MICROCONTROLLER – BASED MULTIPORT COMMUNICATION SYSTEM FOR DIGITAL ELECTRICITY METERS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

FIRAT BEŞTEPE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2004

Approval of the Graduate School of Natural and Applied Sciences.

_____

Prof. Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. İsmet ERKMEN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science

_____

Prof. Dr. Hasan GÜRAN
Supervisor

Examining Committee Members

Prof. Dr. Osman SEVAIOĞLU	(METU,EE)	_____
Prof. Dr. Hasan GÜRAN	(METU,EE)	_____
Assoc. Prof Dr. Gözde BOZDAĞI	(METU,EE)	_____
Asst. Prof. Dr. Cüneyt BAZLAMAÇCI	(METU,EE)	_____
Graduate Engineer Turgay ŞENDUR	(KOSGEB)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Fırat BEŞTEPE

Signature :

ABSTRACT

MICROCONTROLLER-BASED MULTIPORT COMMUNICATION SYSTEM FOR DIGITAL ELECTRICITY METERS

BEŞTEPE, Fırat
Ms., Department of Electrical and Electronics Engineering
Supervisor : Prof. Dr. Hasan GÜRAN

December 2004, 116 pages

This thesis explains the design of a microcontroller-based device, which provides an efficient and practical alternative for the remote reading of digital electricity meters over Public Switch Telephone Network (PSTN). As an alternative application, a system is constructed providing file transfer capability to the PC connected to the port of implemented device in addition to remote reading of digital electricity meters. This thesis also provides detailed explanations about the basics of serial asynchronous communication over modem for PICs (peripheral interface controllers) together with description of each component included by the constructed system, which can be used in energy metering sector commonly.

Keywords: PIC, Microcontroller, Serial Communication

ÖZ

ELEKTRONİK ELEKTRİK SAYAÇLARI İÇİN MİKRODENETLEYİCİ TEMELLİ ÇOKLU PORT HABERLEŞME SİSTEMİ

BEŞTEPE, Fırat
Yüksek Lisans, Elektrik-Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Hasan GÜRAN

Aralık 2004, 116 sayfa

Bu tez elektronik elektrik sayaçlarının, Genel Telefon Ağı (PSTN) üzerinden uzaktan okunması amacına yönelik etkili ve kullanışlı bir alternatif sunan, mikrodenetleyici temelli bir cihazın dizaynını açıklamaktadır. Alternatif bir uygulama olarak da, elektronik elektrik sayaçlarının uzaktan okunmasının yanısıra, dizayn edilen cihazın portlarından birine bağlanan PC ile dosya transferinin de yapılabildiği bir sistem oluşturulmuştur. Ayrıca enerji ölçüm sektöründe yaygın olarak kullanılabilecek bu sistemin tüm temel unsurlarının izahı ile birlikte, çevresel arayüz denetleyicileri (PICs) için modem üzerinden seri ve eşzamanlı olmayan haberleşme temelleri konusunda ayrıntılı bilgi vermektedir.

Anahtar Kelimeler: PIC, Mikrodenetleyici, Seri Haberleşme

ACKNOWLEDGEMENTS

The author wishes to express his deepest gratitude to his supervisor Prof. Dr. Hasan GÜRAN for his guidance, advices, criticism, and encouragements and insight throughout the study.

The technical assistances of Tuncay KANDEMİR, Ali Erkan ERKOL and Kadir ERDOĞAN are gratefully acknowledged.

This study was supported by Aktif Enerji Ltd. Şti., ANKARA.

TABLE OF CONTENTS

CHAPTER 4)- SERIAL COMMUNICATION WITH MICROCONTROLLERS

CHAPTER 5)- DESCRIPTION OF IMPLEMENTED SYSTEM

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

GENERAL DESCRIPTION OF THE SYSTEM

1.1 Introduction

Electricity meters functioned according to electromechanical principles until 1980's. Therefore, desired measures in accuracy, reliability, lifetime and price of metering solutions could not be attained. Since that time, electronics has made a remarkable development. Today electricity meters are mass-produced using the most modern microelectronics. This digital processing technology is not only providing better solutions than the classical measurement technology but also is able to operate with auxiliary devices (e.g. modem). Also these high technology meters enable the consumers to construct very feasible remote reading infrastructures via their advanced communication facilities.

Within this scope, in this thesis, an implemented device will be presented and elucidated as a solution for simplifying the remote reading infrastructures of digital electricity meters by means of reducing the number of components used in the system.

1.2 Common Remote Reading Infrastructure

Figure – 1.1 shows the brief model of a system, which is used very commonly in the metering sector. The system mainly consists of a PC including software for remote reading purpose, a modem at PC side, a telephone line dedicated to the PC side modem, modems for each meter at remote side and dedicated telephone lines for each modem. This kind of infrastructure has been widely used in metering systems, especially in high power plants, multi lined power plants and auto-producers. The main difficulty in this kind of systems is

the necessity of modem and telephone line for each meter. Using a small but sufficient telephone exchange instead of using one dedicated line for each meter can eliminate this hardship a bit. But using a modem for each meter is an unavoidable necessity for these kinds of systems.

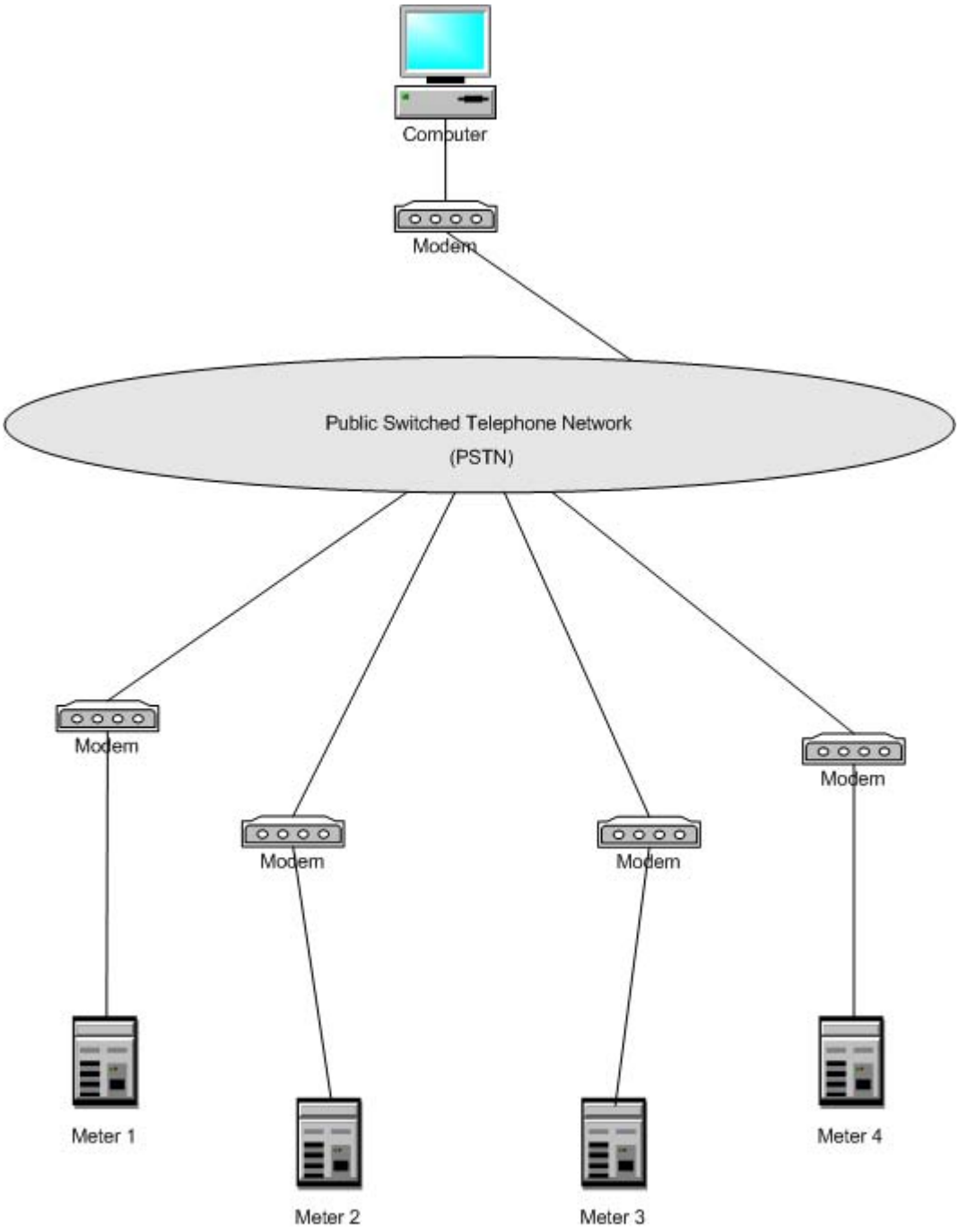


Figure – 1.1 Remote Reading Infrastructure used in metering solutions

1.3 Developed Remote Reading Infrastructure

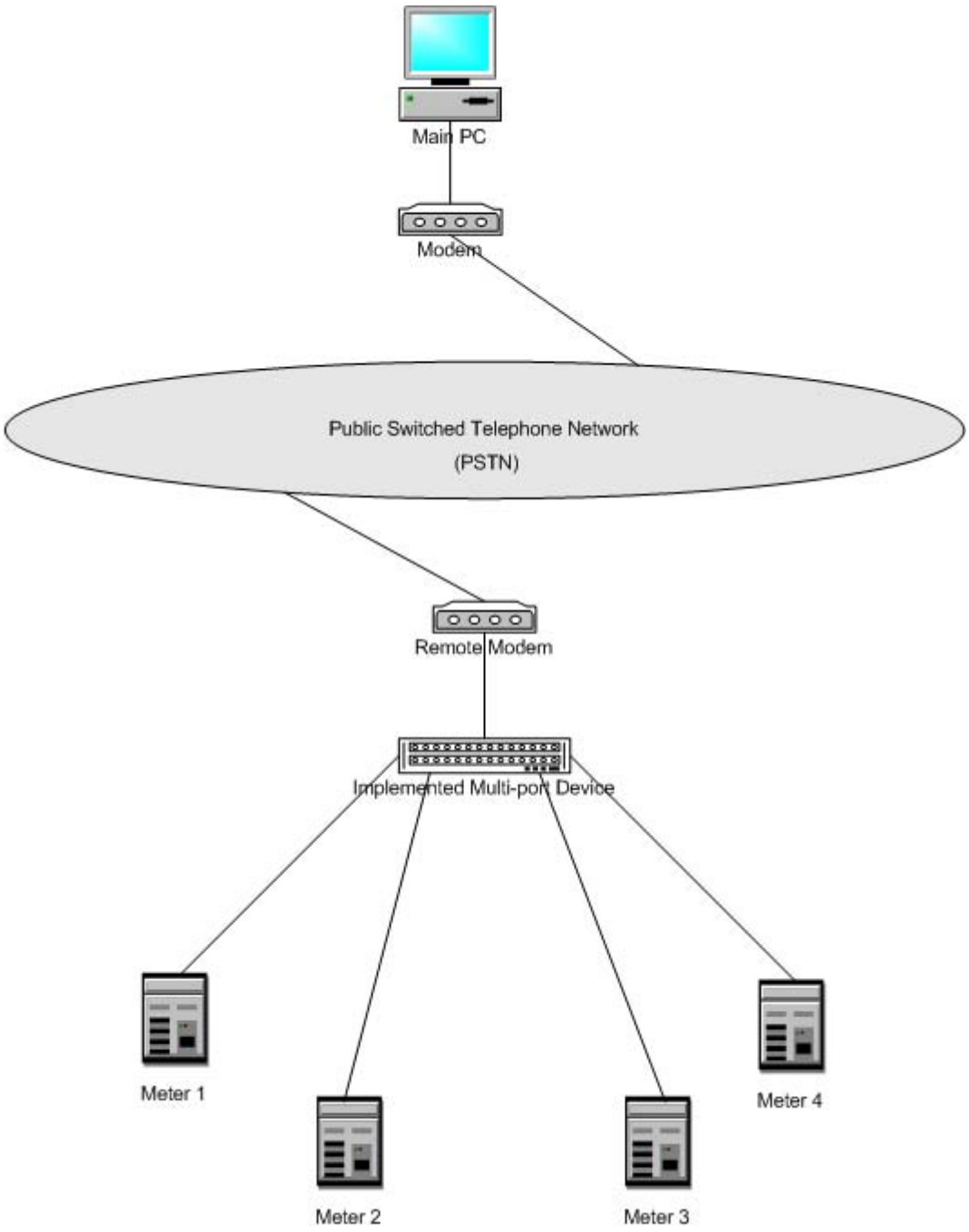
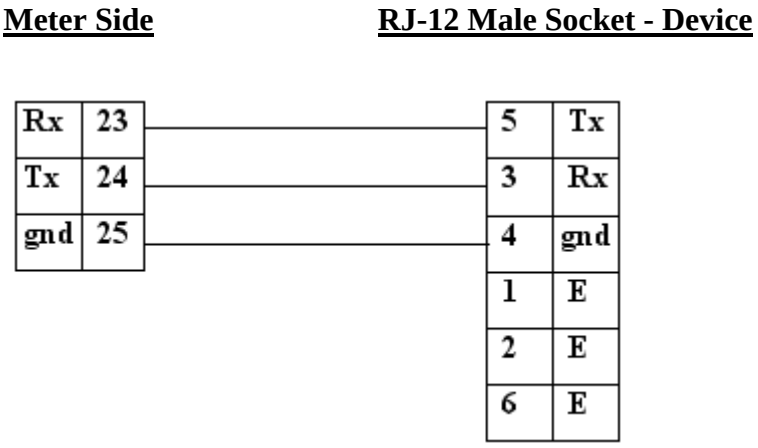


Figure – 1.2 More feasible remote reading solution with Implemented Multi-port Device

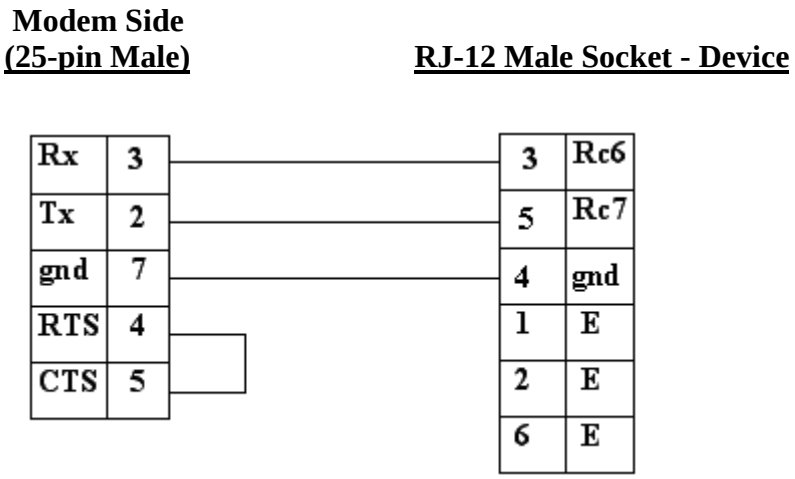
Considering the previous mentioned issues, Figure – 1.2 shows the same system in Figure – 1.1 but materialized by the implemented multi-port device, which is the subject of this thesis. As it can be seen clearly in Figure – 1.2, there is no need to have modems and telephone lines for each meter any more. One telephone line and one modem connected to the multi-port device is now enough to communicate with more than one meter.

In a system shown in Figure – 1.2, the PC establishes the communication with the multi-port device over PSTN. The direct connection between PC and modem is done by original modem cable, which has 25-pin female connector for the end of modem side, and 9-pin female connector at the other end for the serial port of PC. In addition to that, two different types connection cables are also needed in the system; one of which is to be used for connection between remote modem and the multi-port device, and the other of which to be used for connection between multi-port device and each meter. The pin specifications of these cables are shown in Figure – 1.3 and 1.4.



E: empty, not used

Figure – 1.3 Meter-Device connection



E: empty, not used

Figure – 1.4 Modem-Device connection

Using the software running on the PC, which will be described in Chapter 5, the user enters the telephone number of the remote modem for dialing. After the communication is established between two ends, the user chooses only the port of the implemented multi-port device to which the meter is connected. Right after the reception of the port number of the meter by the multi-port device, the microcontroller in the device sends the string to the specified port according to the Flag protocol, which arranges the communication rules of the meter with the peripherals. If the meter is password protected, it will reply with a specific string that indicates a password requirement or if there is no meter connected to this port, time-out occurs and the multi-port device sends a related message to the user station. A similar message is also sent to LCD on the device for local recognition by the users.

If there is a meter connected to a specified port and the meter is not password protected, it will reply in 20 msec after receiving a request according to the protocol. If the request is standard readout request of the meter, the meter will send all the standard readout data character by character to the multi-port device and the multi-port device sends all the characters to the PC as soon as it receives them. During the reception process, an indication message demonstrating a continuing data reception appears on the LCD. After data reception is completed,

the multi-port device sends another message indicating the completed readout process to both LCD and user station and then starts to wait for the next request.

1.4 Alternative Application of Multi-port Device

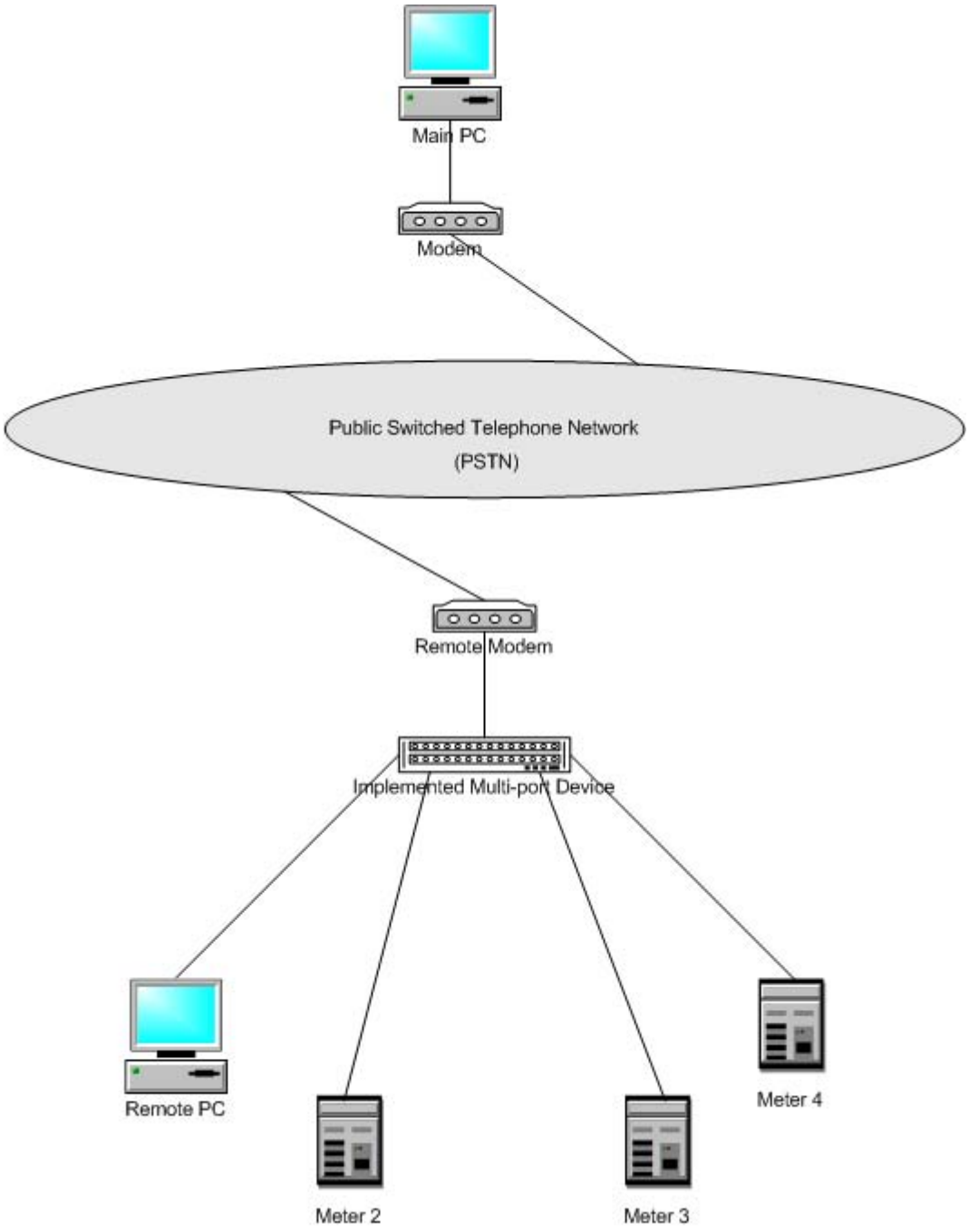
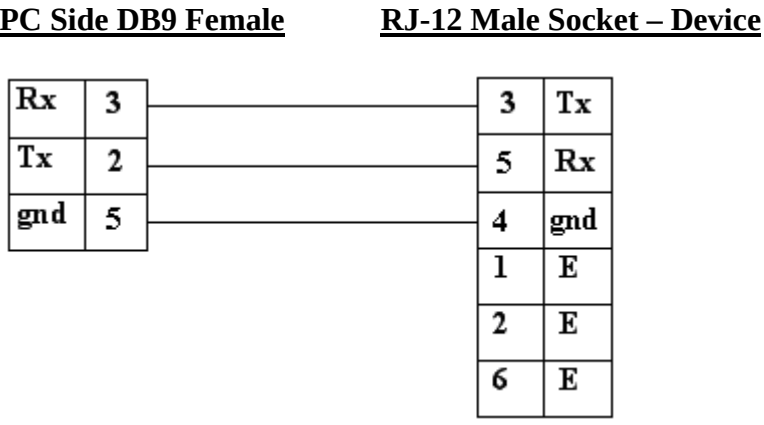


Figure – 1.5 Alternative Application of Implemented Multi-port Device

As shown in Figure – 1.5, the implemented multi-port device has 4 ports (expandable to 8) apart from the port providing the connection to the remote modem. Four digital electricity meters can be connected to these ports as well as one PC and three digital electricity meters. As shown in Figure – 1.5 the device connected to the port 1 may be a PC or a digital electricity meter according to the application. The application with a PC connected to port 1 provides the users to carry out two-way file transfer between the main PC and the remote PC. This two-way file transfer process is performed according to Xmodem file transfer protocol. The details of Xmodem protocol are explained in Appendix D. The pin specifications of the connection cable between the multi-port device and remote PC is as shown in Figure – 1.6.



E: empty, not used

Figure – 1.6 Remote PC-Device connection

1.5 Objective of This Thesis

This thesis aims to reduce the number of components used in the remote reading infrastructures of digital electricity meters and thus simplify the system together with a cost reduction.

The topic of meter remote reading has become very popular in last decade in the world and almost all producers have integrated communication interfaces to their meters, which provide the capability of remote access. Correspondingly,

several companies, such as CallDirect Telecommunications [Ref. 19], Adaptive Networks [Ref. 20], Enermet [Ref. 21], PERAX Remote Automated Management Systems [Ref. 22], started to carry on business related with constructing remote reading infrastructures for meters.

In order to construct a professional remote reading infrastructure, several companies with different proficiencies should work together. For example, The Integral Energy is the second largest state-owned energy corporation in NSW in Australia and their meter remote reading infrastructure has been constructed based on GSM technology by a group being composed of CallDirect Telecommunications, Nokia and a software development company. Another example is Clark Public Utilities, which is a customer-owned utility providing electricity, water and wastewater service in Clark County, Washington and they have more 164.000 electricity customer and 26.000 water customer. The meters of customers are connected to the remote reading system by radio transmitter or telephone line according to meter type and feasibleness.

The remote reading infrastructures illustrated in Figure – 1.1 and 1.2 may sometimes need many more meters as mentioned previous examples, which will also mean more modems and more inter-phone lines or dedicated lines. At most of the sites where a lot of modems will be installed together with inter-phone or dedicated lines, various problems might be come across. For instance, enough space for modems could not be found on the panel where the meters will be put. Additionally, operating and maintaining too many devices is not very simple.

It is obvious that the smaller space a system requires, the more adaptable the system is. Therefore reducing the number of devices used in a system is an engineering matter, which provides several advantages including maintenance ease.

As an example, when we consider a remote reading infrastructure having 32 meters, an implemented 4-port multi-port device will be able to decrease the number of modems and phone lines to 8. This type of a reduction in the system will be more remarkable if the number of ports of implemented multi-port device is expanded to 8.

Moreover, if the implemented multi-port device is connected to a PC together with digital electricity meters as shown in Figure – 1.5, this application will provide two-way file transfer between the main PC and the remote PC. The readout data of a meter contains all the information that the meter has, such as all energy registers according to tariff measurements and their old values, all mean power values, instantaneous date and time, meter serial number, error code giving the current situation of the meter, etc. With regard to that, the readout data of the meters or some evaluation reports of this readout data may be necessary to be shared between the remote ends. Under the circumstances this two-way file transfer opportunity can simplify this process fairly.

The system constructed in the scope of this thesis will be explained in next chapters by giving the details about operating principles of used components and devices, designing of multi-port device and code implementation.

First of all, the basics of serial communication and RS 232 standard, which constitute the base of interface between multi-port device and peripherals such as digital electricity meters and computers, will be explained in Chapter 2. After that the PIC 16F877, which is the microcontroller used in the multi-port device, will be explained in Chapter 3. Then the fundamentals of serial communication with the microcontrollers, which is the main issue performed by the multi-port device, will be explained in details in Chapter 4. Finally, the developed remote access software, modems, implemented multi-port device and digital electricity meters, which are the main parts of the system, will be described in Chapter 5.

The source code of developed remote access software written by using Delphi 7.0 is presented in Appendix A and the code of PIC 16F877 microcontroller, which is written by using PicBasic Pro, is presented in Appendix B. In Appendix C the user manual of multi-port device is given. The Appendix D includes the details of Xmodem, which is the protocol used for the file transfer between main PC and remote PC. The final appendix gives the serial communication modes in PicBasic Pro, which is explained in Chapter 4.

CHAPTER 2

DATA COMMUNICATION INTERFACE

2.1 Introduction

SCI is an abbreviation for Serial Communication Interface, as a special subsystem, and it exists on most microcontrollers. In the early 1960s, a standards committee, today known as the Electronic Industries Association, developed a common interface standard for data communications equipment. At that time, data communications was thought to mean digital data exchange between a centrally located mainframe computer and a remote computer terminal, or possibly between two terminals without a computer involved. These devices were linked by telephone voice lines, and consequently required a modem at each end for signal translation. While simple in concept, the chances for data errors that could occur when transmitting data through an analog channel are considerably high and therefore transmission over phone lines require a relatively complex design. It was thought that a standard was needed first to ensure reliable communication, and second to enable the interconnection of equipment produced by different manufacturers. From these ideas, the RS232 standard, which is the short of Recommended Standard-232, was born. It specified signal voltages, signal timing, signal function, a protocol for information exchange, and mechanical connectors.

Over the 40 years since this standard was developed, the Electronic Industries Association published three modifications, the most recent being the EIA232E standard introduced in 1991 [Ref. 12]. Besides changing the name from RS232 to EIA232, some signal lines were renamed and various new ones were defined, including a shield conductor.

2.2 EIA232 Standard

If the full EIA232 standard is implemented, the equipment at the far end of the connection, which is named as DTE device (Data Terminal Equipment, usually a computer or terminal), has a male DB25 connector, and utilizes 22 of the 25 available pins for signals or ground. Equipment at the near end of the connection (the telephone line interface), which is named as DCE device (Data Circuit-terminating Equipment, usually a modem), has a female DB25 connector, and utilizes the same 22 available pins for signals and ground. The cable linking DTE and DCE devices is a parallel straight-through cable. If all devices exactly followed this standard, all cables would be identical, and there would be no chance that an incorrectly wired cable could be used. Figure – 2.1 shows the orientation and connector types for DTE and DCE devices:

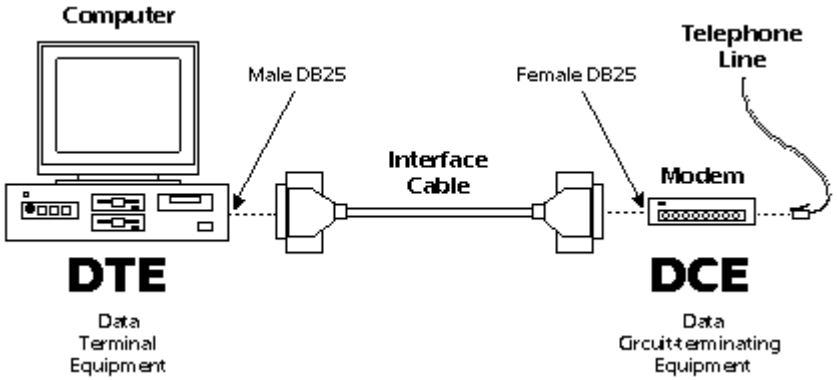


Figure – 2.1 DTE and DCE

Figure – 2.1 also shows EIA232 communication function and connector types for a personal computer and modem. DCE devices are sometimes called “Data Communications Equipment” instead of Data Circuit-terminating Equipment [Ref. 3].

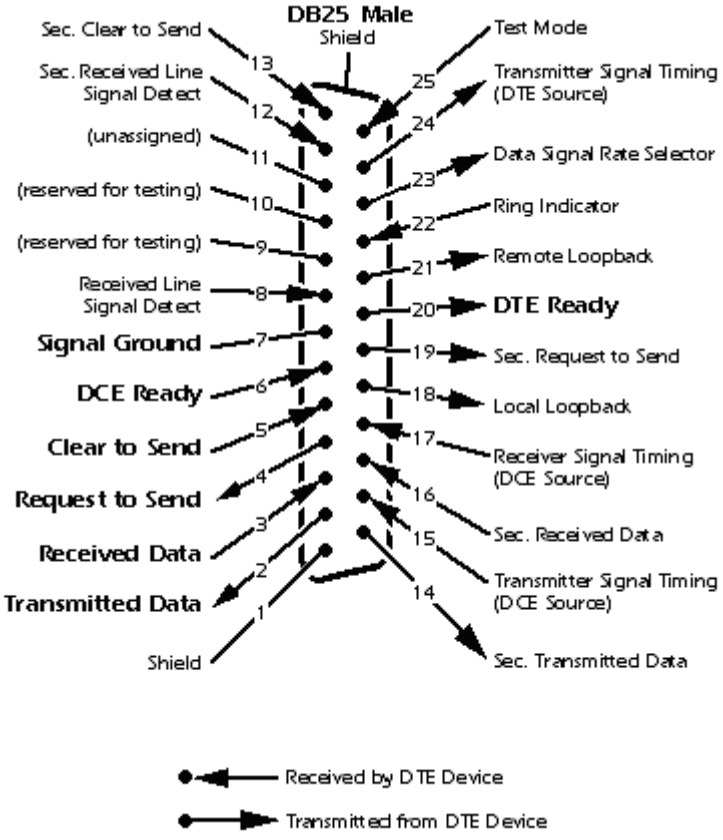


Figure – 2.2 DB25 Male DTE device connector

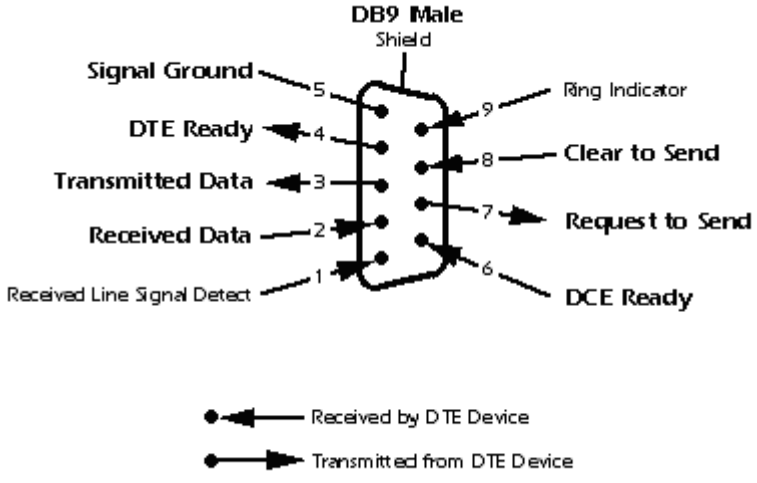


Figure – 2.3 DB9 Male DTE device connector

Figure – 2.2 and Figure – 2.3 is the full EIA232 signal definition for the DTE device (usually the PC). The most commonly used signals are shown in bold.

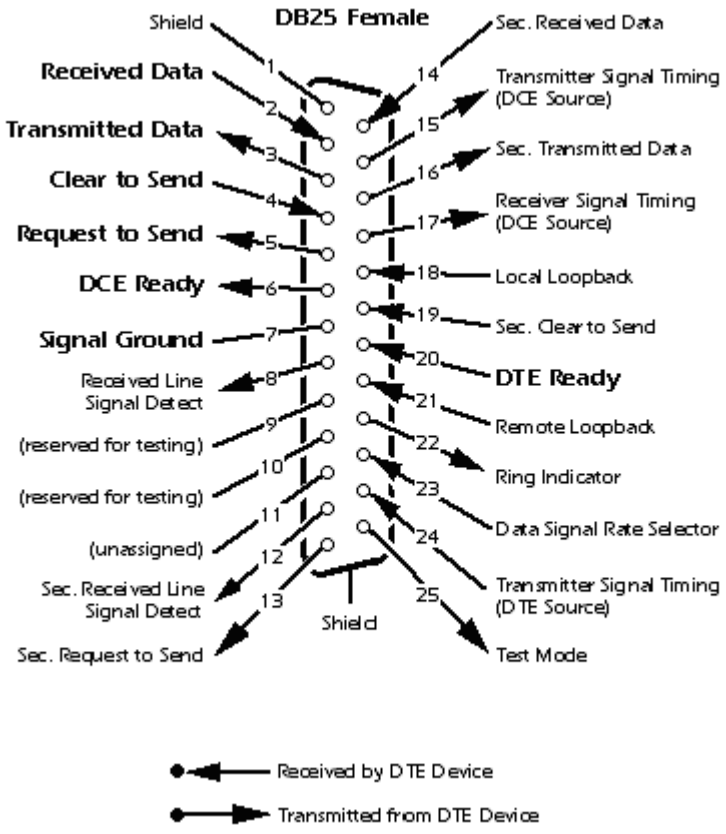


Figure – 2.4 DB25 Female DCE device connector

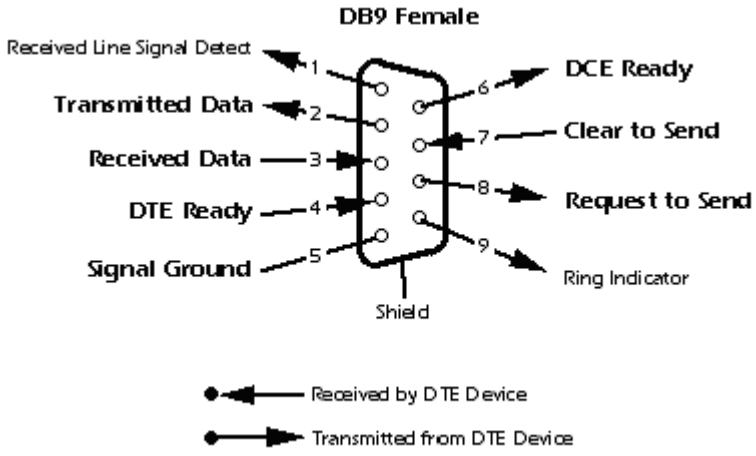


Figure – 2.5 DB9 Female DCE device connector

Figure – 2.4 and Figure – 2.5 show the full EIA232 signal definition for the DCE device (usually the modem). The most commonly used signals are shown in bold.

Many of the 22 signal lines in the EIA232 standard pertain to connections where the DCE device is a modem, and then are used only when the software protocol employs them. For any DCE device that is not a modem, or when two DTE devices are directly linked, fewer signal lines are necessary.

It can be noticed that in the pin-out drawings there is a secondary channel, which includes a duplicate set of flow-control signals. This secondary channel provides the management of the remote modem, enables retransmission when a parity error is detected and also provides other control functions. Furthermore, it enables baud rates to be changed during the flow. This secondary channel, when used, is typically set to operate at a very low baud rate in comparison with the primary channel to ensure reliability in the control path. In addition, it may operate as either a simplex, half-duplex, or full-duplex channel, depending on the capabilities of the modem [Ref. 3].

Transmitter and receiver timing signals (pins 15, 17, and 24) are used only for a synchronous transmission protocol. For the standard asynchronous 8-bit protocol, external timing signals are unnecessary.

Signal names that imply a direction, such as Transmit Data and Receive Data, are named according to DTE device. If the EIA232 standard were strictly followed, these signals would have the same name for the same pin number on the DCE side as well. Unfortunately, most engineers do not do this in practice, probably because no one pays attention to which side is DTE and which is DCE. As a result, direction-sensitive signal names are changed at the DCE side to reflect their drive direction at DCE. Figure – 2.6 gives the conventional usage of signal names:

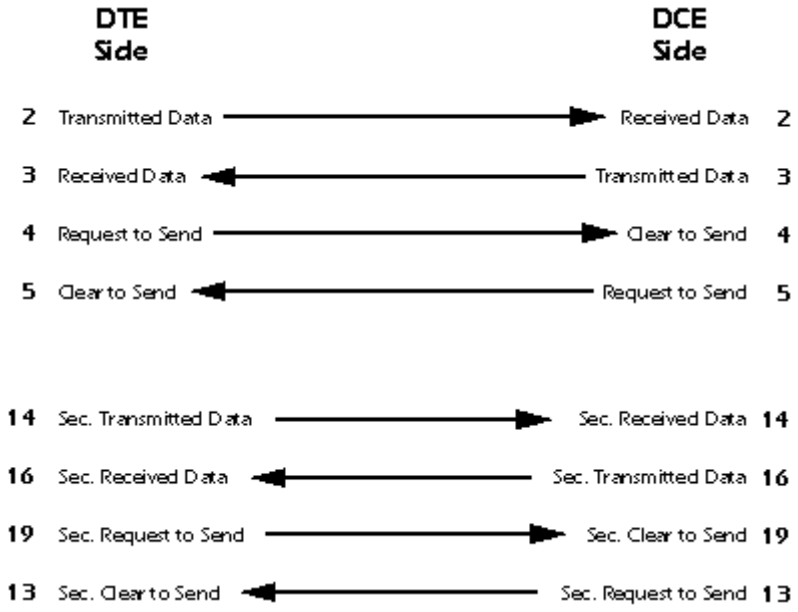


Figure – 2.6 Conventional usage of signal names

2.3 EIA232 Signal Functions

Signal functions in the EIA232 standard can be divided into six subcategories. These categories are summarized as below. After that each signal category is described more comprehensively.

1 - Signal ground and shield.

2 - Primary communications channel: This is used for data interchange and includes flow control signals.

3 - Secondary communications channel: If implemented, this is used for control of the remote modem, requests for retransmission when errors occur.

4 - Modem status and control signals: These signals indicate modem status and provide intermediate checkpoints as the telephone voice channel is established.

5 - Transmitter and receiver timing signals: If a synchronous protocol is used, these signals provide timing information for the transmitter and receiver, which may operate at different baud rates.

6 - Channel test signals: Before data is interchanged, the channel may be tested for its integrity, and the baud rate automatically adjusted to the maximum rate that the channel could support.

2.3.1 Signal Ground and Shield

Pin 7, pin 1, and the shell are included in this category. Cables provide separate paths for each, but internal wiring often connects pin 1 and the cable shell/shield to signal ground on pin 7.

Pin 7 – Ground: All signals are referenced to a common ground, as defined by the voltage on pin 7. This conductor may or may not be connected to protective ground inside the DCE device. The existence of a defined ground potential within the cable makes the EIA232 standard different from a balanced differential voltage standard, such as EIA530, which provides far greater noise immunity.

2.3.2 Primary Communications Channel

Pin 2 - Transmitted Data (TxD): This signal is active when data is transmitted from the DTE device to the DCE device. When no data is transmitted, the signal is held in the mark condition (logic 1, negative voltage).

It should be noted that pin 2 on the DCE device is commonly labeled “Received Data”, although by the EIA232 standard it should still be called “Transmitted Data” because the data is thought to be destined for a remote DTE device.

Pin 3 - Received Data (RxD): This signal is active when the DTE device receives data from the DCE device. When no data is transmitted, the signal is held in the mark condition (logic 1, negative voltage).

It should be noted that pin 3 on the DCE device is commonly labeled “Transmitted Data”, although by the EIA232 standard it should still be called “Received Data” because the data is thought to arrive from a remote DTE device.

Pin 4 - Request to Send (RTS): This signal is changed to space condition (logic 0, positive voltage) to prepare the DCE device for accepting transmitted data from the DTE device. Such preparation might include enabling the receive circuits, or setting up the channel direction in half-duplex applications. When the DCE is ready, it acknowledges by changing Clear To Send to positive voltage.

It should be noted that pin 4 on the DCE device is commonly labeled Clear to Send, although by the EIA232 standard it should still be called Request to Send because the request is thought to be destined for a remote DTE device.

Pin 5 - Clear to Send (CTS): This signal is changed to space condition (logic 0, positive voltage) by the DCE device to inform the DTE device that transmission may begin. RTS and CTS are commonly used as handshaking signals to moderate the flow of data into the DCE device.

It should be noted that pin 5 on the DCE device is commonly labeled "Request to Send", although by the EIA232 standard it should still be called Clear to Send because the signal is thought to originate from a remote DTE device.

2.3.3 Secondary Communications Channel

Pin 14 - Secondary Transmitted Data (STxD)

Pin 16 - Secondary Received Data (SRxD)

Pin 19 - Secondary Request to Send (SRTS)

Pin 13 - Secondary Clear to Send (SCTS)

These signals are equivalent to the corresponding signals in the primary communications channel. The baud rate, however, is typically much slower in the secondary channel for increased reliability.

2.3.4 Modem Status and Control Signals

Pin 6 - DCE Ready (DSR): When originating from a modem, this signal is changed to space condition (logic 0, positive voltage) when the following three conditions are all satisfied:

1 - The modem is connected to an active telephone line that is “off-hook”,

2 - The modem is in data mode, not voice or dialing mode,

3 - The modem has completed dialing or call setup functions and it is generating an answer tone.

If the line goes “off-hook”, a fault condition is detected or a voice connection is established, the DCE Ready signal is changed to mark condition (logic 1, negative voltage).

If DCE Ready originates from a device other than a modem, it may be changed to space condition to indicate that the device is turned on and ready to function, or it may not be used at all. If unused, DCE Ready should be permanently held in space condition (logic 0, positive voltage) within the DCE device or by use of a self-connect jumper in the cable. Alternatively, the DTE device may be programmed to ignore this signal.

Pin 20 - DTE Ready (DTR): This signal is changed to space condition (logic 0, positive voltage) by the DTE device when it wishes to open a communication channel. If the DCE device is a modem, changing DTE Ready to space condition prepares the modem to be connected to the telephone circuit, and once connected, maintains the connection. When DTE Ready is changed to mark condition (logic 1, negative voltage), the modem is switched to "on-hook" to terminate the connection.

If the DCE device is not a modem, it may require DTE Ready to be changed to space condition before the device can be used, or it may ignore DTE Ready altogether. If the DCE device (for example, a printer) is not responding, confirm that DTE Ready is changed to space condition.

Pin 8 - Received Line Signal Detector (CD): It is also called carrier detect. This signal is relevant when the DCE device is a modem. It is changed to space

condition (logic 0, positive voltage) by the modem when the telephone line is “off-hook”, a connection has been established, and an answer tone is being received from the remote modem. The signal is changed to mark condition when no answer tone is being received, or when the answer tone is of inadequate quality to meet the local modem's requirements (perhaps due to a noisy channel).

Pin 12 - Secondary Received Line Signal Detector (SCD): This signal is equivalent to the Received Line Signal Detector (pin 8), but refers to the secondary channel.

Pin 22 - Ring Indicator (RI): This signal is relevant when the DCE device is a modem, and is changed to space condition (logic 0, positive voltage) when a ringing signal is being received from the telephone line. The time of holding in space condition for this signal will approximately equal the duration of the ring signal, and it will be held in mark condition between rings or when no ringing is present.

Pin 23 - Data Signal Rate Selector: This signal may originate either in the DTE or DCE devices (but not both), and is used to select one of two prearranged baud rates. The space condition (logic 0, positive voltage) selects the higher baud rate.

2.3.5 Transmitter and Receiver Timing Signals

Pin 15 - Transmitter Signal Element Timing (TC): It is also called Transmitter Clock. This signal is relevant only when the DCE device is a modem and is operating with a synchronous protocol. The modem generates this clock signal to control exactly the rate at which data is sent on Transmitted Data (pin 2) from the DTE device to the DCE device. The logic 1 to logic 0 (negative voltage to positive voltage) transition on this line causes a corresponding transition to the next data element on the Transmitted Data line. The modem generates this signal continuously, except when it is performing internal diagnostic functions.

Pin 17 - Receiver Signal Element Timing (RC)**:** It is also called Receiver Clock. This signal is similar to TC described above, except that it provides timing information for the DTE receiver.

Pin 24 - Transmitter Signal Element Timing (ETC): It is also called External Transmitter Clock. Timing signals are provided by the DTE device for use by a modem. This signal is used only when TC and RC (pins 15 and 17) are not in use. The logic 1 to logic 0 transition (negative voltage to positive voltage) indicates the time-center of the data element. Timing signals will be provided whenever the DTE is turned on, regardless of other signal conditions.

2.3.6 Channel Test Signals

Pin 18 - Local Loopback (LL): This signal is generated by the DTE device and is used to place the modem into a test state. When local loopback is changed to space condition (logic 0, positive voltage), the modem redirects its modulated output signal, which is normally fed into the telephone line, back into its receive circuitry. This enables data generated by the DTE to be echoed back through the local modem to check the condition of the modem circuitry. The modem changes its Test Mode signal on Pin 25 to space condition in order to acknowledge that it has been placed in local loopback condition.

Pin 21 - Remote Loopback (RL): This signal is generated by the DTE device and is used to place the remote modem into a test state. When remote loopback is changed to space condition (logic 0, positive voltage), the remote modem redirects its received data back to its transmitted data input, thereby re-modulating the received data and returning it to its source. When the DTE initiates such a test, transmitted data is passed through the local modem, the telephone line, the remote modem, and back, to exercise the channel and confirm its integrity. The remote modem signals the local modem to change pin 25 to space condition when the remote loopback test is in progress.

Pin 25 - Test Mode (TM): This signal is relevant only when the DCE device is a modem. When changed to space condition (logic 0, positive voltage), it indicates that the modem is in a local loopback or remote loopback condition. Other internal self-test conditions may also cause Test Mode to be changed to space condition.

2.4 Electrical Standards

The EIA232 standard uses negative, bipolar logic in which a negative voltage signal represents logic 1, and positive voltage represents logic 0. This probably originated with the pre-RS232 current loop standard used in 1950s teletype machines in which a flowing current (and hence a low voltage) represents logic 1. Be aware that the negative logic assignment of EIA232 is the reverse of that found in most modern digital circuit designs.

The EIA232 standard includes a common ground reference on Pin 7, and is frequently joined to Pin 1 and a circular shield that surrounds all 25-cable conductors. Data, timing, and control signal voltages are measured with respect to this common ground. EIA232 cannot be used in applications where the equipment on opposite ends of the connection must be electrically isolated. Optical isolators may be used to achieve ground isolation, however, this option is not mentioned or included in the EIA232 specification [Ref. 3].

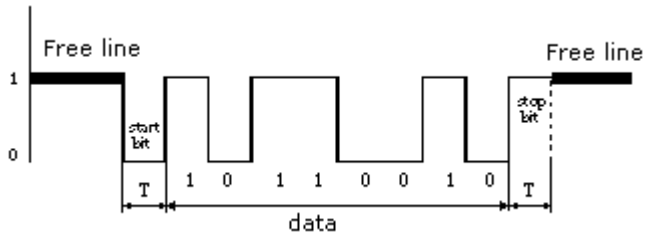


Figure –2.7 Non-Return to Zero format

Standard NRZ (Non Return to Zero) format is also known as 8 (9)-N-1 (8 or 9 data bits, with/without parity bit and with one stop bit). Free line is defined as

the status of logic 1. Start of transmission - Start Bit, has the status of logic 0. The data bits follow the start bit (the first bit is the low significant bit-LSB), and after the bits we place the Stop Bit as logic 1. If there is no more data coming then the receive line will stay in idle state (logic 1). There is another signal called as Break Signal. This is when the data line is held in a logic 0 state for a time long enough to send an entire word. Therefore if the line is not put back into an idle state, then the receiving end will interpret this as a break signal.

The duration of the stop bit “T” depends on the transmission rate and is adjusted according to the needs of the transmission. For the transmission speed of 9600 baud, T is 104 μS.

CHAPTER 3

PIC 16F877 MICROCONTROLLER

3.1 General Overview to Microcontrollers

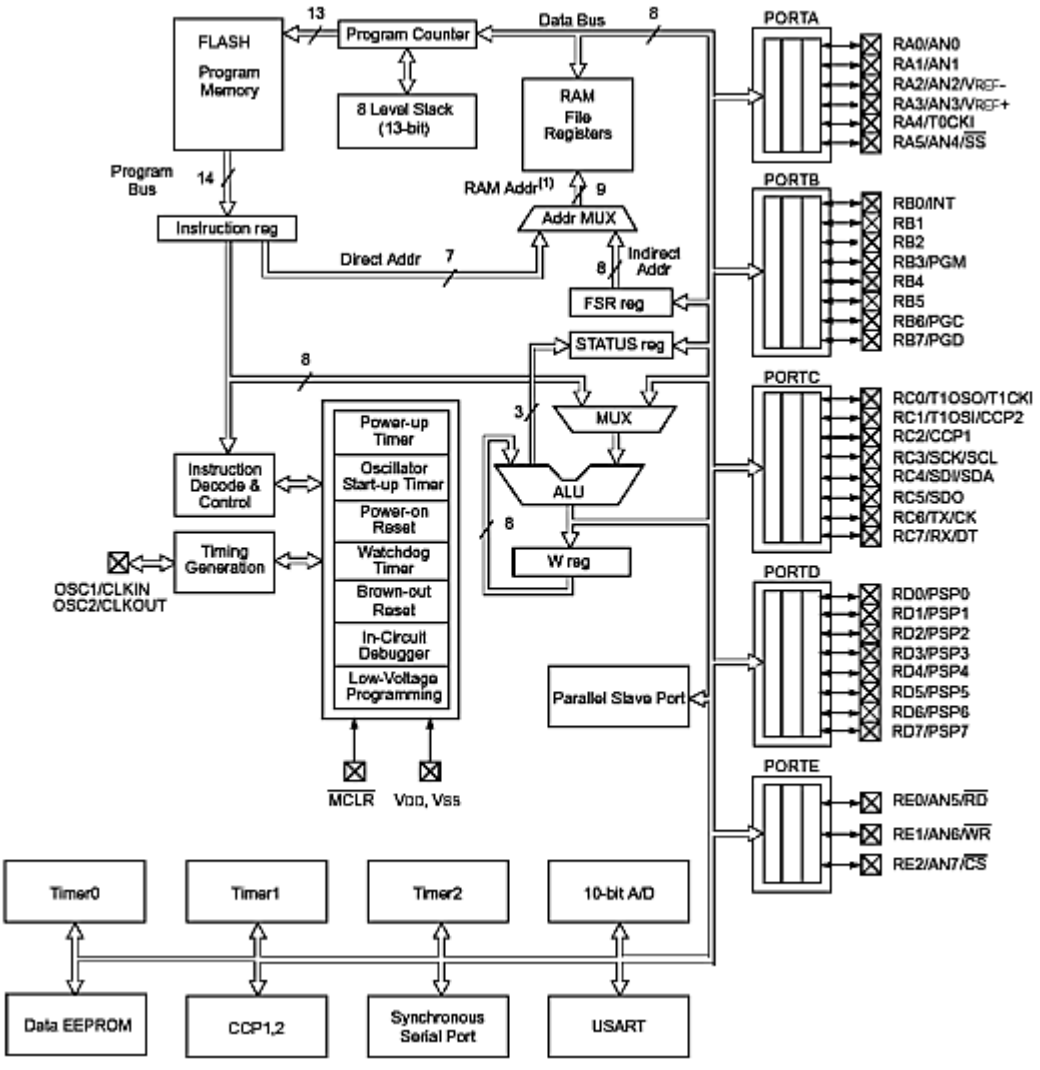


Figure – 3.1 PIC 16F877 Block Diagram [Ref. 8]

Microcontroller is a single integrated circuit containing specialized circuits and functions that are applicable to intelligent and control based systems. The microcontrollers are low-cost and high performance devices and mostly use Reduced Instruction Set Computer (RISC) architecture. The core blocks of a microcontroller are similar to those of a microprocessor. The main difference between microprocessor and microcontroller is that microcontrollers also incorporate onboard RAM, EPROM (for program and data storage), and peripherals, which would be externally interfaced on a microprocessor system. This arrangement simplifies the design of microcontroller systems. On the other hand, the peripherals are generally simpler than external device and the amount of available memory would be much smaller [Ref. 2].

Figure – 3.1 shows the layout of architecture of the PIC 16F877 microcontroller [Ref. 8], which is manufactured by Microchip Technology. The key features of the PIC 16F877 microcontroller are shown in the Table – 3.1 below.

Table – 3.1 Basic features of PIC 16F877

Key Features	**PIC 16F877**
Operating Frequency	DC - 20 MHz
Resets (and Delays)	POR, BOR (PWRT, OST)
FLASH Program Memory	8K x 14bit
Data Memory	368 bytes
EEPROM Data Memory	256 bytes
Interrupts	14
I/O Ports	Ports A,B,C,D,E
Timers	3
Serial Communication	MSSP, USART
Capture/Compare/PWM Modules	2
Parallel Communications	PSP
10-bit Analog-to-Digital Module	8 input channels
Instruction Set	35 instructions

3.1.1 Program Counter and Program Memory EPROM

The program counter is used to fetch each instruction in order. Modifications to the program counter by JUMP instructions affect program flow. Program memory is addressed word-wise, rather than byte-wise, so the first instruction is located at 0x000, the second at 0x001, etc. Program memory is organized into pages. Memory pages can be selected using the PCLATH register. There are three memory blocks in each of the PIC16F877 microcontrollers. The Program Memory and Data Memory have separated buses so that concurrent access can occur. The PIC16F877 microcontrollers have a 13-bit program counter capable of addressing an 8K x 14 program memory space. That means that the PIC 16F877 microcontrollers have 8K x 14 words of FLASH program memory [Ref. 8].

As mentioned before, the program counter (PC) is 13-bits wide. The low byte comes from the PCL register, which is a readable and writable register. The upper 4 bits are not readable, but are indirectly writable through the PCLATH register. On any RESET, the upper 5 bits of the PC will be cleared. Figure – 3.2 shows the two situations for the loading of the PC. The upper example in the figure shows how the PC is loaded on a write to PCL and the lower example in the figure shows how the PC is loaded during a CALL or GOTO instruction.

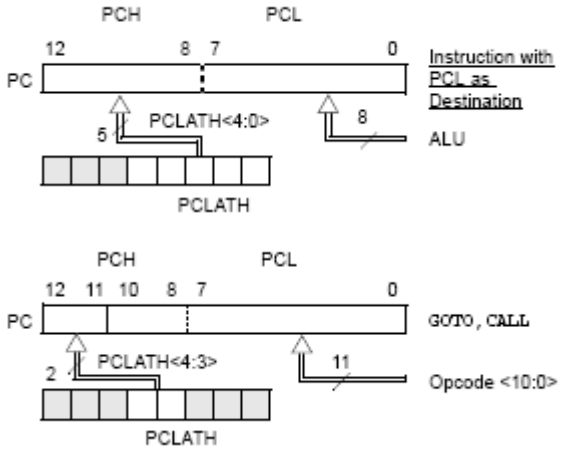


Figure – 3.2 Two situations of loading PC

The PIC 16F877 microcontrollers have an 8-level deep and 13-bit wide hardware stack, which operates as a circular buffer. The stack space is not a part of neither program space nor data space and the stack pointer is not readable or writable. The PC is pushed onto the stack when a CALL instruction is executed, or an interrupt causes a branch. The stack is popped in the event of a RETURN, a RETLW or a RETFIE instruction execution. PCLATH is not affected by a PUSH or POP operation.

3.1.2 Data RAM

Data RAM is organized as file registers (usually 8 bit) which are both readable and writable. Data RAM can be used to hold data for calculations, temporary variable storage, or user settings. The data memory of the PIC 16F877 microcontroller is partitioned into multiple banks each of which has 128 bytes long. A specific bank can be selected by setting special bits (RP0 and RP1) of the STATUS register. All implemented banks contain Special Function Registers, which may be mirrored in another bank for code reduction and quicker access. Special Function Registers are registers used by CPU and peripheral modules for controlling the desired operation of the device. These registers, which are implemented as static RAM, can be classified into two groups: core (CPU) and peripheral.

3.1.3 Arithmetic Logic Unit

Actual arithmetic instructions (add, subtract, shift, AND, OR, etc.) are carried out here. The results of arithmetic instructions are generally stored in the working (W) register, although some instructions allow a file register to be operated on directly. The contents of the W register can be transferred to the I/O ports, or used as an indirect address to access program or data memory.

3.1.4 I/O ports

The PIC 16F877 microcontroller is 40-pin device as shown in Figure – 3.3 [Ref. 8]. V_{dd} and V_{ss} pins provide power and ground references, respectively. There are two ports for external oscillator connection. The Master Clear/Reset pin (MCLR) is used to reset the PIC device externally. This could be tied to a system-reset circuit. If no external reset is used, the pin should be tied to V_{dd}. The PIC 16F877 microcontroller has A (6 pins), B (8 pins), C (8 pins), D (8 pins) and E (3 pins) ports, totally 33 pins.

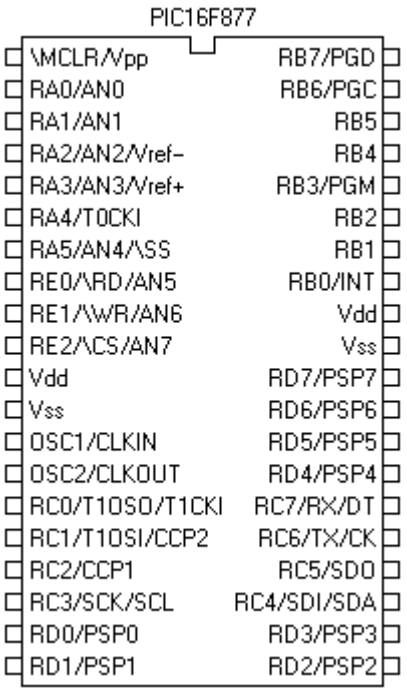


Figure – 3.3 PDIP package

The generic I/O ports are bidirectional pins that can be set up for input (the microcontroller reads the data on the pins) or output (the microcontroller sets the value on the pins) by the code. For output, each pin has an internal latch to drive the output. Some I/O ports are multiplexed with the onboard peripherals. Some pins for these I/O ports are multiplexed with an alternate function for the peripheral features on the device. The function of the pin (input, output, or peripheral) is determined by internal control registers. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

3.1.4.1 PORTA and the TRISA Register

PORTA is a 6-bit wide, bi-directional port and the corresponding data direction register is TRISA. Setting a TRISA bit (one) will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a Hi-Impedance mode). Clearing a TRISA bit (zero) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin). Reading the PORTA register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations. Pin RA4 is multiplexed with the Timer0 module clock input. This pin is a Schmitt Trigger input and an open drain output. All other PORTA pins have TTL input levels and full CMOS output drivers. Other PORTA pins are multiplexed with analog inputs and analog VREF input. The operation of each pin is selected by clearing or setting the control bits in the ADCON1 register (A/D Control Register1). It should be noted that on a Power-on Reset (POR), these pins are configured as analog inputs and read as '0'. The TRISA register controls the direction of the RA pins, when they are being used as analog inputs.

3.1.4.2 PORTB and the TRISB Register

PORTB is an 8-bit wide, bi-directional port and the corresponding data direction register is TRISB. Three pins of PORTB are multiplexed with the Low Voltage Programming function. Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit RBPU (bit 7 of OPTION_REG register). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

Four of the PORTB pins (RB4, RB5, RB6 and RB7) have an interrupt-on-change feature. Only pins configured as inputs can cause this interrupt to occur. These input pins are compared with the old value latched on the last read of PORTB. The mismatched outputs of these pins are OR’ed together to generate the

RB Port Change Interrupt with flag bit RBIF (bit 0 of INTCON register). This interrupt can wake the device from SLEEP. In the Interrupt Service Routine the interrupt can be cleared in the following manner:

a) Any read from or write to PORTB.

b) Clear flag bit RBIF.

This interrupt-on-mismatch feature, together with software configurable pull-ups on these four pins, allow easy interface to a keypad and make it possible for wake-up on key depression. RB0 is an external interrupt input pin and is configured using the INTEDG bit (bit 6 of OPTION_REG register).

3.1.4.3 PORTC and the TRISC Register

PORTC is an 8-bit wide, bi-directional port and the corresponding data direction register is TRISC. PORTC is multiplexed with several peripheral functions, and PORTC pins have Schmitt Trigger input buffers. When enabling peripheral functions, care should be taken in defining TRIS bits for each PORTC pin. Some peripherals override the TRISC bit to make a pin an output, while other peripherals override the TRISC bit to make a pin an input. Since the TRISC bit override is in effect while the peripheral is enabled, read-modify-write instructions (BSF, BCF, XORWF) with TRISC as destination, should be avoided.

3.1.4.4 PORTD and TRISD Registers

PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output. PORTD can be configured as an 8-bit wide microprocessor port (parallel slave port) by setting control bit PSPMODE (bit 4 of TRISE register). In this mode, the input buffers are TTL.

3.1.4.5 PORTE and TRISE Register

PORTE has three pins, which are individually configurable as inputs or outputs. These pins have Schmitt Trigger input buffers. The PORTE pins become the I/O control inputs for the microprocessor port when bit PSPMODE is set. In

this mode, the user must make certain that the bit 0, bit 1 and bit 2 of TRISE are set, and that these pins are made digital inputs by configuring ADCON1 register. In this mode, the input buffers are TTL. TRISE register also controls the parallel slave port operation. PORTE pins are multiplexed with analog inputs. When selected for analog input, these pins will read as ’0’s. The user must make sure to keep the pins configured as inputs when using them as analog inputs. On a Power-on Reset, these pins are configured as analog inputs, and read as ‘0’.

3.1.5 Peripherals

There are many peripherals that maybe included on the chip. The various types included by the PIC 16F877 microcontroller are summarized below:

■ **Analog to Digital Converters:** Analog inputs on some pins are converted to a digital value, which can then be processed by the microcontroller. This is useful for obtaining data from external devices like thermistors, accelerometers, and other sensors. The PIC 16F877 microcontroller has 10-bit ADC module with 8 input channels.

The analog input charges a sample and hold capacitor. The output of the sample and hold capacitor is the input into the converter. The converter then generates a digital result of this analog level via successive approximation. The A/D conversion of the analog input signal results in a corresponding 10-bit digital number. The A/D module has high and low voltage reference input that is software selectable to some combination of VDD, VSS, RA2, or RA3.

The A/D converter has a unique feature of being able to operate while the device is in SLEEP mode. To operate in SLEEP, the A/D clock must be derived from the A/D’s internal RC oscillator.

The A/D module has four registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register0 (ADCON0)
- A/D Control Register1 (ADCON1)

The ADCON0 register controls the operation of the A/D module. The ADCON1 register configures the functions of the port pins.

■ **Universal Synchronous/Asynchronous Receiver/Transmitter (USART):** This device allows the microcontroller to interface serially with other devices through protocols such as RS232. The PIC 16F877 microcontroller USART module will be explained comprehensively in Chapter 4.

■ **Master Synchronous Serial Port (MSSP):** This module provides a mode in which Data bus, RD, WR, and Chip Select are all made available. This allows the microcontroller to be interfaced to a microprocessor, a microcontroller or a peripheral, which may be a serial EEPROM, shift register, display drivers, A/D converters, etc. The MSSP module can operate in one of two modes, such as Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I^2C). The I^2C bus is a proprietary serial bus developed by Microchip Technology. Not all microcontrollers have both I^2C Master function and I^2C Slave function, but 16F877 has.

■ **Timers:** Timers can be used to time events. Timers have an advantage over merely using wait loops in that other processing can take place while the timer is running. 16F877 has three independent timers.

Timer 0 has the following features:

• 8-bit timer/counter
• Readable and writable
• Software programmable prescaler
• Internal or external clock select
• Interrupt on overflow from FFh to 00h
• Edge select for external clock

Timer 1 has the following features:

• 16-bit timer/counter consisting of two 8-bit registers (TMR1H, TMR1L)

• Readable and writable
• Software programmable prescaler
• Internal or external clock select
• Interrupt on overflow from FFFFh to 0000h

Timer 2 has the following features:
• 8-bit timer/counter
• Readable and writable
• Software programmable prescaler and postscaler
• Interrupt on overflow from FFh to 00h

■ **Capture/Compare/PWM (CCP):** This module allows the microcontroller to watch an input until it reaches a certain value, then take some action. It is especially useful for implementing Pulse Width Modulation (PWM) schemes. The PIC 16F877 microcontroller has two CCP module, each contains 16-bit register which can operate as a:

• 16-bit Capture register
• 16-bit Compare register
• PWM Master/Slave Duty Cycle register

Both the CCP1 and CCP2 modules are identical in operation, with the exception being the operation of the special event trigger.

3.2 Microcontroller Programming Procedure

During this thesis study MicroCode Studio programming environment was used to compose the code and to load it to the microcontroller. MicroCode Studio is a powerful, visual Integrated Development Environment (IDE) with In Circuit Debugging (ICD) capability with PicBasic PRO compiler [Ref. 9]. The code explorer allows the user to automatically jump to include files, defines, constants,

variables, aliases and modifiers, symbols and labels that are contained within the source code. It is easy for a user to set up the compiler, assembler and programmer options or the user can let MicroCode Studio do it for him with its built in auto-search feature. Compilation and assembler errors can easily be identified and corrected using the error results window. MicroCode Studio even provides a serial communications window.

The In Circuit Debugger (ICD) enables the user to execute a PicBasic Program on a host PIC microcontroller and view variable values, Special Function Registers (SFR), memory and EEPROM as the program is executing. The user can toggle multiple breakpoints and step through the PicBasic code line by line.

PicBasic programming language, which can be defined as compiler in MicroCode Studio, has nearly 100 different and powerful commands. These commands were determined according to the microcontroller needs, so PicBasic helps the user to write powerful codes much simpler than assembly. MicroCode Studio can be obtained from the following web page [Ref. 16]:

http://www.mecanique.co.uk/code-studio/

After compiling the PicBasic code in MicroCode Studio, hex code is created automatically. This hex code is sent to microcontroller via ICProg PIC programmer, which can be defined as programmer in MicroCode Studio optionally. ICProg allows the user to program all types of serial programmable Integrated Circuits as 12Cxx, 16Cxxx, 16Fxx, 16F87x, 18Fxxx, 16F7x, 24Cxx, 93Cxx, 90Sxxx, 59Cxx, 89Cx051, 89S53, 250x0, PIC, AVR, 80C51 etc. using Windows 95/98/NT/2000/ME/XP. ICProg or other equivalent PIC programmers can be obtained from Internet easily.

CHAPTER 4

SERIAL COMMUNICATION WITH MICROCONTROLLERS

4.1 Introduction

There are two modules embedded in the PIC 16F877 microcontrollers for serial communication purpose. These are USART (Universal Synchronous Asynchronous Receive Transmit) module and MSSP (Master Synchronous Serial Port) module. The USART module is mainly used in order to interface the microcontroller with a peripheral device with RS 232. That is why the USART module is in the scope of this thesis and it is implemented in order to perform the serial communication between the main PC and the implemented multi-port device. Whereas the MSSP module is mainly functioned for communicating with other microcontroller devices and peripherals, such as serial EEPROMs, shift registers, display drivers, A/D converters, etc. So the MSSP module is not covered in this thesis.

In addition to these two serial communication modules, the ordinary I/O pins can be used for serial communication purpose thanks to the advanced I/O features of the PIC 16F877 microcontroller. This technique is named as inverted logic level method. The serial communication between the implemented multi-port device and the digital electricity meters and the remote PC is performed over the 8 RB pins by using this inverted logic level technology. Hereafter the details of the USART module and inverted logic level technology will be explained.

4.2 USART Module

The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the two serial I/O modules embedded in PIC 16F877 microcontrollers. The USART, which is also known as a Serial Communications

Interface or SCI, can be configured as a full duplex asynchronous system that can communicate with peripheral devices such as CRT terminals and personal computers, or it can be configured as a half duplex synchronous system that can communicate with peripheral devices such as A/D or D/A integrated circuits, serial EEPROMs etc.

The USART module can be configured in the following modes:

- Asynchronous (full duplex)
- Synchronous - Master (half duplex)
- Synchronous - Slave (half duplex)

Bit 7 of RCSTA register and bit 7 and bit 6 of TRISC registers have to be set in order to configure pins RC6 and RC7 as the Universal Synchronous Asynchronous Receiver Transmitter.

As mentioned previously, because of the mismatch between the TTL/CMOS signal level and RS 232 signal level, the serial data communication between a microcontroller and a peripheral device with RS 232 interface, via USART module must be fulfilled over a level converter device. So before giving the details of USART module, it should be better to explain signal level conversion process.

4.2.1 Signal Level Conversion

In order to connect a microcontroller to the serial port of a peripheral device (EIA 232 or RS 232) properly, we need to adjust the level of the signal. The signal level on the serial port of a PC is –10 V for logic zero, and +10 V for logic one as shown in Figure – 4.1 and 4.2. Since the signal level on the microcontroller is +5 V for logic one, and 0 V for logic zero, we need an intermediary stage that will convert the levels. There are some chips specially designed for this task. These chips receive signals from -10 to +10 V and convert them into 0 and 5 V [Ref. 1].

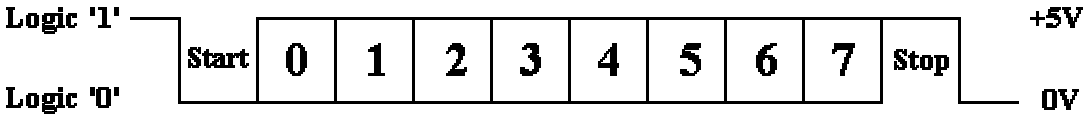


Figure – 4.1 TT/CMOS Serial Logic Waveform

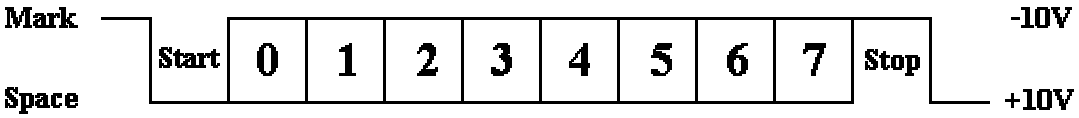


Figure – 4.2 RS-232 Logic Waveform

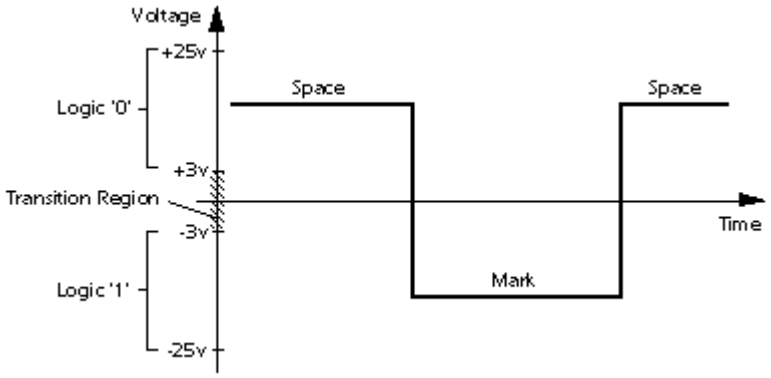


Figure – 4.3 Logical Regions of signal

Voltages of –3 V to –25 V with respect to signal ground pin are considered logic 1 (the marking condition), whereas voltages of +3 V to +25 V are considered as logic 0 (the spacing condition) as shown in Figure – 4.3. The range of voltages between –3 V and +3 V is considered a transition region for which a signal state is not assigned.

Two common RS-232 Level Converters are the 1488 RS-232 Driver and the 1489 RS-232 Receiver. Each package contains 4 inverters, but only one type, either Drivers or Receivers. The driver requires two supply rails, +7.5 to +15 V and -7.5 to –15 V. As it could be understood this may be a problem in many

instances where only a single supply of +5 V is present. However the advantages of these I.C.'s are their cheapness [Ref. 3].

Another device is the MAX 232, which is used in the implemented multi-port device. It includes a charge pump, which generates +10 V and –10 V from a single 5 V supply. MAX 232 also includes two receivers and two transmitters in the same package [Ref. 1]. This is handy in many cases when you only want to use the Transmit and Receive data lines. And also there is no need to use two chips, one for the receive line and one for the transmit line.

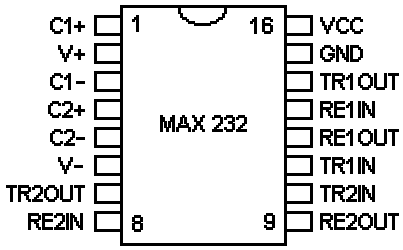


Figure – 4.4 MAX232 pin diagram

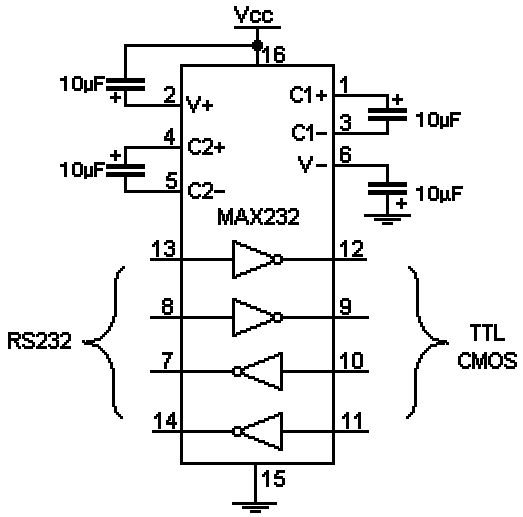


Figure – 4.5 MAX232 circuit diagram

The large valued capacitors, which are used with level converters, are not only bulky, but also expensive. Therefore some level converter devices are available which use smaller capacitors and even some with inbuilt capacitors. However the MAX-232 is the most common one.

4.2.2 The USART Module Asynchronous Mode

In this mode, the USART module uses standard non-return-to zero (NRZ) format with one START bit, eight or nine data bits, and one STOP bit. The most common data format is 8-bit. An on-chip, dedicated, 8-bit baud rate generator can be used to derive standard baud rate frequencies from the oscillator. In order to define appropriate baud rate according to the employed oscillator it should be necessary to look at the PIC 16F877 data sheet [Ref. 8].

The USART module transmits and receives the LSB first. The transmitter and receiver are functionally independent, but use the same data format and baud rate. Parity is not supported by the hardware, but can be implemented in the software (and stored as the ninth data bit). Asynchronous mode, which is activated by clearing bit SYNC (bit 4 of TXSTA register), is stopped during SLEEP.

The USART Asynchronous module consists of the following important elements:

- Baud Rate Generator
- Sampling Circuit
- Asynchronous Transmitter
- Asynchronous Receiver

In PicBasic Pro, the parameters of USART module, which affects both USART receiver and transmitter, must be declared at the beginning of the program with “Define” commands as below [Ref. 5 and 6]:

```
Define HSER_EVEN      1       'If even parity desired
Define HSER_ODD       1       'If odd parity desired
Define HSER_BAUD      4800    'USART baud rate
Define HSER_CLROERR   1       'Clear USART overrun error
```

After these declarations, the transmit and receive processes in the code with commands Hserout and Hserin respectively will be fulfilled according defined parameters. The default settings of USART in PicBasic Pro module are as

8 data bits, none parity, 1 stop bit. Please refer to the PicBasic Pro code in Appendix B.

4.2.2.1 USART Asynchronous Transmitter

The USART transmitter block diagram is shown in Figure – 4.6. The heart of the transmitter is the transmit (serial) shift register (TSR). The shift register obtains its data from the read/write transmit buffer, TXREG. The TXREG register is loaded with data in software. The TSR register is not loaded until the STOP bit has been transmitted from the previous load. As soon as the STOP bit is transmitted, the TSR is loaded with new data from the TXREG register (if available). Once the TXREG register transfers the data to the TSR register (occurs in one T_{CY}), the TXREG register becomes empty and flag bit TXIF (bit 4 of PIR1 register) is set. This interrupt can be enabled/disabled by setting/clearing enable bit TXIE (bit 4 of PIE1 register). Flag bit TXIF will be set, regardless of the state of enable bit TXIE and cannot be cleared in software. It will reset only when new data is loaded into the TXREG register. While flag bit TXIF indicates the status of the TXREG register, another bit TRMT (bit 1 of TXSTA register) shows the status of the TSR register. Status bit TRMT is a read only bit, which is set when the TSR register is empty. No interrupt logic is tied to this bit, so the user has to poll this bit in order to determine if the TSR register is empty. It should be noted that the TSR register is not mapped in data memory, so it is not available to the user and flag bit TXIF is set when enable bit TXEN (bit 5 of TXSTA register) is set, TXIF is cleared by loading TXREG.

Transmission is enabled by setting enable bit TXEN. The actual transmission will not occur until the TXREG register has been loaded with data and the baud rate generator (BRG) has produced a shift clock. The transmission can also be started by first loading the TXREG register and then setting enable bit TXEN. Normally, when transmission is first started, the TSR register is empty. At that point, transfer to the TXREG register will result in an immediate transfer to TSR, resulting in an empty TXREG. A consecutive transfer is thus possible. Clearing enable bit TXEN during a transmission will cause the transmission to be

aborted and will reset the transmitter. As a result, the RC6 pin will revert to hi-impedance.

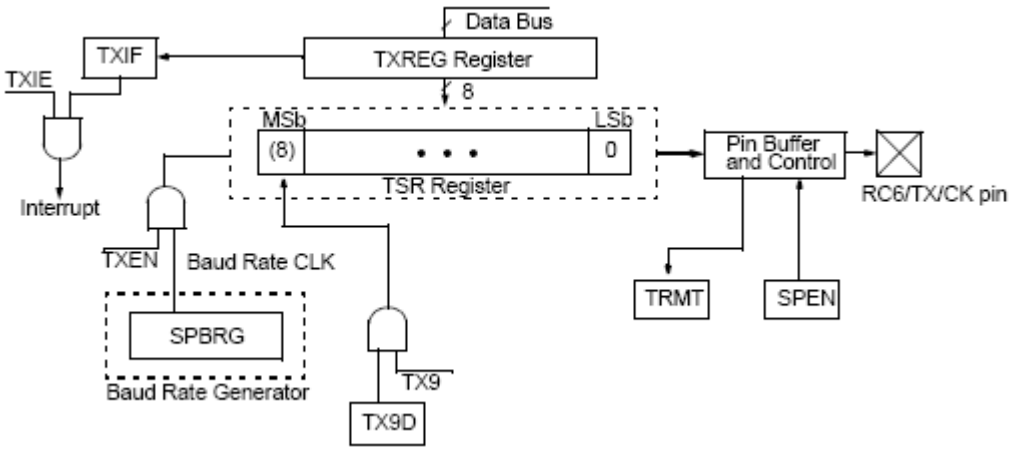


Figure – 4.6 USART Transmit Block Diagram

When setting up an Asynchronous Transmission, the following steps are executed [Ref. 8]:

1 - Initialize the SPBRG register for the appropriate baud rate. If a high-speed baud rate is desired, set bit BRGH.

2 - Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.

3 - If interrupts are desired, then set enable bit TXIE.

4 - If 9-bit transmission is desired, then set transmit bit TX9.

5 - Enable the transmission by setting bit TXEN, which will also set bit TXIF.

6 - If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.

7 - Load data to the TXREG register (starts transmission).

8 - If using interrupts, ensure that GIE bit and PEIE bit (bits 7 and 6 of the INTCON register) are set.

It seems that these steps are troublesome a bit. But owing to the “Hserout” command provided by PicBasic Pro, sending data over USART module becomes fairly practical. At the beginning of the code, the initiation of transmit over USART module must be also defined as below:

```
Define HSER_TXSTA      20h        'USART transmit status init
```

4.2.2.2 USART Asynchronous Receiver

The receiver block diagram is shown in Figure – 4.7. Once Asynchronous mode is selected, reception is enabled by setting bit CREN (bit 4 of RCSTA register). The data is received on the RC7 pin and is passed to the data recovery block. The data recovery block is actually a high-speed shifter, operating at x16 times the baud rate. Whereas, the main receive serial shifter (RSR) operates at the baud rate or at F_{OSC}.

After sampling the STOP bit, the received data in the RSR is transferred to the RCREG register (if it is empty). If the transfer is complete, flag bit RCIF (bit 5 of PIR1 register) is set. The actual interrupt can be enabled/disabled by setting/clearing enable bit RCIE (bit 5 of PIE1 register). Flag bit RCIF is a read only bit, which is cleared by the hardware. It is cleared when the RCREG register has been read and is empty. The RCREG is a double-buffered register (that is, it is a two deep FIFO). It is possible for two bytes of data to be received and transferred to the RCREG FIFO and a third byte to begin shifting to the RSR register. On the detection of the STOP bit of the third byte, if the RCREG register is still full, the overrun error bit OERR (bit 1 of RCSTA register) will be set. The word in the RSR will be lost. The RCREG register can be read twice to retrieve the two bytes in the FIFO. Overrun bit OERR has to be cleared in software. This is done by resetting the receive logic (CREN is cleared and then set). If bit OERR is set, transfers from the RSR register to the RCREG register are inhibited, and no further data will be received. It is therefore, essential to clear error bit OERR if it is set. Framing error bit FERR (bit 2 of RCSTA register) is set if a STOP bit is detected as clear. Bit FERR and the 9th receive bit (RX9D, bit 0 of RCSTA

register) are buffered the same way as the receive data. Reading the RCREG will load bits RX9D and FERR with new values, therefore, it is essential for the user to read the RCSTA register before reading the RCREG register in order not to lose the old FERR and RX9D information.

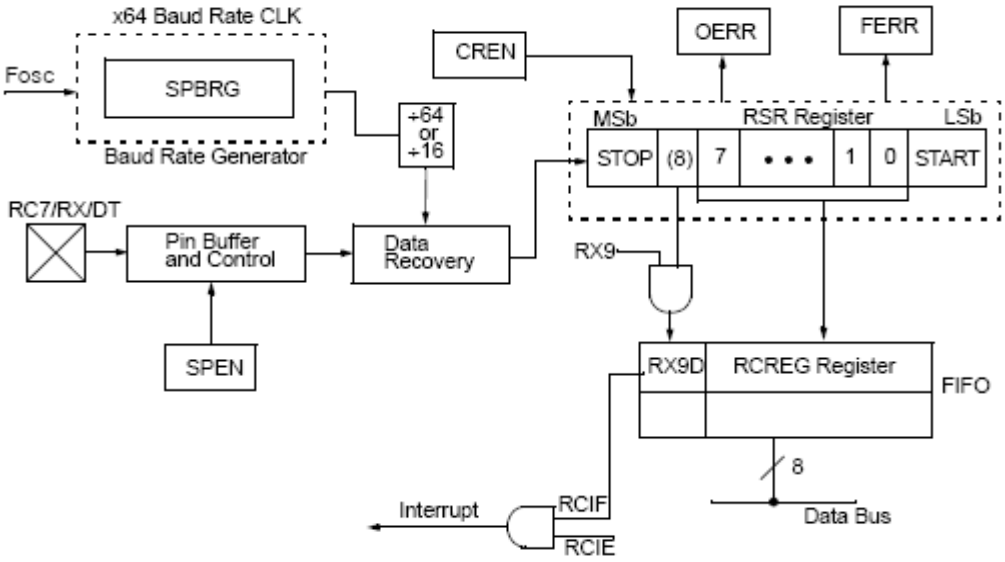


Figure – 4.7 USART Receive Block Diagram

When setting up an Asynchronous Reception, the following steps are executed [Ref. 8]:

1 - Initialize the SPBRG register for the appropriate baud rate. If a high-speed baud rate is desired, set bit BRGH.

2 - Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.

3 - If interrupts are desired, then set enable bit RCIE.

4 - If 9-bit reception is desired, then set bit RX9.

5 - Enable the reception by setting bit CREN.

6 - Flag bit RCIF will be set when reception is complete and an interrupt will be generated if enable bit RCIE is set.

7 - Read the RCSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.

8 - Read the 8-bit received data by reading the RCREG register.

9 - If any error occurred, clear the error by clearing enable bit CREN.

10 - If using interrupts, ensure that GIE and PEIE (bits 7 and 6) of the INTCON register are set.

Similar to the USART transmit process, the USART receive process can be fulfilled very practically by using “Hserin” command provided by PicBasic Pro. At the beginning of the code, the initiation of transmit over USART module must be also defined as below:

```
Define  HSER_TXSTA      90h         'USART receive status init
```

4.3 Inverted Logic

As mentioned previously normal I/O pins can be utilized as serial communication interface thanks to current RS-232 implementation and the excellent I/O specifications of the PIC 16F877. As a matter of fact, utilizing ordinary I/O pins as serial interface is a necessity for this thesis, because the multi-port device needs at least 5 serial communication interfaces, but the PIC 16F877 has only one USART module.

8 RB pins of PIC 16F877 are allocated for serial interface to digital electricity meters and remote PC. As it can be seen from the circuit diagram in Figure – 5.9, there are not any extra component for level conversion other than resistors between RB pins of the microcontroller and the ports for the peripherals, so the communication still has voltage level conflict. Under these circumstances the PIC 16F877 microcontroller provides an effective solution. Voltage level of an I/O pin can be inverted for serial communication purpose by the code. Only requirement is using resistors in order to limit the current. The pins of the microcontroller used as Rx are connected to Tx pins of other peripherals over 22 KΩ resistors, similarly the pins of microcontroller used as Tx are connected to Rx of other peripherals over 1 KΩ resistors as shown in Figure – 4.8 [Ref. 9].

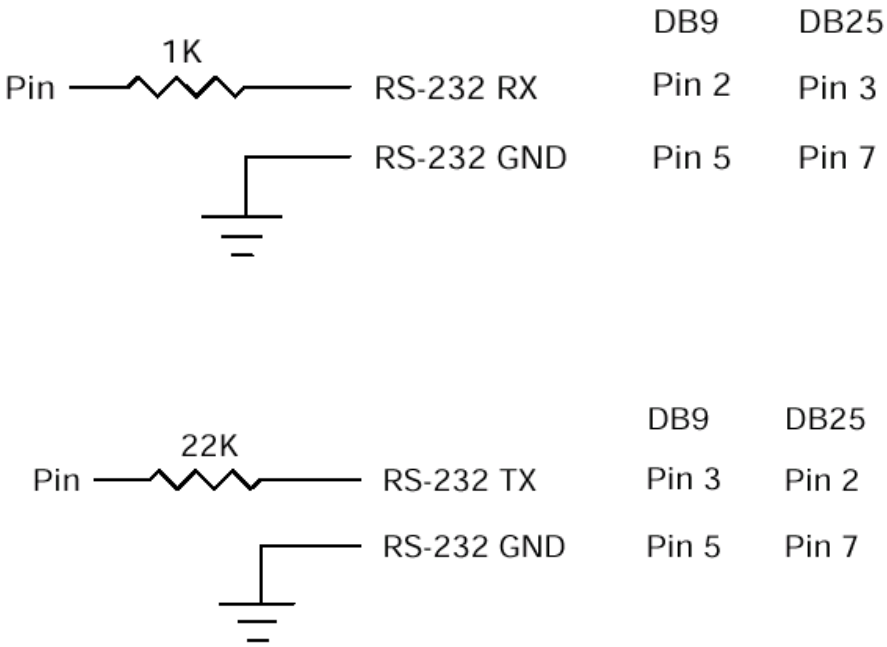


Figure – 4.8 Inverted logic connections

There are several commands used in PicBasic Pro in order to transmit or receive character serially. Serin2 and Serout2 are two of them, which can be utilized as inverted logic or normal logic according to application and hardware. Only the number of data-bits and odd parity enabling (if parity is enabled, even parity is default) of Serout2 and Serin2 commands must be declared at the beginning of the program as below:

```
Define SER2_BITS    8       'Set number of data bits
Define SER2_ODD     1       'Use odd parity instead of even parity
```

The usage of Serin2 command in the code is as following [Ref. 5 and 6]:

```
Serin2 PortB.2,16572,65000,receivernotready,[char]
```

The above code performs the required procedure in order to receive the char character over PortB.2. The parameter 16572 is the mode of receiving, which defines the specifications of receiving such as baud rate, parity enabling, inverted

logic or normal logic. The complete table of Serin2 and Serout2 modes is presented in Appendix E. The number of 65000 is the timeout value (65000 milisec = 65 sec) to receive a character. Please refer to PicBasic code to see the different usages of Serout2 and Serin2 commands. For the detailed information about the PicBasic Pro, please look at http://picbasic.com [Ref. 9].

4.4 Comments on Serial Communication with Microcontrollers

According to given explanation, the remarks listed below should be taken into consideration for properly performing serial communication with microcontrollers:

- Firstly a stable oscillator should be equipped in the circuit. If 4 MHz Crystal or Resonator oscillator is equipped, that would be enough for proper data communication at 4800 baud rate. But if internal or RC type oscillator is used, it would be unlikely to get stable communication above 1200 baud rate [Ref 10].
- Throughout designing level, operating the system starting with lower baud rates would be a good designing method. After success, the baud rate should be increased step by step.
- The signal level conversion between TTL/CMOS logic and RS 232 should be fulfilled properly, otherwise nothing can be transmitted or received between a microcontroller and a peripheral device.
- If a transmission towards a PC is realized, it should be better to terminate the data packets with a <CR> and <LF> characters. These Carriage Return and Line Feed characters terminate the data buffer string at the PC and start a new string with the next data packet.

CHAPTER 5

DESCRIPTION OF IMPLEMENTED SYSTEM

As mentioned briefly in Chapter 1, the overall system contains a main PC, a developed remote access software running on the main PC, one modem connected to main PC, the multi-port device, another modem connected to the multi-port device, one remote PC and three digital electricity meters or no remote PC and four electricity meters. The device connected to the port 1 may be a PC or a digital electricity meter according to the application. Hereafter the main parts of the system and their operating principles will be explained one by one.

5.1 Remote Access Software

The remote access software used in the system was developed with Delphi 7.0 by using APRO (Asynchronous Professional) Libraries to perform serial communication [Ref. 11]. With APRO, the implemented programs can send alphanumeric characters to anywhere in the world, even across the Internet. The user can add Dial-Up networking capabilities to the programs easily together with advanced terminal control, automated scripting for complex communication tasks, and even multi-user fax server solutions. The software developed in this thesis mainly uses the Dial-Up networking features of APRO Libraries.

The developed remote access software has two versions, one of them is for main PC, and the other one is for remote PC. The version operating on remote PC can be stated as lite version, because it does not have many functions such as meter remote reading options. This lite version is specified for the Xmodem file transfer. So the program execution procedure will be explained separately for full version and lite version.

5.1.1 Running of Full Version Remote Access Software

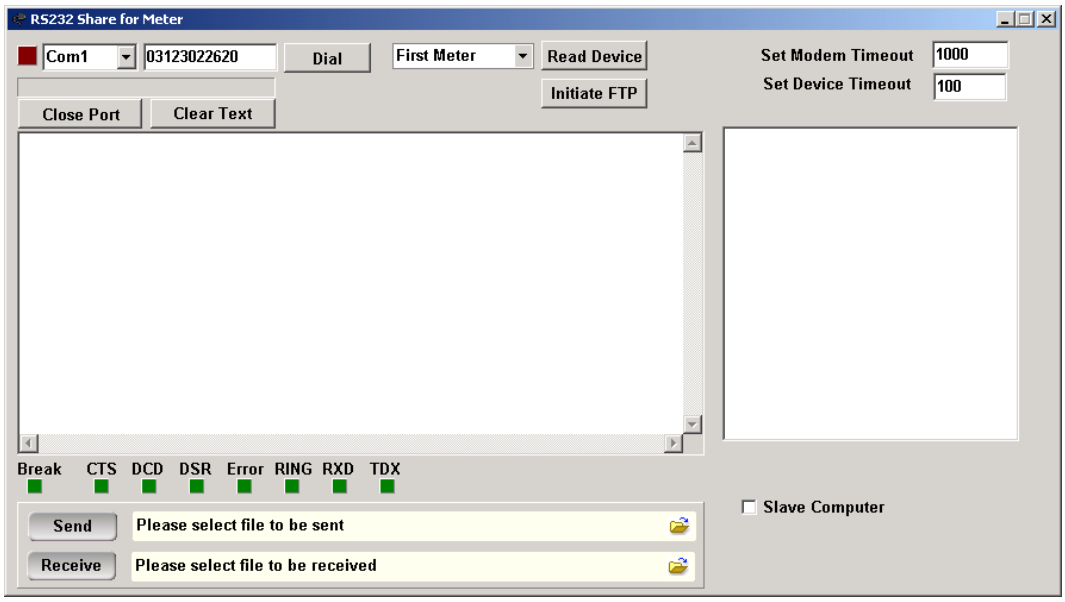


Figure – 5.1 Main screen of Reading Software on main PC

When the program, which is developed by Delphi 7.0 programming language and loaded to main PC, is started, the program window shown in Figure – 5.1 should appear on the screen.

The execution steps of software are described below:

1)- First of all, the serial port of main PC, which will be used for serial communication, should be selected. The active serial ports are listed in the left upper box.

2)- Then the phone number, which will be dialed to establish the communication with the remote side, should be specified in the box at left side of Dial button.

3)- Next, the timeout values for main PC to modem connection and main PC to multi-port device should be entered to right upper box. After that, Dial button can be clicked, thus the "ATDT + Phone Number" command is sent to modem. Then, if the remote modem and telephone lines are available, the communication is established and the program starts to send communication indication string to the multi-port device. After the handshake process between the

multi-port device and main PC is completed, the program will receive the port status of the multi-port device, such as which device is connected to which port. If the handshake process between main PC and multi-port device, which can continue 60 seconds at most, fails anyhow, the connection will be closed automatically by the program. The related indication messages appear on the left log window.

4)- After the communication is established successfully, the request may be to perform a remote reading of a meter or to perform a file transfer to the remote PC.

5)- If the request is to perform a remote reading of a meter, the desired meter should be selected from the box and then Read Device button should be clicked. If the meter is not password protected and also connected to the port properly, the readout data will be listed in reading window on the left. The readout data can be written to a text file by clicking right button of mouse over the window.

6)- This meter readout procedure can be executed for each meter in the same manner. The readout window on the left can be cleared anytime by clicking Clear Text button.

7)- If the request is to perform a file transfer to the remote PC, the request must be initiated by clicking to the Initiate FTP button and after that the file to be sent should be selected by clicking File Open icon at the bottom of the program window at same line with Send button. After clicking to the Send button, the file transfer procedure will wait for 30 seconds for preparation of remote PC to receive a file. Within 30 seconds, when the remote PC passes to receiving file mode, the file transfer process will start according to Xmodem protocol.

8)- If a file transfer request is initiated from remote PC, the multi-port device informs the main PC to be in receiving mode within 30 seconds. When the main PC enters to receiving file mode, the file transfer process will start according to Xmodem protocol.

9)- The connection between main PC and multi-port device can be closed anytime by the user of main PC by clicking Close Port button.

Note: The buttons below the reading window indicates the status of serial communication performed by using APRO Libraries [Ref. 11]. They will be red or green according to the status of process while the program is running. Clear Text button clears the reading window to be prepared for next reading. Clear Text button can be used anytime.

The source code of the reading software is presented in Appendix A.

5.1.2 Running of Lite Version Remote Access Software

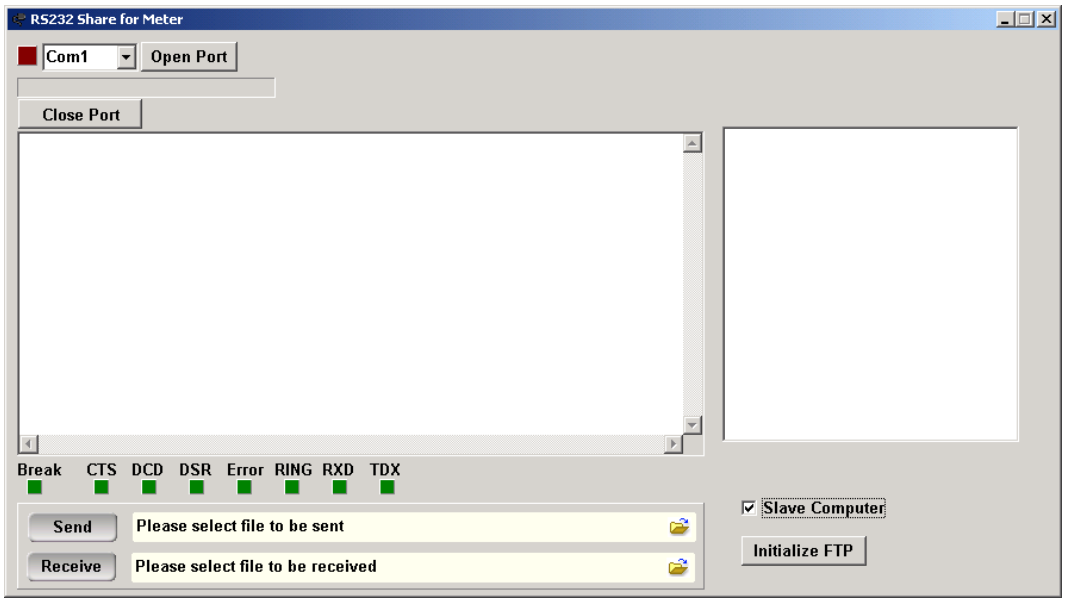


Figure – 5.2 Main screen of Reading Software on remote PC

When the program is started on remote PC and Slave Computer box at the right bottom of window is checked, the program window shown in Figure – 5.2 should appear on the screen of remote PC.

The execution steps of software are described below:

1)- First of all, the serial port to be used for the communication with multi-port device should be specified and opened.

2)- During the start up procedure, the multi-port device sends the port status detection string to all ports. As soon as the remote PC receives this string, it

immediately replies with a specified string that indicates remote PC is connected and ready.

3)- Then the remote PC starts to wait for a file transfer procedure initialization message. When this message is received, the program should be in receiving file mode within 30 seconds. Within 30 seconds, when the remote PC passes to receiving file mode, the file transfer starts immediately.

4)- The remote PC can initiate a file transfer anytime by clicking Initialize FTP button at the right bottom of the program window. If the remote PC receives "Ready for file transfer" message, the file transfer can be started. When Send button is clicked, the file transfer procedure will wait for 30 seconds for preparation of main PC to receive a file. Within 30 seconds, when the main PC passes to receiving file mode, the file transfer process starts according to Xmodem protocol.

5)- After the file transfer from remote PC to main PC is completed, the program execution turns back step 2. Throughout running of lite version remote access software, the specified serial port must be opened and the Slave Computer button must be checked.

Note: The buttons below the reading window indicates the status of serial communication performed by using APRO Libraries. They will be red or green according to the status of process while the program is running. The Clear Text button clears the reading window to be prepared the window for next reading. Clear Text button can be used anytime.

5.1.3 Flow Diagram of Reading Software

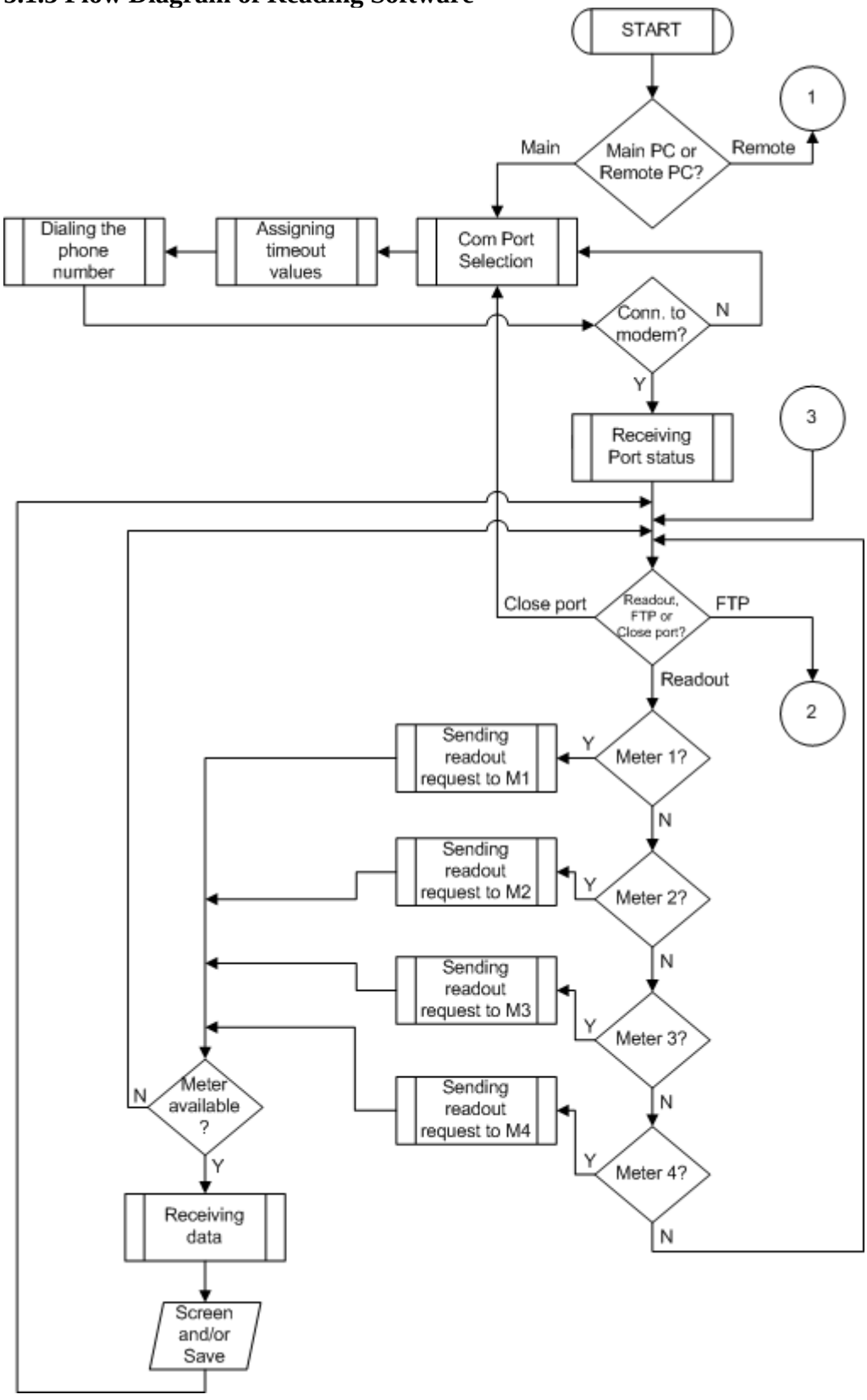


Figure 5.3 Reading software flow diagram (I)

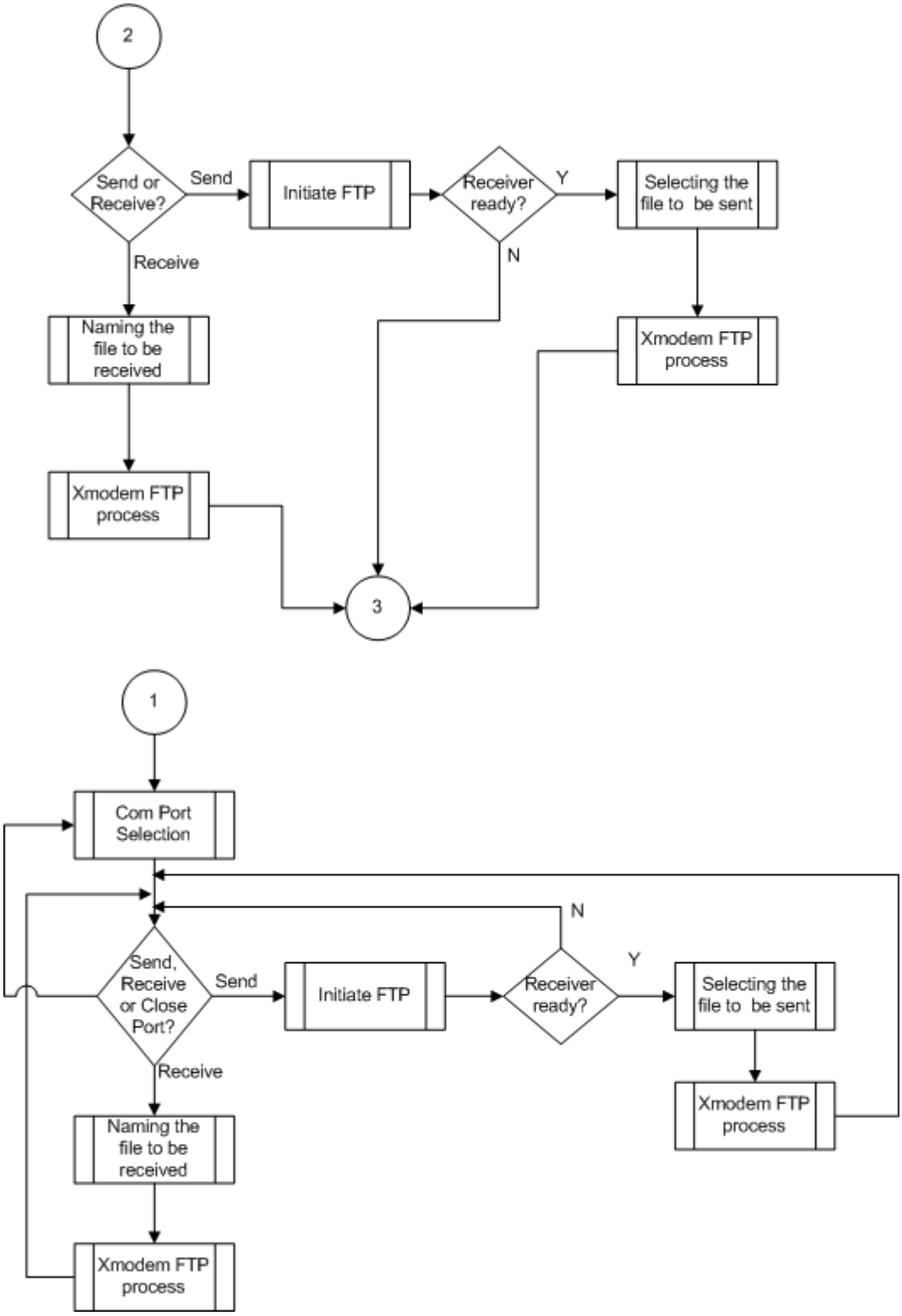


Figure 5.3 Reading software flow diagram (II)

It should be useful to describe the program code by following the flow diagram in order to give the theory.

When the program is started, first of all the Slave PC check box must be checked or not according to operating user station. If the program is running on remote PC, the box must be checked, and after that some options peculiar to main PC and meter readout process will be off by calling the TForm1.ChkboxClick() procedure.

If we assume that the operating user station is main PC, the next step will be establishing communication with remote side. In order to that, first of all the serial port to be used should be specified. When the upper left ComboBox is clicked, the available and properly working serial ports are listed. The Comfind() procedure in the program performs this port detection process. Then the timeout values for the establishment of communication between main PC and remote modem and main PC and the implemented multi-port device should be defined. The usage of timeout values in these EditBoxes by the program will be explained. Finally, the phone number, which will be dialed, should be defined before clicking Dial button.

When the Dial button is clicked, the program calls the TForm1.DialClick() procedure. In this procedure, the program branches to SingOn() procedure and then the selected serial port is assigned to the TApdComPort component of APRO (Asynchronous Professional) serial communication library. After that, the number is dialed by sending the ‘ATDT’ + Edit2.Text to the modem. At the same time, the timers controlling the periods for communication establishment are set by using previously mentioned timeout values defined in EditBoxes. If the remote modem or multi-port device does not respond with specified strings before the timers are expired, the program calls ComEnd() procedure and closes the serial port assigned to TApdComPort. If communication is established properly between main PC and multi-port device, the program will receive the port status information, which is sent by multi-port device. The reception of characters by using TApdComPort component is performed by TForm1.cp2TriggerAvail() procedure. Please look at APRO manual for more detailed information [Ref. 11].

After acquiring the knowledge of port status, the program is ready to perform meter readout or file transfer. For remote reading of a meter, after the device is selected from the ComboBox, the Read Device button should be clicked. By clicking Read Device button, the required string is sent to the multi-port device by TForm1.RdeviceClick() procedure. If there is a problem about the specified meter, the multi-port device informs the user of main PC by sending a message. If everything is OK, the readout data sent by the meters is received by TForm1.cp2TriggerAvail() procedure and screened on the reading window, which is a MemoBox. The readout data on the reading window can be saved to a text file, if it is required.

When a file transfer is wanted to be performed, first of all the Initiate FTP button must be clicked in order to inform the multi-port device. After that, an acknowledgement message should be waited to start file transfer. If the message is received and the file, which will be sent, is selected, the Send button can be clicked. By clicking Send button, the XmodemCRC file transfer protocol is initiated by TForm1.FsendClick () procedure. As soon as the handshake character, which is sent by the receiver (i.e. remote PC), is received, the XmodemCRC file transfer process starts and continues until sending <EOT> character and subsequently receiving <ACK> character by sender. After the file transfer is completed, the TForm1.Prt1ProtocolFinish() procedure is called.

If a message indicating the initiation of file transfer by remote PC is received, the file name should be specified by clicking the folder icon. After that by clicking Receive button, the file transfer process, which is initiated by remote PC, starts. During file reception process according to XmodemCRC protocol, TForm1.FreceiveClick() procedure is executed. After the file transfer is completed, the TForm1.Prt1ProtocolFinish() procedure is called.

The file transfer process performed by the remote PC follows the same steps and executes the same procedures as it can be seen clearly from the flow diagram. As the remote PC is not allowed to perform meter readout, all the related options are disabled under the condition of checked Slave Computer box.

5.2 Modem

Modem term is the acronym for **mo**dulator-**dem**odulator. A modem is a device or a program that enables a computer to transmit digital data in the form of analog waves over telephone lines.

There is a standard interface for connecting external modems to computers which is called as RS-232. By using this interface, any external modem can be attached to any computer that has an RS-232 port, which almost all personal computers have. RS-232 standard is explained comprehensively in Chapter 2. There are also onboard or internal modems as an expansion board that you can insert into a vacant expansion slot.

Dial-up Access refers to connecting a device to a network via a modem and a public telephone network. Dial-up access is really just like a phone connection, except that two ends are the computers (or the devices having at least an RS-232 port) rather than people. Because dial-up access uses normal telephone lines, the quality of the connection is not always good and data rates are limited. In the past, the maximum data rate with dial-up access was 56 Kbps (56,000 bits per second), but new technologies such as ISDN provide faster rates [Ref. 4].

An alternative way to connect two devices is through a leased line, which provides a permanent connection between two devices. Leased lines provide faster throughput and better quality connections, but they are also more expensive.

5.2.1 Communication Protocol

All communications between devices require that the devices agree on the format of the data. The set of rules defining a format is called as protocol. At least, a communication protocol must define the followings:

- Rate of transmission (in baud or bps).
- Whether transmission is to be synchronous or asynchronous.
- Whether data is to be transmitted in half-duplex or full-duplex mode.

In addition, protocols can include sophisticated techniques for detecting and recovering from transmission errors and for encoding and decoding data.

Table – 5.1 summarizes the most commonly used protocols for communication via modems [Ref. 13]. These protocols are almost always implemented in the hardware; that is, they are built into modems.

Table – 5.1 Summary of the most commonly used protocols

Protocol	Maximum Transmission Rate	Duplex Mode
Bell 103	300 bps	Full
CCITT V.21	300 bps	Full
Bell 212A	1,200 bps	Full
ITU V.22	1,200 bps	Half
ITU V.22bis	2,400 bps	Full
ITU V.29	9,600 bps	Half
ITU V.32	9,600 bps	Full
ITU V.32bis	14,400 bps	Full
ITU V.34	36,600 bps	Full
ITU V.90	56,000 bps	Full

In addition to the standard protocols listed in the table, there are numerous protocols that complement these standards by adding extra functions such as error detection, error recovery, data compression, etc. Some of these protocols are Xmodem, Kermit, MNP, and CCITT V.42 [Ref. 17]. These protocols can be implemented either in hardware or software.

Typical modems are known as asynchronous devices. This means that these devices transmit data in an intermittent stream of small packets. Once these are received, the receiving system then reassembles them into a form computer can use.

Stop 1 bit	Data 8 bits	Start 1 bit	Stop 1 bit	Data 8 bits	Start 1 bit
Packet 10 bits			Packet 10 bits		

Figure – 5.4 Asynchronous transmission packets

Figure – 5.4 represents the packet format in an asynchronous transmission over a phone line. In an asynchronous communication, 1 byte (8 bits) is transferred within 1 packet, which is equivalent to one character. However for the computer to receive this information each packet must contain a Start and a Stop bit therefore the complete packet would be 10 bits. The basics of serial communication is described more comprehensively in Chapter 2.

5.3 The Implemented Multi-port Device

5.3.1 General Description

The implemented multi-port device is designed based on the PIC 16F877, which is a microcontroller produced by Microchip Company and explained in Chapter 3. The multi-port device can be examined in three main parts as voltage regulator module, LCD module, and microcontroller module. These modules will be described in this section.

5.3.2 Voltage Regulator

The multi-port device includes an internal voltage regulation circuit. Owing to this voltage regulation circuit, any voltage adaptors between 12-24 V range can be used as a power supply for the multi-port device. Voltage regulation circuit diagram is shown in Figure – 5.5 [Ref. 15].

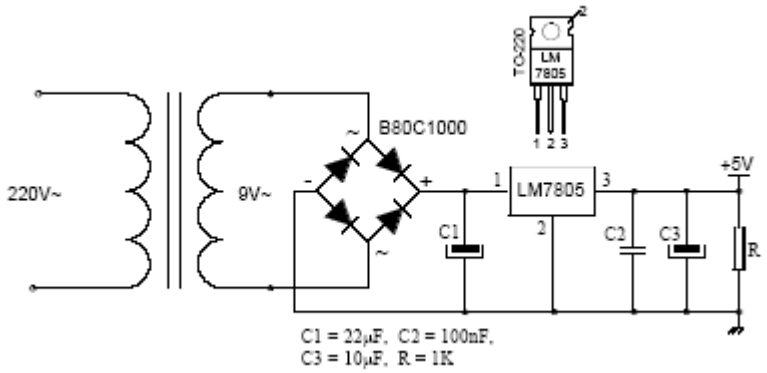


Figure – 5.5 Voltage regulation circuit

Although a voltage adaptor acquired from the market has its own regulation circuit, this extra regulation circuit is used because the microcontrollers require very pure and stabilized voltage level in order to operate properly. The required stabilized +5V is achieved by using the power stabilizer LM-7805. If the output voltage goes down, the LM-7805 regulator will draw more current, forcing the output voltage back to 5V. Thus, the LM-7805 regulator controls the output voltage and keeps it at 5V.

Because the microcontrollers require very clear voltage, a capacitor between V_{dd} and V_{ss} pins of the MCU is also used in order to filter the distortion from the circuit itself.

5.3.3 LCD Module

The LCD used in the multi-port device has 16-pin Hitachi LCD with HD44780 module. These LCD's are inexpensive and easy to use. Hitachi LCDs have a standard ASCII set of characters along with Japanese, Greek and mathematical symbols.

The LCD module can be wired for a 4-bit or an 8-bit mode. In PicBasic Pro, the LCD mode (4-bit or 8-bit) must be defined at the beginning of the program via "`DEFINE LCD_BITS`" instruction, please refer to microcontroller code in Appendix B. In 4-bit mode shown in Figure – 5.6, a byte is sent by microcontroller as two successive 4-bit over the lines D4-D7 default, but it can be changed to D0-D3 by code, if required [Ref. 9 and 15]. Four-bit mode is a good

way to save control lines. The entire LCD module can normally be controlled from 6 control lines. The initialization commands are constructed in such a way that you can instruct the LCD module which interface is to be used just after reset.

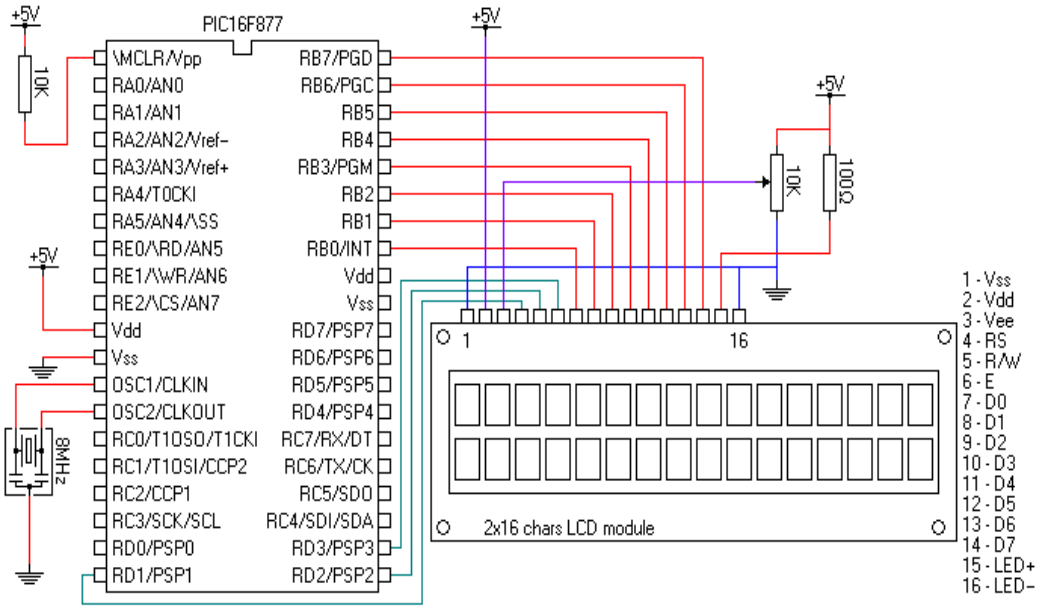


Figure – 5.6 General LCD connection diagram

For an 8-bit data bus, the display requires a +5V supply plus 11 I/O lines. For a 4-bit data bus it only requires the supply lines plus 7 I/O lines. When the LCD display is not enabled, data lines are tri-state which means that they are in a state of high impendence (as though they are disconnected) and this also means that they do not interfere with the operation of the microcontroller when the display is not being addressed.

The LCD requires mainly 3 control lines from the microcontroller:

Read/Write (R/W): This line determines the direction of data between the LCD and microcontroller. When it is low, data is written to the LCD. When it is high, data is read from the LCD.

Register Select (RS): With the help of this line, the LCD interprets the type of data on data lines. When it is low, an instruction is being written to the LCD, and when it is high, a character is being written to the LCD.

Enable (E): This line allows access to the display through R/W and RS lines. When this line is low, the LCD is disabled and ignores signals from R/W and RS. When (E) line is high, the LCD checks the state of the two control lines and responds accordingly.

Logic status on control lines:

E	0 Access to LCD disabled
	1 Access to LCD enabled
R/W	0 Writing data to LCD
	1 Reading data from LCD
RS	0 Instruction
	1 Character

Writing data to the LCD is done in several steps:

- Set R/W bit to low
- Set RS bit to logic 0 or 1 (instruction or character)
- Set data to data lines
- Set E line to high
- Set E line to low

The “LCDOUT” command, which performs the above explained steps, is used for writing data to the LCD in PicBasic Pro.

The definition of LCD data ports and control ports must be declared at the beginning of the program by “Define” command in PicBasic Pro code as shown below:

```
Define  LCD_DREG        PORTD       'LCD data port
Define  LCD_DBIT        0           'LCD data starting bit 0
Define  LCD_RSREG       PORTE       'LCD register select port
Define  LCD_RSBIT       0           'LCD register select bit
Define  LCD_EREG        PORTE       'LCD enable port
Define  LCD_EBIT        1           'LCD enable bit
```

5.3.4 Microcontroller Module

As mentioned previously the microcontroller used in the implemented multi-port device is the PIC 16F877, which is produced by Microchip Company.

There are several reasons to choose PIC 16F877 for this study. First of all the PIC 16F877 is a low-cost and high performance device with RISC CPU architecture. Owing to electrically erasable program memory it provides designing ease. Also it has very powerful I/O features and practical interfacing options with peripherals. Especially the USART module is a very feasible peripheral option for serial communication, which is the main issue of this study. Also owing to large number of I/O pins and their inverted logic feature, which is explained in Chapter 4, the necessity of 8 more (4 of them are in use) serial communication interfaces could be satisfied very practically. It is very obvious that the PIC 16F877 microcontroller is really a good choice for this study because of its powerful and practical interfacing features.

The auxiliary processes and components used with microcontroller in the implemented multi-port device are described in this section. The PIC 16F877 is described comprehensively in Chapter 3.

5.3.4.1 Reset Process

Reset is used for putting the microcontroller into a known condition. That practically means that microcontroller can behave inaccurately under certain undesirable conditions. In order to continue its proper functioning it has to be reset, that is all registers would be placed in a starting position. Reset is not only used when microcontroller does not behave as desired, but also can be used to get a microcontroller ready when loading a program.

The PIC 16F877 microcontroller has MCLR pin for resetting process, which is logic one during normal operation of microcontroller. When the microcontroller is required to be reset, the MCLR pin, which is an active-low pin, has to be made logic zero. In order to prevent from bringing a logical zero to

MCLR pin accidentally it has to be connected via resistor to the positive supply pole. Resistor should be between 5K and 10K. This kind of resistor, whose function is to keep a certain line on a logical one as a preventive, is called as pull up. If MCLR pin will not be used, it has to be connected to positive supply pole (i.e. to +5V) [Ref. 8].

During a reset, RAM memory locations are not being reset. They are unknown during a power up and are not changed at any reset. On the other hand, special function registers are reset to a starting position initial state. One of the most important effects of a reset is setting the program counter (PC) to zero (0000h), which enables the program to start executing from the first written instruction.

5.3.4.2 Oscillator

The microcontroller included by implemented multi-port device has the 4 MHz crystal oscillator operating together with two 33 pF capacitors. The crystal oscillator is a good choice for a reliable triggering required by the microcontrollers. The crystal is usually made of quartz, but can also be made of rubidium or ceramic. Crystal oscillators are the most common source of time and frequency signals. The crystal is sometimes called as timing crystal. They can be embedded in integrated circuits.

5.3.4.3 Serial Communication with USART

The communication between the implemented multi-port device and modem is fulfilled over MAX 232 level converter device by using the RC6 and RC7 pins of the microcontroller. RC6 and RC7 pins are the Rx and Tx pins of USART module respectively. The voltage level conversion process between microcontroller and modem is realized by MAX 232. PicBasic Pro supports the USART module with very powerful and practical commands, such as HSERIN and HSEROUT (mean that “Hardware Serial Input” and “Hardware Serial Output, respectively, please refer to PicBasic Pro code presented in Appendix B). The

detailed description of serial communication with USART module is presented in Chapter 4.

5.3.4.4 Serial Communication with Inverted Logic

In this thesis, the microcontroller has 5 serial interfaces (may increase to 9) with peripheral devices. As mentioned previously one of them is used for asynchronous serial communication with an external modem, which provides the connection to the main PC via USART module. Other 4 ports are specified for the connections to the digital electricity meters and the remote PC. As the PIC 16F877 microcontroller has only one USART module, these asynchronous serial communications must be fulfilled by some other ordinary I/O pins. Naturally there are some differences between these connections and the USART module connection. Inverted logic serial communication can be used very practically via PicBasic Pro, please refer to PicBasic code in Appendix B and check the "Serout2" and "Serin2" commands in the code. The detailed description of serial communication with inverted logic is presented in Chapter 4.

5.3.5 Operational Principles of Multi-port Device

When the implemented multi-port device is powered on, it will take less than one second to be ready for the operation. Then the device starts to wait for a communication initialization from the main PC. For that reason the device listens the Rx pin of USART module (RC7 pin of the PIC 16F877). The communication between main PC and multi-port device is performed with 4800-baud rate and 8N1 (1 start bit, 8 data bits, none parity and 1 stop bit). As soon as the multi-port device receives the communication initialization string from main PC, it triggers a port detection process by sending a specified string to all ports (8 RB pins of the PIC 16F877 utilized as serial port). If a meter is connected to a port, the meter will respond with its serial number. If a PC is connected to the port 1, it will respond by a specified string. If there is not a PC or a meter connected to a port, time-out will occur. After this process the multi-port device could detect the connected

peripherals and informs the user of main PC as which device is connected to which port.

Then the multi-port device starts to wait for the instruction from the main PC and remote PC. If an instruction is received, it will be evaluated immediately and the code loaded to the microcontroller will branch to the related part of the program according to the received instruction.

If the received instruction is the indication string of closing port process sent by main PC, the execution turns back to starting point and waits for the communication initialization again.

If the received instruction is the meter readout request, the serial communication with 4800-baud rate and 7E1 (1 start bit, 7 data bits, even parity and 1 stop bit) will be established between the multi-port device and the meter on the specified port. Then readout request string is sent to the meter. As soon as the meter receives readout request, it starts to transmit all standard readout data, if the meter is not password protected. All received characters by the multi-port device will be transmitted to the main PC over Tx pin of USART module (RC6 of PIC 16F877). After the readout is completed, the execution turns back to the point of waiting instruction.

If the received instruction is the file transfer request sent by main PC, the messages giving the instruction to start the file transfer process are sent to main PC and remote PC. As mentioned previously the file transfer process is fulfilled according to Xmodem protocol with 4800-baud rate and 8N1 (which is a requirement for Xmodem protocol [Ref. 17]). In file transfer mode, firstly, remote PC sends handshake character, which is <C> for CRC error checks, to the main PC, and waits 10 seconds for the transmitter to send a block of data. The main PC starts to send the blocks as soon as it receives handshake character. The remote PC responds with <ACK> character to each successfully received block. The file transfer will end after receiving <EOT> character by the remote PC. The details of Xmodem protocol is given in Appendix D.

If the received instruction is the file transfer request sent by remote PC, the above explained steps will be repeated as the remote PC is sender and the main

PC is receiver. After file transfer is completed, the multi-port device starts to listen to main PC and remote PC.

During the whole operation, all required messages giving the information about the current situation of the process for local recognition appears on the LCD.

The source code loaded to the microcontroller, which is written in PicBasic Pro language is presented in Appendix B. Figure – 5.7 shows the flow diagram of the source code.

5.3.6 Flow Diagram of Microcontroller Code

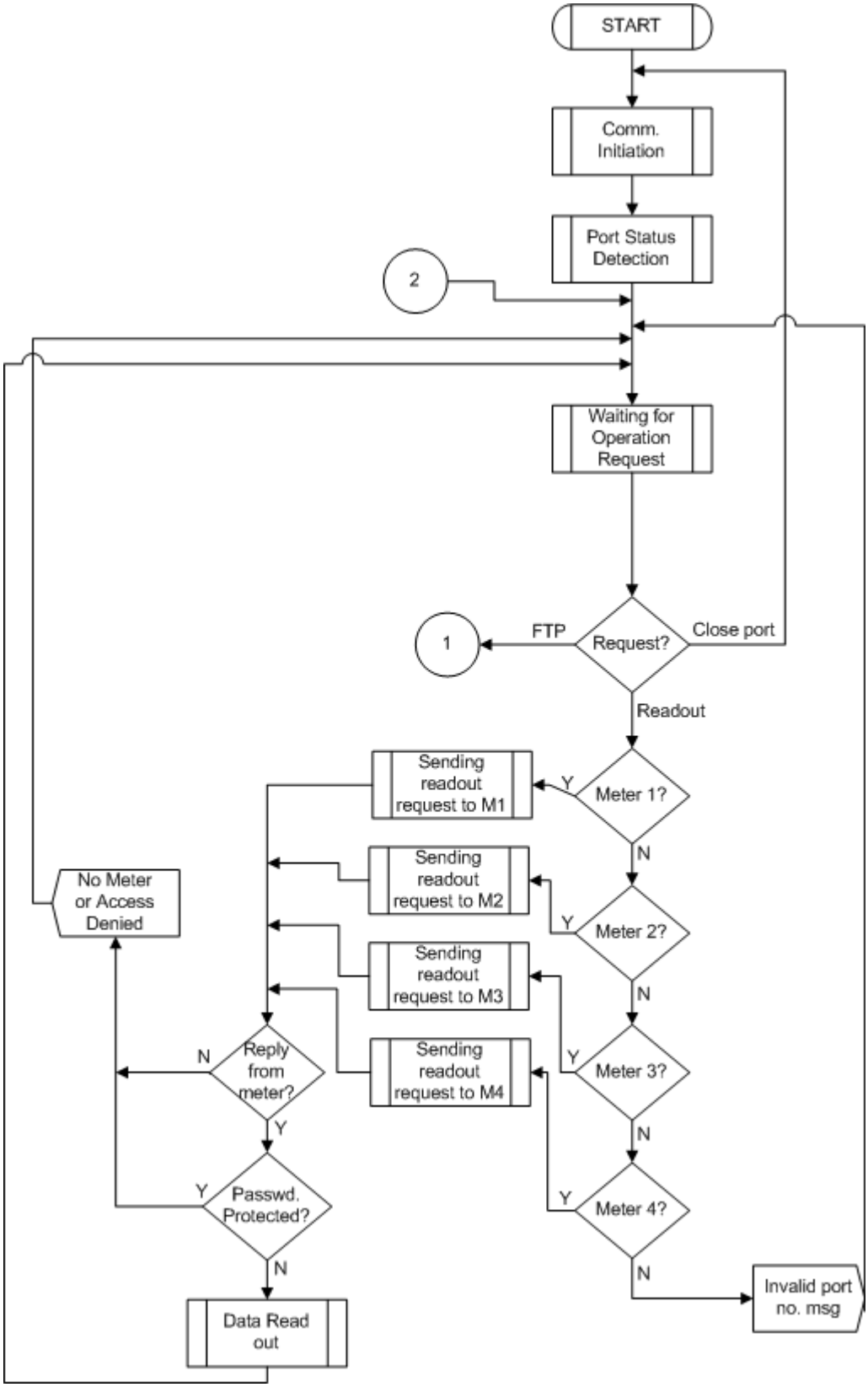


Figure – 5.7 Microcontroller code flow diagram (I)

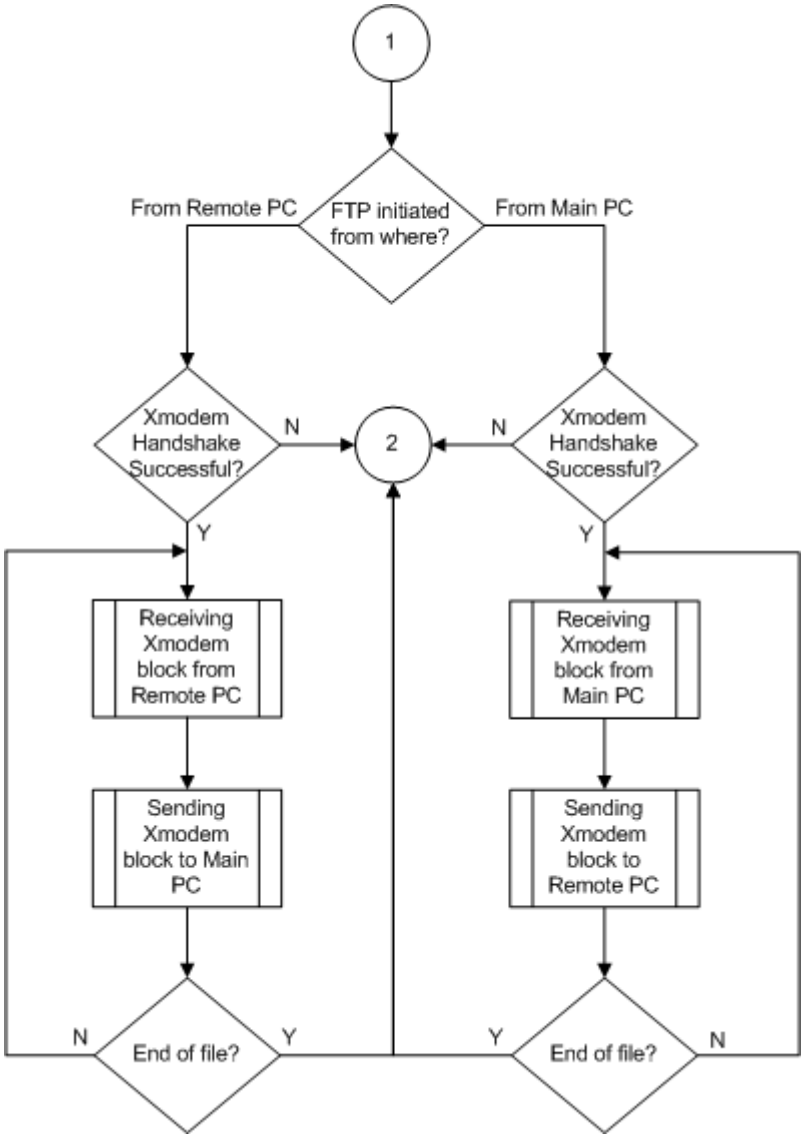


Figure – 5.7 Microcontroller code flow diagram (II)

It should be useful to describe the PicBasic code with the help of sample code fragments by following the flow diagram in order to give the theory of code.

After the implemented multi-port device is powered on, it will start to wait a communication initiation from main PC by means of the PicBasic code:

Hserin[wait("m"),b] .

Hserin command is used for receiving character by utilizing USART module. The parameters of this serial communication are defined at the beginning

of the program such as 4800 baud rate, 8 data bits, no parity bit and 1 stop bit by means of the code such as:

Define HSER_……..…..…..…

If the expected string for communication initiation is received, the port status detection process is triggered by multi-port device. During this operation, the device sends a specific string to each port by means of the code line:

Serout2 PortB.1,24764,[13,10,21,47,63,33,13,10]

then waits for the response by means of the code line:

Serin2 PortB.0,24764,2000,meterdetect1,[str a1\16]

These Serin2 and Serout2 commands can use the inverted logic property of PIC 16F877 I/O pins, which is defined by the mode 24764 in the code [Ref. 6 and 9]. This mode number also defines the parameters of this serial communication, such as 4800 baud rate, 7 data bits, Even parity and 1 stop bit. Please look at Appendix E for Serin2 and Serout2 mode definitions.

The devices connected to each port of implemented multi-port device are determined according to the received responses.

Then the multi-port device starts to wait for operation request, which can be sent by main PC or remote PC. The device performs this process by listening Rx pins of USART module and Port1 in a loop. The program branches from this point to the related part of the code according to what the request is and which PC sends it.

If the request is remote reading of a meter, the port, which the specified meter is connected to, is utilized and the data transfer between the multi-port device and meter is performed over this port according to Flag protocol. The received characters are sent to the main PC over the USART module. After the process is completed, the multi-port device turns back to wait for operation request.

If the request is to perform a file transfer between main PC and remote PC, the program branches to the code part related with Xmodem protocol. The file

transfer process can be performed from main PC to remote PC or from remote PC to main PC. There is only one difference, that is the direction of transfer, but the idea is same; receiving the Xmodem data blocks from sender, transmitting to receiver, and receiving the <ACK> or <NACK> from receiver, transmitting to sender. The below code performs this procedure:

```
        Hserout [13,10,"Ready for file transfer",13,10]
            pause 1000
        Serout2 PortB.1,16572,[13,10,"Be ready for receiving file",13,10]
            pause 1000
        Serin2 PortB.0,16572,30000,receivernotready,[nak]
        Hserout [nak]
loop1:
        Hserin 5000,nofiletransfer,[char1]
        Hserin 500,ftpcompleted,[char2]
        Hserin 1000,problem,[char3]
        Hserin 1000,problem,[str char\65]
        Hserin 1000,problem,[str a\65]
        Serout2 PortB.1,16572,[char1,char2,char3,str char\65,str a\65]
        Serin2 PortB.0,16572,11000,noreplyfromreceiver,[nak]
        Hserout [nak]
goto loop1
```

After the file transfer is completed or failed anyhow, the program turns back to wait for operation request. The file transfer, which is failed, can be initiated again at this point. If the communication is terminated by main PC, the program turns back to the point of waiting for communication initiation.

5.3.7 Circuit Diagram

The PCB (Printed Circuit Board) and the circuit diagram, which are designed and built in the scope of this thesis, are shown in Figure – 5.8 and Figure – 5.9 respectively. The PCB diagram was created by ARES tool in Proteus

Professional 6.0 after circuit diagram was drawn by ISIS tool again in Proteus Professional 6.0 [Ref. 18].

Proteus Professional 6.0 is a software package, which combines a mixed mode circuit simulator with animated component models. And it provides an architecture in which additional animated models may be created by users. Indeed, many types of animated model can be produced without coding. Proteus also provides simulator models for popular microcontrollers and a set of animated models for related peripheral devices such as LED, LCD, keypads, an RS232 terminal and more. It is possible to simulate complete microcontroller systems and thus to develop the software for them without access to a physical prototype. Consequently Proteus Professional 6.0 allows professional engineers to set up a circuit of real designs, run the circuit as interactive simulations and finally construct the PCB of the circuit.

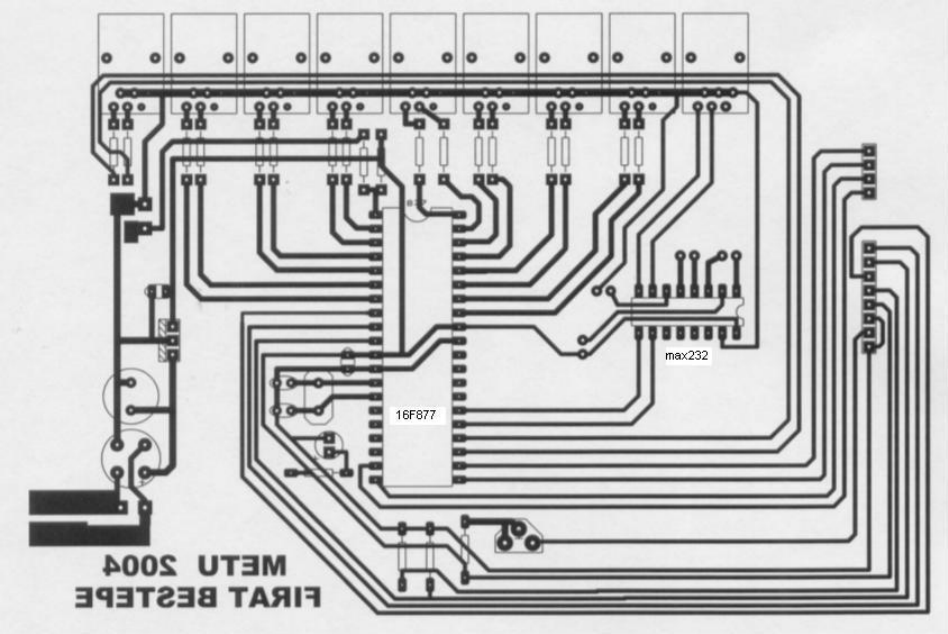


Figure – 5.8 Multi-port device PCB diagram

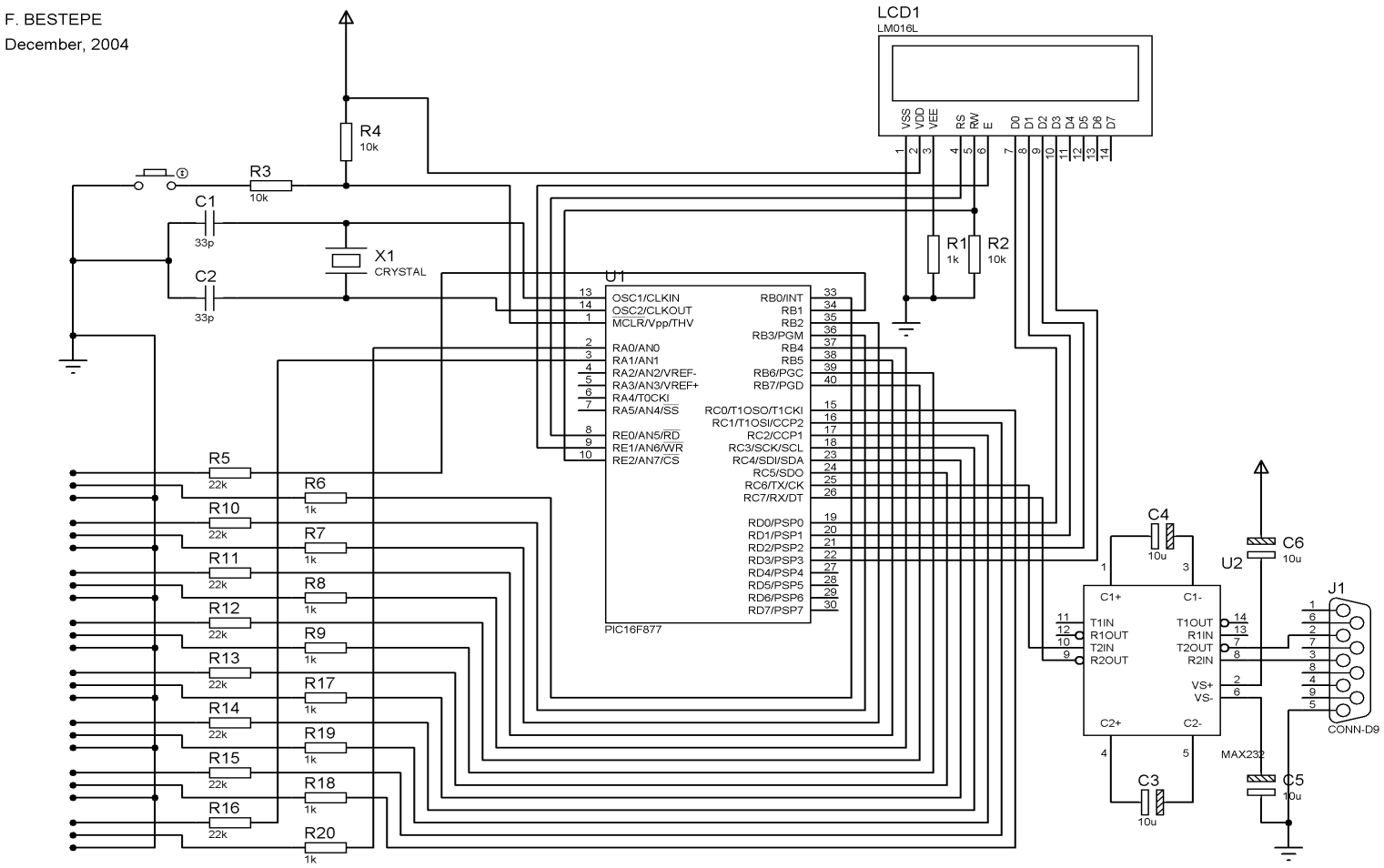


Figure – 5.9 Multi-port device circuit diagram

5.4 The Digital Electricity Meter

5.4.1 General Description

The electricity meters that operate according to electromechanical principles could not satisfy some requirements such as accuracy, reliability, lifetime and compatibility with electronic solutions. After electronics has made a remarkable development, electricity meters have been mass-produced using the most modern microelectronics in order to satisfy the requirements of present age.

Upon closer view, EMH meters, which are used with implemented multi-port device in this thesis, will be examined briefly. EMH is a digital electricity meter producer in Germany. EMH developed digital electricity meters, which are self-contained functioning tariff systems. These have little resemblance to the old meter technology.

The term ‘digital meter’ requires a brief explanation. The digital principle means that voltages and currents in individual phases are measured simultaneously. They are measured at a high sample and hold rate, and transformed into digital words. This information is then fed into a signal processor, which calculates all the desired values.

The control elements used, such as the real time clock with calendar and a universal ripple control receiver, which were previously external, are already integrated into the meter. The result is that a simple meter is transformed into a complete, high performance tariff system. The interfaces of the past have been extended by an optical fibre interface in this meter generation. The information important for the final customer of this system can be separated from the main meter with this interface and fed into an isolating relay box. The relay model can be chosen by the customer.

This concept allows that the relays, which were previously in the meter, can be removed to external boxes. This guarantees the maximum possible insulation from outward interference and also reduces the need for additional isolating relays in the meter cabinet. Before electronics arrived in the meters of the electricity providers, a classical measuring set for special contract customers usually looked like as following: For the measurement set there was a maximum

and a power distribution meter. In order to control the device according to tariffs, an external ripple control receiver or tariff time switch was added. And also the transmission contacts for the meter were separated by additional relays in the measuring set cabinet.

With the introduction of the combi-meter with integrated ripple control receiver, integrated real time clock and optical fibre interface, the costs can be reduced remarkably.

There is no doubt that introduction and equipping of a measurement point with this newest technology will lead to an increase in reliability and a reduction of error sources in the system.

5.4.2 EDIS (Energy Data Identification System)

The EDIS code system was created on the basis of the future-oriented efforts by the German Electrical Engineering Commission of the DIN and VDE (DKE) [Ref. 14]. The purpose of the EDIS is to provide unambiguous identification of data for display and processing originating from devices, which are produced by different manufacturers such as EMH, ACTARIS (Schlumberger), SIEMENS, ABB etc.

Some examples shown in Figure – 5.10 illustrate the structure of EDIS code system. As seen from the examples, the first digit, which can be any number between 1 and 8, defines the measurement, as 1 for active imported, 2 for active exported, 3 for reactive imported, 4 for reactive exported, 5 for reactive-inductive imported, 6 for reactive-capacitive imported, 7 for reactive-inductive exported and 8 for reactive-capacitive exported. The second digit defines which quantity is measured. It may indicate mean power, last recorded mean power, maximum demand and energy. Maximum demand is one of the most important values for plants and factories. If the consumer has a demand agreement with the energy provider, the consumer will be limited for importing power with this undertaken maximum demand value. If the limit is exceeded, the consumer will have to pay fine. So maximum demand value should be measured and controlled very

carefully. The third digit is related with the tariff indication. The last two digits show the historical values of measurement.

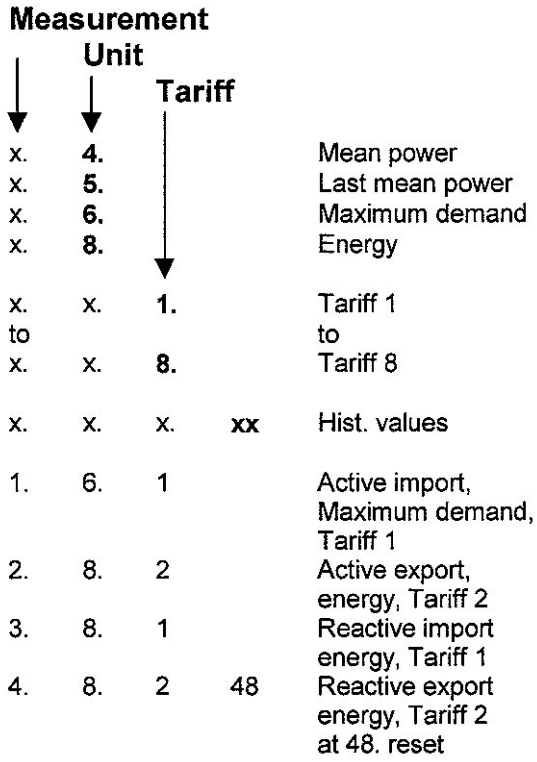


Figure – 5.10 General EDIS code system

5.4.3 Combi Meter / 4-Quadrant Meter

EMH combi meter and 4-quadrant meters are identical in appearance and belong to the same family of devices. The combi meter has become widely used. The 4-quadrant meter is more complex than combi-meter and is state of the art in hardware and software [Ref.14].

5.4.3.1 Combi Meter

The EMH combi meter replaces measurement sets, which consisted of two induction-type meters, an active use meter (which is coded as 1.x.x in EDIS code

system) and a reactive use meter (which is coded as 3.x.x in EDIS code system). In addition, the combi meter can perform reactive use measurement separately in quadrants 1 and 4 (which are coded as 5.x.x and 8.x.x in EDIS code system). The combi meter is capable of depicting the measurements according to the EDIS code system which is illustrated in Figure – 5.11 [Ref. 14].

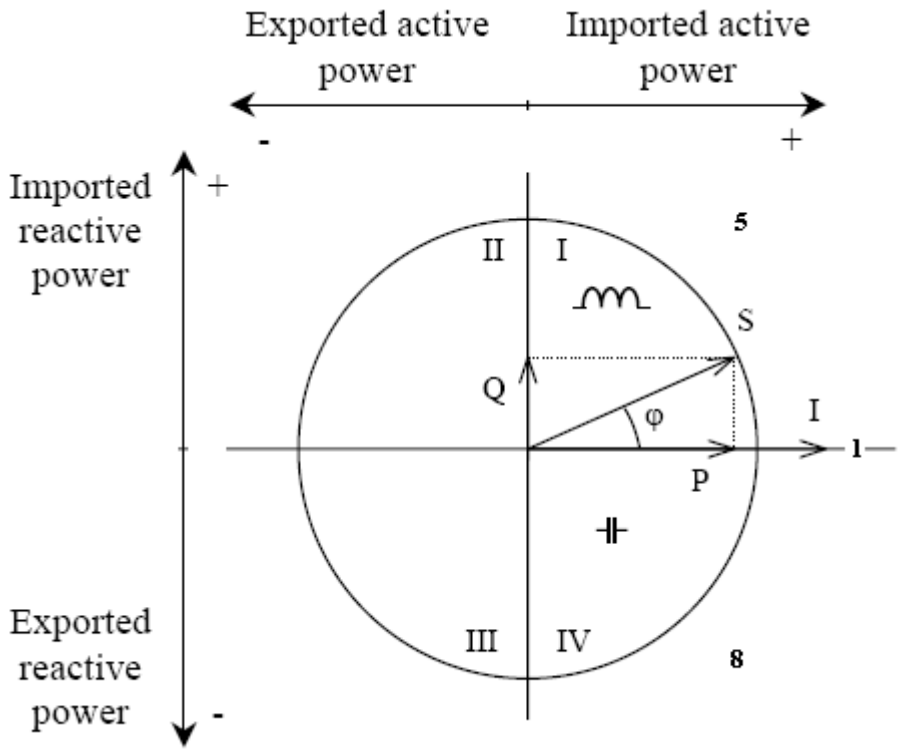


Figure – 5.11 EDIS coding for Combi-meter

5.4.3.2 Four-Quadrant Meter

The EMH 4-quadrant meter replaces the classical 4 induction-type meters each comprising an active consumption meter for imported and exported energy. The 4-quadrant meter is able to display the measurements, the codes of which are shown in Figure – 5.12 from the EDIS code system.

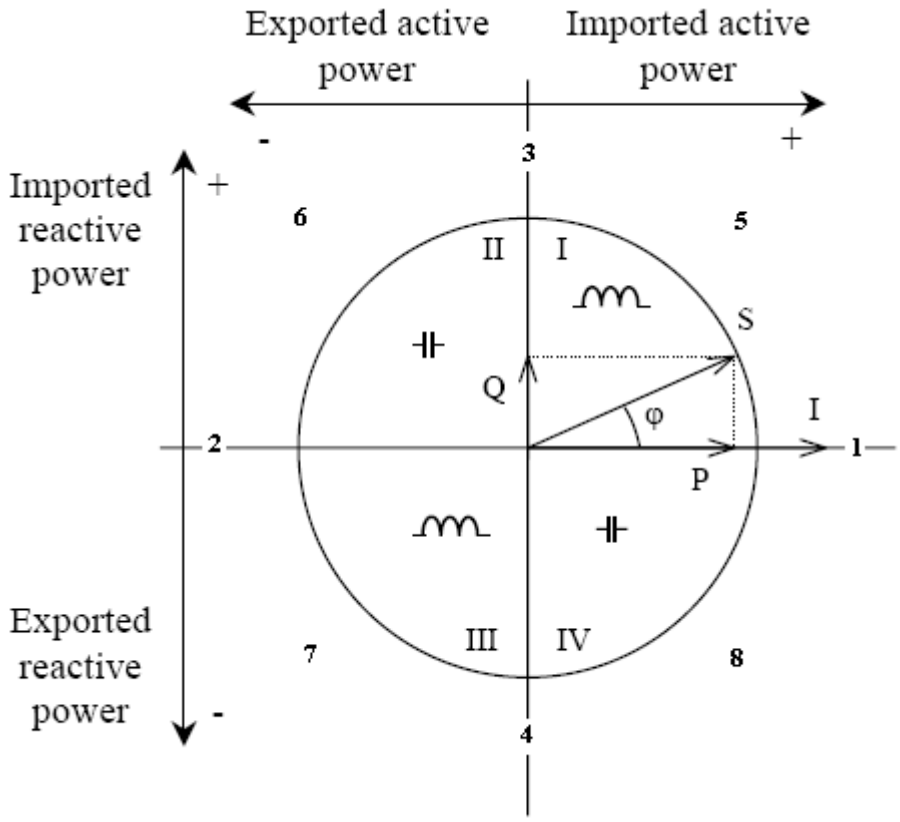


Figure – 5.12 EDIS coding for Four-quadrant meter

5.4.4 Modules

The meter consists of two essential elements as digital measuring mechanism and tariff mechanism.

The measuring mechanism is critical for the accuracy of the meter. It realizes the basic measurements, then transforms them into digital information and transmits them to the tariff mechanism for calculation and processing. EMH meters with LZ prefix belong to accuracy class 0.5S while with PZ prefix are class 0.2S (precision meters). Class 0.5S and 0.2S mean that in normal operation conditions the maximum probable measurement error will not exceed 0.5 % and 0.2 % respectively. The experience in meter testing technology has been used to obtain this high degree of meter accuracy. Both the measuring and the tariff mechanisms are charged with a common power unit.

5.4.4.1 Power Unit

This is a primary switched-mode power unit with a high degree of effectiveness. Even in case of a module defects it is secure against overload or short-circuiting. Potential damage remains limited and consequential damage is avoided.

For EMH meters with LZ prefix with a single-phase connected meter, error-free operation until U_{nom} - 20% is guaranteed. EMH meters with PZ prefix (precision meters) have a wide area power unit that is supplied from external auxiliary power terminals, independent of the measurement path. This prevents unnecessary load on the measurement path and that the meter's measurement path remains functional and ready to measure even in the event of power failure.

While the meter is operating, the current measurements are stored in the working memory (RAM). Every 24 hours, this data is stored in a non-volatile memory. The meter also performs this storage operation when there is a 20% drop below nominal voltage or a total power interruption (loss of 50 Hz signal). In the event of a power outage or drop below the minimum voltage, the electronics will continue to function normally for the next 500 ms. It is powered by the energy in the charger-capacitor. If it is only a short outage of less than 500 ms, then the meter will continue to operate quasi-non-stop. Only in case of longer interruptions will the measuring period be interrupted and the device completely shut down so that a new measuring period can be started when the meter is reactivated. Data remains stored in the non-volatile memory for at least ten years. No buffer battery is needed to preserve the data. The data received is retained alone through the qualities of the storage medium (EEPROM).

5.4.4.2 Suppressor Circuits

The suppressor circuit behind the voltage terminals consists of a combination of surge-proof power resistors and varistors, which dilute the surge energy in the event of an overvoltage. This means that fast, energy-rich

disturbance pulses, which can be caused by turning off reactive loads, are prevented effectively thanks to the microelectronics.

5.4.4.3 Modular Construction

The entire measuring and tariff mechanism including the options, such as clock module, tariff time switch, ripple control receiver and electrical interface, control inputs and outputs. They are all included on a single circuit board. The modular structure of the entire meter means that the meter can be assembled to perform functions in accordance with the customer's desired meter properties. The display is plugged onto the circuit board and can be easily replaced.

The meters have a memory (EPROM or FLASH) containing the program code and an additional memory (EPROM) for storing the relevant billing data, parametering, settings and correction factors for the measuring mechanism which operate independent of any battery.

EMH meters with LZ prefix with optical fibre interface or load profiles have an additional circuit board. On this circuit board is a FLASH containing the program code. Depending on the meter version, there may be a further FLASH for the load profile.

5.4.5 Digital Measuring Mechanism

5.4.5.1 Measurement principle

Measurement is performed by taking voltage and current measurements at very short intervals. The analog voltage and current values are then digitized. The digitized current and voltage values are then transmitted individually to a digital signal processor by means of a multiplexer.

5.4.5.2 Voltage measurement

The terminal voltages generate network-proportional internal voltage levels. These are fed into the input channels of the ADC.

5.4.5.3 Current measurement

The current paths contain ferrite cores that generate current-proportional voltages. These are fed into the two ADC inputs via an instrument amplifier.

5.4.5.4 Digitization

The analog current and voltage proportional instantaneous values are digitized in the ADCs and fed into the digital signal processor (DSP) input through a multiplexer with a high sampling rate.

5.4.5.5 Integral values

The DSP calculates the individual integral of i^2, u^2, u * i and u_{90^o} * i for each phase. These integral values and other information are transmitted to the tariff mechanism and in particular to the tariff controller CPU. Here the P, Q and S for each phase are calculated and assigned to the respective energy and power registers according to the tariff structures configured.

5.4.5.6 Measurement values

The following measurement values can be seen on the display and read out using the data interfaces:

- Instantaneous active, reactive, apparent power for each phase and the combined value,
- Individual line current and line voltages,
- Number of active phases, network frequency and power factor as well as power factor for individual phases.

5.4.5.7 Calibration

EMH's combi meter and 4-quadrant meter are fully static and digital meters. In practice, this means that there are no mechanical moving parts in the measuring device. This also means that the electronic components' tolerances are so matched to each other that a partial calibration between manufacturing steps is unnecessary. Thus the devices can be produced rationally in identical series.

At the end of the production process, the meters are subjected to a final calibration. The meters are submitted to precise normal load on the test stand. Each meter measures this load and transmits its measurement to the test stand via an optical interface. This compares the meter's measurement with its own precise measurement and sends measurement correction factors back to the various points in the meter in the form of measurement constants. These are then stored in the non-volatile memory elements of the meter. This allows the error curve to be improved by offset corrections [Ref. 14].

5.4.6 Tariff Mechanism

Using digitized measurement values, the tariff mechanism calculates electricity consumed or supplied as well as electrical power. It then assigns it to the respective energy or power register according to the tariff control and meter configuration provided.

5.4.7 Data Interface

The data exchange between meters and read out devices is performed by means of the electrical interface (RS-232, RS-485), which is located on the meter’s auxiliary terminal below the sealable terminal cover. The transmission is fixed between 300 – 4800 baud rate and 7E1 (1 stop bit, 7 data bits, even parity bit and 1 stop bit).

CHAPTER 6

CONCLUSION

In the scope of this thesis, a general overview of a remote reading infrastructure for digital electricity meters was given. Then, the design and implementation of a multi-port device, which can reduce the cost of this infrastructure and also simplify it effectively by reducing the number of device used in the system, were described. And also an alternative application of implemented multi-port device, which provides interfacing with a PC together with digital electricity meters, was described.

The operating principles of auxiliary devices used in the system such as modems and digital electricity meters were explained along with implemented multi-port device.

A remote access software, which performs remote reading of digital electricity meters and file transfer between two remote computers, was described. This software was developed in Delphi 7.0 by using APRO (Asynchronous Professional) Libraries.

As the serial communication was the main issue of the design of this multi-port device, the fundamental principles of serial communication with microcontrollers were explained comprehensively. Also the file transfer process between two computers performed over the multi-port device according to Xmodem protocol is described.

The PIC 16F877 microcontroller, which is used in the design, was described in detail. Additionally, the LCD usage, which is one of the popular applications realized with microcontrollers, is also mentioned. The source code of microcontroller written in PicBasic Pro including the procedures of serial communication and LCD application is also presented.

As a future work, the multi-port device can be improved by providing meter readout capability to remote PC with only the help of code modification. Besides, more advanced file transfer protocols with the aspect of speed and error

tolerance, such as Zmodem can be implemented with multi-port device. In fact the multi-port device can be used in order to communicate with more than one device possessing a serial communication port at the same time. Owing to this feature, the multi-port device can be utilized in lots of applications by making required modifications in the code.

REFERENCES

[1] Peatman, J.B. 2003, Embedded Design with the PIC 18F452 Microcontroller, Prentice Hall.

[2] Hintz, K. , Tabak, D. 1992, Microcontrollers : Architecture, Implementation & Programming, McGraw Hill.

[3] Kheir, Michael 1997, The M68HC11 Microcontroller: Applications in control, instrumentation and communication, Prentice Hall.

[4] Anderson, D., Dawson, P., Tribble, M., The Modem Technical Guide

[5] Doğan , İbrahim, PIC BASIC : Programlama ve Projeler, İstanbul Bileşim Yayınları

[6] Altınbaşak, Orhan 2002, PicBasic Pro ile PIC Programlama, Altaş Basım Yayın.

[7] Altınbaşak, Orhan 2003, Microdenetleyiciler ve PIC Programlama, Altaş Basım Yayın.

[8] http://www.microchip.com/

[9] http://picbasic.com

[10] http://www.picbasic.co.uk/forum/

[11] http://www.turbopower.com

[12] http://www16.boulder.ibm.com/pseries/en_US/aixasync/asycomgd/mastertoc.htm#mtoc

[13] http://readthetruth.com/modems.htm

[14] http://www.emh-meter.de

[15] http://www.microelektronika.co.yu

[16] http://www.mecanique.co.uk/code-studio/

[17] http://www.totse.com/en/technology/telecommunications/wxmodem.html

[18] http://labcenter.co.uk

[19] http://www.calldirect.at

[20] http://www.adaptivenetworks.com

[21] http://www.enermet.com

[22] http://www.perax.fr

APPENDIX A – READING SOFTWARE SOURCE CODE

```
unit mdmshare;

interface

uses
  WinTypes, WinProcs, SysUtils, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, AdPort, OoMisc, AdStatLt, Buttons, ExtCtrls,
  cxControls, cxContainer, cxEdit, cxTextEdit, cxMaskEdit, cxDropDownEdit,
  cxImageComboBox, Mask, sCustomComboEdit, sTooledit, AdProtcl, AdPStat,
  sButtonControl, sCustomButton,Registry, ComCtrls, Menus;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    cp2: TApdComPort;
    lc1: TApdSLController;
    ApdStatusLight1: TApdStatusLight;
    ApdStatusLight2: TApdStatusLight;
    ApdStatusLight3: TApdStatusLight;
    ApdStatusLight4: TApdStatusLight;
    ApdStatusLight5: TApdStatusLight;
    ApdStatusLight6: TApdStatusLight;
    ApdStatusLight7: TApdStatusLight;
    ApdStatusLight8: TApdStatusLight;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Edit2: TEdit;
    Dial: TBitBtn;
    Rdevice: TBitBtn;
    Pclose: TBitBtn;
    Panel1: TPanel;
    cbIslem: TcxImageComboBox;
    Clstxt: TButton;
    Panel2: TPanel;
    Fedit1: TsFilenameEdit;
    Fsend: TsBitBtn;
    Freceive: TsBitBtn;
    Fedit2: TsFilenameEdit;
    Prt1: TApdProtocol;
```

```
  PrtSt: TApdProtocolStatus;
  CbComfind: TComboBox;
  Memo2: TMemo;
  PBar: TProgressBar;
  PMenu1: TPopupMenu;
  SavetoTxt1: TMenuItem;
  ClearTxt1: TMenuItem;
  SDlg: TSaveDialog;
  Chkbox: TCheckBox;
  Btnopen: TButton;
  Btnftp: TButton;
  Edit1: TEdit;
  Label9: TLabel;
  Label10: TLabel;
  Edit3: TEdit;
  Button4: TButton;
  procedure cp2PortOpen(Sender: TObject);
  procedure cp2TriggerAvail(CP: TObject; Count: Word);
  procedure RdeviceClick(Sender: TObject);
  procedure PcloseClick(Sender: TObject);
  procedure cp2PortClose(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormCreate(Sender: TObject);
  procedure ClstxtClick(Sender: TObject);
  procedure Fedit1Change(Sender: TObject);
  procedure FsendClick(Sender: TObject);
  procedure Fedit2Change(Sender: TObject);
  procedure FreceiveClick(Sender: TObject);
  procedure Prt1ProtocolFinish(CP: TObject; ErrorCode: Integer);
  procedure CbComfindDropDown(Sender: TObject);
  procedure DialClick(Sender: TObject);
  procedure cp2TriggerData(CP : TObject; TriggerHandle : Word);
  procedure cp2TriggerTimer(CP: TObject; TriggerHandle: Word);
  procedure Tmr(ms:longint);
  procedure ClearTxt1Click(Sender: TObject);
  procedure SavetoTxt1Click(Sender: TObject);
  procedure ChkboxClick(Sender: TObject);
  procedure BtnopenClick(Sender: TObject);
  procedure BtnftpClick(Sender: TObject);
  procedure Button4Click(Sender: TObject);
private
  { Private declarations }
  procedure SignOn;
  procedure Comfind;
  procedure ComEnd;

public
  { Public declarations }
```

```
  end;

var
  Form1: TForm1;
  S: String = '';
  TF:Boolean = False;

     MCnttrg : Word;
     DCnttrg : Word;
     Timeout : Word;
     TCnttrg : Word;
     TDCnttrg : Word;
     PCnttrg : Word;
     RCnttrg :Word;



const
   bos   = #0;
   SOH = #1;
   STX  = #2;
   ETX  = #3;
   EOT  = #4;
   ENQ = #5;
   ACK = #6;
   BEL   = #7;
   BS    = #8;
   TAB   = #9;
   LF    = #10;
   VT    = #11;
   FF    = #12;
   CR    = #13;
   SO    = #14;
   SI    = #15;
   DLE  = #16;
   DC1   = #17;
   DC2   = #18;
   DC3   = #19;
   DC4   = #20;
   NAK = #21;
   SYN  = #22;
   ETB  = #23;
   CAN = #24;
   EM    = #25;
   SUB   = #26;
   ESC   = #27;
   FS   = #28;
   GS    = #29;
```

```
   RS    = #30;
   US    = #31;
   SP    = #32;


implementation

uses Windows;

{$R *.DFM}



procedure TForm1.cp2PortOpen(Sender: TObject);
begin
     lc1.ComPort:=cp2;
     lc1.Monitoring:=True;
     Panel1.Color:=clGreen;
end;

procedure TForm1.cp2TriggerAvail(CP: TObject; Count: Word);
var
  I : Word;
  C : Char;
  S : String;
begin
  S := '';
  for I := 1 to Count do
  begin
    C :=cp2.GetChar;
    S := S+C;
  end;
    memo1.Lines.Text:=memo1.Lines.Text+S;
    end;


procedure TForm1.RdeviceClick(Sender: TObject);
begin
  if cbIslem.Properties.Items[cbIslem.ItemIndex].Tag=1 then
  cp2.OutPut := 'm1'
  else
  begin
   if cbIslem.Properties.Items[cbIslem.ItemIndex].Tag=2 then
   cp2.OutPut := 'm2'
   else
    begin
    if cbIslem.Properties.Items[cbIslem.ItemIndex].Tag=3 then
    cp2.OutPut := 'm3'
```

```
    else
    cp2.Output := 'm4';
    end;
   end;
end;


procedure TForm1.PcloseClick(Sender: TObject);
begin
  cp2.Output:='m9';
  Tmr(100);
  ComEnd;
end;

procedure TForm1.cp2PortClose(Sender: TObject);
begin
  Panel1.Color:=clRed;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  cbIslem.ItemIndex :=0;
  Btnopen.Visible :=False;
  Btnftp.Visible :=False;


end;

procedure TForm1.ClstxtClick(Sender: TObject);
begin
  memo1.Text:= '';
  memo2.Text:='';
  Pbar.Position:=0;
end;

procedure TForm1.Fedit1Change(Sender: TObject);
begin
 Prt1.FileMask := Fedit1.FileName;
end;

procedure TForm1.FsendClick(Sender: TObject);
begin
 PrtSt.Visible:=True;
 Tmr(200);
 Prt1.ProtocolType := ptXmodemCRC ;
 Prt1.StartTransmit;
 S:='';
 Memo1.Text:='';
```

```
 Memo1.Visible:=False;
 Pclose.Enabled:=False;
end;

procedure TForm1.Fedit2Change(Sender: TObject);
begin
 Prt1.FileName:= Fedit2.FileName;
end;

procedure TForm1.FreceiveClick(Sender: TObject);
begin
  S:='';
  Prt1.ProtocolType := ptXmodemCRC;
  memo1.Visible:=False;
  Pclose.Enabled:=False;
  Prt1.StartReceive;


end;

procedure TForm1.Prt1ProtocolFinish(CP: TObject; ErrorCode: Integer);
begin
PrtSt.Visible:=False;
Memo1.Text:='';
Memo1.Visible:=True;
PClose.Enabled:=True;
end;

procedure TForm1.SignOn;
begin
  Application.ProcessMessages;
  cp2.ComNumber:=
StrToInt(Copy(CbComfind.Text,4,Length(CbComfind.Text)-3));
  S:='';
  cp2.Open:=True;
  cp2.OutPut := 'ATDT'+ Edit2.Text+CR;
  MCnttrg := cp2.AddDataTrigger('CONNECT 4800', False);
  DCnttrg := cp2.AddDataTrigger('CNT', False);

  PCnttrg := cp2.AddDataTrigger('/?!',True);
  RCnttrg := cp2.AddDataTrigger('FTP Completed',True);
  TimeOut:= cp2.AddTimerTrigger;
  TCnttrg:= cp2.AddTimerTrigger;
  TDCnttrg:= cp2.AddTimerTrigger;

  Pbar.Position:=0;
end;
```

```
procedure TForm1.CbComfindDropDown(Sender: TObject);
begin
Comfind;
end;

procedure TForm1.Comfind;
var
  reg : TRegistry;
  ts,PL : TStrings;
  i : integer;
begin

  reg := TRegistry.Create;
  reg.RootKey := HKEY_LOCAL_MACHINE;
  reg.OpenKey('hardware\devicemap\serialcomm',false);
  ts := TStringList.Create;
  PL := TStringList.Create;
  reg.GetValueNames(ts);
  for i := 0 to ts.Count -1 do
  begin
    PL.Add( reg.ReadString(ts.Strings[i]) );
  end;
  ts.Free;
  reg.CloseKey;
  reg.free;
  CbComfind.Items:=PL;
  end;

procedure TForm1.ComEnd;
begin

  cp2.Open:=False;
  Dial.Enabled:=True;
  Pbar.Position:=0;
end;


procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  cp2.Open:=False;
  Application.Terminate;
end;


procedure TForm1.DialClick(Sender: TObject);
var i:integer;

begin
```

```
  s:='';
  i:= strtoint(Edit1.Text);
  Dial.Enabled:=False;
  SignOn;
  cp2.SetTimerTrigger(Timeout, i, True);

end;


procedure TForm1.cp2TriggerData(CP : TObject; TriggerHandle : Word);
var i,t:integer;
begin

   PBar.Min:=1;
   PBar.Max:=12;
   i:=0;
   t:= StrToInt(Edit3.Text);
    if TriggerHandle = MCnttrg then
    begin
    cp2.SetTimerTrigger(Timeout, 36, False);
    Pclose.Enabled:=False;
    TF:= True;
    While TF = True do
      begin
      i:=i+1;
      Pbar.Position:= i;
      cp2.Output:='m9';
      Tmr(2000);
      cp2.Output:='m9';
      Tmr(2000);
      cp2.Output:='m0';
      Tmr(5000) ;
      if i=12 then
        begin
        TF:=False;
        ComEnd;
        Memo2.Lines.Add('Not Connected to Device '+ DateTimeToStr(Now));
        end
      end
    end
    else if TriggerHandle = DCnttrg then
    begin
    TF:= False;
    Cp2.SetTimerTrigger(TDCnttrg,t,True);
    end

    else if TriggerHandle = PCnttrg then
    begin
```

```
   cp2.output:='Comp';
   end
   else if TriggerHandle = RCnttrg then
   begin
   PrtSt.Visible:=False;
   Memo1.Lines.Add('FTP Completed');
   end;
  end;

procedure TForm1.Tmr ;
var t : longint;
begin
 t:= GetTickCount;
 while (GetTickCount -t)< ms do
  Application.ProcessMessages;
 end;


procedure TForm1.cp2TriggerTimer(CP: TObject; TriggerHandle: Word);
begin
  if TriggerHandle = Timeout then
  begin
    Memo2.Lines.Add('Modem Time Out.. '+ DateTimeToStr(Now));
    ComEnd;
    end
  else if TriggerHandle = TDCnttrg then
  begin
   Memo2.Lines.Add('Connected to device '+ DateTimeToStr(Now));
   Pclose.Enabled:=True;
    Pbar.Position:=0;

  end;

 end;

procedure TForm1.ClearTxt1Click(Sender: TObject);
begin
Memo1.Text:='';
end;

procedure TForm1.SavetoTxt1Click(Sender: TObject);
begin
if SDlg.Execute then
  Memo1.Lines.SaveToFile(SDlg.FileName);
end;
```

```
procedure TForm1.ChkboxClick(Sender: TObject);
begin

If Chkbox.Checked =True then
Edit2.Visible:=False;
Dial.Visible:=False;
Clstxt.Visible:=False;
cbIslem.Visible:=False;
Rdevice.Visible:=False;
Btnopen.Visible:=True;
Btnftp.Visible:=True;
Label9.Visible:=False;
Label10.Visible:=False;
Edit1.Visible:=False;
Edit3.Visible:=False;
Button4.Visible:=False;
if Chkbox.Checked =False then
begin
Edit2.Visible:=True;
Dial.Visible:=True;
Clstxt.Visible:=True;
cbIslem.Visible:=True;
Rdevice.Visible:=True;
memo1.Visible:=True;
memo2.Visible:=True;
Btnopen.Visible:=False;
Btnftp.Visible:=False;
Label9.Visible:=True;
Label10.Visible:=True;
Edit1.Visible:=True;
Edit3.Visible:=True;
Button4.Visible:=True;
end
end;

procedure TForm1.BtnopenClick(Sender: TObject);
begin
cp2.ComNumber:= StrToInt(Copy(CbComfind.Text,4,Length(CbComfind.Text)-
3));
cp2.Open:=True;
PCnttrg := cp2.AddDataTrigger('CRLFNAK/?!CRLF',False);
end;

procedure TForm1.BtnftpClick(Sender: TObject);
begin
If cp2.Open = False then
begin
Memo2.Lines.Add('The Port Is Not Opened '+ DateTimeToStr(Now));
```

```
end
else if cp2.Open = True then
cp2.Output:='m6';
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
cp2.Output:='m5';
end;


end.
```

APPENDIX B – SOURCE CODE IN PICBASIC

```
Include "modedefs.bas"


Define  OSC 4
Define LOADER_USED      1

Define  LCD_BITS        4               'LCD bus size 4 or 8'
Define  LCD_DREG        PORTD           'LCD data port'
Define  LCD_DBIT        0               'LCD data starting bit 0'
Define  LCD_RSREG       PORTE           'LCD register select port'
Define  LCD_RSBIT       0               'LCD register select bit'
Define  LCD_EREG        PORTE           'LCD enable port'
Define  LCD_EBIT        1               'LCD enable bit'

Define  HSER_RCSTA      90h     'Hser receive status init'
Define  HSER_TXSTA      20h     'Hser transmit status init'
Define  HSER_BAUD       4800    'Hser baud rate'
Define  HSER_CLROERR    1       'Hser CLEAR overrun error'

Define  SER2_BITS       8       'Set number of data bits for Serin2'
                                'and  Serout2'

a1      var     byte[16]
char0   var     byte

char    var     byte[65]
a       var     byte[65]
b       var     byte

char1   var     byte
char2   var     byte
char3   var     byte

nak     var     byte


    ADCON1 = 7      'Set PORTA and PORTE to digital, otherwise portA ADC'
    Low PORTE.2     'LCD R/W line low (W)'
    Pause 500       'Wait for LCD to startup'

    Lcdout $fe, 1, "  Waiting for"
    Lcdout $fe, $C0, "initialization!"
```

```
‘Waiting for communication initialization by listening Tx port of USART’

start:
Hserin 6000,notinitialized,[wait ("m"), b]

Hserout ["CNT"]
        Lcdout $fe, 1, "  Communication "
        Lcdout $fe, $C0, "  initialized!  "

pause 1000


‘Port status detection procedure’

Serout2 PortB.1,24764,[13,10,21,47,63,33,13,10]
Serin2 PortB.0,24764,2000,meterdetect1,[str a1\16]
Hserout [13,10,"Port 1 = Meter",13,10]

detect2:
Serout2 PortB.3,24764,[13,10,21,47,63,33,13,10]
Serin2 PortB.2,24764,2000,meterdetect2,[str a1\16]
Hserout [13,10,"Port 2 = Meter",13,10]

detect3:
Serout2 PortB.5,24764,[13,10,21,47,63,33,13,10]
Serin2 PortB.4,24764,2000,meterdetect3,[str a1\16]
Hserout [13,10,"Port 3 = Meter",13,10]

detect4:
Serout2 PortB.7,24764,[13,10,21,47,63,33,13,10]
Serin2 PortB.6,24764,2000,meterdetect4,[str a1\16]
Hserout [13,10,"Port 4 = Meter",13,10]

loop:
      Lcdout $fe, 1, "Status of Ports "
      Lcdout $fe, $C0, "  detected!  "


pause 2000
      Lcdout $fe, 1, "  Waiting for"
      Lcdout $fe, $C0, "  request!!! "
      Hserout [13,10,"Multi-port device is waiting for request!",13,10]



‘Getting the operation request by listenning the USART serial port and’
‘ inverted logic serial port in a loop’
```

```
listenstart:

Hserin 2000,listennext,[wait ("m"),char0]
goto listen

listennext:
Serin2 PortB.0,24764,2000,listenstart,[wait ("m"),char0]
goto listen


listen:

Select Case char0
   Case "1"

      Lcdout $fe, 1, " Data Receiving "
      Lcdout $fe, $C0, " from 1    "

      ‘Send the readout string to meter’
      Serout2 PortB.1,24764,[47,63,33,13,10]

      ‘Start to receive data(meter id), otherwise branch to noconnection’
      Serin2 PortB.0,24764,3010,noconnection,[str a1\16]
      Hserout [str a1\16]
      pause 10


     ‘Data receiving after getting Meter id from meter, if ended goto datason’
Don1:
      Serin2 PortB.0,24764,3010,datason,[char0]
      Hserout [char0]
      Goto Don1



   Case "2"

      Lcdout $fe, 1, " Data Receiving "
      Lcdout $fe, $C0, " from 2    "

      Serout2 PortB.3,24764,[47,63,33,13,10]

      Serin2 PortB.2,24764,3000,noconnection,[str a1\16]
      Hserout [str a1\16]
      pause 10
```

```
Don2:
        Serin2 PortB.2,24764,3000,datason,[char0]
        Hserout [char0]
        Goto Don2



    Case "3"

        Lcdout $fe, 1, " Data Receiving "
        Lcdout $fe, $C0, " from 3    "

        Serout2 PortB.5,24764,[47,63,33,13,10]
        pause 10

        Serin2 PortB.4,24764,3000,noconnection,[str a1\16]
        Hserout [str a1\16]
        pause 10


Don3:
        Serin2 PortB.4,24764,3000,datason,[char0]
        Hserout [char0]
        Goto Don3



    Case "4"

        Lcdout $fe, 1, " Data Receiving "
        Lcdout $fe, $C0, " from 4    "

        Serout2 PortB.7,24764,[47,63,33,13,10]
        pause 10

        Serin2 PortB.6,24764,3000,noconnection,[str a1\16]
        Hserout [str a1\16]
        pause 10


Don4:
        Serin2 PortB.6,24764,3000,datason,[char0]
        Hserout [char0]
        Goto Don4



    Case "5"
```

```
‘File transfer procedure is initiated from main PC’

        Lcdout $fe, 1, " FTP initiated "
        Lcdout $fe, $C0, "   by main PC"

        Hserout [13,10,"Ready for file transfer",13,10]
        pause 1000

        Serout2 PortB.1,16572,[13,10,"Be ready for receiving file",13,10]
        pause 1000

        Serin2 PortB.0,16572,30000,receivernotready,[nak]
        Hserout [nak]

loop1:
        Hserin 5000,nofiletransfer,[char1]
        Hserin 500,ftpcompleted,[char2]
        Hserin 1000,problem,[char3]

        Hserin 1000,problem,[str char\65]
        Hserin 1000,problem,[str a\65]

        Serout2 PortB.1,16572,[char1,char2,char3,str char\65,str a\65]

        Serin2 PortB.0,16572,11000,noreplyfromreceiver,[nak]
        Hserout [nak]

    goto loop1



    Case "6"
‘File transfer procedure is initiated from remote PC’

        Lcdout $fe, 1, " FTP initiated "
        Lcdout $fe, $C0, "  by remote PC"

        Serout2 PortB.1,16572, [13,10,"Ready for file transfer",13,10]
        pause 1000

        Hserout [13,10,"Be ready for receiving file",13,10]
        pause 1000

        Hserin 30000,receivernotready1,[nak]
        Serout2 PortB.1,16572,[nak]


loop12:
```

```
        Serin2 PortB.0,16572,5000,nofiletransfer1,[char1]
        Serin2 PortB.0,16572,500,ftpcompleted1,[char2]
        Serin2 PortB.0,16572,1000,problem1,[char3]

        Serin2 PortB.0,16572,1000,problem1,[str char\65]
        Serin2 PortB.0,16572,1000,problem1,[str a\65]

        Hserout [char1,char2,char3,str char\65,str a\65]

        Hserin 11000,noreplyfromreceiver1,[nak]
        Serout2 PortB.1,16572,[nak]

    goto loop12



    Case "9"

Serout2 PortB.1,16572,["Modem connection is closed by remote end!!"]

        Lcdout $fe, 1, "  Waiting for"
        Lcdout $fe, $C0, "initialization!"

    goto start


    Case Else

        Lcdout $fe, 1, "FALSE REQUEST!"


End Select

goto loop


notinitialized:
goto start


meterdetect1:
Hserout [13,10,"Port 1 = Computer",13,10]
Serout2 PortB.1,16572,[13,10,"The main PC is online now!!!",13,10]
goto detect2

meterdetect2:
Hserout [13,10,"Port 2 = No connection",13,10]
goto detect3
```

```
meterdetect3:
Hserout [13,10,"Port 3 = No connection",13,10]
goto detect4

meterdetect4:
Hserout [13,10,"Port 4 = No connection",13,10]
goto loop



‘After Data reading is completed’
datason:

    Lcdout $fe, 1, " Data Received "
    Hserout [13,10,"Data receiving is completed!",13,10]
    pause 3000
    goto loop


‘If no access to meter at related port’
noconnection:
    Lcdout $fe, 1, " No connection!"
    Hserout [13,10,"No meter connected to this port!",13,10]
    pause 1000
    goto loop

‘Below program parts are related with ftp process’

‘If ftp process from main PC to remote PC is not started’
nofiletransfer:

    Lcdout $fe, 1, "No data received"
    Hserout ["No data received"]
    Serout2 PortB.1,16572,["No data received from sender"]
    pause 1000
    goto loop

‘If ftp process from main PC to remote PC is completed’
ftpcompleted:

        Lcdout $fe, 1, "FTP completed!"
        pause 2000
        Serout2 PortB.1,16572,[4]
        Hserout [6]
        pause 100
        Hserout [13,10," FTP Completed",13,10]
        Serout2 PortB.1,16572,[13,10," FTP Completed",13,10]
```

```
        goto loop
```

‘If a problem is occured during file transfer from main PC to remote PC’
problem:

```
        Hserout [24]
        Serout2 PortB.1,16572,[24,24,24]
        Lcdout $fe, 1, "problem"
        pause 2000
        goto loop
```

‘If receiver does not send handshake char’
receivernotready:

```
       Lcdout $fe, 1, "    Receiver"
       Lcdout $fe, $C0, "  not ready!! "
       goto loop
```

‘If receiver does not respond with ACK or NACK to last received block’
noreplyfromreceiver:

```
   Lcdout $fe, 1, "noreplyfromrcvr"
   Serout2 PortB.1,16572,["You did not reply with ACK or NACK for last
block"]
   goto loop
```

‘If ftp process from remote PC to main PC is not started’
nofiletransfer1:

```
       Lcdout $fe, 1, "No data received"
       Hserout ["No data received from sender"]
       Serout2 PortB.1,16572,["Data could not be received!!"]
       pause 1000
       goto loop
```

‘If ftp process from remote PC to main PC is completed’
ftpcompleted1:

```
          Lcdout $fe, 1, "FTP completed!"
          pause 2000
          Serout2 PortB.1,16572,[6]
          Hserout [4]
          pause 100
          Hserout [13,10," FTP Completed",13,10]
```

```
        Serout2 PortB.1,16572,[13,10," FTP Completed",13,10]
        goto loop

‘If a problem is occured during file transfer from remote PC to main PC’
problem1:

      Hserout [24]
      Serout2 PortB.1,16572,[24,24,24]
      Lcdout $fe, 1, "problem"
      pause 1000
      goto loop

‘If receiver does not send handshake char’
receivernotready1:

     Lcdout $fe, 1, "notready"
     Hserout ["Be in receving file mode!"]
     goto loop

‘If receiver does not respond with ACK or NACK to last received block’
noreplyfromreceiver1:

     Lcdout $fe, 1, "noreplyfromrcvr"
     Hserout ["You did not reply with ACK or NACK for last block"]
     goto loop
End
```

APPENDIX C – MULTIPORT DEVICE USER MANUAL

1)- Dimensions of the Device

Figure – 1 shows the dimensions of the multi-port device. As it can be seen from the figure multi-port device can be located easily thanks to its reasonable dimensions.

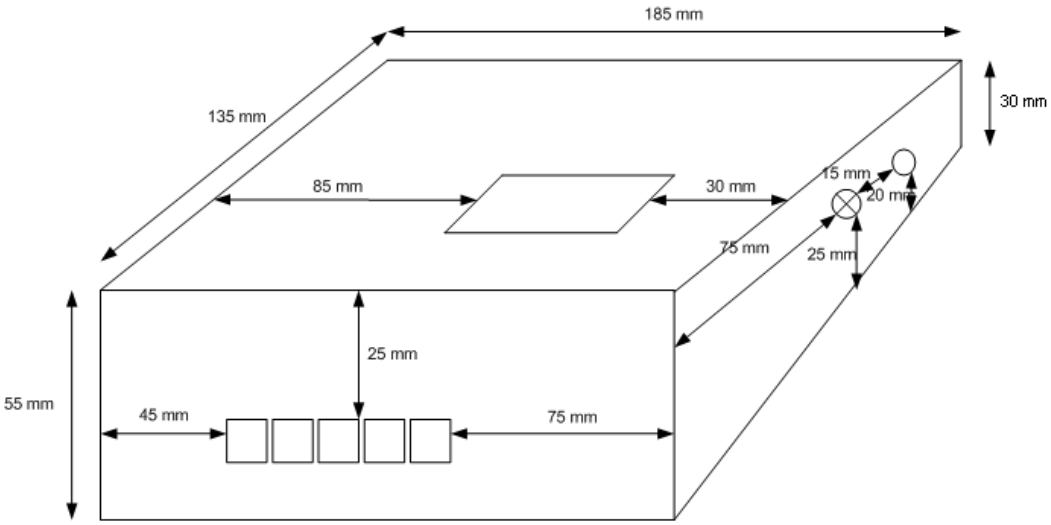


Figure – 1 Dimensions of the device

2)- Connection Specifications

Connection point in the system: The multi-port is connected between the remote modem and digital electricity meters as shown in Figure – 2:

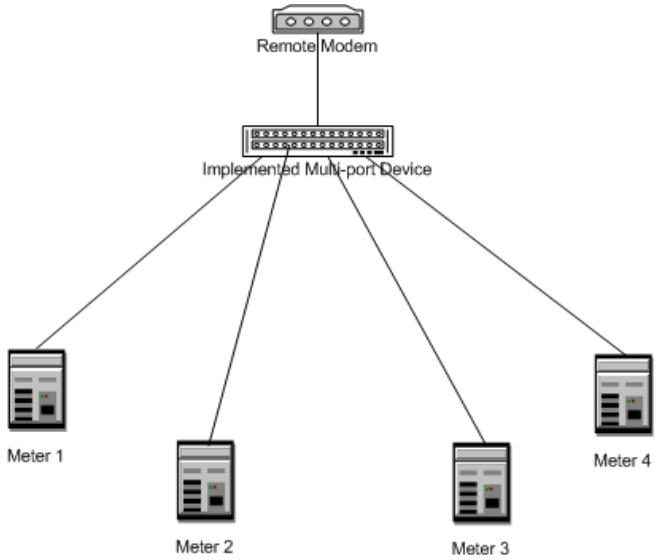


Figure – 2 The multi-port device and digital electricity meters

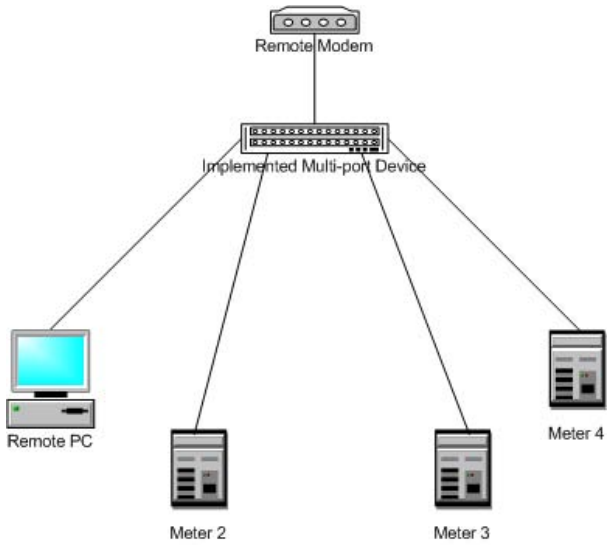


Figure – 3 The multi-port device, remote PC and digital electricity meters

Port Specifications: The connection points of modem, PC and electricity meters are as shown in Figure – 4. The connections are realized by RJ12 connectors as specified in Figure 5 and 6.

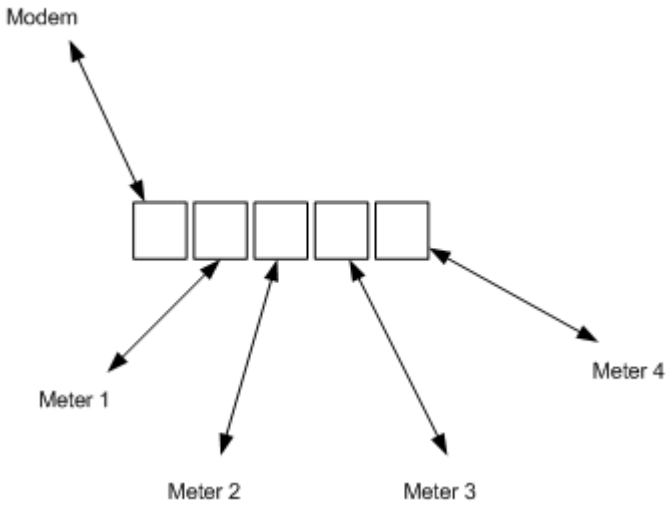


Figure – 4 Connections to the ports

Meter Side **RJ-12 Male Socket - Device**

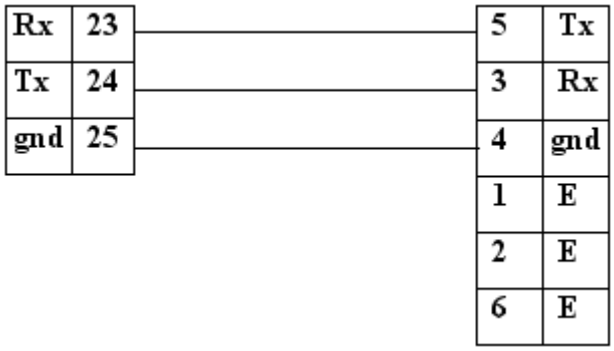


Figure – 5 Meter-Device Connection

Modem Side (25-pin Male) **RJ-12 Male Socket - Device**

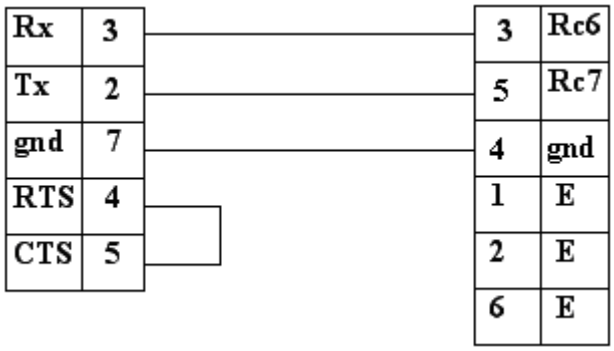


Figure – 6 Modem-Device Connection

Remote PC Side DB9 Female **RJ-12 Male Socket – Device**

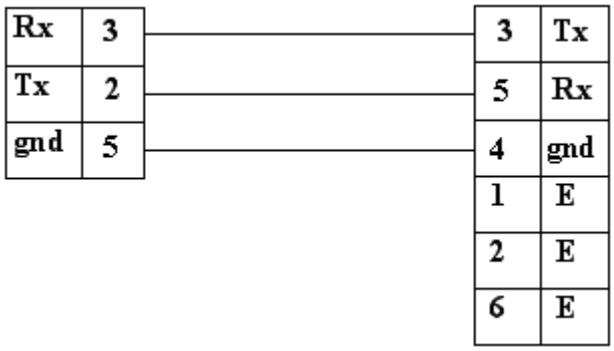


Figure – 7 Modem-Device Connection

E: empty, not used

Power Supply: The multi-port device can be operated by using a standard voltage adaptor. The voltage of the adaptor can be changed between 12 V –24 V. The recommended voltage is 12 V DC from an adaptor.

3)- Minimum Requirements for the Main Parts of the System

PC: The PC might be a standard personal computer, which has Windows 2000 or XP operating system. For other Windows or Linux, Unix some settings about the operating system might be needed. The main and easy requirement for the PC is a properly working serial com. Port.

PC Side Modem : It is recommended by the Digital Electricity Meter producers that the modem should be Hayes compatible, under this circumstances the modem might be internal or external. The recommended and tested as properly working modem settings with AT command sets are as following:

AT &F1
AT+MS=9,0,4800,0,0,4800
AT&M0
AT%C0
ATX3
ATE0
AT&W0
AT&W1
AT&Y0

But it should be noted that these settings are not the only way for solution and might be changed according to user and system preferences.

Remote Side Modem : It is recommended by the Digital Electricity Meter producers that the modem should be Hayes compatible. The recommended and tested as properly working modem settings with AT command sets are as following:

AT &F1
AT+MS=9,0,4800,4800,0,0,4800
ATS0=1
AT&M0
AT%C0
ATE0
AT&D0
AT&W0
AT&W1
AT&Y0

But it should be noted that these settings are not the only way for solution and might be changed according to user and system preferences.

Meters : The meters which can be read properly in this system must have at least one RS232 communication port and very importantly they must communicate with the peripherals according to the FLAG protocol. FLAG protocol is created by FLAG Association as guide to IEC Meter Communications Specifications.

4)- Reading Software

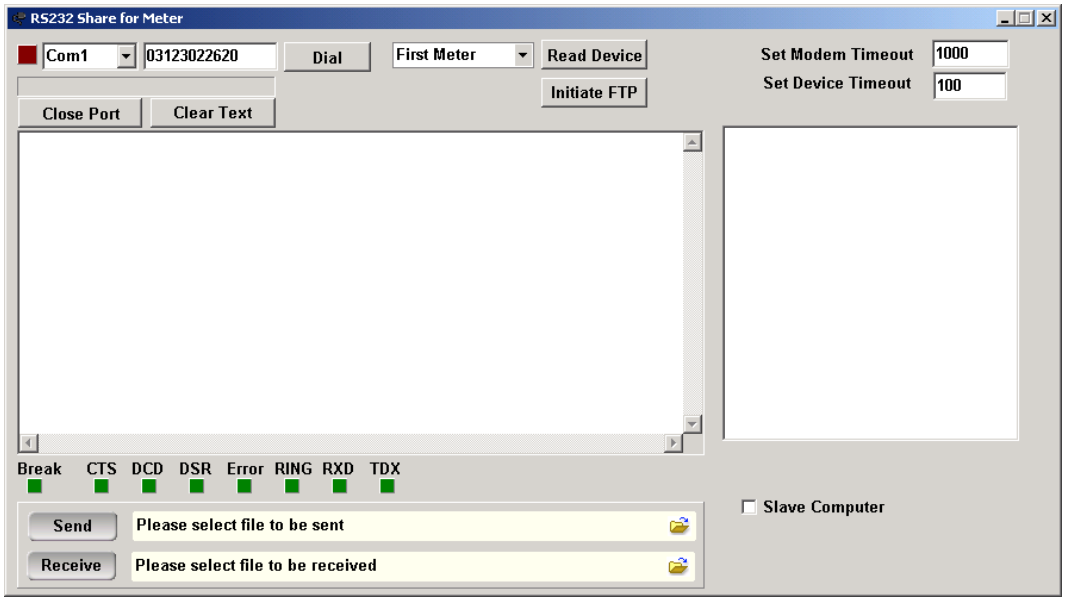


Figure – 8 Main Screen of Reading Software on Main PC

When the program is started on the main PC, the program window shown in Figure – 2.1 should appear on the main PC screen.

The execution steps of software are described below:

1)- First of all, the port of main PC, which will be used for serial communication, should be selected. The active ports are listed in the left upper box.

2)- Then the phone number, which will be dialed to establish the communication to the remote side, should be specified in the box at left side of Dial button.

3)- Next, the timeout values for main PC to modem connection and main PC to multi-port device should be entered to right upper box. After that, Dial button should be clicked, and thus the "ATDT + Phone Number" command is sent to modem. Then connection is established and the program starts to send connection indication string to the multi-port device. After the handshake process between the multi-port device and main PC is completed, the program will receive the port status of the multi-port device, such as which device is connected to

which port. If the handshake process between main PC and multi-port device, which can continue 60 seconds at most, fails anyhow, the connection will be closed automatically by the program. The related indication messages appear on the left log window.

4)- After the connection is established successfully, the request may be to perform a remote reading procedure of a meter or to perform a file transfer to the remote PC.

5)- If the request is to perform a remote reading procedure of a meter, the desired meter should be selected from the box and then Read Device button should be clicked. If the meter is not password protected and also connected to the port properly, the readout data will be listed in reading window on the left. The readout data can be written to a text file by clicking right button of mouse over the window.

6)- This meter readout procedure can be executed for each meter in the same manner. The readout window on the left can be cleared at anytime by clicking Clear Text button.

7)- If the request is to perform a file transfer to the remote PC, the request must be initiated by clicking to the Initiate FTP button and after that the file to be sent should be selected by clicking File Open icon at the bottom of the program window at same line with Send button. After clicking to the Send button, the file transfer procedure will wait for 30 seconds for preparation of remote PC to receive a file. Within 30 seconds, when the remote PC passes to receiving file mode, the file transfer process will start according to Xmodem protocol.

8)- If a file transfer request is received from remote PC, the multi-port device informs the main PC to be in receiving mode within 30 seconds. When the main PC enters to receiving file mode, the file transfer process will start according to Xmodem protocol.

9)- The connection between main PC and multi-port device can be closed at anytime by the user of main PC by clicking Close Port button.

Note: The buttons below the reading window indicates the status of serial communication performed by using APRO Libraries. They will be red or green according to the status of process while the program is running. Clear Text button clears the reading window to prepare for next reading. Clear Text button can be used at anytime.

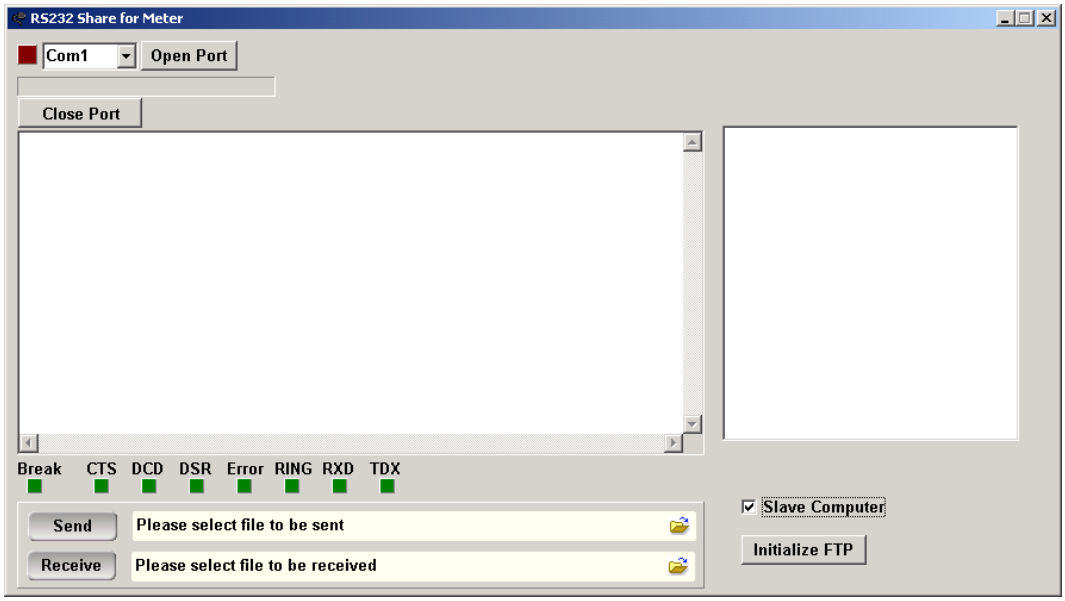


Figure – 9 Main Screen of Reading Software on Remote PC

When the program is started on the remote PC, the program window shown in Figure – 2.2 should appear on the remote PC screen.

The execution steps of software are described below:

1)- First of all, the Slave Computer box must be checked in order to turn the lite version on. As soon as the program is started, the serial port to be used for the communication with multi-port device should be specified and opened.

2)- While the start up procedure of the multi-port device, the multi-port device sends the port status detection messages to all ports. As soon as the remote PC receives this message, it immediately replies with a specified string that indicates remote PC is connected and ready.

3)- Then the remote PC starts to wait for a file transfer procedure initialization message. When this message is received, the program should be in receiving file mode within 30 seconds. After a few seconds, the file transfer starts.

4)- The remote PC can initiate a file transfer anytime by clicking Initialize FTP button at the right bottom of the program window. If the remote PC receives “Ready for file transfer” message, the file transfer can be started. When Send button is clicked, the file transfer procedure will wait for 30 seconds for preparation of main PC to receive a file. Within 30 seconds, when the main PC passes to receiving file mode, the file transfer process will start according to Xmodem protocol.

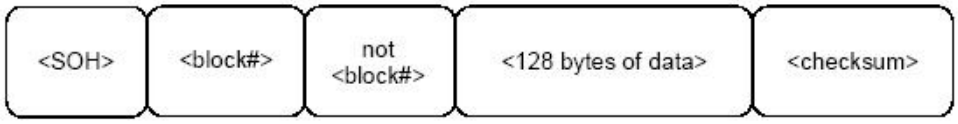
5)- After the file transfer from remote PC to main PC is completed, the program execution turns back step 2. Throughout running of lite version remote access software, the specified serial port must be opened and the Slave Computer button must be checked.

APPENDIX D – XMODEM FILE TRANSFER PROTOCOL

Xmodem

Xmodem, which is one of the oldest, simplest and slowest file transfer protocols, was developed and first implemented by Ward Chrlstensen in 1977 and placed in the public domain. Since then, it has become an extremely popular protocol and continues in use today (although at a diminished frequency).

Xmodem uses blocks of only 128 bytes and requires an acknowledgment of each block. It uses only a simple checksum for data integrity.What follows is a simplified description of the Xmodem protocol, although it describes far more than is required to actually use the protocol.



The format for XModem blocks.

Figure – 1 Xmodem block structure

The <SOH> character marks the start of the block. Next comes a one byte block number followed by a ones complement of the block number. The block number starts at one and goes up to 255 where it rolls over to zero and starts the cycle again. Following the block numbers are L28 bytes of data and a one-byte checksum. The checksum is calculated by adding together all the data bytes and ignoring any carries that result. Below table describes a typical Xmodem protocol transfer.

Transmitter		Receiver
	<---	<NAK>
<SOH><1><254><128 data bytes><chk>	--->	
	<---	<ACK>
<SOH><2><253><128 data bytes><chk>	--->	
	<---	<ACK>
<EOT>	--->	
	<---	<ACK>

Figure – 2 Xmodem protocol transfer steps

The receiver always starts the protocol by issuing a <NAK>, also called the handshake character. It waits 10 seconds for the transmitter to send a block of data. If it does not get a block within 10 seconds, it sends another <NAK>. This continues for 10 retries, after which the receiver gives up.

If the receiver does get a block, it compares the checksum. It calculates to the received checksum. If the checksums differ, the receiver sends a <NAK> and the transmitter resends the block. If the checksums match, the receiver accepts the block by sending an <ACK>. This continues until the complete file is transmitted. The transmitter signifies this by sending an <EOT>, which the receiver acknowledges with an <ACK>.

From this description several things become clear. First, this protocol does not transfer any information about the file being transmitted. Hence, the receiver must assign a name to the incoming file.

The receiver also does not know the exact size of the file, even after it is completely received. The received file size is always a multiple of the block size. This Xmodem implementation fills the last partial block of a transfer with characters of value BlockFillChar, whose default is ^Z.

Xmodem often exhibits a start-up delay. The transmitter always waits for a <NAK> from the receiver as its start signal If the receiving program was started first, the transmitter probably missed the first <NAK> and must wait for the receiver to time out and send another <NAK>.

The only merit of the basic Xmodem protocol is that it is so widespread that it is probably supported by any microcomputer communications program you can find, thus providing a lowest common denominator between systems.

Xmodem Extensions

Xmodem has been tweaked and improved through the years. Some of these variations have become standards of their own.

The first of these improvements is called Xmodem CRC, which substitutes a 16 bit CRC (cyclic redundancy check) for the original checksum. This offers a much higher level of data integrity. When given the opportunity, you should always choose Xmodem CRC over plain Xmodem.

The receiver indicates that it wants to use Xmodem CRC by sending the character ‘C’ instead of <NAK> to start the protocol. If the transmitter doesn't respond to the ‘C’ within three attempts, the receiver assumes the transmitter is not capable of using Xmodem CRC. The receiver automatically drops back to using checksums by sending a <NAK>.

Another popular extension is called Xmodem 1K. This derivative builds on Xmodem CRC by using 1024 byte blocks instead of 128 byte blocks. When Xmodem 1K is active, each block starts with an <STX> character instead of an <SOH>. Xmodem 1K also uses a 16 bit CRC as the block check.

A larger block size can greatly speed up the protocol because it reduces the number of times the transmitter must wait for an acknowledgment. However, it can actually reduce throughput over noisy lines because more data must be retransmitted when errors are encountered.

APPENDIX E – COMPLETE LIST OF SERIN2/SEROUT2 MODES

Baud Rate	Bit 15 (Output)	Bit 14 (Conversion)	Bit 13 (Parity)	Mode Number
300	Driven	True	None	3313
300	Driven	True	Even*	11505
300	Driven	Inverted	None	19697
300	Driven	Inverted	Even*	27889
300	Open	True	None	36081
300	Open	True	Even*	44273
300	Open	Inverted	None	52465
300	Open	Inverted	Even*	60657
600	Driven	True	None	1646
600	Driven	True	Even*	9838
600	Driven	Inverted	None	18030
600	Driven	Inverted	Even*	26222
600	Open	True	None	34414
600	Open	True	Even*	42606
600	Open	Inverted	None	50798
600	Open	Inverted	Even*	58990
1200	Driven	True	None	813
1200	Driven	True	Even*	9005
1200	Driven	Inverted	None	17197
1200	Driven	Inverted	Even*	25389
1200	Open	True	None	33581
1200	Open	True	Even*	41773
1200	Open	Inverted	None	49965
1200	Open	Inverted	Even*	58157
2400	Driven	True	None	396
2400	Driven	True	Even*	8588
2400	Driven	Inverted	None	16780
2400	Driven	Inverted	Even*	24972
2400	Open	True	None	33164
2400	Open	True	Even*	41356
2400	Open	Inverted	None	49548
2400	Open	Inverted	Even*	57740
4800	Driven	True	None	188
4800	Driven	True	Even*	8380
4800	Driven	Inverted	None	16572
4800	Driven	Inverted	Even*	24764
4800	Open	True	None	32956
4800	Open	True	Even*	41148
4800	Open	Inverted	None	49340
4800	Open	Inverted	Even*	57532
9600 baud may be unreliable with 4MHz clock				
9600	Driven	True	None	84
9600	Driven	True	Even*	8276
9600	Driven	Inverted	None	16468

9600	Driven	Inverted	Even*	24660
9600	Open	True	None	32852
9600	Open	True	Even*	41044
9600	Open	Inverted	None	49236
9600	Open	Inverted	Even*	57428
baud rates below require 8MHz clock or faster				
14400	Driven	True	None	49
14400	Driven	True	Even*	8241
14400	Driven	Inverted	None	16433
14400	Driven	Inverted	Even*	24625
14400	Open	True	None	32817
14400	Open	True	Even*	41009
14400	Open	Inverted	None	49201
14400	Open	Inverted	Even*	57393
baud rates below require 10MHz clock or faster				
19200	Driven	True	None	32
19200	Driven	True	Even*	8224
19200	Driven	Inverted	None	16416
19200	Driven	Inverted	Even*	24608
19200	Open	True	None	32800
19200	Open	True	Even*	40992
19200	Open	Inverted	None	49184
19200	Open	Inverted	Even*	57376
baud rates below require 16MHz clock or faster				
28800	Driven	True	None	15
28800	Driven	True	Even*	8207
28800	Driven	Inverted	None	16399
28800	Driven	Inverted	Even*	24591
28800	Open	True	None	32783
28800	Open	True	Even*	40975
28800	Open	Inverted	None	49167
28800	Open	Inverted	Even	57359
33600	Driven	True	None	10
33600	Driven	True	Even*	8202
33600	Driven	Inverted	None	16394
33600	Driven	Inverted	Even*	24586
33600	Open	True	None	32778
33600	Open	True	Even*	40970
33600	Open	Inverted	None	49162
33600	Open	Inverted	Even	57354
baud rates below require 20MHz clock or faster				
38400	Driven	True	None	6
38400	Driven	True	Even*	8198
38400	Driven	Inverted	None	16390
38400	Driven	Inverted	Even*	24582
38400	Open	True	None	32774
38400	Open	True	Even*	40966
38400	Open	Inverted	None	49158
38400	Open	Inverted	Even	57350

**Parity is odd when DEFINE SER2_ODD 1 is used.*

[Ref. 9]