLOAD (SAMPLES)





☐ PREVIOUS DAY'S MEASUREMENTS (256)



☐ COMMUNICATIONS CHECK (8)





☒ COMMUNICATIONS CHECK (40)

UPLOADED DATA

Text1







```

' Program for REMOTE stations (station.bas)

'- DEFINITIONS AND VARIABLES-----

DEFINE      OSC 20          ' Defines 20 MHz oscillator.
DEFINE      LCD_EREG PORTE  ' Defines LCD enable port as PORTE.
DEFINE      LCD_EBIT 0      ' Defines LCD enable bit as PORTE.0.

TX_Begin    con $24         ' Beginning location of packet to transmit.
RX_Begin    con $40         ' Beginning location of packet to receive.

type        var bit         ' Variables used in program.
temp        var byte
command     var byte
n           var byte
n2          var byte
sample      var byte
calc        var byte

PKT_TX      var byte[22] $24 ' Transmit buffer, start at location $24.
PKT_RX      var byte[22] $40 ' Receive buffer, beginning at location $40.
RPC_C       var byte        ' RPC_C control byte.
Address_Source var byte     ' Station address variable.

buf1        var byte[18]    ' Energy data buffer.
buf2        var byte[18]    ' Voltage frequency data buffer.

ASM                     ' Defines RPC\PIC handshaking lines
RPC                   EQU 06 ' ... for assembly subroutines.
TXA                   EQU 3
TXR                   EQU 2
RXA                   EQU 1
RXR                   EQU 0
ENDASM

D3              var PORTB.7  ' Defines RPC\PIC handshaking lines
D2              var PORTB.6  ' ... for PBASIC code.
D1              var PORTB.5
D0              var PORTB.4
TXA             var PORTB.3
TXR             var PORTB.2
RXA             var PORTB.1
RXR             var PORTB.0

RST            var PORTC.0   ' Defines RPC !MCLR line.

'- PROGRAM -----

goto START

'- SUBROUTINES -----

' Subroutine to reset the remote station by initializing data buffers and variables.

RESET_UNIT:
sample = 0

for n = 0 to 21
PKT_RX[n]=$FF          ' Fill receive buffer with $FF
PKT_TX[n]=$FF          ' Fill transmit buffer with $FF
next n

for n = 0 to 17
buf1[n]=$44            ' Fill energy buffer with $44.
buf2[n]=$45            ' Fill volt/freq. buffer with $45.

```

```

next n

TRISA = 0          ' PORTA (LCD) PIN Direction (0=out 1=in).
TRISC=%11111110   ' PORTC PIN Direction
Address_Source = $01 ' Station address, must be changed on programming
GOSUB RESET_RPC    ' Reset the RPC chip.

RETURN
-----

' Subroutine to transmit an ACK packet.

TX_ACK:

PKT_TX[0] = $05      ' "RPC_C Field" - Bytes in packet = 5
PKT_TX[1] = $41      ' "C Field" - Packets = 4 , type = 1 (ACK).
PKT_TX[2] = Address_Source ' "Address_Source Field" - Station address.
PKT_TX[3] = $00      ' "Address_Destination Field" = Central
PKT_TX[4] = temp      ' "Information Field" - Acknowledging the command stored
                     ' ... in temp.

RPC_C      = $05      ' Bytes in packet = 5.
GOSUB TX_PACKET      ' Send it.

RETURN
-----

' Subroutine to transmit Energy DATA packet.

TX_DATA44:

PKT_TX[0] = $16      ' "RPC_C Field" - Bytes in packet = 22.
PKT_TX[1] = $44      ' "C Field" - Packets in MSG = 4, type = 4 (Energy DATA).
PKT_TX[2] = Address_Source ' "Address_Source Field" - Station address.
PKT_TX[3] = $00      ' "Address_Destination Field" - Central is destination.
for n = 0 to 17      ' "Information Field" - Copy DATA from Energy buffer to
    n2=n+4            ' ... packet transmit buffer "Information Field".
    PKT_TX[n2] = buf1[n]
next n

RPC_C      = $16      ' Bytes in packet = 22.
GOSUB TX_PACKET      ' Send it.

RETURN
-----

' Subroutine to transmit Voltage/Frequency DATA packet.

TX_DATA45:

PKT_TX[0] = $16      ' "RPC_C Field" - Bytes in packet = 22.
PKT_TX[1] = $45      ' "C Field" - Packets in MSG = 4, type = 5
                     ' ... (Volt/freq DATA).
PKT_TX[2] = Address_Source ' "Address_Source Field" - Address of the station sending
                     ' ... packet.
PKT_TX[3] = $00      ' "Address_Destination Field" - Central is destination.
for n = 0 to 17      ' "Information Field" - Copy DATA from Volt/freq buffer to
    n2=n+4            ' ... packet transmit buffer "Information Field".
    PKT_TX[n2] = buf2[n]
next n

RPC_C      = $16      ' Bytes in packet = 22
GOSUB TX_PACKET      ' Send it.

RETURN
-----

```

```
' Subroutine to transmit TOKEN packet
```

```
TX_TOKEN:
```

```
PKT_TX[0] = $05      ' "RPC_C Field" - Bytes in packet = 5.
PKT_TX[1] = $43      ' "C Field" - Packets in MSG = 4 , type = 3 (TOKEN)
PKT_TX[2] = Address_Source ' "Address_Source Field" - Address of the station sending.
PKT_TX[3] = Address_Source+1 ' "Address_Destination Field" - Destination is next sta.
PKT_TX[4] = $AA      ' "Information Field" - Place TOKEN $AA.

RPC_C      = $05      ' Bytes in packet = 5.
GOSUB TX_PACKET      ' Send it.
```

```
RETURN
```

```
' Subroutine sets Energy and Voltage/frequency data buffers with token data.
```

```
ACQUIRE_DATA:
```

```
for n=0 to 17
  buf1[n]=$00      ' Fill Energy buffer with zeroes.
  buf2[n]=$00      ' Fill Volt/freq buffer with zeroes.
next n
```

```
calc = cos sample      ' Calculate token Energy data.
```

```
buf1[0] = calc      ' EC REACTIVE MSB
buf1[3] = calc      ' EC ACTIVE MSB
buf1[6] = calc      ' EB REACTIVE MSB
buf1[9] = calc      ' EB ACTIVE MSB
buf1[12] = calc     ' EA REACTIVE MSB
buf1[15] = calc     ' EA ACTIVE MSB
```

```
calc = sin sample      ' Calculate token Volt/freq data.
```

```
buf2[0] = calc      ' fC MSB
buf2[3] = calc      ' VC MSB
buf2[6] = calc      ' fB MSB
buf2[9] = calc      ' VB MSB
buf2[12] = calc     ' fA MSB
buf2[15] = calc     ' VA MSB
```

```
sample = sample + 1      ' Increment the sample.
```

```
RETURN
```

```
' Subroutine to download a variable length packet to RPC.
```

```
TX_PACKET:
```

```
IF PORTB.0 = 0 THEN      ' First test to see if received packet but ignore it
  POKE $04,RX_Begin      ' ... this is done to solve hang-ups.
  CALL IN_BYTE
```

```
for n=2 to PKT_RX[0]
  INCF FSR
  CALL IN_BYTE
next n
```

```
CALL LISTEN_BUS
```

```
endif
```

```
POKE $04,TX_Begin
```

```
For n=1 to RPC_C
```

```
CALL OUT_BYTE
```

```
' Transmit the packet beginning at TX_Begin.
```

```
' Transfer variable number of bytes
' ... to RPC chip as determined by RPC_C byte.
```

```

@   INCF FSR
    next n

    CALL LISTEN_BUS          ' Resets the RPC/PIC bus.

    RETURN
-----
' Subroutine to upload a variable length packet from RPC.

RX_Packet:
    POKE $04,RX_Begin
    CALL IN_BYTE

    for n=2 to PKT_RX[0]      ' Transfer variable number of bytes
@   INCF FSR                  ' ... as determined by first byte received.
    CALL IN_BYTE
    next n

    CALL LISTEN_BUS          ' Resets the RPC/PIC bus.

    RETURN

'- ASSEMBLY SUBROUTINES -----
ASM
;-----
;
; Subroutine to configure PORTB of the PIC16C77 to wait for a packet. Credit Radiometrix.
;
_LISTEN_BUS    BSF     STATUS,RP0
               MOVLW   0xF9          ;B'11111001'
               MOVWF   TRISB
               BCF     STATUS,RP0
               RETURN
;-----
;
; Subroutine to download one byte to the RPC stored in location pointed to by FSR
; ... FSR must be previously loaded. Credit Radiometrix.
;
_OUT_BYTE      SWAPF   INDF,W          ;GET LS NIBBLE FROM FILE (VIA FSR) INTO
;                                           ;BITS 4 to 7 of W
;
;               ANDLW   0xF0          ;JUST THE NIBBLE
;               IORLW   0x02          ;SET TXR LOW, LEAVE RXA HIGH
;               MOVWF   RPC          ;SET TXR LOW, OUTPUT NIBBLE
;
WACCEPT      BTFSC   RPC,TXA          ;WE GOT A TX ACCEPT BACK YET?
               GOTO    WACCEPT      ;NO, SO LOOP BACK AND WAIT
;
;WE GOT ACCEPTANCE SO IT'S OK TO DRIVE BUS
;
               BSF     STATUS,RP0      ;SELECT PAGE 1
               MOVLW   0x09          ;DRIVE BUS
               MOVWF   TRISB
               BCF     STATUS,RP0      ;SELECT PAGE 0 BUS IS NOW DRIVING
;
               BSF     RPC,TXR          ;REMOVE REQUEST, DATA IS ON BUS
WDUN           BTFSS   RPC,TXA          ;HAS DATA BEEN READ?
               GOTO    WDUN           ;WAIT TILL RPC REMOVES ACCEPT
;
;LS NIBBLE OF (FSR) IS SENT , NOW DO MS NIBBLE
;
               MOVF    INDF,W          ;GET MS NIBBLE FROM FILE (VIA FSR)
;
               ANDLW   0xF0          ;JUST THE MS NIBBLE
               IORLW   0x02          ;SET TXR LOW (BIT 2), RXA STAYS HIGH

```

```

MOVWF  RPC          ;OUTPUT NIBBLE + TXR LOW
;
WACCEPT1  BTFSC  RPC,TXA      ;WE GOT A TX ACCEPT BACK YET?
GOTO     WACCEPT1    ;NO, SO LOOP BACK AND WAIT
;
BSF      RPC, TXR      ;REMOVE REQUEST, DATA IS ON BUS
;
WDUN1     BTFSS  RPC,TXA      ;HAS DATA BEEN READ?
GOTO     WDUN1        ;WAIT TILL RPC REMOVES ACCEPT
;
RETURN
;
-----
; Subroutine to upload one byte from the RPC and store it in location determined by the
; ... FSR, FSR must be previously loaded. Credit Radiometrix.
;
_IN_BYTE  BTFSC  RPC,RXR      ;WE GOT A RX REQUEST YET?
GOTO     _IN_BYTE    ;NO , SO LOOP BACK AND WAIT
;
; READ THE LS NIBBLE FROM THE RPC
;
BCF      RPC, RXA      ;ACCEPT THE REQUEST (SET ACCEPT LOW)
;
AWAITDATA BTFSS  RPC,RXR      ;HAS REQUEST GONE UP? data is present
GOTO     AWAITDATA    ;LOOP BACK TILL IT DOES
;
NOP      ;TIME DELAY TO ENSURE DATA STABLE
; ;BEFORE READ
;
MOVF     RPC,W         ;READ THE LS NIBBLE FROM THE BUS
BSF      RPC, RXA      ;TELL RPC WE GOT NIBBLE (ACCEPT = 1)
ANDLW    0xF0         ;JUST THE DATA
;
MOVWF    INDF          ;SAVE LS NIBBLE IN TARGET FILE (VIA FSR)
;
SWAPF    INDF          ;RIGHT JUSTIFY LS NIBBLE
;
; NOW GET MS NIBBLE FROM THE RPC
;
;
INNIBBLE  BTFSC  RPC,RXR      ;WE GOT NEXT RX REQUEST YET ?
GOTO     INNIBBLE    ;NO , SO LOOP BACK AND WAIT
;
BCF      RPC, RXA      ;ACCEPT REQUEST (SET ACCEPT LOW)
;
AWAITD1   BTFSS  RPC,RXR      ;HAS REQUEST GONE UP? data is present
GOTO     AWAITD1     ;LOOP BACK TILL IT DOES
;
NOP      ;TIME DELAY TO ENSURE DATA STABLE
; ;BEFORE READ
;
MOVF     RPC,W         ;READ THE MS NIBBLE FROM THE BUS
BSF      RPC, RXA      ;TELL RPC WE GOT NIBBLE (ACCEPT=1)
ANDLW    0xF0         ;JUST THE DATA
;
IORWF    INDF          ;COMBINE MS NIBBLE WITH LS NIBBLE
;ALREADY
;
RETURN    ;IN THE FILE (VIA FSR)RETURN
;
; A BYTE HAS BEEN READ FROM THE RPC INTO ADDRESS POINTED AT BY FSR
;
ENDASM
;
-----

```

```

' Subroutine to reset the RPC chip.

RESET_RPC:

    CALL LISTEN_BUS

    TXR=1
    RXA=1
    RST=0                ' !MCLR low.
    pause 2
    RST=1                ' !MCLR high.
    pause 2
    RETURN

'- END SUBROUTINES -----

'- MAIN PROGRAM -----

START:
    CLEAR                ' Zero all variables.
    Pause 500            ' 500ms delay.

    Lcdout $fe, 1        ' Clear LCD screen.
    Lcdout "Resetting Unit"
    Pause 1000          ' Wait 1.0 seconds.

    GOSUB RESET_UNIT    ' Reset the station.

    Lcdout $fe, 1        ' Clear LCD screen.
    Lcdout "Unit Address = ", dec2 Address_Source
    Pause 1000          ' Wait 1.0 seconds

command_wait:

    Lcdout $fe, 1        ' Clear LCD screen
    Lcdout "COMMAND STANDBY"

wait:
    GOSUB RX_Packet      ' Wait until receive a packet

' If not a NACK to me or previous station then test if command type of packet else we
' ... have been nacked so proceed to wait for token.

    IF PKT_RX[1] != $12 THEN proceed
    IF PKT_RX[3] <= Address_Source THEN token_wait

proceed:
    IF PKT_RX[1] != $20 THEN wait    ' If command then excecute, wait for token.
    ' ... else keep waiting.
    temp = PKT_RX[4]                ' ... save command.

command_test:                ' Test the command.

    command = temp & %11110000
    pause 1

    IF command = $A0 THEN case1    ' Test if RESET
    IF command = $B0 THEN case1    ' Test if RECHECK
    IF command = $C0 THEN case2    ' TEST if TDM DATA

    goto command_wait            ' Keep waiting if not a valid command.

case1: GOSUB RESET_UNIT          ' If RESET or RECHECK reset unit.
    goto token_wait             ' Wait for token.

```



```

case2:
    GOSUB ACQUIRE_DATA          ' Acquire data.

token_wait:

    Lcdout $fe, 1                ' Clear LCD screen
    Lcdout "Token wait"

    GOSUB RX_Packet              ' Wait until receive a packet.

' Test packet, test if addressed to me and if packet is token. If not, keep waiting.

    temp2=PKT_RX[1] & %00001111
    IF PKT_RX[3] != Address_Source THEN token_wait
    IF temp2 != $03 THEN token_wait

    pause 25                     ' If token has been received back off before
                                ' ... transmitting message.

' Check to see if received NACK in critical period,if not a NACK to me or previous
' ... station then proceed to begin to transmit the message.

    IF PORTB.0 = 0 THEN
        GOSUB RX_Packet
        IF PKT_RX[1] != $12 THEN proceed2
        IF PKT_RX[3] <= Address_Source THEN token_wait
        goto token_wait
    endif

proceed2:
    GOSUB TX_ACK                 ' Send ACK frame
    pause 25                     ' Back off a bit

' Check to see if received NACK in critical period,if not a NACK to me or previous
' ... station then proceed to continue to transmit the message.

    IF PORTB.0 = 0 THEN
        GOSUB RX_Packet
        IF PKT_RX[1] != $12 THEN proceed3
        IF PKT_RX[3] <= Address_Source THEN token_wait
        goto token_wait
    endif

proceed3:
    GOSUB TX_DATA44              ' send Energy DATA frame
    pause 50

' Check to see if received NACK in critical period,if not a NACK to me or previous
station then
' ... proceed to continue to transmit the message.

    IF PORTB.0 = 0 THEN
        GOSUB RX_Packet
        IF PKT_RX[1] != $12 THEN proceed4
        IF PKT_RX[3] <= Address_Source THEN token_wait
        goto token_wait
    endif

proceed4:
    GOSUB TX_DATA45              ' Send Volt/freq DATA frame
    pause 50

' Check to see if received NACK in critical period,if not a NACK to me or previous
' ... station then proceed to continue to transmit the message.

    IF PORTB.0 = 0 THEN

```

```

        GOSUB RX_Packet
        IF PKT_RX[1] != $12 THEN proceed5
        IF PKT_RX[3] <= Address_Source THEN token_wait
        goto token_wait
    endif

proceed5:
    GOSUB TX_TOKEN          ' Send TOKEN frame
    pause 25

    GOSUB RESET_RPC        ' Reset to prevent hang ups.

' Message done now wait for either an ACK or NACK from central.

    Lcdout $fe, 1          ' Clear LCD screen
    Lcdout "ACK/NACK wait"

wait2: GOSUB RX_Packet

' If packet received not a NACK to me or previous station then test if ACK
' ... else wait for token with previous data transmitted saved in RAM buffer.

    IF PKT_RX[1] != $12 THEN proceed6
    IF PKT_RX[3] <= Address_Source THEN token_wait

proceed6:

' Test to see if packet received was an ACK if not then keep waiting for it.

    IF PKT_RX[1] = $11 THEN
        goto command_wait    ' Done so wait for command.
    else
        goto wait2
    endif
end
end

' - END OF MAIN -----

```

```

' Program for BASE STATION (base.bas)

'- DEFINITIONS AND VARIABLES-----

Include "modedefs.bas"          ' Include serial modes

DEFINE OSC      20              ' Defines 20 MHz oscillator.
DEFINE HSER_RCSTA 90h           ' Set up USART.
DEFINE HSER_TXSTA 20h
DEFINE HSER_BAUD 9600           ' ... USART baud = 9600.
DEFINE LCD_EREG PORTE          ' Defines LCD enable port as PORTE.
DEFINE LCD_EBIT 0              ' Defines LCD enable bit as PORTE.

DPIN   var   PORTC.4           ' I2C data pin.
CPIN   var   PORTC.5           ' I2C clock pin.

' Definition of constants used in program.

cont1      con %10100000      ' eeprom 1 control byte.
cont2      con %10100010      ' eeprom 2 control byte.
TX_Begin   con $20            ' Beginning location of packet to transmit.
RX_Begin   con $25            ' Beginning location of packet to receive.

RESET      con $A1
RECHECK    con $B1
TDM_DATA   con $C1
from_portb con %0
from_tmr0   con %1
fail        con $FF
resend_max  con $0F
first       con $AA
second      con $BB
third       con $CC
fourth      con $DD

' Definition of variables used in program.

cont        var byte
temp        var byte
temp2       var byte
x           var byte
type        var bit
command     var byte
n           var byte
flag        var byte
flag_save   var byte
sample      var word
sample_max  var word
sample_min  var word
sample_save var word

current_packet var byte
current_station var byte
station_max    var byte

PKT_TX        var byte[5] $20
PKT_RX        var byte[22] $25
RPC_C         var byte
C             var byte
Address_Source var byte
Address_Destination var byte

data_buf1     var byte[40]
data_buf2     var byte[40]
data_buf3     var byte[40]

```

```
offset      var word
pointer     var word
eeaddr      var word
```

```
ASM
RPC          EQU 06
TXA          EQU 3
TXR          EQU 2
RXA          EQU 1
RXR          EQU 0
ENDASM
```

```
D3          var PORTB.7
D2          var PORTB.6
D1          var PORTB.5
D0          var PORTB.4
TXA          var PORTB.3
TXR          var PORTB.2
RXA          var PORTB.1
RXR          var PORTB.0
```

```
RST          var PORTC.0
led          var PORTE.1
```

```
'- PROGRAM -----
```

```
    pause 500
```

```
    clear                      ' Zero all variables
    OPTION_REG = $15           ' Configure Interrupts, set Prescaler
```

```
    On Interrupt Goto myint    ' Define interrupt handler
```

```
    goto wait_flag            ' Wait for command from laptop
```

```
'- SUBROUTINES -----
```

```
' Interrupt handler, tests if interrupt was due to PORTB.0 or TIMER0. The result of
' ... this test is returned in variable "type".
```

```
    Disable                    ' Disable interrupts past this point
myint: IF INTCON.2 = 1 THEN
    INTCON.2 = 0
    type = from_tmr0           ' if timer interrupted
    TMR0=0
    x=x+1
ELSE
    INTCON.1 = 0
    type = from_portb          ' if PORTB.0 interrupted

ENDIF
    Resume                     ' Return to main program
    Enable                     ' Enable interrupts past this point
```

```
'-----
```

```
'----- ASSEMBLY SUBROUTINES -----
```

```
ASM
```

```
;-----
; Subroutine to configure PORTB of the PIC16C77 to wait for a packet.
; ... credit Radiometrix.
```

```
;
_LISTEN_BUS    BSF    STATUS,RP0
                MOVLW    0xF9                ;B'11111001'
```

```

MOVWF  TRISB
BCF    STATUS,RP0
RETURN

```

```

;-----
; Subroutine to download one byte to the RPC stored in location pointed to by FSR
; ... FSR must be previously loaded. Credit Radiometrix.
;

```

```

_OUT_BYTE    SWAPF    INDF,W          ;GET LS NIBBLE FROM FILE (VIA FSR) INTO
;                                                    ;BITS 4 to 7 of W
;
;           ANDLW    0xF0          ;JUST THE NIBBLE
;           IORLW    0x02          ;SET TXR LOW, LEAVE RXA HIGH
;           MOVWF    RPC          ;SET TXR LOW, OUTPUT NIBBLE
;
WACCEPT      BTFSC    RPC,TXA        ;WE GOT A TX ACCEPT BACK YET?
;           GOTO     WACCEPT      ;NO, SO LOOP BACK AND WAIT
;

```

```

;WE GOT ACCEPTANCE SO IT'S OK TO DRIVE BUS
;

```

```

;           BSF      STATUS,RP0     ;SELECT PAGE 1
;           MOVLW    0x09          ;DRIVE BUS
;           MOVWF    TRISB
;           BCF      STATUS,RP0     ;SELECT PAGE 0 BUS IS NOW DRIVING
;
;           BSF      RPC,TXR        ;REMOVE REQUEST, DATA IS ON BUS
WDUN         BTFSS    RPC,TXA        ;HAS DATA BEEN READ?
;           GOTO     WDUN          ;WAIT TILL RPC REMOVES ACCEPT
;

```

```

;LS NIBBLE OF (FSR) IS SENT , NOW DO MS NIBBLE
;

```

```

;           MOVF     INDF,W          ;GET MS NIBBLE FROM FILE (VIA FSR)
;
;           ANDLW    0xF0          ;JUST THE MS NIBBLE
;           IORLW    0x02          ;SET TXR LOW (BIT 2), RXA STAYS HIGH
;           MOVWF    RPC          ;OUTPUT NIBBLE + TXR LOW
;
WACCEPT1     BTFSC    RPC,TXA        ;WE GOT A TX ACCEPT BACK YET?
;           GOTO     WACCEPT1     ;NO, SO LOOP BACK AND WAIT
;
;           BSF      RPC,TXR        ;REMOVE REQUEST, DATA IS ON BUS
;
WDUN1        BTFSS    RPC,TXA        ;HAS DATA BEEN READ?
;           GOTO     WDUN1        ;WAIT TILL RPC REMOVES ACCEPT
;
;           RETURN
;

```

```

;-----
; Subroutine to upload one byte from the RPC and store it in location
; ... determined by the FSR, FSR must be previously loaded
; ... credit Radiometrix.
;

```

```

_IN_BYTE     BTFSC    RPC,RXR        ;WE GOT A RX REQUEST YET?
;           GOTO     _IN_BYTE     ;NO , SO LOOP BACK AND WAIT
;

```

```

; READ THE LS NIBBLE FROM THE RPC
;

```

```

;           BCF      RPC,RXA        ;ACCEPT THE REQUEST (SET ACCEPT LOW)
;
AWAITDATA    BTFSS    RPC,RXR        ;HAS REQUEST GONE UP? data is present
;           GOTO     AWAITDATA    ;LOOP BACK TILL IT DOES
;
;           NOP                    ;TIME DELAY TO ENSURE DATA STABLE
;           ;BEFORE READ
;

```

```

;
;      MOVF    RPC,W           ;READ THE LS NIBBLE FROM THE BUS
;      BSF     RPC,RXA         ;TELL RPC WE GOT NIBBLE (ACCEPT = 1)
;      ANDLW   0xF0            ;JUST THE DATA
;
;      MOVWF   INDF            ;SAVE LS NIBBLE IN TARGET FILE (VIA FSR)
;
;      SWAPF   INDF            ;RIGHT JUSTIFY LS NIBBLE
;
; NOW GET MS NIBBLE FROM THE RPC
;
;
INNIBBLE    BTFSC   RPC,RXR     ;WE GOT NEXT RX REQUEST YET ?
;           GOTO    INNIBBLE     ;NO , SO LOOP BACK AND WAIT
;
;           BCF     RPC,RXA      ;ACCEPT REQUEST (SET ACCEPT LOW)
;
AWAITD1     BTFSS   RPC,RXR     ;HAS REQUEST GONE UP? data is present
;           GOTO    AWAITD1      ;LOOP BACK TILL IT DOES
;
;           NOP                 ;TIME DELAY TO ENSURE DATA STABLE
;                               ;BEFORE READ
;
;           MOVF    RPC,W           ;READ THE MS NIBBLE FROM THE BUS
;           BSF     RPC,RXA         ;TELL RPC WE GOT NIBBLE (ACCEPT=1)
;           ANDLW   0xF0            ;JUST THE DATA
;
;           IORWF   INDF            ;COMBINE MS NIBBLE WITH LS NIBBLE
;ALREADY
;                               ;IN THE FILE (VIA FSR)RETURN
;           RETURN
;
; A BYTE HAS BEEN READ FROM THE RPC INTO ADDRESS POINTED AT BY FSR
;
ENDASM

```

'Subroutine to reset the central by initializing the data buffers and variables.

```

RESET_UNIT:
    sample = 0

    for n = 0 to 27
        PKT_RX[n]=$FF          ' Fill receive buffer with $FF
        PKT_TX[n]=$FF          ' Fill transmit buffer with $FF
    next n

    for n = 0 to 39
        data_buf1[n]=$AA        ' Fill station 1 buffer with $AA
        data_buf2[n]=$BB        ' Fill station 2 buffer with $BB
        data_buf3[n]=$CC        ' Fill station 3 buffer with $CC
    next n

    Address_Source = 0          ' Central address

    GOSUB RESET_RPC             ' Reset the RPC chip

    RETURN

```

' Subroutine to send a command packet to remote stations.

```

TX_COMMAND:

    PKT_TX[0] = $05             ' "RPC_C Field" - Bytes in packet = 5.
    PKT_TX[1] = $20             ' "C Field" - Packets in msg = 2 , type = 0 (COMMAND)

```

```

PKT_TX[2] = Address_Source      ' "Address_Source Field" - Station address.
PKT_TX[3] = $FF                ' "Address_Destination Field" - All stations.
PKT_TX[4] = flag                ' "Information Field" - Command to send.

```

```

RPC_C      = $05                ' Bytes in packet = 5
GOSUB TX_PACKET                  ' Send it.

```

```

RETURN

```

```

'-----
' Subroutine to transmit an ACK packet.

```

```

TX_ACK:

```

```

PKT_TX[0] = $05                ' "RPC_C Field" - Bytes in packet = 5.
PKT_TX[1] = $11                ' "C Field" - Packets in msg = 4 , type = 1 (ACK).
PKT_TX[2] = $00                ' "Address_Source Field" - Station address.
PKT_TX[3] = $FF                ' "Address_Destination Field" - All is ACK destination
PKT_TX[4] = temp                ' "Information Field" - Acknowledging the command.

```

```

RPC_C      = $05                ' Bytes in packet = 5.
GOSUB TX_PACKET                  ' Send it.

```

```

RETURN

```

```

'-----
' Subroutine to transmit a NACK packet.

```

```

TX_NACK:

```

```

PKT_TX[0] = $05                ' "RPC_C Field" - Bytes in packet = 5.
PKT_TX[1] = $12                ' "C Field" - Packets in msg = 1 , type = 2 (NACK).
PKT_TX[2] = Address_Source      ' "Address_Source Field" - Station address.
PKT_TX[3] = current_station     ' "Address_Destination Field" - All Go back to Nacked
station.                        '
PKT_TX[4] = current_packet      ' "Information Field" - particular packet Nacked.

```

```

RPC_C      = $05                ' Bytes in packet = 5.
GOSUB TX_PACKET                  ' Send it.

```

```

RETURN

```

```

'-----
' Subroutine to transmit a TOKEN packet.

```

```

TX_TOKEN:

```

```

PKT_TX[0] = $05                ' "RPC_C Field" - Bytes in packet = 5.
PKT_TX[1] = $23                ' "C Field" - Packets in msg = 2 , type = 3 (TOKEN).
PKT_TX[2] = Address_Source      ' "Address_Source Field" - Station address.
PKT_TX[3] = Address_Destination ' "Address_Destination Field" - Station destination.
PKT_TX[4] = $AA                ' "Information Field" - TOKEN.

```

```

RPC_C      = $05                ' Bytes in packet = 5.
GOSUB TX_PACKET                  ' Send it.

```

```

RETURN

```

```

'-----
' Subroutine to download a variable length packet to the RPC.

```

```

TX_PACKET:

```

```

IF PORTB.0 = 0 THEN            ' First test to see if received packet but ignore it
    POKE $04,RX_Begin          ' ... this is done to solve hang ups.
    CALL IN_BYTE

```

```

for n=2 to PKT_RX[0]

```

```

@    INCF FSR

```

```

        CALL    IN_BYTE
        next n

        CALL    LISTEN_BUS
    endif

    POKE        $04,TX_Begin      ' Load FSR with address of first byte.

    for n=1 to RPC_C              ' Transfer variable number of bytes
        CALL    OUT_BYTE          ' ..to RPC chip as determined by RPC_C byte.
    @      INCF    FSR
    next n

        CALL    LISTEN_BUS        ' Resets the RPC7/PIC bus.

    RETURN
-----

' Subroutine to upload a variable length packet from the RPC.
RX_Packet:
    POKE        $04,RX_Begin      ' Upload the packet and store beginning at RX_Begin.
    CALL        IN_BYTE

    for n=2 to PKT_RX[0]          ' Transfer variable number of bytes
    @      INCF    FSR              ' ..as determined by first byte received.
        CALL    IN_BYTE
    next n

    CALL        LISTEN_BUS

    RETURN
-----

' Subroutine to reset the RPC.
RESET_RPC:

    TRISA = 0                      ' PORTA (LCD) PIN Direction (0=out 1=in)
    CALL        LISTEN_BUS
    TRISC=%11111110                ' PORTC PIN Direction

    TXR=1
    RXA=1
    RST=0                          ' !MCLR low
    pause 10
    RST=1                          ' !MCLR high
    pause 10
    RETURN
-----

' - END SUBROUTINES -----
' - MAIN PROGRAM -----

START:
    pause        500                ' 500ms delay.

    Lcdout $fe, 1                  ' Clear LCD screen.
    Lcdout "Resetting Central"
    Pause 1000                    ' Wait 1.0 seconds.

    GOSUB RESET_UNIT

    Lcdout $fe, 1                  ' Clear LCD screen.
    Lcdout "Central Address = ", dec2 Address_Source

```



```

    Pause 1000                ' Wait 1.0 seconds.
check:
    Lcdout $fe, 1             ' Clear LCD screen.
        Lcdout "Resetting Units"
    pause 2000                ' Wait 2.0 seconds.
call_units:
    station_max=3             ' Programmable variable number of stations.
    sample=0                  ' Initialize sample to zero.
    current_station=$00       ' Initialize the current station as 0.
    Address_Destination = $01 ' Send Message to remote station 01.
main_loop:
    sample = sample+1         ' Increment the sample.
    GOSUB TX_COMMAND          ' Transmit COMMAND.
    pause 25                  ' Back off before transmitting TOKEN.
    GOSUB TX_TOKEN            ' Transmit TOKEN.

wait:  current_packet=$41     ' Current packet we are waiting for is an ACK.
    current_station=current_station+1 ' Current station we are waiting for.
    flag = flag + 1          ' Increment the flag, in case of retransmit.

' Set up the interrupts on PORTB.0 and TIMER0. Formula to calculate delay:
'
' delay = (Prescaler)*($FF-TMR0)*(x of loops)*(200 nsec)

wait2:
    type = from_tmr0
    x = 0
    TMR0=0

nacked: INTCON = $B0          ' Enable the interrupts.

loop:  IF type = from_portb THEN pb_cont ' Test the type of interrupt.
    IF x = 25 THEN adrian          ' If from TMR0 then test if x = max.
    goto loop                      ' No then keep on.

' This portion of program is used for retransmission of command + token packets to
' ... a station if it has failed. If it fails then moves on to next station by
' ... sending that station's token.

adrian: x = 0                   ' Timer expired, resend.
    type = from_tmr0
    INTCON = 0

    temp2= flag & %00001111
    IF temp2=resend_max THEN      ' Test if exceeded max number of retransmissions.
        temp = fail
        if current_station = 1 then store1
        if current_station = 2 then store2
        if current_station = 3 then store3
store1:
    data_buf1[0]=sample.highbyte
    data_buf1[1]=sample.lowbyte
    data_buf1[2]=1
    data_buf1[3]=temp
    for n = 4 to 21
        data_buf1[n]=$AA ' Store token data if station 1 failed.
    next n
    for n = 22 to 39
        data_buf1[n]=$BB
    next n
    goto adrian2
store2:
    data_buf2[0]=sample.highbyte
    data_buf2[1]=sample.lowbyte
    data_buf2[2]=2
    data_buf2[3]=temp

```

```

        for n = 4 to 21
            data_buf2[n]=$AA    ' Store token data if station 2 failed.
        next n
        for n = 22 to 39
            data_buf2[n]=$BB
        next n
        goto adrian2
store3:
    data_buf3[0]=sample.highbyte
    data_buf3[1]=sample.lowbyte
    data_buf3[2]=3
    data_buf3[3]=temp
    for n = 4 to 21
        data_buf3[n]=$AA    ' Store token data if station 3 failed.
    next n
    for n = 22 to 39
        data_buf3[n]=$BB
    next n

' A station has failed so we must send that station's token to keep communication going.

adrian2:
    Address_Source=current_station
    Address_Destination=current_station + 1
    Gosub Tx_Token
    Address_Source=0

    goto case4

ELSE

    flag = flag + 1            ' Station has not failed yet.
    GOSUB TX_COMMAND          ' Resend command.
    pause 50
    Address_Destination=current_station
    GOSUB TX_TOKEN
    current_packet=$41
    goto wait2
ENDIF

' PORTB.0 generated the interrupt.  Packet is waiting in the RPC buffer.

pb_cont:
    x = 0                    ' Reset x and type variables.
    type=from_tmr0
    INTCON=0                 ' Disable PB.0 interrupt.
    GOSUB RX_Packet          ' Upload the packet to test it.

' Test the packet to see if it is from the current station and whether or not it is
' ... the correct packet in the sequence.

    IF PKT_RX[2] != current_station THEN nack_it ' Test source address.
    IF PKT_RX[1] != current_packet THEN nack_it ' Test the C field.

' Depending on which packet we are waiting for and given that we have received it then
' ... we need to store the information according to the unit transmitting the packet.

    IF PKT_RX[1] = $41 THEN case1
    IF PKT_RX[1] = $44 THEN case2
    IF PKT_RX[1] = $45 THEN case3
    goto case4
case1:
    temp = PKT_RX[4]          ' Store acked command in all cases.
    current_packet = $44

    IF current_station = 1 THEN store_ack1

```

```

    IF current_station = 2 THEN store_ack2
    IF current_station = 3 THEN store_ack3
store_ack1:
    data_buf1[0]=sample.highbyte ' Store sample number.
    data_buf1[1]=sample.lowbyte
    data_buf1[2]=1 ' Store station number.
    data_buf1[3]=PKT_RX[4]
    ' Store acked command.
    goto next_packet2
store_ack2:
    data_buf2[0]=sample.highbyte ' Store sample number.
    data_buf2[1]=sample.lowbyte
    data_buf2[2]=2
    ' Store station number.
    data_buf2[3]=PKT_RX[4]
    ' Store acked command.
    goto next_packet2
store_ack3:
    data_buf3[0]=sample.highbyte ' Store sample number.
    data_buf3[1]=sample.lowbyte
    data_buf3[2]=3 ' Store station number.
    data_buf3[3]=PKT_RX[4] ' Store acked command.
next_packet2:
    INTCON = $B0 ' Enable the interrupts.
    goto loop ' Wait for next packet.

case2: current_packet = $45

    IF current_station = 1 THEN store_44data_1
    IF current_station = 2 THEN store_44data_2
    IF current_station = 3 THEN store_44data_3
store_44data_1:
    for n = 4 to 21
        data_buf1[n]=PKT_RX[n] ' Store the energy data.
    next n
    goto next_packet3
store_44data_2:
    for n = 4 to 21
        data_buf2[n]=PKT_RX[n] ' Store the energy data.
    next n
    goto next_packet3
store_44data_3:
    for n = 4 to 21
        data_buf3[n]=PKT_RX[n] ' Store the energy data.
    next n
next_packet3:
    INTCON = $B0 ' Enable the interrupts.
    goto loop ' Wait for next packet.

case3: current_packet = $43

    IF current_station = 1 THEN store_45data_1
    IF current_station = 2 THEN store_45data_2
    IF current_station = 3 THEN store_45data_3
store_45data_1:
    for n = 4 to 21
        data_buf1[n+18]=PKT_RX[n] ' Store the volt/freq data.
    next n
    goto next_packet4 ' Then wait for next packet.
store_45data_2:
    for n = 4 to 21
        data_buf2[n+18]=PKT_RX[n] ' Store the volt/freq data.
    next n
    goto next_packet4 ' Then wait for next packet.
store_45data_3:
    for n = 4 to 21

```

```

        data_buf3[n+18]=PKT_RX[n] ' Store the volt/freq data.
    next n
next_packet4:
    INTCON = $B0 ' Enable the interrupts.
    goto loop ' Wait for next packet.

' In case we have received a TOKEN we should test to see if this was from last station.
' ... if not from last station then we should wait for the next station's message.

case4:
    IF current_station = station_max THEN ack_it

    Lcdout $fe, 1 ' Clear LCD screen
    Lcdout "Expecting Next Station "

    flag = flag & %11110000 ' Reset the flag.
    goto wait ' Wait for packet from next station.

' A station has been nacked. Send NACK three times (jamming) to gain control of channel.

nack_it:
    GOSUB TX_NACK ' Send NACK 1st time.
    pause 10 ' ... Back off a bit.
    GOSUB TX_NACK ' Send NACK 2nd time.
    pause 10 ' ... Back off a bit.
    GOSUB TX_NACK ' Send NACK 3rd time.
    pause 10 ' ... Back off a bit.
    Address_Destination=current_station
    GOSUB TX_TOKEN ' Send token to station nacked.

    current_packet=$41 ' Current packet will now be an ACK.

    goto wait2 ' Wait for the packet.

ack_it: pause 50 ' Back off before sending ACK.
    GOSUB TX_ACK ' Send ACK.

' Now proceed to store this sample to the eeprom.

    Lcdout $fe, 1 ' Clear LCD screen
    Lcdout "Storing SAMPLE ", dec sample

    pointer = sample * 120 - 120

ee_prom:

    IF sample > 273 THEN ' Test if we need to switch eeproms.
        cont=cont2
        pointer = sample*120 - 32880
    else
        cont=cont1
        pointer = sample*120 - 120
    endif

    for n = 0 to 39
        temp2=data_buf1[n]
        I2CWRITE DPIN,CPIN,cont,pointer,[temp2] ' Store data from station 1 to eeprom.
        pause 10
        pointer = pointer + 1
    next n

    for n = 0 to 39
        temp2=data_buf2[n]
        I2CWRITE DPIN,CPIN,cont,pointer,[temp2] ' Store data from station 2 to eeprom.
        pause 10

```

```

        pointer = pointer + 1
    next n

    for n = 0 to 39

        temp2=data_buf3[n]
        I2CWRITE DPIN,CPIN,cont,pointer,[temp2] ' Store data from station 3 to eeprom.
        pause 10
        pointer = pointer + 1
    next n

    IF sample = sample_max THEN prompt          ' Test if we are done.

    ' If not done then next sample needs to be taken. This section generates an inter-sample
    ' ... wait which would in the actual power monitoring system application be a five minute
    ' ... wait implemented with a real-time clock. This is to be integrated at a future
    ' ... time. For now, software delay.

    temp = 10
sample_delay:
    if temp = 0 then go_on
        Lcdout $fe, 1                          ' Clear LCD screen
        Lcdout "Next Sample ",dec temp
        pause 200
        GOSUB TX_ACK                            ' Send ACK periodically, for hang-ups.
        temp = temp - 1
        goto sample_delay

    ' ... End of software delay next sample must be taken.

go_on:
    current_station=0                          ' Reset the variables.
    flag = flag_save                          ' Reset the original command.
    goto main_loop                            ' Take next sample.

    ' Base station is done with communication and data adquisition. Prompt user to upload.

prompt: Lcdout $fe, 1                          ' Clear LCD screen
        Lcdout "Upload to PC? y/n "

        sample = 1                            ' This is used to set eeaddr.

        HSERIN [temp2]                        ' Wait for y/n from user.

        if temp2 = "n" THEN wait_flag          ' No then prompt for next command.
        IF temp2 != "y" THEN prompt            ' Yes, upload number of samples taken.

        Lcdout $fe, 1                          ' Clear LCD screen
        Lcdout "Uploading...."

    ' The sample_max variable must be tested to see if we are uploading 288 samples. If we
    ' ... are this is done differently than if it were 3 samples or 10 samples to upload.
    ' ... 288 samples, 120 bytes per sample are stored in two eeproms therefore we must know
    ' ... when to switch eeproms. Also, the Visual basic data buffer only allows 32kbytes at
    ' ... once. The data must therefore be uploaded in packets to Visual Basic. Although
    ' ... real data consists of 34,560 bytes (120 bytes/samples * 288 samples) since the data
    ' ... must be sent tab delimited and with carriage returns, this easily doubles the
    ' ... amount of bytes sent to Visual Basic. Therefore the data is uploaded in four
    ' ... packets if 288 samples are uploaded otherwise only one packet is uploaded for 3 and
    ' ... 10 samples.

    IF sample_max = 288 Then
        goto packet1
    ELSE
        pointer=0                            ' Not 288 samples then either 3 or 10 samples to upload.
    continue:

```

```

cont = cont1

offset = 120 * sample - 120          ' Calculate the offset of the address.

for n = 0 to 80 step 40              ' Do three times.

    pointer = 0 + n                  ' Calculate the pointer.
    eeaddr = offset + pointer        ' Calculate the address of byte to read.

    I2CREAD DPIN,CPIN,cont,eeaddr,[sample_save] ' Read sample (two bytes).
    HSEROUT [dec sample_save,tab]      ' Send word to laptop.

    pointer = 2 + n                  ' Calculate the pointer.
    eeaddr = offset + pointer        ' Calculate the address of byte to read.

    I2CREAD DPIN,CPIN,cont,eeaddr,[temp]      ' Read station (one byte).
    HSEROUT [dec temp,tab]                  ' Send byte to laptop.

    pointer = 3 + n                  ' Calculate the pointer.
    eeaddr = offset + pointer        ' Calculate the address of byte to read.

    I2CREAD DPIN,CPIN,cont,eeaddr,[temp] 'Read command acked (reads one byte).
    HSEROUT [hex2 temp,tab]              ' Send byte to laptop.

    for pointer = (4+n) to (39+n) ' Read Power data (reads 36 bytes).
        eeaddr = offset + pointer
        I2CREAD DPIN,CPIN,cont,eeaddr,[temp]
        HSEROUT [hex2 temp,tab]
    next pointer
next n

HSEROUT [cr]          ' Send carriage return after uploading 120 byte line.

IF sample = sample_max THEN          ' Send "EOF" if done.
    HSEROUT["EOF"]
    goto finished3                  ' Done.
endif

sample = sample + 1                  ' Not done, increment sample.
goto continue                      ' Send next 120 byte line.
endif

' Number of samples to upload is 288, must now send in four packets.
'
' ... first packet = samples 1 to 92 (on eeprom 1)
' ... second packet = samples 93 to 182 (on eeprom 1)
' ... third packet = samples 183 to 273 (on eeprom 1)
' ... fourth packet = samples 274 to 288 (on eeprom 2)

packet1:
    pointer=0
    cont=cont1
    sample_min=1
    sample_max=92
    temp2=first
continue2:
    for sample = sample_min to sample_max

        offset = 120*sample-120

        for n = 0 to 80 step 40

            pointer = 0 + n
            eeaddr = offset + pointer

```

```

I2CREAD DPIN,CPIN,cont,eeaddr,[sample_save] ' Read/send sample.
HSEROUT [dec sample_save,tab]

pointer = 2 + n
eeaddr = offset + pointer

I2CREAD DPIN,CPIN,cont,eeaddr,[temp]      ' Read/send station.
HSEROUT [dec temp,tab]

pointer = 3 + n
eeaddr = offset + pointer

I2CREAD DPIN,CPIN,cont,eeaddr,[temp]      ' Read/send acked command.
HSEROUT [hex2 temp,tab]

for pointer = (4+n) to (39+n) ' Read/send Energy, Voltage, Frequency data.
    eeaddr = offset + pointer
    I2CREAD DPIN,CPIN,cont,eeaddr,[temp]
    HSEROUT [dec temp,tab]
next pointer

next n

HSEROUT[cr]                                ' Send carriage return after uploading 120 byte line.

next sample

' Done uploading a packet to laptop, send "EOF", test which packet has been sent, change
parameters
' ...

finished1:
    HSEROUT["EOF"]                          ' Done with packet so send "EOF", test packet sent.

    if temp2 = first then
        pointer=0
        temp2 = second
        cont = cont1
        sample_max=182
        sample_min=93
        goto prompt2
    endif
    if temp2 = second then
        pointer=0
        temp2 = third
        cont = cont1
        sample_max=273
        sample_min=183
        goto prompt2
    endif
    if temp2 = third then
        pointer=0
        temp2 = fourth
        cont = cont2
        sample_max=15
        sample_min=1
        goto prompt2
    endif
    if temp2 = fourth then finished3          'If fourth was sent then done.

' Not finished must upload another packet, base controller waits for ok, Visual Basic
' ... will then append to the file data.txt.

prompt2:Lcdout $fe, 1                        ' Clear LCD screen
    Lcdout "Click OK"

```

```

HSERIN[temp]
IF temp != "g" THEN prompt2

Lcdout $fe, 1          ' Clear LCD screen
Lcdout "Uploading....."

goto continue2

' Done uploading, now will wait for next command.

finished3:
  Lcdout $fe, 1          ' Clear LCD screen
  Lcdout "Upload Complete "
  Pause 2000

wait_flag:
  Lcdout $fe, 1          ' Clear LCD screen
  Lcdout "Waiting for command"

  HSERIN [temp]

  IF temp = "A" THEN do_1
  IF temp = "B" THEN do_2
  IF temp = "C" THEN do_3

  Goto wait_flag

  Lcdout $fe, 1          ' Clear LCD screen
  Lcdout "Uploading Data to PC"
  pause 2000

  IF temp = "D" THEN do_4
  IF temp = "E" THEN do_5
  IF temp = "F" THEN do_6

  goto wait_flag

do_1:  flag_save=RESET
       flag=RESET
       sample_max=3
       Lcdout $fe, 1          ' Clear LCD screen
       Lcdout "RESETTING SYSTEM"

       pause 2000
       goto START

do_2:  flag_save=RECHECK
       flag=RECHECK
       sample_max=10
       Lcdout $fe, 1          ' Clear LCD screen
       Lcdout "RECHECKING UNITS"

       pause 2000
       goto call_units

do_3:  flag_save=TDM_DATA
       flag=TDM_DATA
       sample_max=288
       Lcdout $fe, 1          ' Clear LCD screen
       Lcdout "TDM_DATA"

       pause 2000
       goto call_units

do_4:  sample_max=288

```



```
        goto prompt
do_5:   sample_max=3
        goto prompt
do_6:   sample_max=10
        goto prompt
end
'- END MAIN -----
```

VITA

The author was born in Monterrey, Nuevo León México on October 21, 1973.

He received the Bachelor of Science in Electrical Engineering degree in December 1997 from Texas Tech University and entered into the Electrical Engineering graduate program at Texas Tech University and Instituto Tecnológico y de Estudios Superiores de Monterrey (I.T.E.S.M. Campus Monterrey, N.L. México) in January 1998 and August 1998 respectively.

He will receive the Master of Science in Electrical Engineering degree from Texas Tech University in August 1999.

His areas of interest included electronic design, instrumentation, and the efficient use of electrical energy. He is a member of the Institute of Electrical and Electronics Engineers and alumnus of the Alpha Tau Omega fraternity Texas Zeta Eta.

Permanent Address: 8412 Avenue X
Lubbock, Texas 79423
United States of America

Electronic Mail: pineda@ieee.org