Wireless Sensor Data Logging System Design

By

Noorshafrina Binti Zulkalnain

FINAL REPORT

Submitted to the Electrical & Electronic Engineering Programme
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronic Engineering)

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Wireless Sensor Data Logging System Design

By

Noorshafrina Binti Zulkalnain

An Final Report Submitted to the
Electrical & Electronic Engineering Programme
Universiti Teknologi PETRONAS
in Partial Fulfillment of the Requirement for the
Bachelor of engineering (Hons)
(Electrical & Electronic Engineering)

Approved by,

(Dr. Mohd Zuki Bin Yusoff)
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

MAY 2011

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

(Noorshafrina Binti Zulkalnain)

DEDICATION

To

my parents,
my mother, Norlela
father, Zulkalnain and
my siblings

ACKNOWLEDGEMENT

It's been a long journey in UTP, 5 years living as a student and now finally as a final year student. The journey was great and many special moments I have been through together with my beloved friends. Learning is always a lifelong journey that demands a lot of dedication, passion and patience. Transitioning from a student to a scholar, one's route to knowledge quest is always unique in one's way. Some people manage to achieve their search for knowledge in a sweet and straightforward way. On contrary; some other people's knowledge seeking paths are bitter and full of winding ridges. Many sweats and brain juggling have been generated and many mind boggling of this Degree study proves that ultimate life balance and perseverance do pay.

During one year of my Final Year Project, I have worked with a great number of people whose contribution in assorted ways to the research and the making of the thesis deserved special mention. It is a pleasure to convey my gratitude to them all in my humble acknowledgment.In the first place,I would like to record my gratitude to my supervisor, Dr.Mohd Zuki supervision, advice, and guidance from the very early stage of this research as well as giving me great ideas throughout the work. Above all and the most needed, he provided me unflinching encouragement and support in various ways. Many thanks go in particular En.Azhar, the EE lab technician for giving ideas and advice to conduct the sensors experiment. I gratefully thank to the chemical lab technician, En.Sulaimen for lending me help in conducting the humidity and temperature sensor accuracy experiment. I would also acknowledge Mr. Hasrul Firdaus for his help in producing the printed circuit board for my project.I would also like to thank my friends for supporting me all along the way and creating a great friendship in the Universitiy. Finally, I would like to thank to my beloved mother, Norlela and father, Zulkalnain, who up brought me to be what I am today, and who always extends their prayers and best wishes.Last but not least,I would like to thank my siblings Noorazreen, Khairi, Asyraf, Syahmi for being there whenever needed and also for supporting me.

ABSTRACT

Wireless Sensor Data Logging System Design is a standalone electronic sensor device that captures and stores data through wireless communication. This system comprises two main integrated components; the Radio Frequency module and the Microcontroller based system. The main goal of this project is to design and construct a data logging system that effectively monitors the device's measurement values. In real life applications, most data monitoring system is a passive system. This type of system requires manned guarding on site to manage the devices. Therefore, a standalone data logging system offers a better enhancement system to replace the manned guarding method. The standalone data logger system can be applied by leaving the device alone in any place that requires the measurement of humidity and temperature. These data can be retrieved from EEPROM and transferred to a PC whenever needed by a user. A radio frequency module enables these data travels through wireless transmission medium, whereas the serial communication interface enables communication between the devices and PC. For diverse applications, an alarm system can be implemented if assets and security are the major concerns. The final report presents the development of a data logger system which is an integration of radio frequency module and the microcontroller-based system. The system monitors the device's measurement value via a Graphical User Interface. Basically, the system introduces a RF module to replace the hard wired scheme and produce a dynamic data transmission system. It is geared up with a PIC16F877A microcontroller to drive the outputs besides providing communication between devices and a PC. Overall, the project is the best platform to improve the traditional monitoring system and ignites another innovative invention in the future.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

PCB	Printed Circuit Board
UART	Universal Asynchronous Receiver Transmitter
EEPROM	Electrically Erasable Programmable Read Only Memory
I2C	Inter-Integrated Circuit
MCU	Microcontroller
PC	Personal Computer
LCD	Liquid Crystal Display
GUI	Graphical User Interface
FYP	Final Year Project
IDE	Integrated Development Environment
CCS	Code Composer Studio
PIC	Programmable Interface Controller
ICSP	In-Circuit Serial Programming
MCLR	Master Clear
VCC	Voltage Common Collector
PGD	Programming Data
PGC	Programming Clock
EN	Enable
R/W	Read or Write
SDA	Serial Data
SCL	Serial Clock

CHAPTER 1
INTRODUCTION

Wireless Sensor Data Logger System Design preface and background will be explained comprehensively in this chapter. All the information gathering and research were undertaken through many resources such as internets, books, journals and guidelines from lecturers. The elements that will be emphasized in this chapter are the background of study, problem statement, objectives and scope of study. The details discussed throughout this chapter will help the readers grasp the idea of the project and understand the concepts and principles applied.

1.1 Background of Study

The Wireless Sensor Data Logger System Design was designed based on the problems faced in the passive manned guarding system. Usually, the passive manned guarding system requires a person to monitor and guard the devices for the required time. Therefore, the data logger is designed to replace this less reliable and ineffective system. A data logger is a standalone sensor device that has the ability to store data in an external or internal memory. It refers to a system that is used to effectively measure and record important physical parameters such as humidity and temperature measurement. They are great portable device which can function independently without anyone to guard it. It can be taken anywhere and used in various situations. Whenever the measured data is needed, it is being collected. The sensors are the important hardware tools that actually take these measurements [1, 2].

Many industries around the world rely on the very regular use of such system especially the food and beverage industry. The data logger is helpful in controlling certain aspects when it comes to dealing with products that are being stored. The quality of a product must be assured in a good condition when it arrives to its final destination [3]. These devices are useful in restaurants to save a record of food temperature which are kept in the refrigerator. Bacteria increases and grows rapidly as it reaches the temperature between 4°C to 60°C [4]. Therefore, it is very important for the food industry to make sure the foods are at their proper temperature in order to prevent people from getting sick. Humidity measurement also plays an important role for the transport of some goods such as flowers [1, 3]. Humidity must be in high condition to avoid the flowers from getting dried out and wilt. These plants need to hold onto their vital moisture content to live [5].

Therefore, this wireless sensor data logger system was designed to satisfy the requirements stated above. It was developed based on the integration of a number of subsystems; the radio frequency module, the microcontroller-based systems, external EEPROM integration, sensor device, serial communication interface and GUI (Graphical User Interface). This integrated device is an efficient system where it is portable, accurate, less expensive and light weight [1]. These great qualities made the system very competitive and reliable for real application.

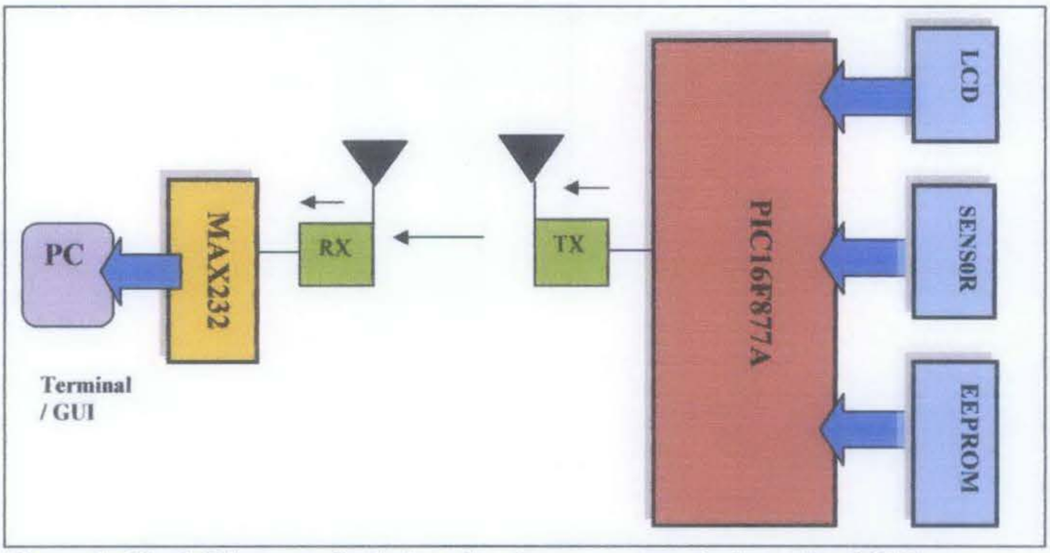


Figure 1: Block Diagram of wireless data logger system design using MCU.

The system is basically illustrated in the block diagram of Figure 1. Apparently; the measured data is obtained from the data logging system. The system was designed to store data received from the sensor that is attached to the microcontroller. The microcontroller is installed in the data loggers in order to interface with computer programs [1]. The value of each data will be transmitted via wireless communication medium utilizing radio frequency module at 433MHz operating frequency. Simultaneously, the data is transferred to serial communication interface via the serial communication system to enable communication between device and a PC. The system is made interactive with the aid of GUI for easy-handling purposes. Besides, the system is capable of monitoring the status in textual form via Terminal program. The system will operate in one direction communication where the value of the sensor is sent to the PC. Overall, the Wireless Sensor Data Logger System comprises the integration of hardware and software that offers an interactive, effective and reliable data logging system.

1.2 Problem Statement

1.2.1 Problem identification

Typically, most of traditional monitoring systems perform passive guarding system. Apparently, these systems are less reliable and ineffective. The hasty changes in technology nowadays made this existing system merely inconvenient. Consequently, a better approach should be implemented for an advancement of this passive system. The time management and quality of a product should be managed wisely in order to achieve a productive and smooth operation process. Some decisions must be made based upon the data and these decisions are important for the safety of human being. For instance the temperature of food must be recorded to keep of track the growth of bacteria in the food. Therefore, the *Wireless Sensor Data Logger System Design* perhaps introduces revolution for the passive manned guarding system.

1.2.2 Problem solution

The project essentially offers a dynamic and efficient system that handles the recorded data through a wireless standalone device that is displayed to PC. Basically, the project is based on the problem analysis basis and extends it to problem solving before it is implemented as a whole. It acts as a self contained unit that does not require any help from hosts to operate. Compared to conventional interface devices, this data logger has the capability to dump or transfer the data to a host system, if required. These data can be saved and analyzed for historical archive purposes [6]. The *Wireless Sensor Data Logger System Design* demonstrates the integration and application of theories in engineering discipline, which is a good platform for better understanding on engineering principle applications.

1.3 Project Objectives

The objectives of the project focus on the steps towards the final design of the data logger system which is based on engineering fundamentals and problem solving basis. The aim and goal of this project are as follows:

i. To design a wireless system that can record and save the sensor data into an external EEPROM chip for the required range of time.

ii. To integrate the radio frequency module with a microcontroller-based system to enable the data transfer through wireless transmission medium.

iii. To reduce and manage the wiring of a circuit by designing its printed circuit board.

iv. To design a graphical user interface that can display the values of measured and recorded data. This interface contributes a user-friendly system to the real environment.

1.4 Scope of Work

The scope of work in this chapter is based on elements listed as below:

i. Integration of sensor and EEPROM device with microcontroller.
ii. Wireless data transmission via a radio frequency module.
iii. Interface the data logger system with serial communication.
iv. Implementation of graphical user interface software design.

The work is based on the elements above which apparently consist of a wireless data logging system. The microcontroller offers various advancement designs for the whole system. The Radio frequency module introduces an alternative solution for data transmission and it is applicable for various fields. Radio frequency transmission medium offers wider coverage area compared to other mediums.

The serial communication is designed in order to communicate with PC which practically improves the data logger system. Finally, the graphical user is implemented to improve the data logging system which makes the system interactive to the end user.

CHAPTER 2
LITERATURE REVIEW

A literature review is one of the important development stages where the knowledge of each element used in the project is introduced. This particular chapter actually discusses the fundamental concepts applied in the project. The resources obtained through research via various sources are covered in this chapter. The literature review comprises external EEPROM interface, PIC16F877A microcontroller, serial communication interface, characteristics of wireless system, printed circuit board and graphical user interface via Visual C++.

2.1 Introduction of Microcontroller

A microcontroller is a single silicon chip which includes at a minimum microprocessor, program memory, data memory and an input output device. The word 'micro' reflects that the device is small while the word 'controller' refers to the use of it in control applications. An embedded controller is another term for a microcontroller since most of microcontrollers are built in the devices they control. The main difference between a microprocessor and a microcontroller is that a microprocessor requires several other components for its operation such as program memory, data memory, input output devices and an external clock circuit. On the other hand, a microcontroller consists of all the support chips embedded inside its single chip. Other additional components such as timers, counters and analog-to digital converters are included in certain microcontrollers. Thus, a microcontroller system can act as a large computer with hard disks, floppy disks and printers to a single-chip embedded controller. They also can be used and embedded into household goods and other electronic controlled devices such as refrigerators, implantable medical devices, remote controls, office machines, toys, appliances, microwave ovens and cookers [7].

A set of instructions stored in a memory of microcontroller can be operated by a microcontroller by fetching the instructions from its memory one by one; then these instructions are being decoded to carry out the required operations. The program languages used to program a microcontroller can either be an assembly language or a high level language. An assembly language is faster compared to a high level language, but it is hard to learn and maintain the program written because an assembly program consists of mnemonics. Different firms manufacture microcontrollers with different assembly languages; therefore, the user has to learn a new language for every new microcontroller used. High level languages are well known languages that facilitate the development of large and complex programs. User Programs which are loaded in the microcontroller's memory are executed. The data are received from input devices, manipulated and sent to output devices [7].

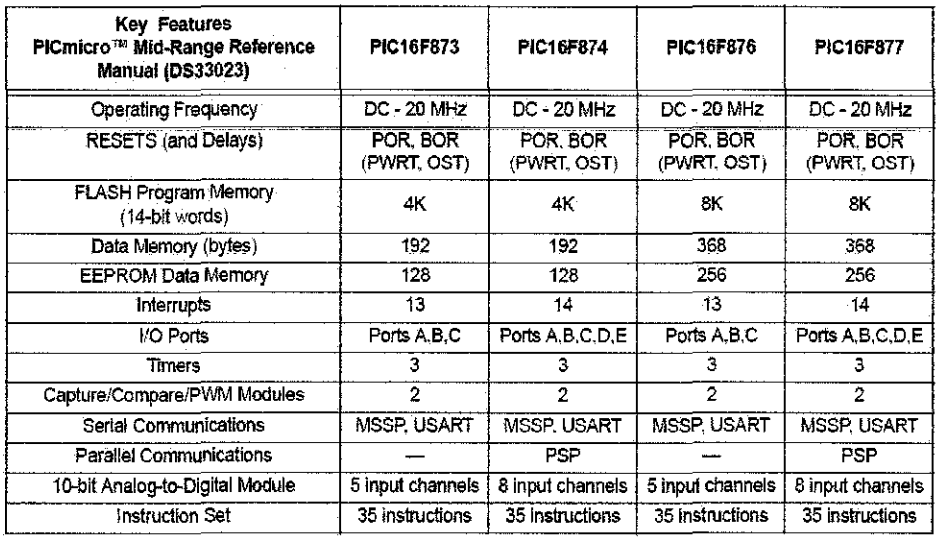
As a powerful tool, a microcontroller allows designers to design sophisticated input output data manipulation under program controls. They are classified by the number of bits they process. In most microcontroller-based applications, 8 bits are widely used and popular among the users. The 16 bits and 32 bits are expensive and not required in small or medium size general purpose applications compared to 8 bits, but they are much more powerful. The architecture of a microcontroller consists of a microprocessor, a memory and an input output. The central processing unit (CPU) and the control unit (CU) are the elements of the microprocessor. The CPU is referred as the brain of the microcontroller. Arithmetic and logic operations are performed here. The required instructions can be carried out by letting the CU control the internal operations of the microprocessor and send signals [7].

2.2 Architecture Overview of PIC16F877A

One of the most advanced and well known microcontrollers from Microchip is PIC16877A. In modern applications, the controller is widely used for experimental purposes since it is less expensive, high quality and easily available in market. It can be applied in various applications such as machine control applications, measurement devices, study purposes and so on. Compared to other microcontroller family series, PIC16F877A features all the components which modern microcontrollers normally have and also has more advanced and developed features [8]. The features, pin diagrams and specifications of PIC16F877A are shown in the datasheets from **APPENDIX E.**

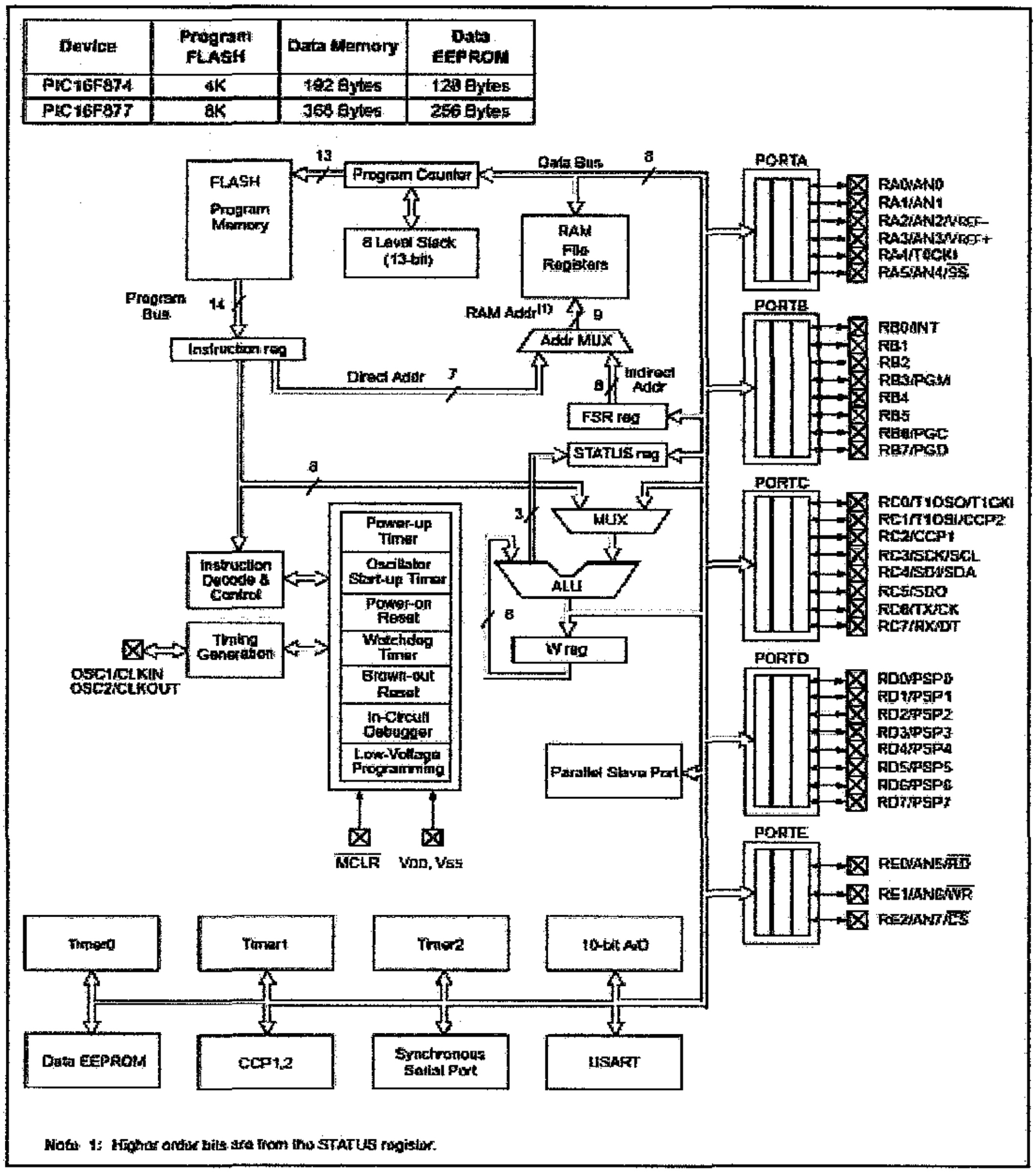
From its data specifications, the microcontroller has 8K Word which is 14.2Kbytes Flash, 368 RAM, 256 EEPROM and 20MHz of operating frequency. The synchronous serial port can be configured as either 3 wired Serial Peripheral Interface or as the 2 wired Inter Integrated Circuit bus and a Universal Asynchronous Receiver Transmitter. These features makes this microcontroller chip ideal for more advanced level analog to digital applications in automotive, industrial, appliance and consumer applications [9]. Table 1 below summarizes the PIC16F877A specifications and other PIC16F87X as well:

Table 1: PIC16F877A and PIC16877X Microchip specifications [9].



Key Features **PICmicro™ Mid-Range Reference Manual (DS33023)**	**PIC16F873**	**PIC16F874**	**PIC16F876**	**PIC16F877**
Operating Frequency	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz
RESETS (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Instruction Set	35 instructions	35 instructions	35 instructions	35 instructions

By understanding the block diagram of PIC MCU, the idea of how to execute programs and manipulate data in the PIC MCU is easily understood. The block diagram is actually the architectural drawing of its inner workings. Processor block diagrams are basically similar for each of PIC MCU processor families. There are only certain things that might not be the same such as how data is accessed in different register banks, how data is indexed and stored in stacks. The internal block diagram of PIC16F877A microcontroller is shown in Figure 2 [10].



Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes

Note 1: Higher order bits are from the STATUS register.

Figure 2: Internal block diagram of PIC16877A.

The arithmetic logic unit (ALU) provides basic arithmetic and bitwise operations for the processor of PIC microcontroller. The input-output registers and the data storage RAM registers are specific use registers that control the operation of the CPU. These registers sometimes can be called as hardware registers. It depends on the function they perform. Hardware registers can provide direct manipulation of functions that are invisible to the programmer such as the program counter which allows advanced program functions. The data storage registers, RAM, are known as file registers by Microchip. The registers have their own spaces because they are separated from the program memory [10] .This is called the Harvard architecture which is shown in Figure 3 below:

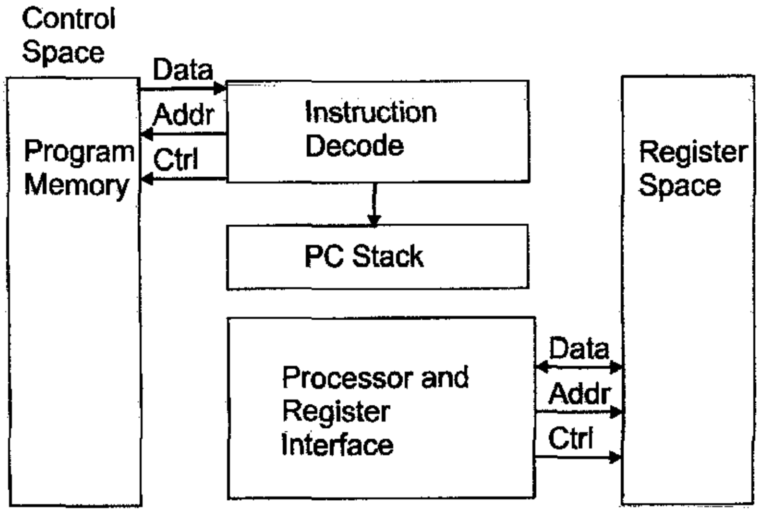


Figure 3: Harvard Architecture block diagram [10].

The purpose of this separation is to allow the program memory read instructions while the processor is accessing data and processing it. Therefore, the PIC microcontroller has the capability to execute software faster than many of its contemporaries. Instruction executions are performed based on the four clock cycles shown in Figure 4 below. Program memory will fetch the next instruction to be executed during an instruction execution cycle. The fetched instructions are latched in a holding or decode register. After an instruction has been fetched and is latched in a holding or decodes register, the program counter is incremented. This is shown in the first cycle, Q1, of Figure 4 [10].

In the next cycle, Q2, the data to be processed are read and put into temporary buffers. The data processing operations takes place during the third cycle, Q3. Last but not least, the resulting data value is stored during the last cycle, Q4 [10].

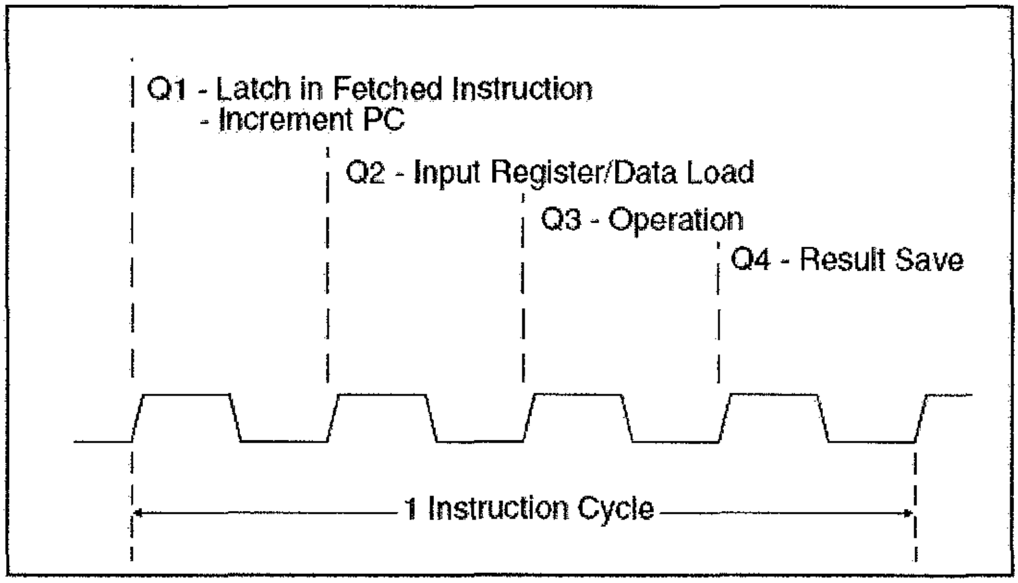


Figure 4: Four clock cycles for instruction execution.

The 7 address bits are explicitly defined as part of the instructions when accessing the PIC16 microcontroller family series. These 7 bits can specify up to 128 addresses in an instruction. The 128 register addresses can also be known as a bank. For the program counter, it maintains the current program instruction address in the program memory which contains the instructions for the PIC microcontroller processor. Each one is read out in sequence and being stored in the instruction register. The instruction decode and control circuitry will decode the program. The code that is executed takes place in the program memory. At each address, the content of the program memory consists of a full instruction. From the block diagram, a temporary holding register known as an accumulator is required to save a temporary value while the instruction fetches data from another register. Another alternative is by passing a constant value from the instructions. In this case, the accumulator used is the working register which is also called as the *w register* [10].

2.2.1 Parallel input/output ports

With respect to PIC16F877A, microcontroller ports are recognized and set according to the functionality of the required system. PIC16F877A is a family of PIC16 series which is more powerful in terms of its updated technology, enhancement of capacity and speed [10]. The pin diagram and its associated description are depicted in Figure 5 and Table 2 below, respectively.

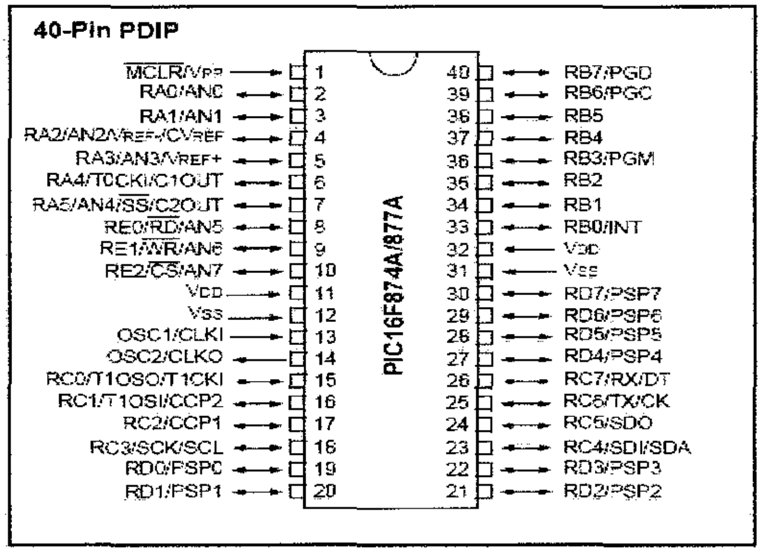


Figure 5: Pin diagram for PIC16F877A.

Table 2: Pin description for PIC16F877A.

ASSIGNED PIN	DESCRIPTION
RB7/PGD	Programming data for ICSP
RB6/PGC	Programming clock for ICSP
MCLR	Master clear for ICSP
VDD	Voltage power
VSS	Voltage ground
OSC1, OSC2	Oscillator 1 and 2 for crystal
RD4 - RD7	Port D4 to port D7 for input/output
SDA, SCL	Serial data and serial clock for I2C activity

2.2.2 The clock oscillator and instruction cycle

Any microcontroller is a complex electronic circuit, made up of sequential and combinational logic. At certain speed it steps in turn through a series of complex states, each state being dependent on the instruction sequence it is executing. Overall, the speed of the microcontroller operation depends on the clock frequency. Many essential timing functions are also derived based on clock frequency ranging from counters and timers functions to serial communications. Overall, the power consumption of the microcontroller strictly depends on the clock frequency where high operation speed uses more power compared to slow speed. Basically, the microcontroller has its specified range for its clock frequency. The selection of the clock frequency is up to the designer. The main clock signal is divided down by a fixed value into a lower frequency within a microcontroller. Each cycle of this slower signal is known as a machine cycle or an instruction cycle. In the action of the processor, the instruction cycle becomes the primary unit of time. For instance, it can be used to measure how long an instruction takes to execute. Basically, the original clock signal is retained to create time stages within the instruction cycle. In order to produce the instruction cycle time, the main oscillator signal in PIC16 series is divided by 4[11].

2.2.3 The timers module

In any microcontrollers, a timer is one of the important elements. Generally, a timer is a counter which is driven from either an external clock pulse or the microcontroller's internal oscillator. It can be either 8 bits or 16 bits wide. Under the control of the program control, timers can load data. The program control can stop or start the timers. An interrupt can be generated by configuring the timers when a certain count is reached. The interrupts can be used by user program to carry out accurate timing-related operations inside the microcontroller [7].

2.2.4 Power supply and its operating conditions

The standard logic voltage of most microcontrollers are 5V. There are also microcontrollers that can operate as low as 2.7V and some will tolerate 6V without any problem. In the datasheets, the information about the allowed limits of power supply voltage is stated. Basically, the voltage regulator is used to obtain the required power supply voltage when the device is operated from a main adapter or batteries. For instance, a 5V regulator is required in order to operate the microcontroller from 5V using a 9V battery.

2.2.5 The power on reset

In microcontrollers, there is a built in power on reset that keeps the microcontroller in the reset state until all the internal circuitry has been initialized. It can start the microcontroller program back to the beginning and is known as the state on power up. The microcontroller also can be reset by an external reset button.

2.3 Programming PIC Microcontrollers

After the program is written and translated into executable code, the resulting HEX file is loaded to the target microcontroller's program memory with the help of a device programmer. Some microcontroller development kits include on-board device programmers, so the microcontroller chip does not need to be removed and inserted into a separate programming device.

2.3.1 In circuit serial programming (ICSP)

The In Circuit Serial Programming (ICSP) circuit must be connected to the MCU in order to burn the chip. ICSP is actually a method where it is easier to program a PIC Microchip without removing the chip from the development board.

The connection of ICSP with the microchip is simple. ICSP provides five connections from the PIC ICSP programmer to the developer's board as described in Table 3.

Table 3: ICSP pin connections to microcontroller.

PIN	DESCRIPTION
MCLR (MASTER CLEAR)	Programming voltage, reset button can connect here to reset the program of chip
VCC(VOLTAGE COLLECTOR)	Power voltage, usually 5V is used
GND(GROUND)	Zero voltage
PGD (PROGRAMMING DATA)	Connected to RB7 which is the ICSP Data (ICSPDAT)
PGC(PROGRAMMING CLOCK)	Connected to RB6 which is the ICSP Clock (ICSPCLK)

Figure 6 below shows the physical pin connections of ICSP with the PIC16877A microcontroller:

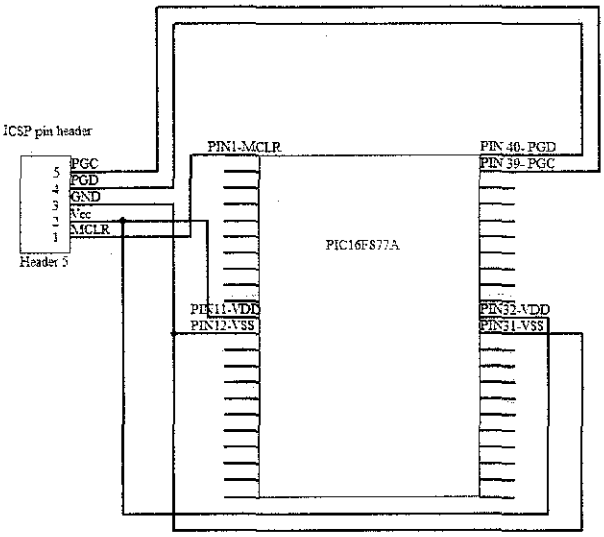


Figure 6: Pin connections of ICSP with PIC16F877A.

2.4 The Human and Physical Interface

Human interface can be devices that can give input and data response from the input data.Switches, keypads, sensors are some examples of input devices. While the output devices are the device that responds to the input device. It can be liquid crystal displays, motors, LEDs and so on.

2.4.1 Liquid crystal display interface

Liquid Crystal Displays (LCDs) consist of many types such as 1 line, 2 line and 4 line LCDs. We will be using a 1 line version with 16 characters. An LCD usually has 1 controller which can support about 80 characters. The LCD used has 14 pins with 2 extra pins. It is classified into 2 groups, which are serial and parallel connections. This LCD is a device where alphanumeric output can be displayed from microcontroller-based circuits. In serial LCD, it requires less input or output resources but they execute slower than the parallel LCD. LCD can be interfaced with various microcontrollers whether 4 bit or 8 bit [12]. Using a 4 bit LCD interface, one can reserve other ports of microcontroller for other functions. Figure 7 below illustrates the serial LCD connection. The description of the pins can be referred in **APPENDIX E.**

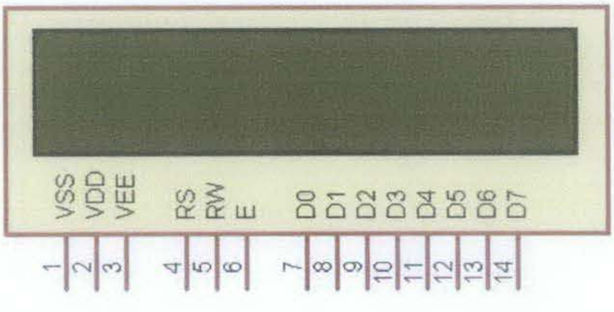


Figure 7: Serial LCD connection [12].

2.4.2 Humidity and temperature sensor

Humidity sensor is a sensor which measures and regularly reports the relative humidity in air. It is designed to sense relative humidity which measures both air temperature and moisture. The relative humidity is usually expressed as percentage which is the ratio of actual moisture in the air to the highest amount of moisture air of the measured environment. The warmer the air is, the more moisture it will be. Therefore, the relative humidity actually changes with fluctuations in temperature [13].

2.5 Serial Communication Overview

The serial communication is basically a method to send data into PC and vice versa. The serial communication interface makes communication between microcontroller and PC significant to a system. In addition, the computer programs are capable of sending data in bytes to transmit pin output and retrieve bytes from the receive pin input. The serial port converts data from parallel to serial forms; besides it changes the electrical representation of the data. Figure 8 below depicts a connection between PC and MAX232.

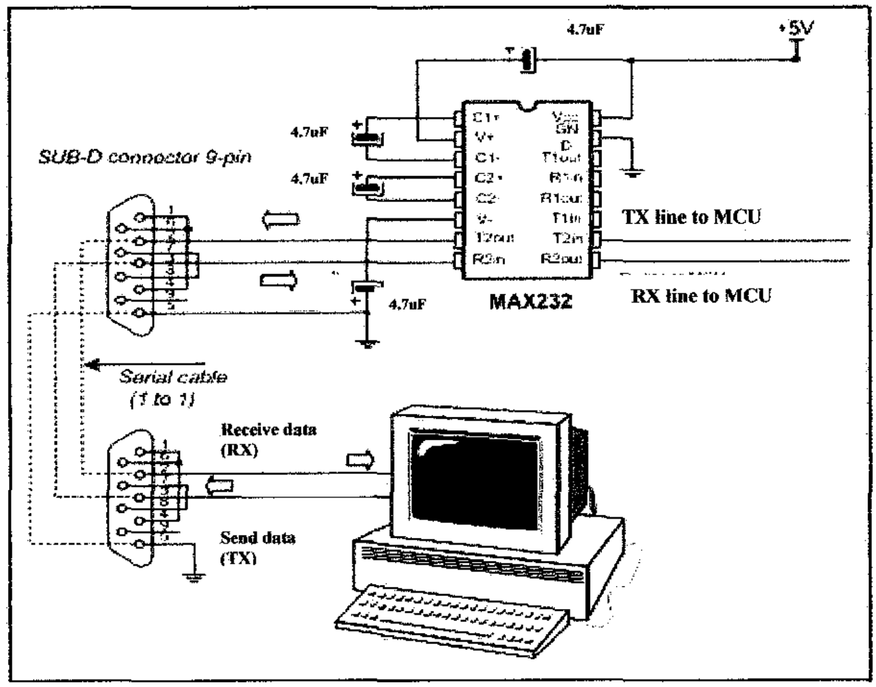


Figure 8: Connection between PC and MAX232.

2.5.1 Inter-intergrated circuit (I2C) protocol

The data communication of sensor is based on I2C method where the two I2C signals are serial data (SDA) and serial clock (SCL). Together, these signals make it possible to support serial transmission of 8-bit bytes of 7-bit-data device addresses plus control bits-over the two-wire serial bus. The device that initiates a transaction on the I2C bus is termed the master. The master normally controls the clock signal. A device being addressed by the master is called a slave. The data from humidity sensor is transmitted and received by the serial data and serial clock signals and being sent to the microcontroller [14].

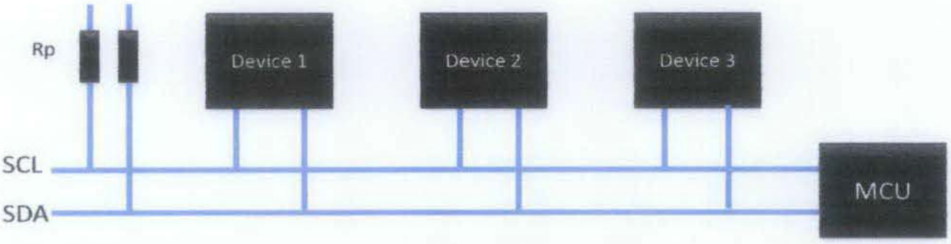


Figure 9: Communication configuration for I2C activity [14].

2.6 Radio Frequency (RF) Module for Wireless Communication

Radio frequency module ia an essential sub-system in the data logger system design. The subsystem is a wireless data link comprising radio frequency transmitter and receiver. TX434 and RX434 are selected for the system. These radio frequency modules require no licensing since the transmitter and receiver are used in accordance with low power devices such as in data logger applications.

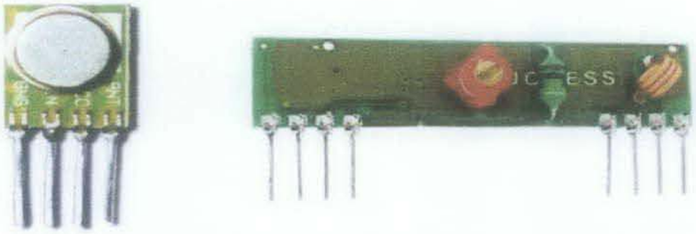


Figure 10: The transmitter and receiver for RF module.

2.7 External EEPROM Memory Device

The external EEPROM is a storage device that can store data for a long term since it has more than 200 years of data retention. The external EEPROM can be connected through I2C protocol or Serial Peripheral Interface (SPI) protocol. It depends on the chip we used. We plan to use the 24L256 EEPROM chip to interface with the microcontroller by using the I2C protocol. This storage device interface concept can be applied to Secure Digital (SD) card and Universal Serial Bus (USB) device. This advanced, low power device has a write capability of up to 64 bytes of data and capable of both random and sequential reads up to 256K boundary. By using the external EEPROM, we can connect more than 1 EEPROM chip to create a memory of more than 256K bytes instead of using the built-in internal EEPROM. EEPROM uses floating gate technology. Its dimension is finer, so that it can exploit another means of charging its floating gate. This is known as Nordheim Fowler tunneling. With this method, it is possible to electrically erase the memory cell, as well as write to it. To allow this to happen, a number of switching transistors need to be included around the memory element itself, so the high density of EEPROM is lost. Generally, EEPROM can be written to and erased on a byte-by-byte basis. This makes it especially useful for storing single items of data. Both writing and erasing take finite time, up to several milliseconds, although a read can be accomplished at normal semiconductor memory access times [11].

2.8 Printed Circuit Board

This design is one of the last stages of development of circuit after the testing, troubleshooting and result of circuit passes the requirements. A Printed Circuit Board (PCB) design is also known as a printed wiring board. PCB has copper tracks connecting the holes through where the components are placed. It basically serves two purposes; it places the mounted components and provides the electrical connections between the components. The fabrication of printed circuit board is achieved by an etching process based on the Gerber file created.

CHAPTER 3
METHODOLOGY

Methodology is one of the important parts of the project development. We will explain the procedure identification process, the tools that will be used and also the proposed work overview. The process of the project development is segregated into few parts within two semester's time frame. In general, the process comprises the integration of hardware and software.

3.1 Procedure Identification

The procedures involved in Wireless Sensor Data Logger System Design are basically based on the overall block diagram illustrated in Figure 1 from Chapter 1. The procedures are identified to ensure that the project can be accomplished within the time frame provided. From the block diagram, the specific flow chart is illustrated in Figures 11 and 12 for FYP1 and FYP2 respectively. The first flow chart shows the procedure for the targeted work to be accomplished during FYP1, while the second flow chart shows the procedure work for FYP2. Generally, the process starts with some research of literature review and knowledge about the project such as the microcontroller features, serial communications, and wireless communications. By identifying their functions, we can start designing the circuit part by part.

For the first step, we plan to display the sensor data through the LCD display and integrate the sensor data with the microcontroller. Then, further research about the external EEPROM chip is done in order to connect the chip with the microcontroller. The serial interface allows communication between microcontroller and PC via serial port. The radio frequency module is integrated to the microcontroller to conduct a wireless data transmission. The last stage of system design is the design of PCB and GUI in order to make the project more presentable.

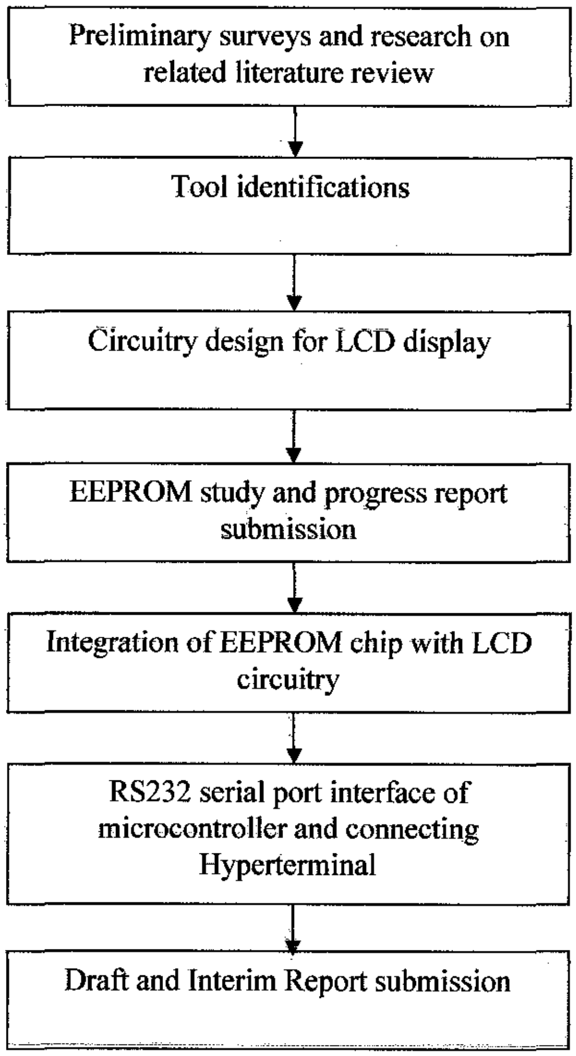

Preliminary surveys and research on related literature review
Tool identifications
Circuitry design for LCD display
EEPROM study and progress report submission
Integration of EEPROM chip with LCD circuitry
RS232 serial port interface of microcontroller and connecting Hyperterminal
Draft and Interim Report submission

Figure 11: Procedure identification flow for FYP1.

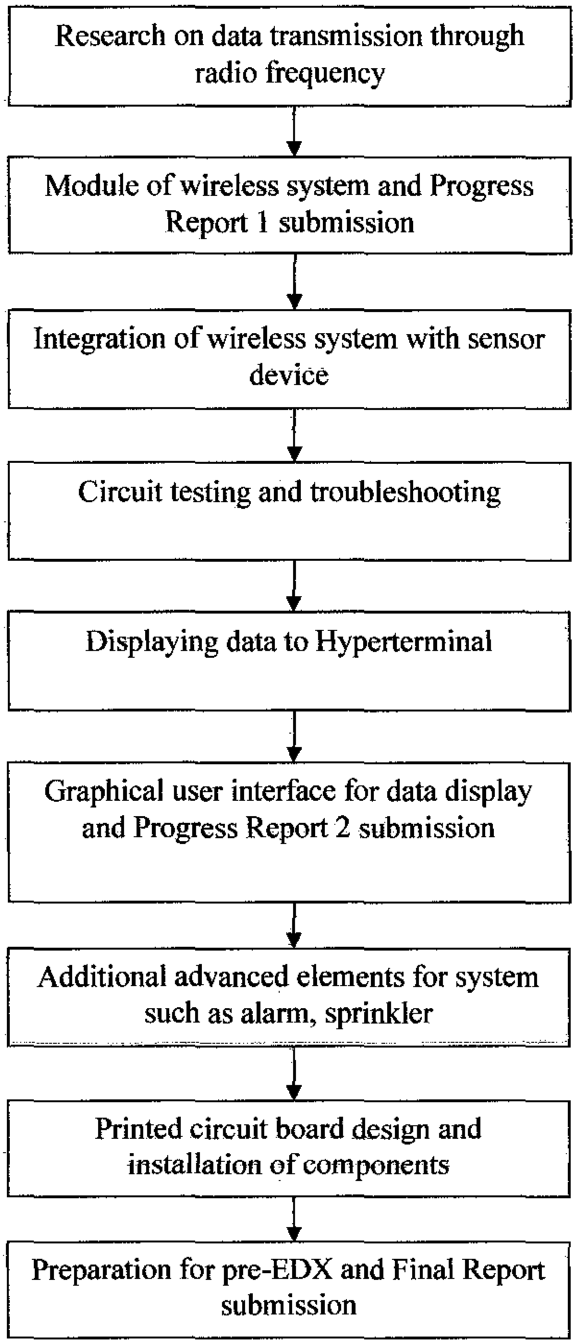

Research on data transmission through radio frequency
Module of wireless system and Progress Report 1 submission
Integration of wireless system with sensor device
Circuit testing and troubleshooting
Displaying data to Hyperterminal
Graphical user interface for data display and Progress Report 2 submission
Additional advanced elements for system such as alarm, sprinkler
Printed circuit board design and installation of components
Preparation for pre-EDX and Final Report submission

Figure 12: Flow procedure for FYP2.

3.2 Tools and Equipment Required

Tools play important roles in developing this data logger system. Since the project will be involving the software and hardware integration, both of software and hardware development tools are required. The tools that are proposed are the common software which mostly is widely used in electronic industries.

3.2.1 Software development tools

- MPLAB IDE and PICKIT 3
- CCS compiler
- ALTIUM Summer Designer - PROTEL
- Microsoft Visual Studio C++
- REALTERM

3.2.2 Hardware development tools

- Microcontroller PIC16F877A
- Humidity and temperature sensor
- Liquid crystal display (LCD)
- External EEPROM 24L256 chip
- MAX 232 level converter IC
- RS232 /RS485 serial port
- Personal computer

3.3 Proposed Work for Prototype Installation

The flow of installation for the prototype includes many crucial processes that should be done during the time frame given. Microcontroller theories are being applied to the project and new knowledge is discovered.

3.3.1 Clock oscillator calculation for PIC16F877A

For the microcontroller to operate, a clock is required to give a clock cycle. We use the crystal/ceramic timing devices that can be connected to the microcontroller through oscillator port denoted by OSC1 and OSC2. This timing device consists of a crystal oscillator plus two small capacitors. An instruction is executed by fetching it from the memory and then decoding it. This usually takes several clock cycles and is known as instruction cycle. The calculation for capacitors and crystal component are as follows:

Cp: Parasitic Capacitance, usually about 8pF
Ca: Actual value of capacitor, the capacitor used is 15pF
(Ca + Cp)/2 = (15p + 8p)/2 = 11.5
Therefore, the crystal that we will be using is 12 MHz.

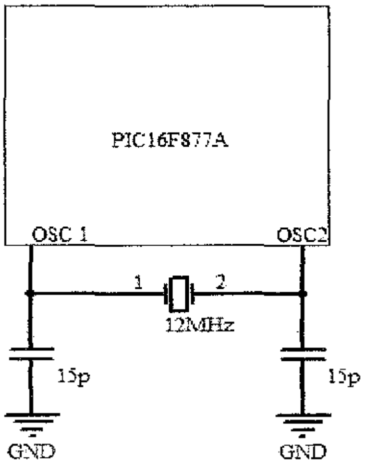


Figure 13: Schematic diagram for clock oscillator with PIC16F877A.

3.3.2 Programming development process

This flow diagram describes the basic process of developing a program using the microcontroller.

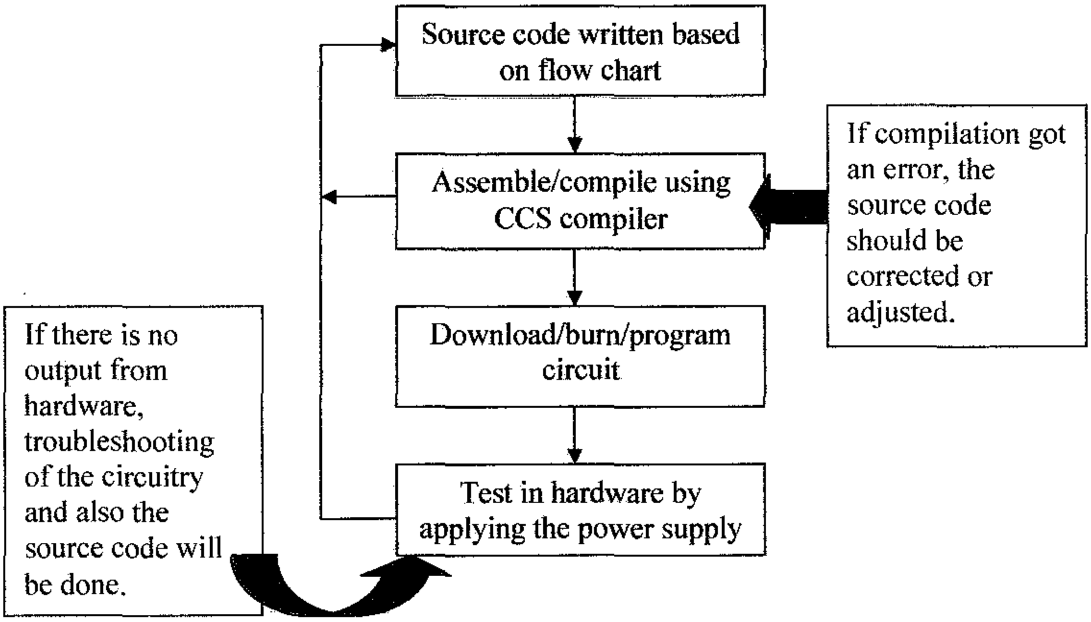

Source code written based on flow chart
Assemble/compile using CCS compiler
Download/burn/program circuit
Test in hardware by applying the power supply
If compilation got an error, the source code should be corrected or adjusted.
If there is no output from hardware, troubleshooting of the circuitry and also the source code will be done.

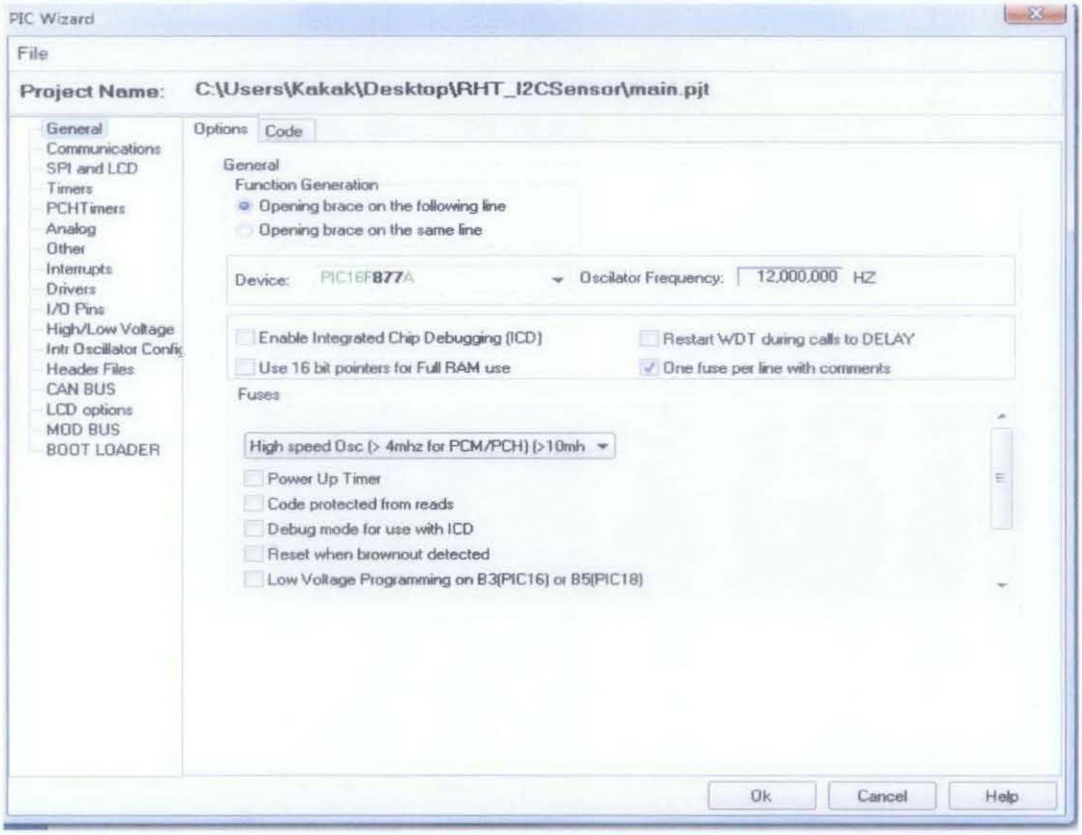
Figure 14: Flow diagram for programming development process.

3.3.3 Software programming installation

A circuit will not work without the microcontroller being programmed. Therefore, we learn and discovered the MPLAB IDE software together with the CCS compiler to create the program for digital alarm clock design. This software design can actually be created by various types of programming software. But we preferred using CCS compiler because the wizard and built-in functions in CCS compiler make it easy to create the basic settings based on the hardware device. They are also user friendly. The files are then imported to MPLAB IDE in order to use the PICKIT3 device to burn the program into the microcontroller chip.

3.3.4 Initialization mode of MCU chip

The initialization setting for the circuit is created by using the PIC Wizard in CCS Compiler.The screenshot for the initilazation of the microcontroller chip is depicted in Figure 15.



PIC Wizard

Project Name: C:\Users\Kakak\Desktop\RHT_I2CSensor\main.pjt

General

Function Generation

● Opening brace on the following line

○ Opening brace on the same line

Device: PIC16F877A

Oscillator Frequency: 12,000,000 HZ

□ Enable Integrated Chip Debugging (ICD)

□ Restart WDT during calls to DELAY

□ Use 16 bit pointers for Full RAM use

☑ One fuse per line with comments

Fuses

High speed Osc (> 4mhz for PCM/PCH) (>10mh

□ Power Up Timer

□ Code protected from reads

□ Debug mode for use with ICD

□ Reset when brownout detected

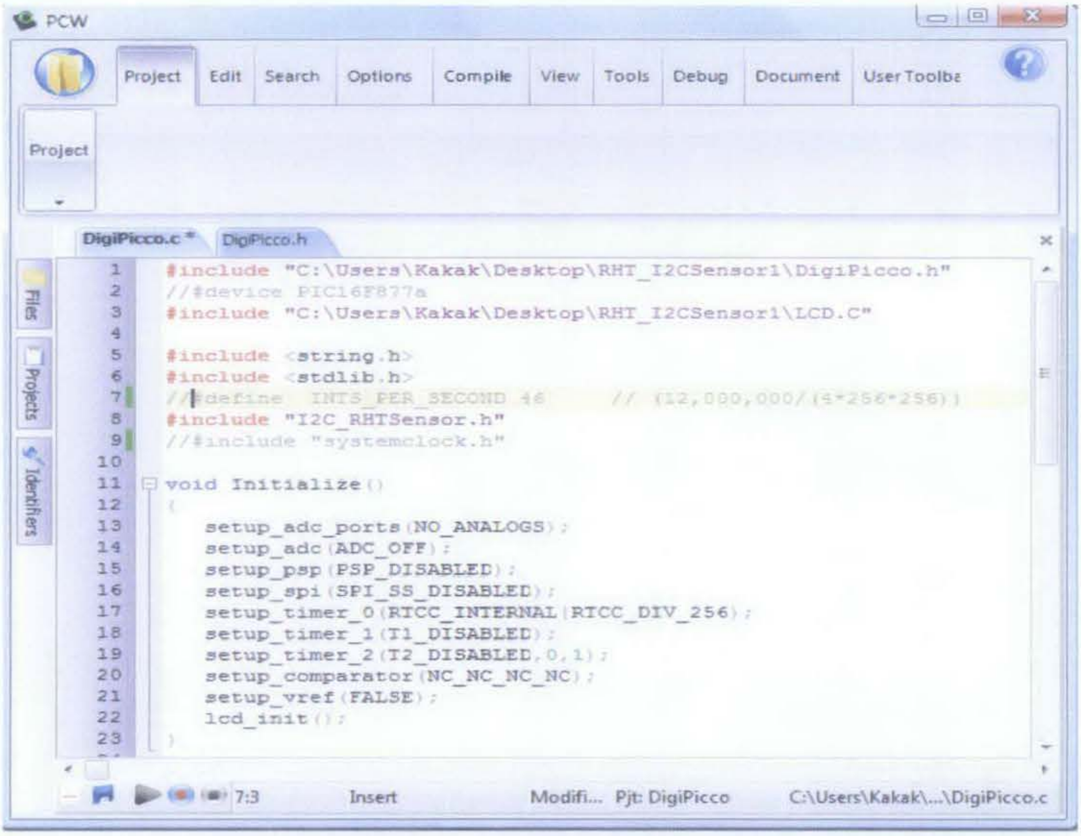
□ Low Voltage Programming on B3(PIC16) or B5(PIC18)

Figure 15: Initializing the microcontroller chip using the PIC Wizard.

The oscillator frequency used is about 12 MHz. For slower execution, a lower value of oscillator frequency can be used.The files included together in DigiPicco.c:

- Header file > **DigiPicco.h**
- LCD driver > **LCD.c**
- String library > **string.h**
- Standard Library > **stdlib.h**

Figure 16 below illustrates a sample C code for PIC16F877A initialization.



```c
#include "C:\Users\Kakak\Desktop\RHT_I2CSensor1\DigiPicco.h"
//#device PIC16F877a
#include "C:\Users\Kakak\Desktop\RHT_I2CSensor1\LCD.C"

#include <string.h>
#include <stdlib.h>
//#define  INTS_PER_SECOND 46        // (12,000,000/(4*256*256))
#include "I2C_RHTSensor.h"
//#include "systemclock.h"

void Initialize()
{
    setup_adc_ports(NO_ANALOGS);
    setup_adc(ADC_OFF);
    setup_psp(PSP_DISABLED);
    setup_spi(SPI_SS_DISABLED);
    setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256);
    setup_timer_1(T1_DISABLED);
    setup_timer_2(T2_DISABLED,0,1);
    setup_comparator(NC_NC_NC_NC);
    setup_vref(FALSE);
    lcd_init();
}
```

Figure 16: C programming code for initialization of PIC16F877A.

3.3.5 The interface for humidity and temperature sensor

The interface of humidity sensor to PIC16F877A based on the humidity and temperature sensor datasheet is given in Figure 17 below.

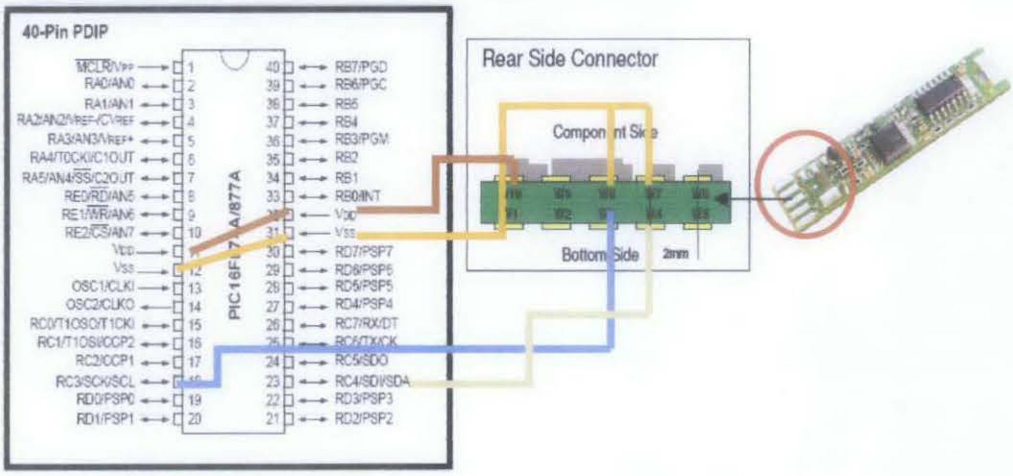
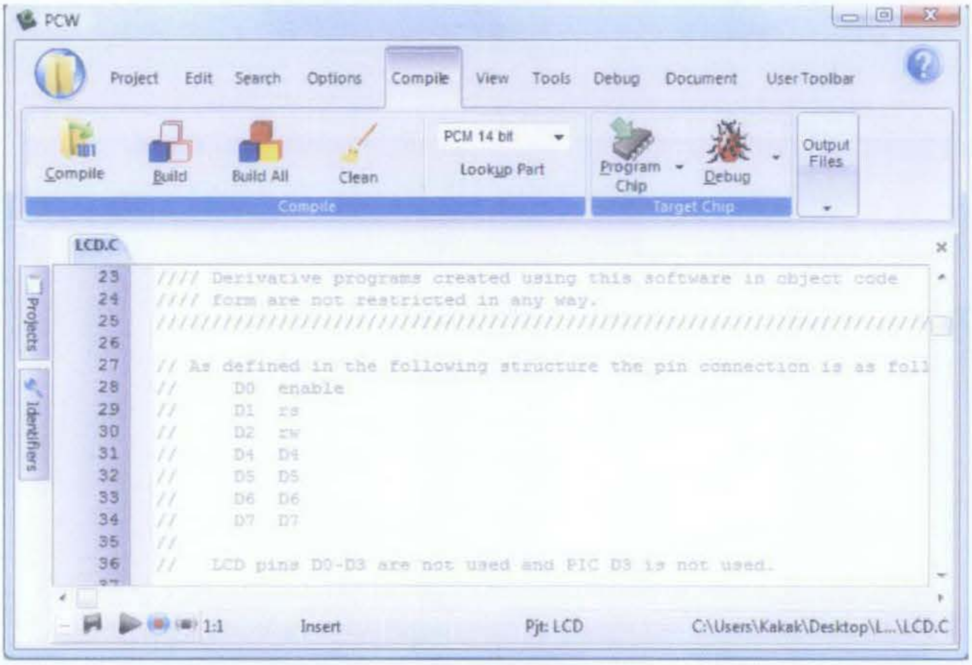


Figure 17: Connection of humidity sensor with PIC16F877A.

3.3.6 Interfacing 4-bit LCD

The first step to display output from LCD would require a sequence of program to be created. But in CCS compiler, a LCD driver has already created the sequence of the program. The following pins are the pin connections from LCD to microcontroller and these have been fixed by the driver. Only 4 bits are used for the LCD interfaces which are D7 to D4. It saves the use of other pins but responds slower than an 8 bit interface. Figure 18 below shows a LCD driver from CCS compiler.



```
//// Derivative programs created using this software in object code
//// form are not restricted in any way.
/////////////////////////////////////////////////////////////////////

// As defined in the following structure the pin connection is as foll
//     D0  enable
//     D1  rs
//     D2  rw
//     D4  D4
//     D5  D5
//     D6  D6
//     D7  D7
//
//   LCD pins D0-D3 are not used and PIC D3 is not used.
```

Figure 18: LCD driver from CCS compiler.

From the program code of LCD in **APPENDIX C,** the lcd_putc module is used to display a text or number on the LCD. The program below displays the value of relative humidity in percentage.

```
// The formula to convert hexadecimal of humidity and temperature into decimal value.
   Humidity = make32(0,0,HumH,HumL)*100/0x7FFF;
   Temperature = make32(0,0,TempH,TempL)*165/0x7FFF-40;
// lcd_putc module from driver is used to display value of humidity
   printf(lcd_putc,"\fR.Humidity=%LU%%\n",Humidity);
   printf(lcd_putc,"Temperature=%Lu C\n",Temperature);
```

3.3.7 Experimenting the accuracies of sensor

The accuracy of sensor was tested for both, the temperature and the humidity. Its accuracies can be detected by comparing the measured value with the actual value from other sources. The experiment took place at the Chemical Lab where the Isotech Model Jupiter 650 of constant temperature bath was used as the heating device. The temperature reading of sensor is compared with the master standard units of a digital thermometer. The data was recorded starting from 30 degree Celsius of temperature bath to 70 degrees of temperature bath. Figure 19 below shows the setup for the temperature heating experiment. Both sensors and master standard unit device were put into the space in the Isotech Jupiter machine. Here the surroundings inside the machine acted as the heating element. The values of both measured devices were recorded in a table and displayed in a plotted graph. The accuracy test for humidity is conducted by comparing the sensor with the hygrometer and the anemometer device. Both are humidity measurement devices that are basically used in industry applications.

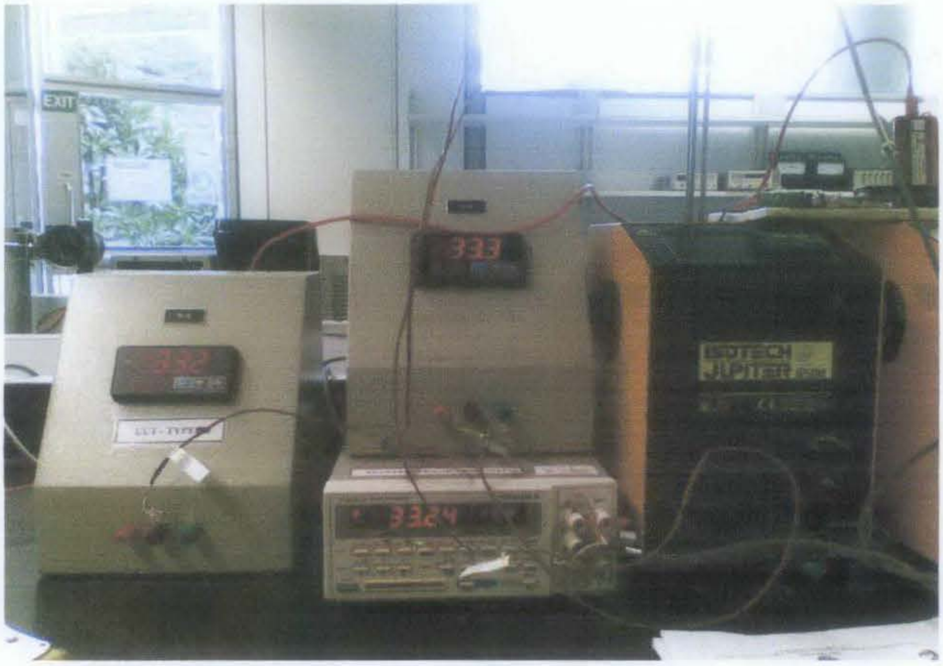


Figure 19: Setup of temperature measurement experiment.

The whirling hygrometer determines the percentage relative humidity (RH) by measuring the evaporation of water into the surrounding air. Two thermometers are placed in flowing air; one thermometer bulb was covered by a wet wick. The RH can be read off the slide rule calculator integrated into the hygrometer. To take the measurement of humidity, the instrument was opened by withdrawing the inner frame from the case. Then, thoroughly the wick was wet by placing the exposed end under cold running water or immersing it in water for about 30 seconds. This would wet both the exposed wick and that coiled in the wick container. The frame was rotated for 30 to 60 seconds at between 2 and 3 revolutions per second. When the hygrometer is closed the slide rule can be used to calculate the relative humidity percentage directly from the wet and dry temperatures. The calculator has two scales; the upper scale should be used for dry bulb temperature up to 20 degree Celsius. For higher temperature, the lower scale should be used. The steps to read the humidity reading of hygrometer, first, locate the wet bulb temperature on the relevant scale. The dry bulb temperature is aligned with the wet bulb. The reading of relative humidity is read from the centre scale at the location of the arrow.

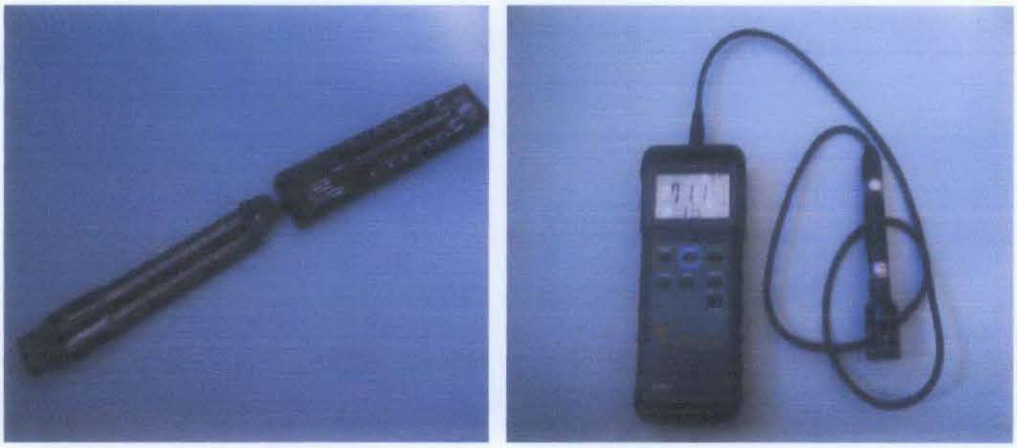


Figure 20: The humidity measuring device; hygrometer and anemometer.

The anemometer is a digital humidity measurement device. This is straight forward compared to hygrometers because the reading of humidity is displayed directly from the anemometer. The devices measured the humidity of the surroundings from 8 am in the morning to 8 pm. The data measurements of sensor between devices are compared and plotted in a bar chart.

3.3.8 Serial Communication Interface

The serial communication allows the communication between the microcontroller and PC. The sensor data should be sent through this serial and displayed in the terminal called Realterm. The pin outs for serial communication is connected as illustrated in Figure 21 below. Two of input output pins of PIC16F877A microcontroller were configured as transmit pin and receive pin and connected directly to MAX232 level converter IC at pin 11 and 12, respectively. The MAX232 level converter IC converts microcontroller signal level 0V and +5V to +3V/+12V from a single supply of 5V. This is due to the fact that PIC16F877A microcontroller sends data serially in logic level of 0V for low logic, and +5V for high logic. However, RS232 serial port uses different logic levels, +3V and +12V for communication. Therefore, MAX232 converts the TTL logic level during data transmission.

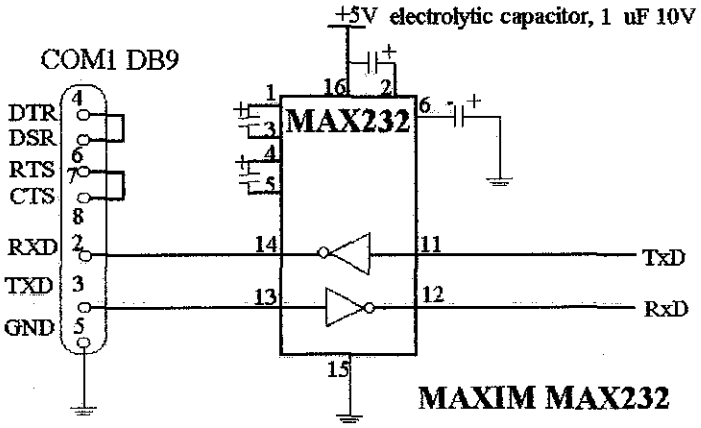


Figure 21: MAX 232 interface layout.

In order to allow the communication, several initializations are required to be executed. The initialization properties are set in the realterm as follows:

- Baud rate (bits per second): 2400
- Data bits: 8
- Parity bit: None
- Stop Bits: 1

3.3.9 External EEPROM Access

The closed up view for pin configuration and connections of EEPROM depicted in **APPENDIX B** is shown in Figure 22.The pin A0 is connected to 5V to get logic 1.Therefore, its address does not clash with the sensor's address that shares the same pin in the MCU circuitry. It is important for us to make sure different addresses are used between devices that share the same port of the MCU in order for them to function properly and be recognized. The EEPROM driver of 24LC256 is included from CCS Compiler library which is shown in **APPENDIX C**. As we see in program code number 5, the 24LC256 EEPROM library shows that both write and read operations follow the I2C protocol. In I2C protocol, the master initiates communications on the bus and controls the bus with one or more slave devices. Basically, it begins with a start condition and ends up with a stop condition.

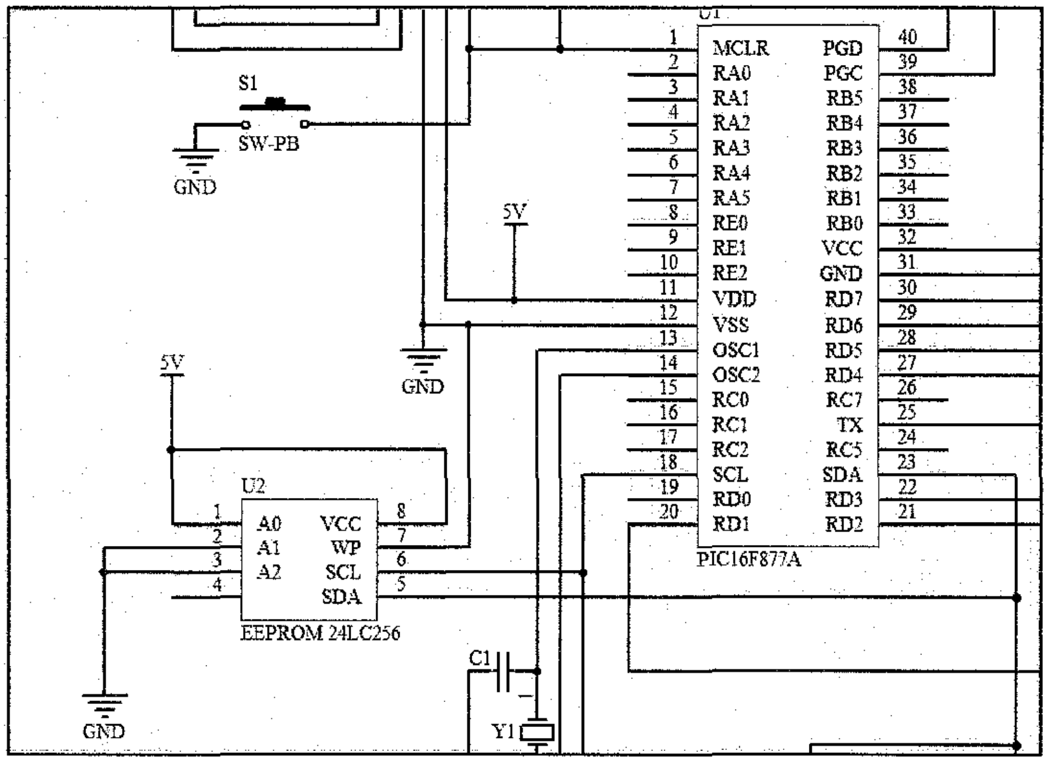


Figure 22: Close up view of EEPROM connections in the MCU circuitry.

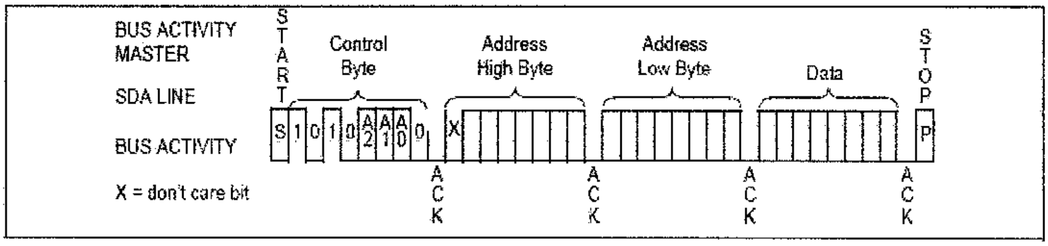


Figure 23: Write operation mode for 24LC256 EEPROM.

In Figure 23, the start condition is generated first. This figure relates with the code of the write operation where the slave actually writes the device address 0XA2 to get 1010 0010. From Figure 23, the first 4 bits refers to the control byte of EEPROM device for both write and read operation. The next three bits are the chip select bits A0, A1 and A2. These chip select bits depends on the user whether to use chip select A0, A1 or A2. The selected chip is connected to high logic to tell that we are using that chip select bits. In read operation, the $R/\overline{W}$ bit should be 0. From the program, it shows that A0 chip select was set to 5V. As shown in the datasheets in **APPENDIX E**, there are page-write and byte-write operations. For a byte write, one byte of data transfer taken place from the MCU to the EEPROM; the transfer is then acknowledged by the EEPROM. While in page write, data transfers can allow up to 16 bytes. The master generates a stop condition when everything was completed. A sample code to achieve a byte-write is shown below.

```
void write_ext_eeprom(long int address, BYTE data)
{
   short int status;
   i2c_start();
   i2c_write(0xa2);//a0=1
   i2c_write(address>>8);
   i2c_write(address);
   i2c_write(data);
   i2c_stop();
   i2c_start();
   status=i2c_write(0xa2);
}
```

Moreover, Figure 24 below depicts the EEPROM read operation.

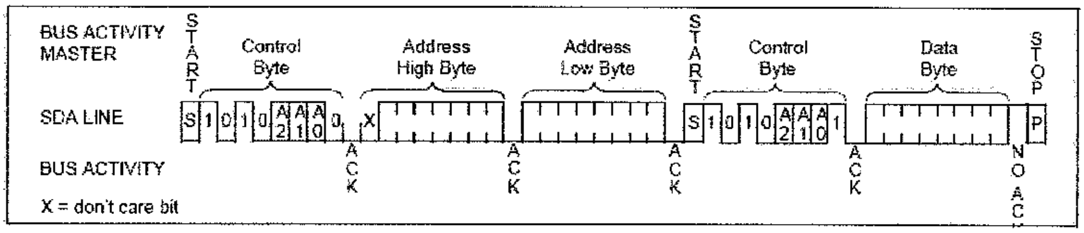


Figure 24: Read operation mode for 24LC256 EEPROM.

In the read mode, the $R/\overline{W}$ should be 1 to indicate it is in the read mode. The following sample code shows that the program starts with a start (i.e.; i2c_start( )) and ends with a stop (i2c_stop( )) conditions.

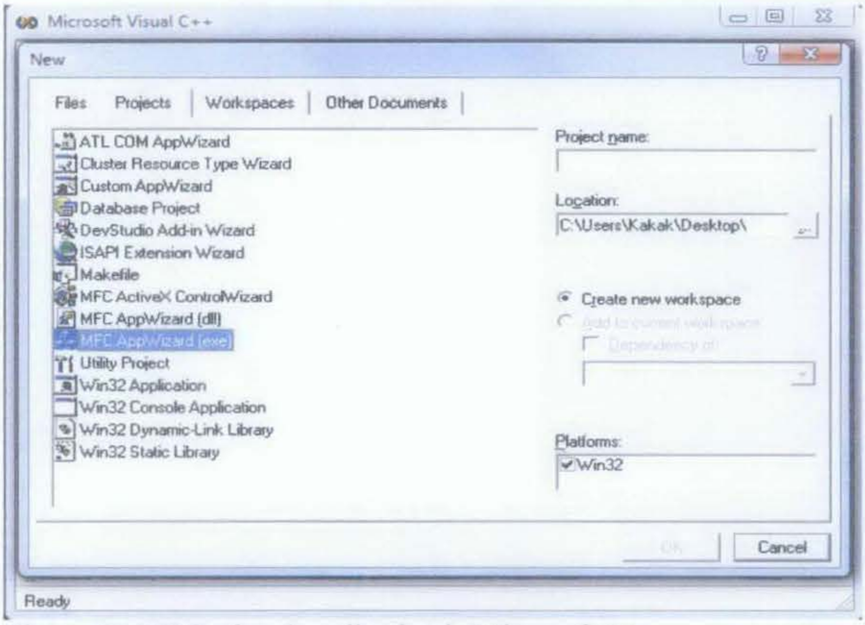
```
BYTE read_ext_eeprom(long int address) {
   BYTE data;
   i2c_start();
   i2c_write(0xa2);//a0=1
   i2c_write(address>>8);
   i2c_write(address);
   i2c_start();
   i2c_write(0xa3);
   data=i2c_read(0);
   i2c_stop();
   return(data);
}
```

In order to test whether or not the functionality of the EEPROM device works, or we used the input.c driver and included it in the main code. This input.c driver allows the user to kick in the data that he/she wants to save into one of the locations in the EEPROM device using a keyboard, and also allows the user to read back the value from the location. The program below explains that when the letter 'R' is received, the value from the EEPROM is read; while the letter 'W' indicates that the user wants to write a value to be saved in any location of the EEPROM.

```
#include "input.c

void main() {

   BYTE value, cmd;
   EEPROM_ADDRESS address;
   Initialize();
   printf("\r\nWelcome and Hye Noorshafrina\r\n");
   init_ext_eeprom();

   do {
      do {
         printf("\r\nRead or Write: ");
         cmd=getc();
         cmd=toupper(cmd);
         putc(cmd);
      } while ( (cmd!='R') && (cmd!='W') );

      printf("\n\rLocation: ");
```

3.4.0 Graphical User Interface Design

We had used the Microsoft Foundation Class wizard to create a new project workspace for the graphical user interface design. The location of project can be changed to a location that we want to save. In this project, the Win32 platform which refers to the recent versions of the windows operating system that runs in a 32-bit mode is used. This application generates an application that has built in functionality which when compiled; it implements the basic features of windows executable application. The Microsoft Foundation Class wizard is depicted in Figure 25.



Project name:

Location: C:\Users\Kakak\Desktop\

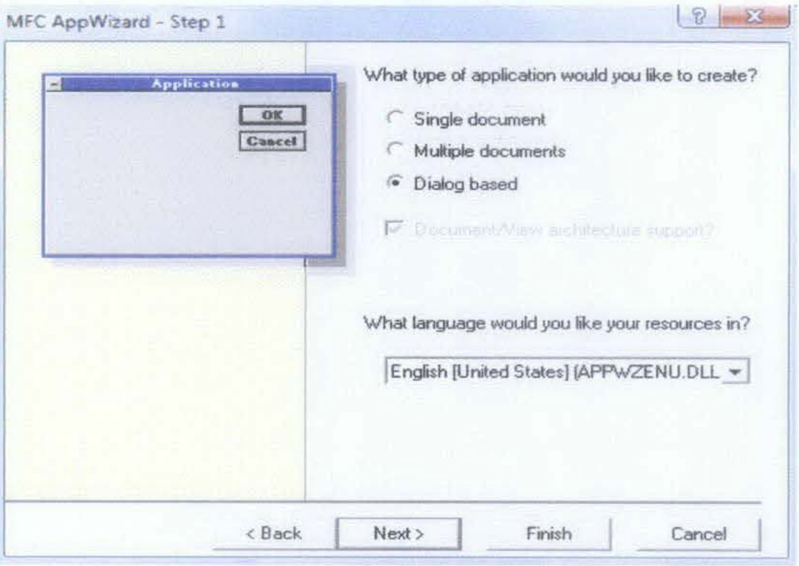
● Create new workspace

Platforms: ☑ Win32

Figure 25: MFC wizard application in Microsoft C++.

A dialog-based GUI is implemented and created using a text file called a resource file which has file extension ".rc". This dialog box is a window that holds other windows controls and can be referred as a container. It is actually the primary interface that involves interaction between the user and computer. We can design the dialog box by selecting the boxes and placing them in the worksheet. But the variable for each of these boxes must be assigned because the user must program these variables according to their desired functions.

The dialog based type is selected as in Figure 26 below.



MFC AppWizard - Step 1

What type of application would you like to create?

○ Single document
○ Multiple documents
● Dialog based

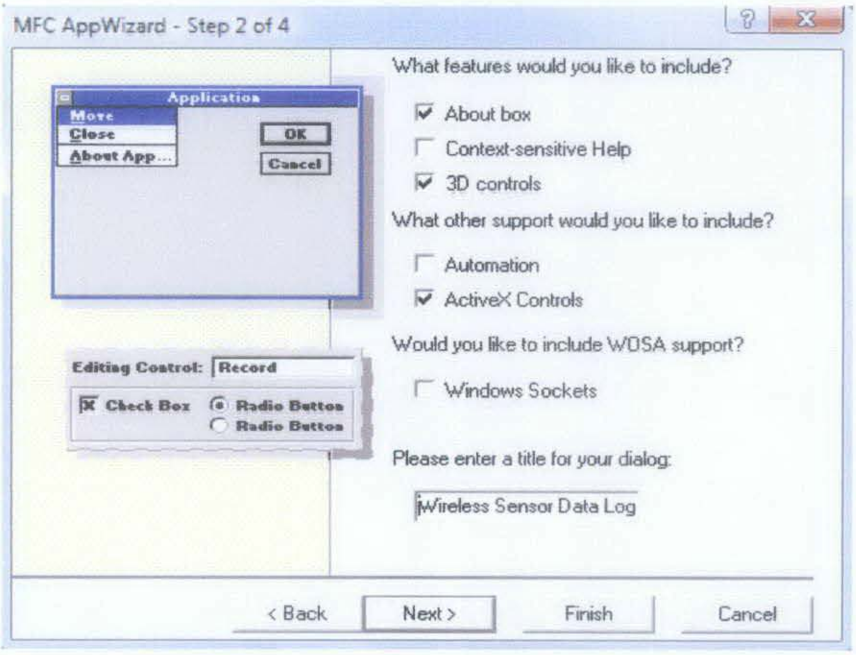
☑ Document/View architecture support?

What language would you like your resources in?

English [United States] (APPWZENU.DLL

Figure 26: Screenshot for step 1 of GUI design.

The title of the dialog box for this project was entered as 'Wireless Sensor Data Logger System Design'. Figure 27 shows step 2 for MFC wizard application.



MFC AppWizard - Step 2 of 4

What features would you like to include?

☑ About box
☐ Context-sensitive Help
☑ 3D controls

What other support would you like to include?

☐ Automation
☑ ActiveX Controls

Would you like to include WOSA support?

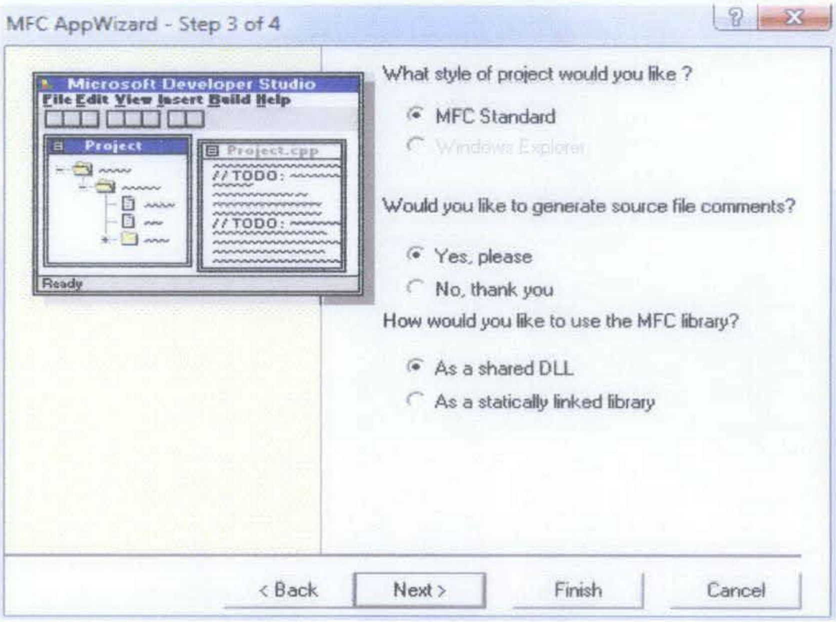
☐ Windows Sockets

Please enter a title for your dialog:

Wireless Sensor Data Log

Figure 27: Screenshot for step 2 of GUI design.

Source code files will automatically be generated and any code modification can be made easily; Figure 28 below illustrates the options to generate such a source file.



MFC AppWizard - Step 3 of 4

What style of project would you like ?

● MFC Standard

○ Windows Explorer

Would you like to generate source file comments?

● Yes, please

○ No, thank you

How would you like to use the MFC library?

● As a shared DLL

○ As a statically linked library

Figure 28: Screenshot for step 3 of GUI design.

Figure 29 below shows the design of graphical user interface with buttons and empty boxes which are arranged according to our desired functions. The 'close comm' and 'open comm' buttons allow the used port to be recognized.

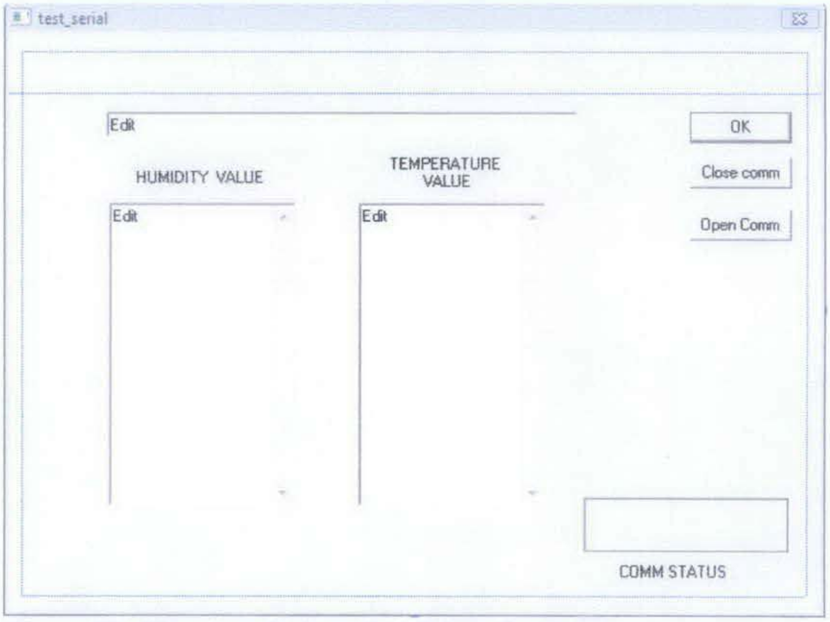
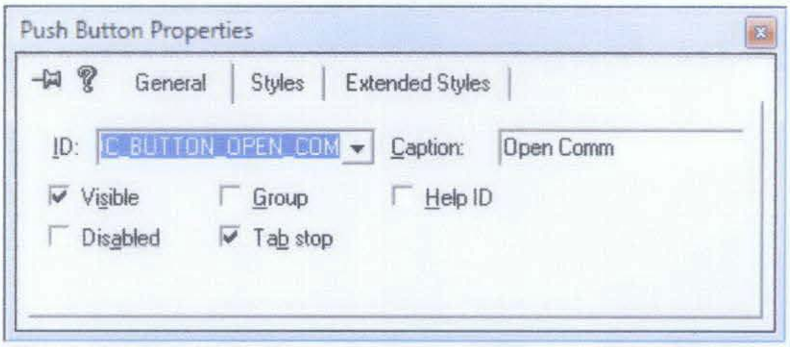


Figure 29: The view design of GUI interface.

The variables can be assigned by clicking the right side of mouse on the respective button box and by selecting the properties. This can be shown in Figure 30 below:



Push Button Properties

ID: C_BUTTON_OPEN_COM   Caption: Open Comm

☑ Visible   □ Group   □ Help ID

□ Disabled   ☑ Tab stop

Figure 30: Properties for assigning variables to selected buttons.

A sample code below shows a situation where if the "open comm" button is pressed, the message of "comm. Port already open" will pop up.

```
void CTest_serialDlg::OnButtonOpenCom()
{

if(m_serial_flag)

{
        MessageBox("comm port already open") ;
        return  ;
} }
```

The serial comm.unication used is 2400 baud rate with no parity bit and one stop bit

```
PortDCB.BaudRate = 2400;
PortDCB.fBinary = TRUE;
PortDCB.fParity = TRUE;
PortDCB.fOutxCtsFlow = FALSE;
PortDCB.fOutxDsrFlow = FALSE;
PortDCB.fDtrControl = DTR_CONTROL_ENABLE;

PortDCB.fAbortOnError = FALSE;

PortDCB.ByteSize = 8;
PortDCB.Parity = NOPARITY;
PortDCB.StopBits = ONESTOPBIT;
```

CHAPTER 4
RESULTS AND DISCUSSION

The results and findings obtained from the project are discussed thoroughly in this section. The process of project development involved masses of information and engineering principles. Apparently, the results presented in this chapter are the sensor data analysis through wireless and wired communication, PIC16F877A microcontroller circuitry and GUI interface. There are also other findings which will be explained further in the remaining sections of this chapter.

4.1 Sensor Output Display Test on LCD

From the first stage of circuit development, the measured output from the sensor is displayed by the LCD. Temperature and humidity data are sensed by the integrated sensor. The PICKIT3 connected through USB port supplies power to the circuit with 5 volts. The range of temperature that can be read by this sensor is around -25 to +85 degrees Celsius. In a normal condition, the room temperature should be around 27 degrees Celsius while the humidity is about 60 percents and above. The output display in Figure 31 shows the value of temperature and humidity on 1 November, 2010 at 1.36pm in room environment. Therefore, the measured data shows that the environment is in normal condition. Table 4 below shows the reading from the LCD display.

Table 4: LCD display of measured value from sensor.

SENSOR	MEASURED VALUE
HUMIDITY	67 %
TEMPERATURE	28 °C

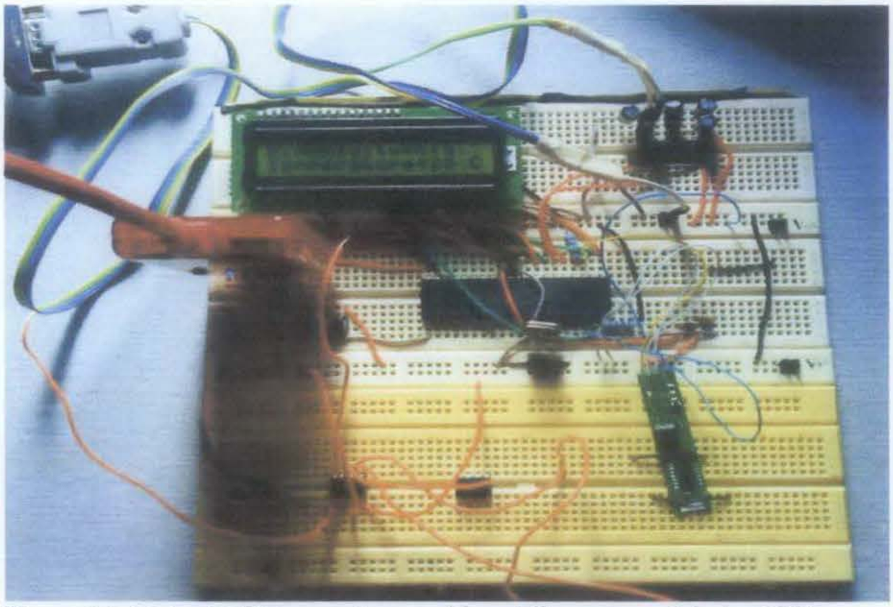


Figure 31: Output of temperature and humidity sensor data to LCD.

4.2 Accuracies of Sensor Board Compared to Other Devices

The accuracy test of the sensors is being run to compare them with standard measurements and to prove that they are accurate. By comparing these measurements with the standard reading, the accuracy level of the sensors could be determined. This is important since the sensors would be applied in real applications. The data analysis for the measurements of humidity and temperature are both tested in different methods. The method used has been already explained and discussed in the methodology section of the report.

4.2.1 Humidity data analysis

The humidity reading is compared with two different humidity devices, the hygrometer and the anemometer. These devices are common humidity measuring devices. The readings of the outside surrounding were taken from 8am in the morning until 8pm. Table 5 below shows the measured humidity data of the environment in normal weather, which is not too windy, not too hot and not raining.

Table 5: Measured value of humidity and temperature for measuring devices.

TIME	HYGROMETER	ANEMOMETER	SENSOR
8:00 AM	85	88	86
10:00 AM	72	77	75
12:00 AM	65	67	68
2:00 PM	71	68	70
4:00 PM	58	56	54
6:00 PM	62	64	66
8:00 PM	78	81	76

From the measured value, it shows that the reading of the integrated sensor is almost the same as the other two devices. The highest humidity percent is in the morning which is around 85 percents and above. This shows that the quantity of water in the air during 8am in the morning is high. While the lowest humidity percent is during 4pm in the evening. The weather at this time was quite hot; therefore the quantity of water is lower. The hygrometer and anemometer are common devices which gives the standard reading of the humidity in real life. Therefore, the integrated sensor is applicable in daily life applications and has an accurate humidity reading since its measured value is almost the same as the value of the devices. The result of humidity percent measured from the three devices is summarized in Figure 32.

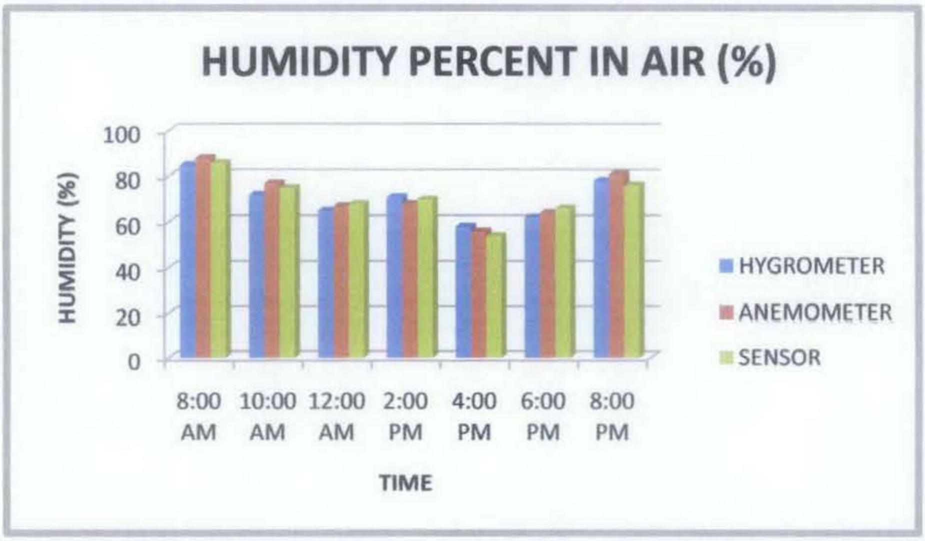


Figure 32: The measurements of humidity percentage in air.

4.2.2 Temperature data analysis

The accuracy of temperature is determined by comparing the measured data of the integrated sensor with the master sensor unit measurement. The bath temperature was used as a heating element. The temperature value of master standard unit (MSU) and value of sensor were recorded for each increasing value of bath temperature. The measurement was taken from 30 to 70 degree Celsius.

Table 6: The reading of temperature measured.

UNIT UNDER TEST (UUT)			SENSOR
MASTER STANDARD UNIT (MSU)			3 WIRE RESISTANCE THERMOMETER
NO	BATH TEMP	MSU READING	UUT READING
1	30	32.04	29
2	40	40.67	37
3	50	52.12	49
4	60	60.39	58
5	70	65.31	62

From the table, as the temperature bath increases, the measrurements of both MSU and sensors increase. The accuracy of the sensors are calculated based on the plotted graph using the standard deviation equation. A graph is plotted based on the recorded data shown in Figure 33 below.

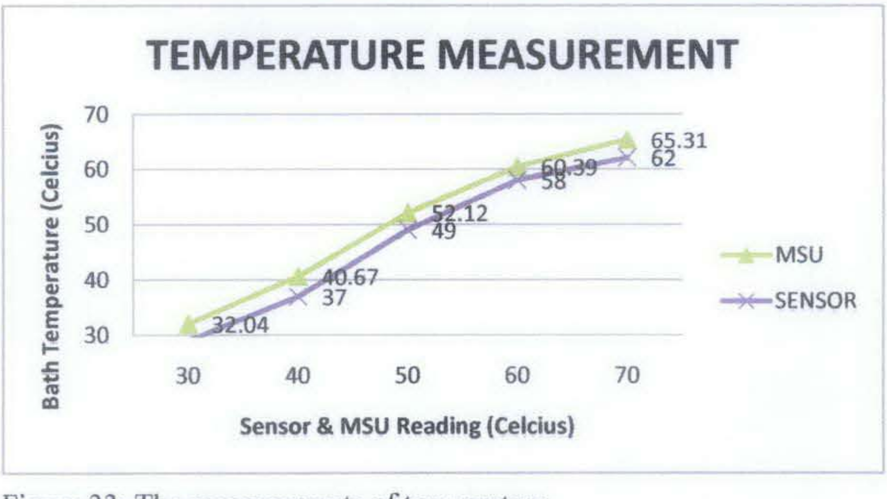


Figure 33: The measurements of temperature.

Using the standard deviation formula, the accuracy of the sensor can be determined as follows:

$$\sigma = \sqrt{\frac{\sum(x-\overline{x})^2}{N}}$$

σ = the standard deviation

x = each value in the population

$\bar{x}$ = the mean of the values

N = the number of values of population

The mean for sensor is calculated using:

$$\overline{x} = \frac{\Sigma x}{N}$$

$$= \frac{29 + 37+49+58+62}{5}$$

$$= 47$$

Using the mean to calculate:

$$\sum(x - \overline{x})^2 = (29-47)^2 + (37-47)^2 + (49-47)^2 + (58-47)^2 + (62\text{-}47)^2$$

$$= 774$$

Therefore, the standard deviation of sensor is around:

$$\sigma = \sqrt{774/5} = 12.44$$

The mean for MSU is calculated using:

$$\overline{x} = \frac{\Sigma x}{N}$$

$$= \frac{30.04 + 40.67 +52.12 +60.39+65.31}{5}$$

$$= 49.71$$

Using the mean to calculate:

$$\begin{aligned}\sum(x-\overline{x})^2 &= (30.04-49.71)^2 + (40.67-49.71)^2 + \\ &\quad (52.12-49.71)^2 + (60.39-49.71)^2 + \\ &\quad (65.31\text{-}49.71)^2 \\ &= 831.86\end{aligned}$$

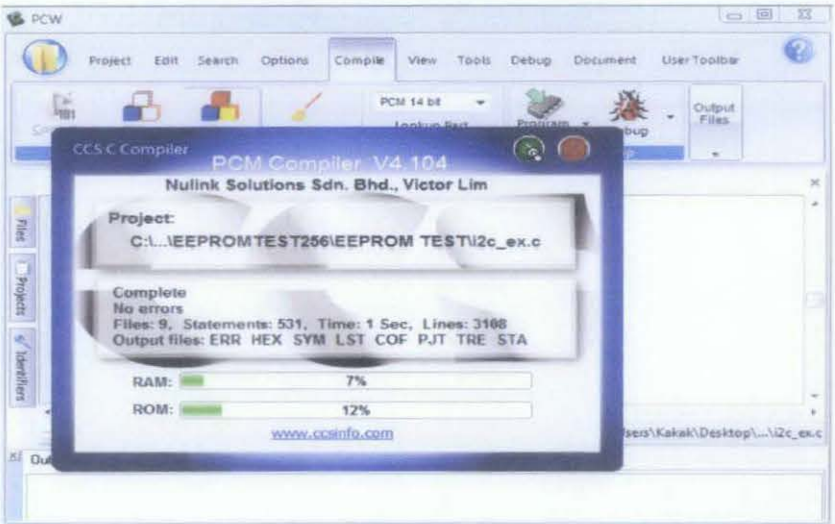
Therefore, the standard deviation of sensor is around:

$$\sigma = \sqrt{831.86/5} = 12.9$$

From the calculation, it proves that the sensor gives an accurate temperature reading because the value of standard deviation of sensor is near with the value of the MSU standard deviation. Therefore, the sensor is applicable in industry applications like other temperature measurement devices. But, the sensor is easily integrated to a circuit compared to other devices.

4.3 EEPROM Output Test on Terminal

At the end of the EEPROM program test, the program manages to come out with a final output from the compilation of EEPROM 24256 C programming. The compilation is completed successfully with no errors. This is proved by the CCS compilation window in Figure 34 below.



Nulink Solutions Sdn. Bhd., Victor Lim

Project:
C:\...\EEPROMTEST256\EEPROM TEST\i2c_ex.c

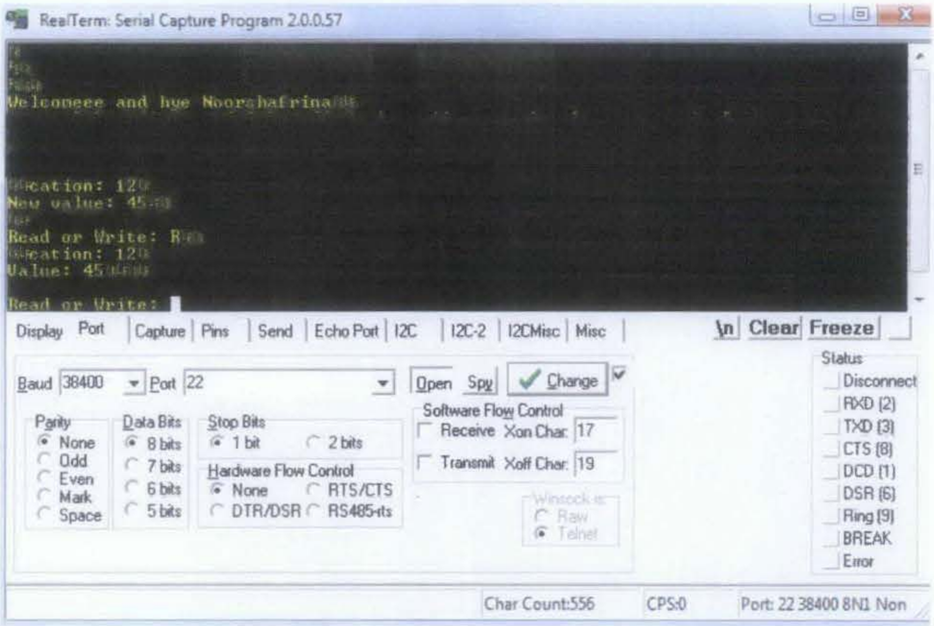
```
Complete
No errors
Files: 9,  Statements: 531,  Time: 1 Sec,  Lines: 3168
Output files: ERR  HEX  SYM  LST  COF  PJT  TRE  STA
```

RAM: 7%
ROM: 12%

www.ccsinfo.com

Figure 34: Compilation of EEPROM program.

The result illustrated in Figure 35 below shows that the value 45 is written by the user and saved in location 12. When the user wishes to read back the value from location 12, the result will show the value 45, which is the value that had been stored in location 12. This proves that the EEPROM chip program functions well.



RealTerm: Serial Capture Program 2.0.0.57

```
Welcomeee and hye Noorshafrina


cation: 12
New value: 45

Read or Write: R
cation: 12
Value: 45

Read or Write:
```

Baud 38400 Port 22

Parity: ● None ○ Odd ○ Even ○ Mark ○ Space

Data Bits: ● 8 bits ○ 7 bits ○ 6 bits ○ 5 bits

Stop Bits: ● 1 bit ○ 2 bits

Hardware Flow Control: ● None ○ RTS/CTS ○ DTR/DSR ○ RS485-rts

Software Flow Control: □ Receive Xon Char: 17 □ Transmit Xoff Char: 19

Winsock is: ○ Raw ● Telnet

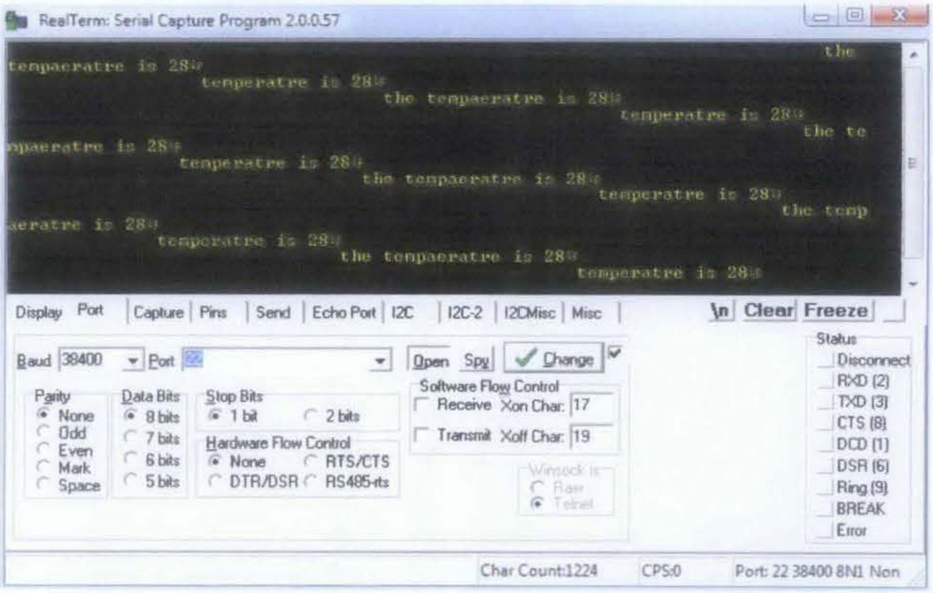
Status: □ Disconnect □ RXD (2) □ TXD (3) □ CTS (8) □ DCD (1) □ DSR (6) □ Ring (9) □ BREAK □ Error

Char Count:556 CPS:0 Port: 22 38400 8N1 Non

Figure 35: Result for external EEPROM programming with microcontroller.

4.4 Serial Communication Test on Terminal

Figure 36 below shows the result from RS232 communication in Realterm terminal.



RealTerm: Serial Capture Program 2.0.0.57

```
                                                                       the
tempaeratre is 28
              temperatre is 28
                             the tempaeratre is 28
                                                 temperatre is 28
                                                                  the te
mpaeratre is 28
            temperatre is 28
                           the tempaeratre is 28
                                               temperatre is 28
                                                                the temp
aeratre is 28
          temperatre is 28
                         the tempaeratre is 28
                                             temperatre is 28
```

Baud 38400 Port 22

Parity: ● None ○ Odd ○ Even ○ Mark ○ Space

Data Bits: ● 8 bits ○ 7 bits ○ 6 bits ○ 5 bits

Stop Bits: ● 1 bit ○ 2 bits

Hardware Flow Control: ● None ○ RTS/CTS ○ DTR/DSR ○ RS485-rts

Software Flow Control: □ Receive Xon Char: 17 □ Transmit Xoff Char: 19

Winsock is: ○ Raw ● Telnet

Status: □ Disconnect □ RXD (2) □ TXD (3) □ CTS (8) □ DCD (1) □ DSR (6) □ Ring (9) □ BREAK □ Error

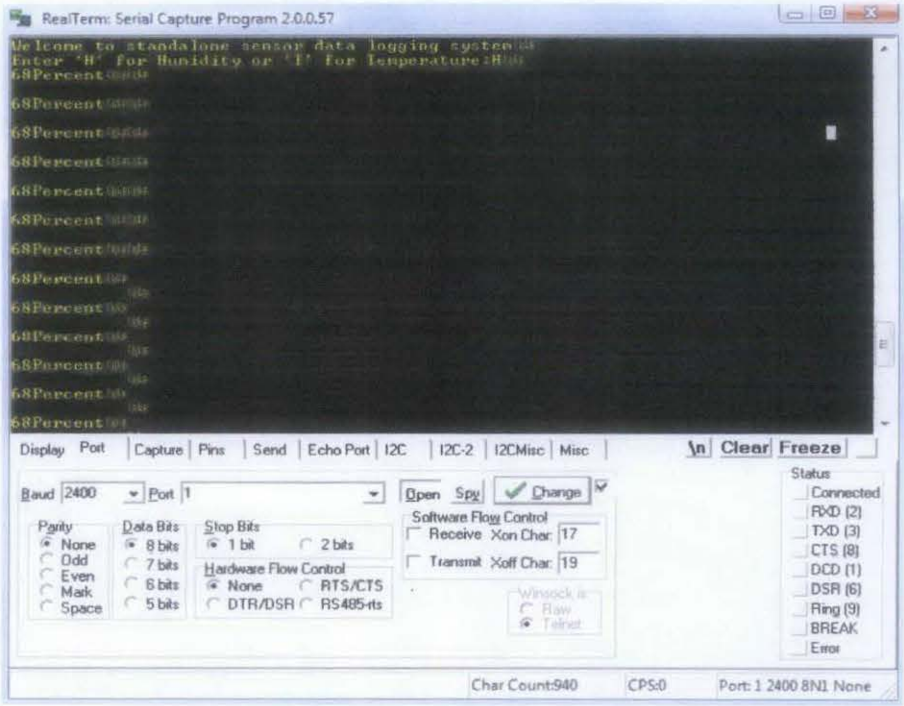
Char Count:1224 CPS:0 Port: 22 38400 8N1 Non

Figure 36: Result from RS232 communication in Realterm terminal.

This test is conducted in order to check the functionality of the serial communication before integrating the wireless communication device. From the program tested, the output display shows the temperature reading of the sensor module. From the result illustrated in Figure 36 above, it can be concluded that the serial port communicates with the PC and the value read by the sensors can be displayed in the RealTerm software.

4.4.1 Data Logger display through wired communication

The result from hard wired connection with PC using the serial port is the displayed data stored by EEPROM. From the program, the location for storing both data consists of 50 locations. The first 25 locations are the data of humidity, while the remaining locations are the data of temperature. The value from the locations are all the same because the time delay for each value to be read is 25micro seconds; therefore, it executes fast and the changes could not be easily detected during this duration of time. A faster delay time is purposely used in the experiment in order to avoid a slow response. In real applications, the exact delay that should be used can be changed in the program. Figure 37 and 38 below shows the reading of the stored data:



RealTerm: Serial Capture Program 2.0.0.57

```
Welcome to standalone sensor data logging system
Enter 'H' for Humidity or 'T' for Temperature:H
68Percent

68Percent

68Percent

68Percent

68Percent

68Percent

68Percent

68Percent

68Percent

68Percent

68Percent

68Percent

68Percent
```

Baud 2400 Port 1

Parity: ● None ○ Odd ○ Even ○ Mark ○ Space

Data Bits: ● 8 bits ○ 7 bits ○ 6 bits ○ 5 bits

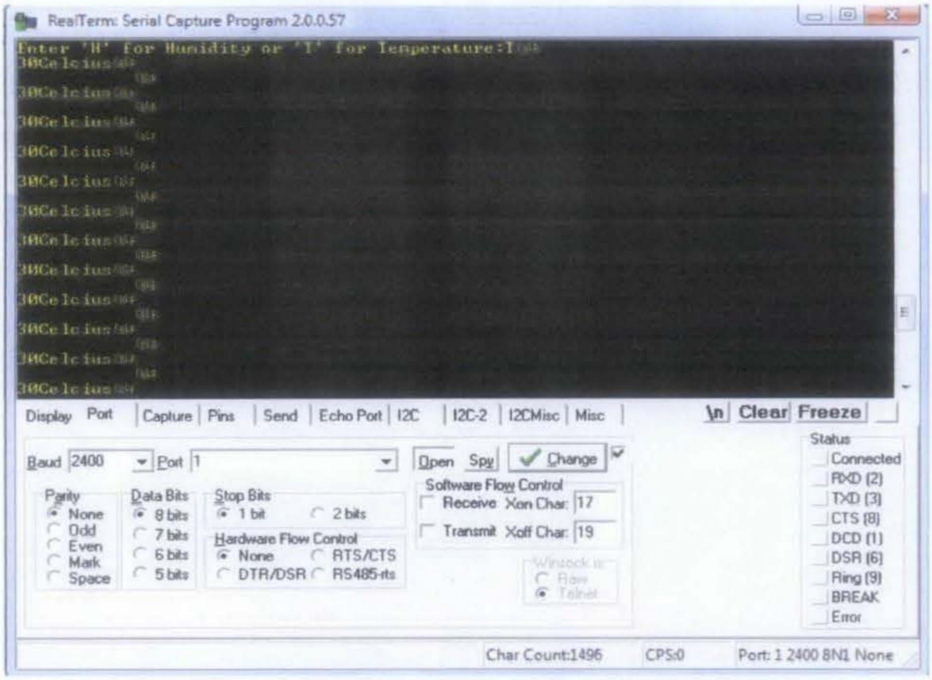
Stop Bits: ● 1 bit ○ 2 bits

Hardware Flow Control: ● None ○ RTS/CTS ○ DTR/DSR ○ RS485-rts

Software Flow Control: □ Receive Xon Char: 17 □ Transmit Xoff Char: 19

Char Count:940 CPS:0 Port: 1 2400 8N1 None

Figure 37: Display of 25 data of humidity from EEPROM.



RealTerm: Serial Capture Program 2.0.0.57

```
Enter 'H' for Humidity or 'T' for Temperature:T
30Celcius
30Celcius
30Celcius
30Celcius
30Celcius
30Celcius
30Celcius
30Celcius
30Celcius
30Celcius
30Celcius
30Celcius
```

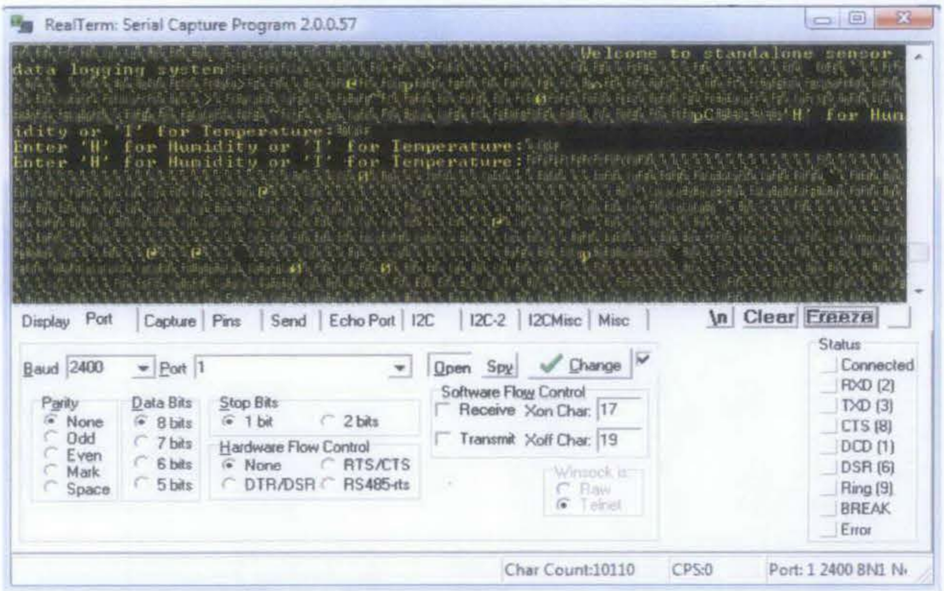
Baud 2400 Port 1 Parity ● None Data Bits ● 8 bits Stop Bits ● 1 bit Hardware Flow Control ● None Winsock is: ● Telnet Xon Char: 17 Xoff Char: 19

Char Count:1496 CPS:0 Port: 1 2400 8N1 None

Figure 38: The display of 25 data of temperature from EEPROM.

4.4.2 Data logger display through wireless communication

Figure 39 below shows the data logger from a wireless communication using the radio frequency module. There are a lot of contaminations (garbage) due to noise and interference displayed in Realterm because of the instability of the RF wireless link. The terminal is basically a dummy terminal where it receives all the incoming data including the garbage.



RealTerm: Serial Capture Program 2.0.0.57

```
Welcome to standalone sensor
data logging system
'H' for Hum
idity or 'T' for Temperature:
Enter 'H' for Humidity or 'T' for Temperature:
Enter 'H' for Humidity or 'T' for Temperature:
```

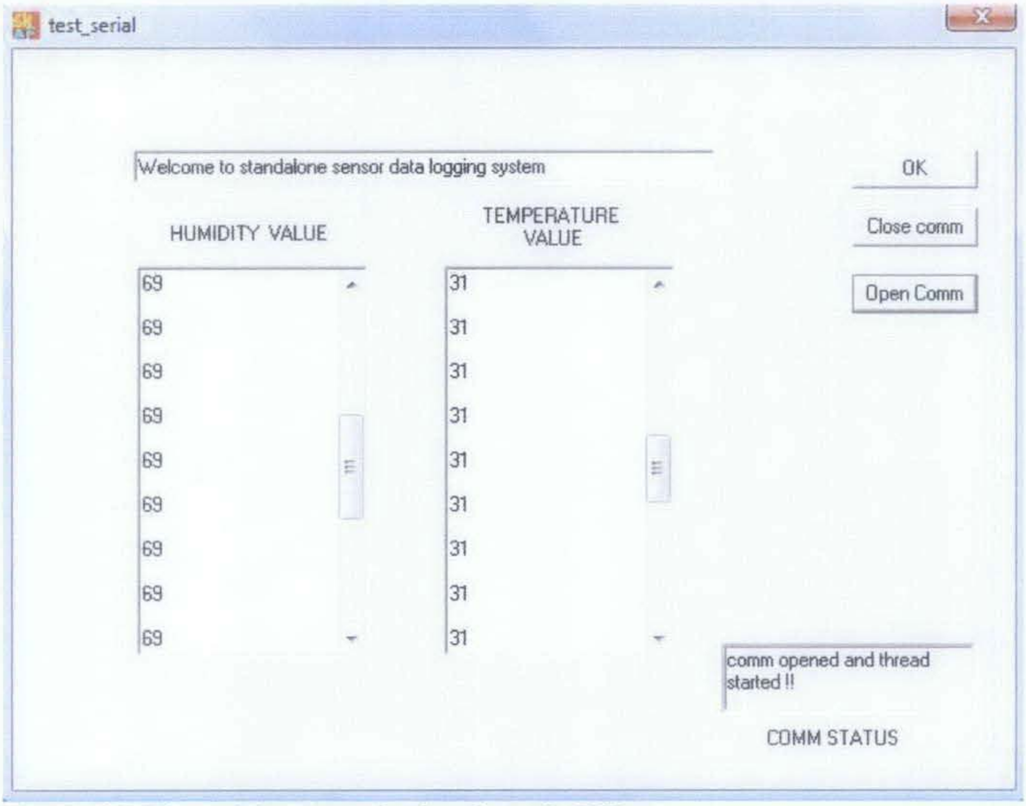
Baud 2400 Port 1 Parity ● None Data Bits ● 8 bits Stop Bits ● 1 bit Hardware Flow Control ● None Winsock is: ● Telnet Xon Char: 17 Xoff Char: 19

Char Count:10110 CPS:0 Port: 1 2400 8N1 N

Figure 39: The display from wireless communication.

4.5 Graphical User Interface Display

As a result from the Figure 40, it shows that the data is displayed vertically. The amount of data sent are 50 data, 25 data for humidity values and another 25 values for temperature. We purposely programmed the EEPROM to read and record only 50 locations from the addresses in order to reduce the time of system development and prepare for other unfinished process. By pressing the "Open Comm" button, the "COMM STATUS" shows that the port has been opened and thread is started.



Welcome to standalone sensor data logging system

```
HUMIDITY VALUE    TEMPERATURE VALUE
69                31
69                31
69                31
69                31
69                31
69                31
69                31
69                31
69                31
```

comm opened and thread started !!

COMM STATUS

Figure 40: The serial communication through GUI.

CHAPTER 5
CONCLUSION AND RECOMMENDATIONS

As a conclusion, at the end of project development process, the project basically meets the objectives and works appropriately as expected. However, there are some spaces or room of improvements for future enhancement. Conclusions and a few recommendations are explained in the section below.

5.1 Conclusion

In a nutshell, the Wireless Sensor Data Logger System proceeded as scheduled and has met its objectives. Through research, analytical and critical thinking, time management, planning and laboratory work; the objectives are met. From the objectives, the wireless communication between the integrated sensor MCU circuits with PC is communicating but the radio frequency module is not very stable and sends garbage to terminal. The printed circuit board has been fabricated and the GUI interface design has been working. The project gives a good practice to us in the embedded system knowledge and technical work which is a useful hands-on work in the future.

The project development was very tough and challenging due to time constraint and components availability. The project requires a frequent testing and troubleshooting processes which are very time consuming. The Wireless Sensor Data Logger system offers a reliable dynamic data logging system. Moreover, the project provides a platform for further advancement with better reliability and various applications in diverse fields.

5.2 Recommendations

In this section, recommendations will be made towards the Wireless Sensor Data Logger System. The recommendations are based on enhancement and improvement of the system besides reducing the mistakes that exists in the designed system. The recommendations for enhancement of the system will be beneficial especially when it deals with real applications. Thus, the recommendations are as below:

- **Implementing a recovery and security system that can accommodate the industries needs**

  A security system should be implemented together with the data logger system in order to respond to fault occurrences and be able to offer warning alarms and immediate corrective actions to the devices.

- **Improving the wireless communication by using a more stable radio frequency module or other better modules**

  The purpose of improving the wireless communication is to reduce or eliminate the garbage that is transferred to PC. A stabilized wireless link is required to make sure the correct data is transferred and sent to PC.

- **Implementing the time and date in the data logger design**

  For more advanced and systematic applications, the time and date should be implemented in order to record the data for the time and date required. Initially, the design of data logger system is designed without the date and time because of time limitations.

REFERENCES

[1] Enzine Articles, "*The importance of a data logger*", http://ezinearticles.com/?The-Importance-Of-A-DataLogger&id=4828095

[2] XZcution, "*For those who are curious about the data logger*", http://www.xzcution.com/for-those-who-are-curious-about-the-data-logger/

[3] Data Acquisition Networks, "*The Importance of data collection*", http://www.danmonitoring.com/data-loggers/the-importance-of-data collection.html

[4] USDA, "*The temperatures affect food*", http://www.fsis.usda.gov/factsheets/how_temperatures_affect_food/index.asp

[5] Post Harvest, "*Post Harvest Cooling/Storage for Cut flowers*",Extension Agriculture Engineer,University of Maryland http://www.bre.umd.edu/portacooler1.htm

[6] One omega, "*Introduction to data logging systems*", http://www.omega.com/techref/pdf/loggerintro.pdf

[7] Dogan Ibrahim*Advanced PIC Microcontroller Projects in C*,The Newnes,Elsevier,2008

[8] Circuits Today, "*Introduction to PIC16877*", http://www.circuitstoday.com/pic-16f877-architecture-and-memory-organization

[9] Modtronix, "PIC16F877A", http://www.modtronix.com/product_info.php?products_id=29

[10] Myke Predko.Programming and Customizaing the PIC Microcontroller,TAB Electronics,Mc Graw Hill,3rd Edition,2008.

[11] Tim Wilmshurst.Designing embeded systems with PIC Microcontroller,The Newnes,Elsevier,1st Edition,2007.

[12] Rickeys World, "*LCD interfacing with microcontroller's tutorial*", http://www.8051projects.net/lcd-interfacing/introduction.php

[13]Wise Geek, "*What is humidity sensor*", http://www.wisegeek.com/what-is-a-humidity-sensor.htm

[14]Robot Electronics, "*Using the I2C Bus*", http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html

APPENDICES

APPENDIX A

GANT CHART

Gantt Chart for the First Semester Final Year Project

No.	Detail/ Week	1	2	3	4	5	6	7		8	9	10	11	12	13	14
1	Selection of Project Topic	▨	▨						Mid-semester break							
2	Preliminary Research Work		▨	▨	▨											
3	Submission of Preliminary Report				●											
4	Seminar 1 (optional)					▨	▨	▨								
5	Project Work					▨	▨	▨								
										●						
6	Submission of Progress Report															
7	Seminar 2 (compulsory)										▨	▨	▨	▨		
8	Project work continues									▨	▨	▨	▨	▨		
															●	
9	Submission of Interim Report Final Draft															●
10	Oral Presentation															
			●													

● Suggested milestone

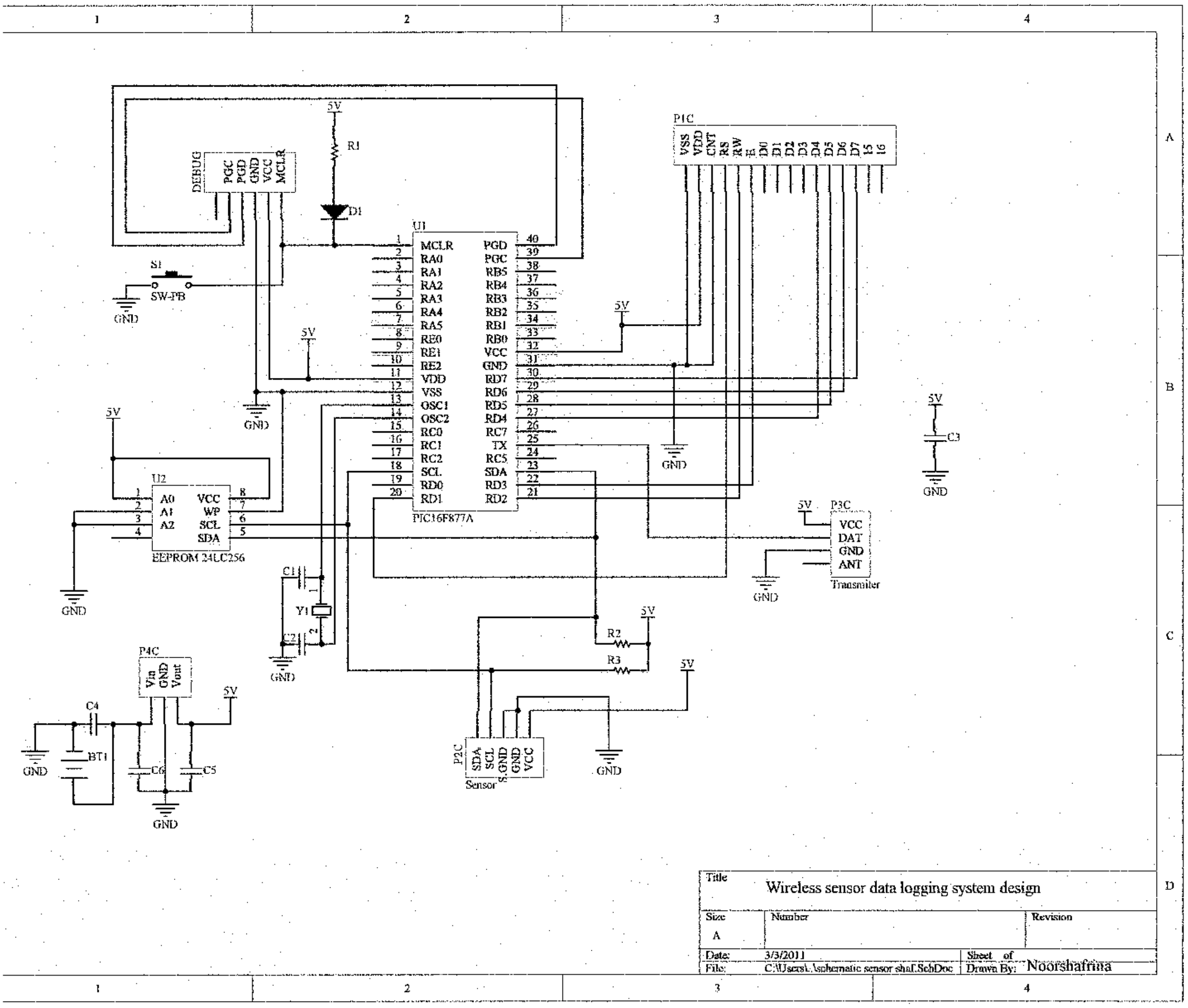
▨ Process

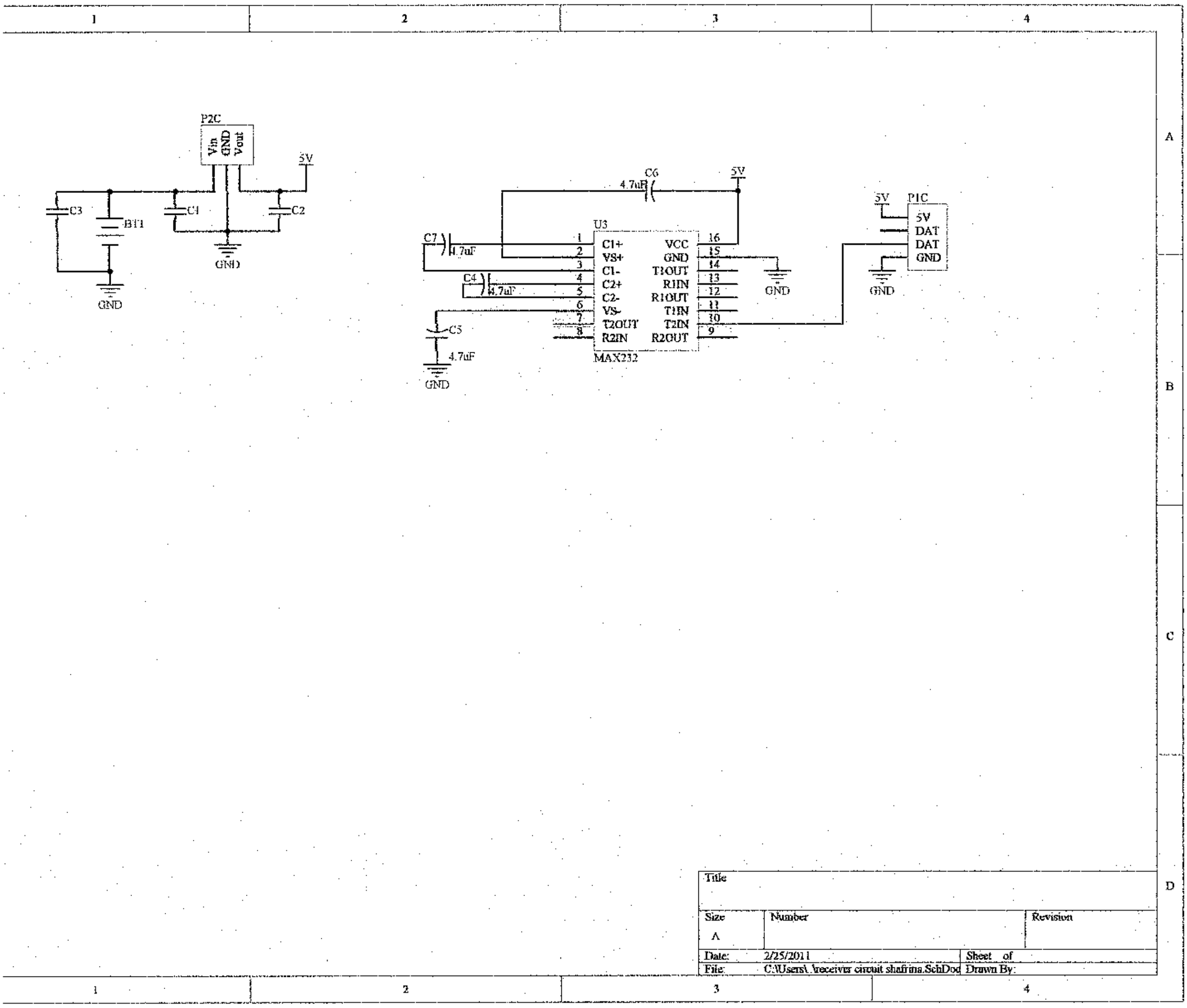
Gantt Chart for the Second Semester Final Year Project

No.	Detail/ Week	1	2	3	4	5	6	7		8	9	10	11	12	13	14
1	Project Work Continue	▨	▨	▨					Mid-Semester Break							
2	Submission of Progress Report 1				●											
3	Project Work Continue				▨	▨	▨	▨								
4	Submission of Progress Report 2									●						
5	Seminar (compulsory)										▨	▨	▨			
5	Project work continue									▨	▨	▨	▨			
6	Poster Exhibition											●				
7	Submission of Dissertation (soft													●		
8	Oral Presentation														●	
9	Submission of Project Dissertation															●

● Suggested milestone

▨ Process


1
2
3
4
A
B
C
D
DEBUG
PGC
PGD
GND
VCC
MCLR
5V
R1
D1
S1
SW-PB
GND
U1
1 MCLR
2 RA0
3 RA1
4 RA2
5 RA3
6 RA4
7 RA5
8 RE0
9 RE1
10 RE2
11 VDD
12 VSS
13 OSC1
14 OSC2
15 RC0
16 RC1
17 RC2
18 SCL
19 RD0
20 RD1
PGD 40
PGC 39
RB5 38
RB4 37
RB3 36
RB2 35
RB1 34
RB0 33
VCC 32
GND 31
RD7 30
RD6 29
RD5 28
RD4 27
RC7 26
TX 25
RC5 24
SDA 23
RD3 22
RD2 21
PIC16F877A
P1C
VSS
VDD
CNT
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
15
16
5V
GND
5V
C3
GND
5V
U2
1 A0
2 A1
3 A2
4
VCC 8
WP 7
SCL 6
SDA 5
EEPROM 24LC256
GND
C1
Y1
C2
GND
5V
P3C
VCC
DAT
GND
ANT
Transmiter
GND
5V
R2
R3
5V
P4C
Vin
GND
Vout
5V
C4
BT1
GND
C6
C5
GND
P2C
SDA
SCL
S.GND
GND
VCC
Sensor
GND
Title
Wireless sensor data logging system design
Size
A
Number
Revision
Date: 3/3/2011
Sheet of
File: C:\Users\..\schematic sensor shaf.SchDoc
Drawn By: Noorshafrina



Title			
Size A	Number		Revision
Date:	2/25/2011	Sheet of	
File:	C:\Users\ \receiver circuit shafrina.SchDoc	Drawn By:	

APPENDIX C
PROGRAM CODES

PROGRAM CODE 1: MAIN CODE

```
#include"C:\Users\Kakak\Desktop\LCDplusEEPROM\RHT_I2CSensor2\DigiPicco.h"
#include "C:\Users\Kakak\Desktop\LCD plus EEPROM\RHT_I2CSensor2\LCD.C"

#include <string.h>
#include <stdlib.h>

#include "I2C_RHTSensor.h"

#define EEPROM_SDA PIN_C4
#define EEPROM_SCL PIN_C3

#include "input.c"
#include "24256ab.c"


void Initialize()
{
   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_OFF);
   setup_psp(PSP_DISABLED);
   setup_spi(SPI_SS_DISABLED);
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_comparator(NC_NC_NC_NC);
   setup_vref(FALSE);
   lcd_init();
   init_ext_eeprom();

}


void main()
{
   byte data,value,cmd;
   EEPROM_ADDRESS address;
   Initialize();

   printf("abczWelcome to standalone sensor data logging system*");

   lcd_putc("\fRH & T Readings\n");
   lcd_putc("from DigiPicco..");
   delay_ms(200);

   Init_RHTSensor();

   for(address=0;address <50;)
    {
     Read_RHTSensor();

     value = humH ;
     WRITE_EXT_EEPROM( address, value );
```

```
 address++ ;
 value = humL ;
 WRITE_EXT_EEPROM( address, value );
 address++ ;

 delay_ms(25);
}


for(address=50;address <100;)
{
 Read_RHTSensor();

 value = TempH ;
 WRITE_EXT_EEPROM( address, value );
 address++ ;
 value = TempL ;
 WRITE_EXT_EEPROM( address, value );
 address++ ;

 delay_ms(25);
}


 for(address=0;address <50;address++)
     {

       humH = READ_EXT_EEPROM( address );
       address++;
       humL = READ_EXT_EEPROM( address );

        Humidity = make32(0,0,humH,humL)*100/0x7FFF;

        printf("abcy%X*\r\n",Humidity);
        delay_ms(200);
     }
for(address=50;address <100;address++)
     {

      TempH = READ_EXT_EEPROM( address );
      address++;
      TempL = READ_EXT_EEPROM( address );
      Temperature = make32(0,0,TempH,TempL)*165/0x7FFF-40;
      printf("abcw%X*\r\n",Temperature);
      delay_ms(200);

     }
     }
```

```
#include <16F877A.h>
#device adc=8
#FUSES NOWDT                    //No Watch Dog Timer
#FUSES HS                       //High speed Osc
#FUSES PUT                      //Power Up Timer
#FUSES NOPROTECT                //Code not protected from reading
#FUSES NODEBUG                  //No Debug mode for ICD
#FUSES NOBROWNOUT               //No brownout reset
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16)
#FUSES NOCPD                    //No EE protection
#FUSES NOWRT                    //Program memory not writes protected

#use delay(clock=12000000)
#use rs232(baud=2400,xmit=PIN_C6,rcv=PIN_C7)
```

PROGRAM CODE 2: HUMIDITY AND TEMPERATURE SENSOR SOURCE CODE

```
#define RHTSensor_SDA  PIN_C4
#define RHTSensor_SCL  PIN_C3


#use i2c(master, sda=RHTSensor_SDA, scl=RHTSensor_SCL)

void init_RHTSensor()
{
   output_high(RHTSensor_SDA);
   output_high(RHTSensor_SCL);
}

  BYTE HumL,HumH,TempL,TempH;
  int32 Humidity, Temperature;

BYTE Read_RHTSensor()
{  // Returns degrees F (0-255)

   i2c_start();
   i2c_write(0xF1);
   HumH=i2c_read();
   HumL=i2c_read();
   TempH=i2c_read();
   TempL=i2c_read(0);

   i2c_stop();
   printf(lcd_putc,"\fR.Humidity=%LU%%\n",Humidity);
   printf(lcd_putc,"Temperature=%Lu C\n",Temperature);
   return 0;
}
```

PROGRAM CODE 3:LCD CODING

```
////                              LCD.C
////
////                    Driver for common LCD modules
////
////
////
////  lcd_init()   Must be called before any other function.
////
////
////
////  lcd_putc(c)  Will display c on the next position of the LCD.
////
////                      The following have special meaning:
////
////                       \f  Clear display
////
////                       \n  Go to start of second line
////
////                       \b  Move back one position
////
////
////
////  lcd_gotoxy(x,y) Set write position on LCD (upper left is 1,1)
////
////
////
////  lcd_getc(x,y)   Returns character at position x,y on LCD
////
////
////
////  CONFIGURATION
////
////  The LCD can be configured in one of two ways: a.) port access or
////
////  b.) pin access.  Port access requires the entire 7 bit interface
////
////  connected to one GPIO port, and the data bits (D4:D7 of the LCD)
////
////  connected to sequential pins on the GPIO.  Pin access
////
////  has no requirements, all 7 bits of the control interface can
////
////  can be connected to any GPIO using several ports.
////
////
////
////  To use port access, #define LCD_DATA_PORT to the SFR location of
////
////  of the GPIO port that holds the interface, -AND- edit LCD_PIN_MAP
////
////  of this file to configure the pin order.  If you are using a
////
////  baseline PIC (PCB), then LCD_OUTPUT_MAP and LCD_INPUT_MAP also
must  ////
////  be defined.
////
////
////
////  Example of port access:
```

```
////
////     #define LCD_DATA_PORT getenv("SFR:PORTD")
////
////
////
////  To use pin access, the following pins must be defined:
////
////     LCD_ENABLE_PIN
////
////     LCD_RS_PIN
////
////     LCD_RW_PIN
////
////     LCD_DATA4
////
////     LCD_DATA5
////
////     LCD_DATA6
////
////     LCD_DATA7
////
////
////
////  Example of pin access:
////
////     #define LCD_ENABLE_PIN  PIN_E0
////
////     #define LCD_RS_PIN      PIN_E1
////
////     #define LCD_RW_PIN      PIN_E2
////
////     #define LCD_DATA4       PIN_D4
////
////     #define LCD_DATA5       PIN_D5
////
////     #define LCD_DATA6       PIN_D6
////
////     #define LCD_DATA7       PIN_D7
////
////
////
////////////////////////////////////////////////////////////////////////
///////
////        (C) Copyright 1996,2009 Custom Computer Services
////
//// This source code may only be used by licensed users of the CCS C
////
//// compiler.  This source code may only be distributed to other
////
//// licensed users of the CCS C compiler.  No other use, reproduction
////
//// or distribution is permitted without written permission.
////
//// Derivative programs created using this software in object code
////
//// form are not restricted in any way.
////
////////////////////////////////////////////////////////////////////////
///
```

```
// define the pinout.

   #define LCD_ENABLE_PIN  PIN_E0
////
   #define LCD_RS_PIN      PIN_E1                                   ///
   #define LCD_RW_PIN      PIN_E2
////
   #define LCD_DATA4       PIN_D4
////
   #define LCD_DATA5       PIN_D5
////
   #define LCD_DATA6       PIN_D6
////
   #define LCD_DATA7       PIN_D7



// this is to improve compatability with previous LCD drivers that
accepted
// a define labeled 'use_portb_lcd' that configured the LCD onto port B.
#if ((defined(use_portb_lcd)) && (use_portb_lcd==TRUE))
 #define LCD_DATA_PORT getenv("SFR:PORTB")
#endif

#if defined(__PCB__)
   // these definitions only need to be modified for baseline PICs.
   // all other PICs use LCD_PIN_MAP or individual LCD_xxx pin
definitions.
/*                                    EN, RS,   RW,   UNUSED,  DATA  */
 const LCD_PIN_MAP LCD_OUTPUT_MAP =  {0,  0,    0,    0,       0};
 const LCD_PIN_MAP LCD_INPUT_MAP =   {0,  0,    0,    0,       0xF};
#endif



////////////////////// END CONFIGURATION
///////////////////////////////////

#ifndef LCD_ENABLE_PIN
   #define lcd_output_enable(x) lcdlat.enable=x
   #define lcd_enable_tris()   lcdtris.enable=0
#else
   #define lcd_output_enable(x) output_bit(LCD_ENABLE_PIN, x)
   #define lcd_enable_tris()  output_drive(LCD_ENABLE_PIN)
#endif

#ifndef LCD_RS_PIN
   #define lcd_output_rs(x) lcdlat.rs=x
   #define lcd_rs_tris()   lcdtris.rs=0
#else
   #define lcd_output_rs(x) output_bit(LCD_RS_PIN, x)
   #define lcd_rs_tris()  output_drive(LCD_RS_PIN)
#endif

#ifndef LCD_RW_PIN
   #define lcd_output_rw(x) lcdlat.rw=x
   #define lcd_rw_tris()   lcdtris.rw=0
#else
   #define lcd_output_rw(x) output_bit(LCD_RW_PIN, x)
   #define lcd_rw_tris()  output_drive(LCD_RW_PIN)
#endif

// original version of this library incorrectly labeled LCD_DATA0 as
LCD_DATA4,
// LCD_DATA1 as LCD_DATA5, and so on.  this block of code makes the
driver
// compatible with any code written for the original library
#if (defined(LCD_DATA0) && defined(LCD_DATA1) && defined(LCD_DATA2) &&
defined(LCD_DATA3) && !defined(LCD_DATA4) && !defined(LCD_DATA5) && !
defined(LCD_DATA6) && !defined(LCD_DATA7))
   #define  LCD_DATA4    LCD_DATA0
   #define  LCD_DATA5    LCD_DATA1
   #define  LCD_DATA6    LCD_DATA2
   #define  LCD_DATA7    LCD_DATA3
#endif

#ifndef LCD_DATA4
#ifndef LCD_DATA_PORT
   #if defined(__PCB__)
      #define LCD_DATA_PORT      0x06     //portb
      #define set_tris_lcd(x)   set_tris_b(x)
   #else
     #if defined(PIN_D0)
      #define LCD_DATA_PORT      getenv("SFR:PORTD")     //portd
     #else
      #define LCD_DATA_PORT      getenv("SFR:PORTB")     //portb
     #endif
   #endif
#endif

#if defined(__PCB__)
   LCD_PIN_MAP lcd, lcdlat;
   #byte lcd = LCD_DATA_PORT
   #byte lcdlat = LCD_DATA_PORT
#elif defined(__PCM__)
   LCD_PIN_MAP lcd, lcdlat, lcdtris;
   #byte lcd = LCD_DATA_PORT
   #byte lcdlat = LCD_DATA_PORT
   #byte lcdtris = LCD_DATA_PORT+0x80
#elif defined(__PCH__)
   LCD_PIN_MAP lcd, lcdlat, lcdtris;
   #byte lcd = LCD_DATA_PORT
   #byte lcdlat = LCD_DATA_PORT+9
   #byte lcdtris = LCD_DATA_PORT+0x12
#elif defined(__PCD__)
   LCD_PIN_MAP lcd, lcdlat, lcdtris;
   #word lcd = LCD_DATA_PORT
   #word lcdlat = LCD_DATA_PORT+2
   #word lcdtris = LCD_DATA_PORT-0x02
#endif
#endif   //LCD_DATA4 not defined

#ifndef LCD_TYPE
   #define LCD_TYPE 2           // 0=5x7, 1=5x10, 2=2 lines
#endif

#ifndef LCD_LINE_TWO
   #define LCD_LINE_TWO 0x40    // LCD RAM address for the second line
#endif
```

```
BYTE const LCD_INIT_STRING[4] = {0x20 | (LCD_TYPE << 2), 0xc, 1, 6};
                             // These bytes need to be sent to the LCD
                             // to start it up.

BYTE lcd_read_nibble(void);

BYTE lcd_read_byte(void)
{
   BYTE low,high;

 #if defined(__PCB__)
   set_tris_lcd(LCD_INPUT_MAP);
 #else
  #if (defined(LCD_DATA4) && defined(LCD_DATA5) && defined(LCD_DATA6) &&
defined(LCD_DATA7))
   output_float(LCD_DATA4);
   output_float(LCD_DATA5);
   output_float(LCD_DATA6);
   output_float(LCD_DATA7);
  #else
   lcdtris.data = 0xF;
  #endif
 #endif

   lcd_output_rw(1);
   delay_cycles(1);
   lcd_output_enable(1);
   delay_cycles(1);
   high = lcd_read_nibble();

   lcd_output_enable(0);
   delay_cycles(1);
   lcd_output_enable(1);
   delay_us(1);
   low = lcd_read_nibble();

   lcd_output_enable(0);

 #if defined(__PCB__)
   set_tris_lcd(LCD_INPUT_MAP);
 #else
  #if (defined(LCD_DATA4) && defined(LCD_DATA5) && defined(LCD_DATA6) &&
defined(LCD_DATA7))
   output_drive(LCD_DATA4);
   output_drive(LCD_DATA5);
   output_drive(LCD_DATA6);
   output_drive(LCD_DATA7);
  #else
   lcdtris.data = 0x0;
  #endif
 #endif

   return( (high<<4) | low);
}

BYTE lcd_read_nibble(void)
{
  #if (defined(LCD_DATA4) && defined(LCD_DATA5) && defined(LCD_DATA6) &&
defined(LCD_DATA7))
   BYTE n = 0x00;
   /* Read the data port */
   n |= input(LCD_DATA4);
   n |= input(LCD_DATA5) << 1;
   n |= input(LCD_DATA6) << 2;
   n |= input(LCD_DATA7) << 3;

   return(n);
  #else
   return(lcd.data);
  #endif
}

void lcd_send_nibble(BYTE n)
{
  #if (defined(LCD_DATA4) && defined(LCD_DATA5) && defined(LCD_DATA6) &&
defined(LCD_DATA7))
   /* Write to the data port */
   output_bit(LCD_DATA4, bit_test(n, 0));
   output_bit(LCD_DATA5, bit_test(n, 1));
   output_bit(LCD_DATA6, bit_test(n, 2));
   output_bit(LCD_DATA7, bit_test(n, 3));
  #else
   lcdlat.data = n;
  #endif

   delay_cycles(1);
   lcd_output_enable(1);
   delay_us(2);
   lcd_output_enable(0);
}

void lcd_send_byte(BYTE address, BYTE n)
{
   lcd_output_rs(0);
   while ( bit_test(lcd_read_byte(),7) ) ;
   lcd_output_rs(address);
   delay_cycles(1);
   lcd_output_rw(0);
   delay_cycles(1);
   lcd_output_enable(0);
   lcd_send_nibble(n >> 4);
   lcd_send_nibble(n & 0xf);
}

void lcd_init(void)
{
   BYTE i;

 #if defined(__PCB__)
   set_tris_lcd(LCD_OUTPUT_MAP);
 #else
  #if (defined(LCD_DATA4) && defined(LCD_DATA5) && defined(LCD_DATA6) &&
defined(LCD_DATA7))
   output_drive(LCD_DATA4);
   output_drive(LCD_DATA5);
   output_drive(LCD_DATA6);
   output_drive(LCD_DATA7);
  #else
   lcdtris.data = 0x0;
  #endif
```

```
   lcd_enable_tris();
   lcd_rs_tris();
   lcd_rw_tris();
 #endif

   lcd_output_rs(0);
   lcd_output_rw(0);
   lcd_output_enable(0);

   delay_ms(15);
   for(i=1;i<=3;++i)
   {
       lcd_send_nibble(3);
       delay_ms(5);
   }

   lcd_send_nibble(2);
   for(i=0;i<=3;++i)
      lcd_send_byte(0,LCD_INIT_STRING[i]);
}

void lcd_gotoxy(BYTE x, BYTE y)
{
   BYTE address;

   if(y!=1)
      address=LCD_LINE_TWO;
   else
      address=0;

   address+=x-1;
   lcd_send_byte(0,0x80|address);
}

void lcd_putc(char c)
{
   switch (c)
   {
      case '\f'   :  lcd_send_byte(0,1);
                     delay_ms(2);
                     break;

      case '\n'   : lcd_gotoxy(1,2);        break;

      case '\b'   : lcd_send_byte(0,0x10);  break;

      default     : lcd_send_byte(1,c);     break;
   }
}

char lcd_getc(BYTE x, BYTE y)
{
   char value;

   lcd_gotoxy(x,y);
   while ( bit_test(lcd_read_byte(),7) ); // wait until busy flag is low
   lcd_output_rs(1);
   value = lcd_read_byte();
   lcd_output_rs(0);
   return(value);
}
```

PROGRAM CODE 4: EXTERNAL EEPROM OUTPUT PROGRAMMING TEST

```
////////////////////////////////////////////////////////////////////////
// Author : Noorshafrina
// Organization: UTP
// Date : Thursday 21st October 2010
// Purpose: To READ and WRITE to serial EEPROM 24LC256 using I2C
interface
////////////////////////////////////////////////////////////////////////
/* By default the compiler treats SHORT as one bit, INT as 8 bits and
LONG as
  16 bits.  The traditional C convention is to have INT defined as the
most
  efficient size for the target processor.  This is why it is 8 bits on
the PIC.
  In order to help with code compatibility a new directive has been added
that
  will allow these types to be changed.  #TYPE can redefine these
keywords.
  For example:
            #TYPE  SHORT=8, INT=16, LONG=32
  Note that the commas are optional.  Since #TYPE may render some sizes
  inaccessible (like a one bit int in the above) four new keywords have
  been added to represent the four ints:  INT1, INT8, INT16 and INT32.
  Be warned CCS example programs and include files may not work right if
  you use #TYPE in your program.*/


//#char port_a = 0x05
//#char port_b = 0x06
//#char port_c = 0x07
#include <16F877A.h>
#device adc=8
#FUSES NOWDT                    //No Watch Dog Timer
#FUSES HS                       //High speed Osc (> 4mhz for PCM/PCH)
(>10mhz for PCD)
#FUSES PUT                      //Power Up Timer
#FUSES NOPROTECT                //Code not protected from reading
#FUSES NODEBUG                  //No Debug mode for ICD
#FUSES NOBROWNOUT               //No brownout reset
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or
B5(PIC18) used for I/O
#FUSES NOCPD                    //No EE protection
#FUSES NOWRT                    //Program memory not write protected

#use delay(clock=12000000)
#use rs232(baud=38400, xmit=PIN_C6,rcv=PIN_C7)
#include "C:\Users\Kakak\Desktop\RHT_I2CSensor1\LCD.C"

#include <string.h>
#include <stdlib.h>
#define EEPROM_SDA  PIN_C4
#define EEPROM_SCL  PIN_C3
#include "input.c"
```

```
#include "24256ab.c"

void Initialize()
{
   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_OFF);
   setup_psp(PSP_DISABLED);
   setup_spi(SPI_SS_DISABLED);
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_comparator(NC_NC_NC_NC);
   setup_vref(FALSE);

  // SET_TRIS_A(2); //configure port a1 as input  and a0 as output
}

void main() {

   BYTE value, cmd;
   EEPROM_ADDRESS address;
   Initialize();
   printf("\r\nWelcome and Hye Noorshafrina\r\n");

   do {
      do {
         printf("\r\nRead or Write: ");
         cmd=getc();
         cmd=toupper(cmd);
         putc(cmd);
      } while ( (cmd!='R') && (cmd!='W') );

      printf("\n\rLocation: ");

#if sizeof(EEPROM_ADDRESS)==1
      address = gethex();
#else
   #if EEPROM_SIZE>0xfff
         address = gethex();
   #else
      address = gethex1();
   #endif

   address = (address<<8)+gethex();
#endif
      if(cmd=='R')
         printf("\r\nValue: %X\r\n",READ_EXT_EEPROM( address ) );

      if(cmd=='W') {
         printf("\r\nNew value: ");
         value = gethex();
         printf("\n\r");
         WRITE_EXT_EEPROM( address, value );
      }} while (TRUE);}
```

PROGRAM CODE 5: LIBRARY FOR 24LC256 SERIAL EEPROM

```
/////////////////////////////////////////////////////////////////////////
//
////   Library for a 24LC256 serial EEPROM
////
////
////
////   init_ext_eeprom();    Call before the other functions are used
////
////
////
////   write_ext_eeprom(a, d);  Write the byte d to the address a
////
////
////
////   d = read_ext_eeprom(a);   Read the byte d from the address a
////
////
////
////   The main program may define eeprom_sda
////
////   and eeprom_scl to override the defaults below.
////
////
////
/////////////////////////////////////////////////////////////////////////
//
////        (C) Copyright 1996,2003 Custom Computer Services
////
//// This source code may only be used by licensed users of the CCS C
////
//// compiler.  This source code may only be distributed to other
////
//// licensed users of the CCS C compiler.  No other use, reproduction
////
//// or distribution is permitted without written permission.
////
//// Derivative programs created using this software in object code
////
//// form are not restricted in any way.
////
/////////////////////////////////////////////////////////////////////////
//


#ifndef EEPROM_SDA

#define EEPROM_SDA  PIN_C4
#define EEPROM_SCL  PIN_C3

#endif
```

```
#use i2c(master, sda=EEPROM_SDA, scl=EEPROM_SCL)

#define EEPROM_ADDRESS long int
#define EEPROM_SIZE   32768

void init_ext_eeprom()
{
   output_float(EEPROM_SCL);
   output_float(EEPROM_SDA);

}


void write_ext_eeprom(long int address, BYTE data)
{
   short int status;
   i2c_start();
   i2c_write(0xa2);//a0=1
   i2c_write(address>>8);
   i2c_write(address);
   i2c_write(data);
   i2c_stop();
   i2c_start();
   status=i2c_write(0xa2);
   while(status==1)
   {
   i2c_start();
   status=i2c_write(0xa2);
   }
   i2c_stop();
}


BYTE read_ext_eeprom(long int address) {
   BYTE data;
   i2c_start();
   i2c_write(0xa2);//a0=1
   i2c_write(address>>8);
   i2c_write(address);
   i2c_start();
   i2c_write(0xa3);
   data=i2c_read(0);
   i2c_stop();
   return(data);
}
```

PROGRAM CODE 6: CODE OF INPUT RECEIVED BY USER

```
///////////////////////////////////////////////////////////////////////////
//
////        (C) Copyright 1996,2003 Custom Computer Services
////
//// This source code may only be used by licensed users of the CCS C
////
//// compiler.  This source code may only be distributed to other
////
//// licensed users of the CCS C compiler.  No other use, reproduction
////
//// or distribution is permitted without written permission.
////
//// Derivative programs created using this software in object code
////
//// form are not restricted in any way.
////
///////////////////////////////////////////////////////////////////////////
//


#include <ctype.h>

BYTE gethex1() {
   char digit;

   digit = getc();

   putc(digit);

   if(digit<='9')
     return(digit-'0');
   else
     return((toupper(digit)-'A')+10);
}

BYTE gethex() {
   unsigned int8 lo,hi;

   hi = gethex1();
   lo = gethex1();
   if(lo==0xdd)
     return(hi);
   else
     return( hi*16+lo );
}

void get_string(char* s, unsigned int8 max) {
   unsigned int8 len;
   char c;

   --max;
   len=0;
   do {
     c=getc();
     if(c==8) {  // Backspace
        if(len>0) {
          len--;
          putc(c);
          putc(' ');
          putc(c);
        }
     } else if ((c>=' ')&&(c<='~'))
       if(len<=max) {
         s[len++]=c;
         putc(c);
       }
   } while(c!=13);
   s[len]=0;
}

// stdlib.h is required for the ato_ conversions
// in the following functions
#ifdef _STDLIB
#if !defined(__PCD__)
signed int8 get_int() {
  char s[5];
  signed int8 i;

  get_string(s, 5);

  i=atoi(s);
  return(i);
}
#endif

#if defined(__PCD__)
signed int16 get_int() {
  char s[5];
  signed int16 i;

  get_string(s, 7);

  i=atoi(s);
  return(i);
}
#endif

#if !defined(__PCD__)
signed int16 get_long() {
  char s[7];
  signed int16 l;

  get_string(s, 7);
  l=atol(s);
  return(l);
```

```
}
#endif

#if defined(__PCD__)
signed int32 get_long() {
  char s[7];
  signed int32 l;

  get_string(s, 10);
  l=atoi32(s);
  return(l);
}
#endif

float get_float() {
  char s[20];
  float f;

  get_string(s, 20);
  f = atof(s);
  return(f);
}

#endif
```

PROGRAM CODE 7: GRAPHICAL USER INTERFACE SOURCE CODE

```
st_serialDlg.cpp : implementation file

te: Jan27th 2011
st Serial Communication

ude "stdafx.h"
ude "test_serial.h"
ude "test_serialDlg.h"

ne WM_POSTMESSAGE 1029
f _DEBUG
ne new DEBUG_NEW
f THIS_FILE
c char THIS_FILE[] = __FILE__;
f

hwnd ;
bTerminateThread_flag ;
ONFIG cc;
ortDCB;
IMEOUTS to;
ROP cp;
E hComm, hThread;
R lpszCommName = "COM1";
ouff[200]="0";

gflag=0 ;
count =0 ;
*ptr ;


hread *m_receive_thread ;
receive_thread (LPVOID pThreadParam) ;
Read_data(void);
send_char(BYTE inbuff);
Ascii2hex( char temp, char  * hex_data);

////////////////////////////////////////////////////////////////////////////
boutDlg dialog used for App About

 CAboutDlg : public CDialog

c:
AboutDlg();

alog Data
/{{AFX_DATA(CAboutDlg)
num { IDD = IDD_ABOUTBOX };
/}}AFX_DATA

/ ClassWizard generated virtual function overrides
/{{AFX_VIRTUAL(CAboutDlg)
rotected:
irtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
/}}AFX_VIRTUAL

plementation
cted:
/{{AFX_MSG(CAboutDlg)
/}}AFX_MSG
ECLARE_MESSAGE_MAP()
```

```
tDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)

/{{AFX_DATA_INIT(CAboutDlg)
/}}AFX_DATA_INIT


CAboutDlg::DoDataExchange(CDataExchange* pDX)

Dialog::DoDataExchange(pDX);
/{{AFX_DATA_MAP(CAboutDlg)
/}}AFX_DATA_MAP


_MESSAGE_MAP(CAboutDlg, CDialog)
/{{AFX_MSG_MAP(CAboutDlg)
   // No message handlers
/}}AFX_MSG_MAP
ESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////////
est_serialDlg dialog

_serialDlg::CTest_serialDlg(CWnd* pParent /*=NULL*/)
 CDialog(CTest_serialDlg::IDD, pParent)

/{{AFX_DATA_INIT(CTest_serialDlg)
_data_rx = _T("");
_rx_message = _T("");
_data_rx2 = _T("");
/}}AFX_DATA_INIT
/ Note that LoadIcon does not require a subsequent DestroyIcon in Win32
_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);


CTest_serialDlg::DoDataExchange(CDataExchange* pDX)

Dialog::DoDataExchange(pDX);
/{{AFX_DATA_MAP(CTest_serialDlg)
DX_Text(pDX, IDC_EDIT_DATA_RX, m_data_rx);
DV_MaxChars(pDX, m_data_rx, 20);
DX_Text(pDX, IDC_EDIT_RX_MESSAGE, m_rx_message);
DX_Text(pDX, IDC_EDIT_DATA_RX2, m_data_rx2);
/}}AFX_DATA_MAP


_MESSAGE_MAP(CTest_serialDlg, CDialog)
/{{AFX_MSG_MAP(CTest_serialDlg)
N_WM_SYSCOMMAND()
N_WM_PAINT()
N_WM_QUERYDRAGICON()
N_BN_CLICKED(IDC_BUTTON_OPEN_COM, OnButtonOpenCom)
N_BN_CLICKED(IDC_BUTTON_CLOSE_COM, OnButtonCloseCom)
N_BN_CLICKED(IDC_BUTTON_TEST, OnButtonTest)
N_MESSAGE(WM_POSTMESSAGE,OnPostMessage)
N_BN_CLICKED(IDC_BUTTON_HUMIDITY, OnButtonHumidity)
N_BN_CLICKED(IDC_BUTTON_TEMP, OnButtonTemp)
/}}AFX_MSG_MAP
ESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////////
est_serialDlg message handlers

CTest_serialDlg::OnInitDialog()

Dialog::OnInitDialog();
```

```
/ Add "About..." menu item to system menu.

/ IDM_ABOUTBOX must be in the system command range.
SSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
SSERT(IDM_ABOUTBOX < 0xF000);

Menu* pSysMenu = GetSystemMenu(FALSE);
f (pSysMenu != NULL)

   CString strAboutMenu;
   strAboutMenu.LoadString(IDS_ABOUTBOX);
   if (!strAboutMenu.IsEmpty())
   {
       pSysMenu->AppendMenu(MF_SEPARATOR);
       pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
   }


/ Set the icon for this dialog.  The framework does this automatically
/  when the application's main window is not a dialog
etIcon(m_hIcon, TRUE);         // Set big icon
etIcon(m_hIcon, FALSE);        // Set small icon

/ TODO: Add extra initialization here
_serial_flag = 0 ; // to indicate serial comm not open yet
wnd = GetSafeHwnd() ; // use this handle when I call class function from global function


eturn TRUE;  // return TRUE  unless you set the focus to a control


CTest_serialDlg::OnSysCommand(UINT nID, LPARAM lParam)

f ((nID & 0xFFF0) == IDM_ABOUTBOX)

   CAboutDlg dlgAbout;
   dlgAbout.DoModal();

lse

   CDialog::OnSysCommand(nID, lParam);



 you add a minimize button to your dialog, you will need the code below
o draw the icon.  For MFC applications using the document/view model,
his is automatically done for you by the framework.

CTest_serialDlg::OnPaint()

f (IsIconic())

   CPaintDC dc(this); // device context for painting

   SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

   // Center icon in client rectangle
   int cxIcon = GetSystemMetrics(SM_CXICON);
   int cyIcon = GetSystemMetrics(SM_CYICON);
   CRect rect;
   GetClientRect(&rect);
   int x = (rect.Width() - cxIcon + 1) / 2;
   int y = (rect.Height() - cyIcon + 1) / 2;
   // Draw the icon
   dc.DrawIcon(x, y, m_hIcon);

lse

   CDialog::OnPaint();



ə system calls this to obtain the cursor to display while the user drags
ıe minimized window.
ↃR CTest_serialDlg::OnQueryDragIcon()

ǝturn (HCURSOR) m_hIcon;


ↃTest_serialDlg::OnButtonOpenCom()

/ TODO: Add your control notification handler code here
* open COM port #1 for reading and writing */

f(m_serial_flag)

   MessageBox("comm port already open") ;
   return  ;


f ((hComm=CreateFile (lpszCommName, GENERIC_READ|GENERIC_WRITE,
, NULL, OPEN_EXISTING, 0, NULL)) == INVALID_HANDLE_VALUE)

   /* handle error */
   AfxMessageBox ("Error opening COM port");
   exit(1) ;
   /* end if (error creating read thread) */


etCommState (hComm, &PortDCB); //Setting Port

ortDCB.BaudRate = 2400;
ortDCB.fBinary = TRUE;
ortDCB.fParity = TRUE;
ortDCB.fOutxCtsFlow = FALSE;
ortDCB.fOutxDsrFlow = FALSE;
ortDCB.fDtrControl = DTR_CONTROL_ENABLE;

ortDCB.fDsrSensitivity = FALSE;
ortDCB.fTXContinueOnXoff = FALSE;
ortDCB.fOutX = FALSE;
ortDCB.fInX = FALSE;
ortDCB.fErrorChar = FALSE;
ortDCB.fRtsControl = RTS_CONTROL_DISABLE;

ortDCB.fAbortOnError = FALSE;

ortDCB.ByteSize = 8;
ortDCB.Parity = NOPARITY;
ortDCB.StopBits = ONESTOPBIT;

f (!SetCommState (hComm, &PortDCB))

   AfxMessageBox ("Unable to configure the serial port");
   exit(1) ;


///////////////////////////////////////////////////////
```

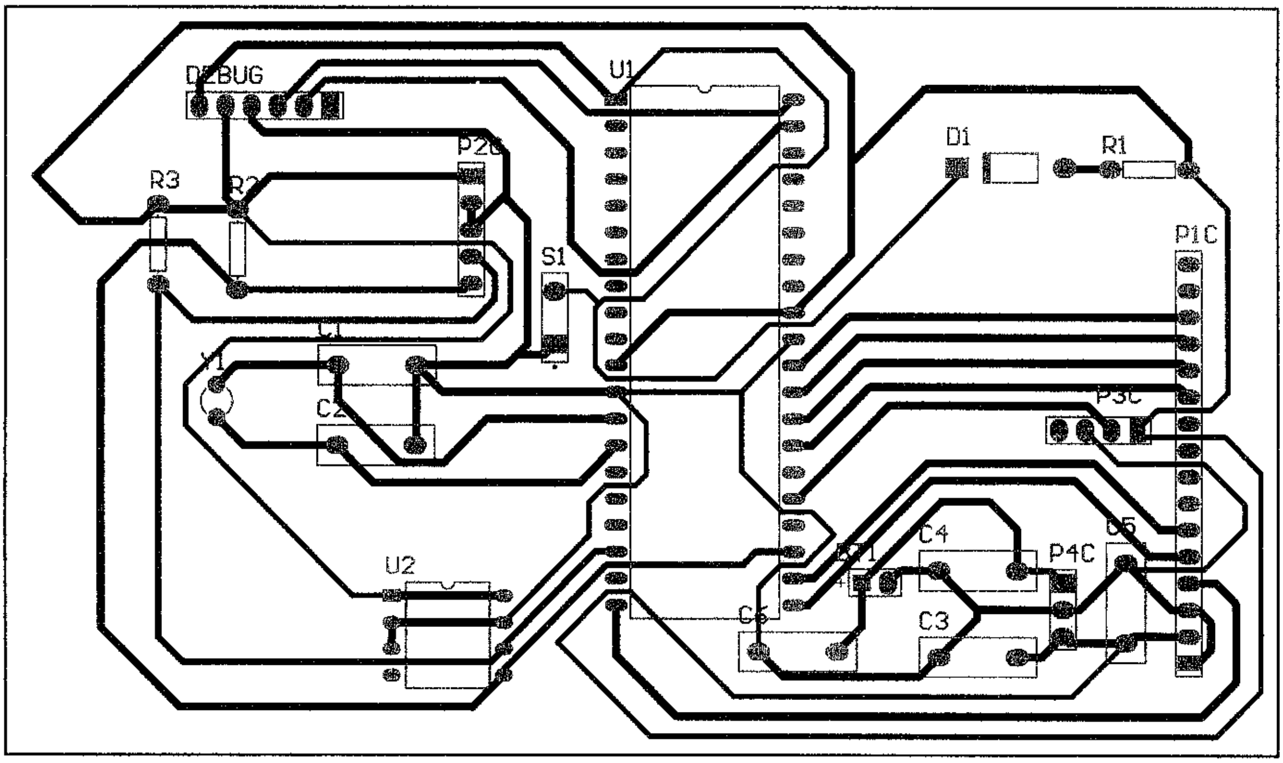
```
* the next four lines set the total timeout interval */
///////////////////////////////////////////////////////
/memset(&to, 0, sizeof(to) ) ;
/to.ReadTotalTimeoutMultiplier = 5 ;
).ReadIntervalTimeout = 30 ;
/to.ReadIntervalTimeout = MAXDWORD ;
/SetCommTimeouts(hComm,&to) ;

irgeComm(hComm,PURGE_RXCLEAR) ;



_serial_flag = 1 ;

?VOID pThreadParam = NULL ;
[erminateThread_flag =  FALSE  ;
_receive_thread = ::AfxBeginThread(receive_thread, pThreadParam,THREAD_PRIORITY_NORMAL,0,0,NU
 // to initiate thread for serial port function
/bTerminateThread_flag =  FALSE  ;
/ enable timer
/SetTimer(1,10,NULL) ;

etDlgItemText(IDC_STATIC_READSTATUS,"comm opened and thread started !!");



CTest_serialDlg::OnButtonCloseCom()

/ TODO: Add your control notification handler code here
iar str[20] ;


f(m_serial_flag==0)

   MessageBox("com port not open") ;




lse



   bTerminateThread_flag=TRUE ;
   Sleep(300) ;

   //CloseHandle(hComm) ;
   sprintf(str,"Thread Ends !! !");
   //MessageBox("com port closed") ;
   SetDlgItemText(IDC_STATIC_READSTATUS,str);
   m_serial_flag = 0 ;





CTest_serialDlg::OnButtonTest()

/ TODO: Add your control notification handler code here
r message[]={"Hello apakhabar\0"};

ata_rx = message ;
```

```
r message[] = {"xdsxxx%###shaf123456*\0"} ;



pdateData(FALSE) ;



CTest_serialDlg::OnPostMessage()

har data_char ;
har temp_buff[40] ;
nsigned int data_long ;


nt temp ;

/MessageBox("masukkkkk") ;

f(buff[0]=='z')
   m_rx_message = &buff[1] ;



lse if(buff[0]=='y')

   //m_data_rx += &buff[1] ;
   //m_data_rx += "\r\n" ;

    /////////////////////////////////////
   Ascii2hex(buff[1] , &data_char ) ;
   data_long = data_char << 4 ;

   Ascii2hex(buff[2] , &data_char ) ;
   data_long = data_long | data_char ;

   data_long = data_long << 4 ;
   Ascii2hex(buff[3] , &data_char );
   data_long = data_long | data_char ;

   data_long = data_long << 4 ;
   Ascii2hex(buff[4] , &data_char  );
   data_long = data_long | data_char ;
   //////////////////////////////////////

   data_long = (data_long * 100) / 0x7FFF ;


   sprintf(temp_buff,"%lu\n",data_long) ;
   //printf(temp_buff) ;

   //printf("hex value= %x integer value %d\n",data_long,data_long) ;
   m_data_rx += temp_buff ;
   m_data_rx += "\r\n" ;



lse if(buff[0]=='w')
```

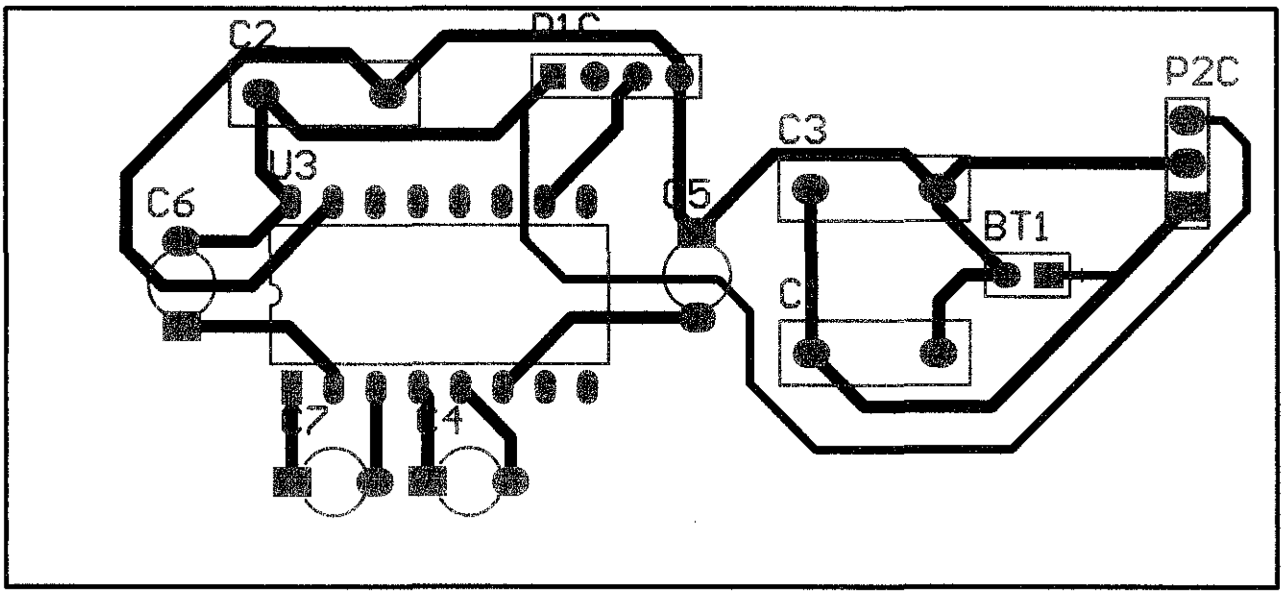
```
   Ascii2hex(buff[1] , &data_char ) ;
   data_long = data_char << 4 ;

   Ascii2hex(buff[2] , &data_char ) ;
   data_long = data_long | data_char ;

   data_long = data_long << 4 ;
   Ascii2hex(buff[3] , &data_char );
   data_long = data_long | data_char ;

   data_long = data_long << 4 ;
   Ascii2hex(buff[4] , &data_char  );
   data_long = data_long | data_char ;
   //////////////////////////////////////

   data_long = ((data_long * 165) /0x7FFF ) -40;


   sprintf(temp_buff,"%lu\n",data_long) ;
   //printf(temp_buff) ;

   //printf("hex value= %x integer value %d\n",data_long,data_long) ;
   m_data_rx2 += temp_buff ;
   m_data_rx2 += "\r\n" ;




   //m_data_rx2 += &buff[1] ;
   //m_data_rx2 += "\r\n" ;


odateData(FALSE) ;




receive_thread(LPVOID pThreadParam)


YTE inbuff;
WORD nBytesRead;

/AfxMessageBox ("Thread starts");
hile(bTerminateThread_flag==FALSE)


   Read_data();



/AfxMessageBox ("Thread ends");
fxEndThread(0) ;
eturn 0 ;



Read_data(void)

YTE inbuff;
WORD nBytesRead;
String m_data  ;
```

```
it i;

/ detect "abc" sequence before transferring into buffer,
/ the character '*' is used as the data terminator
/ this way garbage data read from serial port will be ignored
/ This is especially true for transmitting data using the RF channel
 = 0;
o

   if(bTerminateThread_flag == TRUE)
       return ;

   if(!ReadFile(hComm,&inbuff,1,&nBytesRead,NULL)) // Readfile is a function to read from ser
ort
       {
           AfxMessageBox ("Error"); return ;
       }
 while(inbuff!='a');

o

   if(bTerminateThread_flag == TRUE)
       return ;
   if(!ReadFile(hComm,&inbuff,1,&nBytesRead,NULL))
       {
           AfxMessageBox ("Error");return ;
       }
 while(inbuff!='b');

o

   if(bTerminateThread_flag == TRUE)
       return ;
   if(!ReadFile(hComm,&inbuff,1,&nBytesRead,NULL))
       {
           AfxMessageBox ("Error");return ;
       }
 while(inbuff!='c');




o

   if(bTerminateThread_flag == TRUE)
       return ;

   if(!ReadFile(hComm,&inbuff,1,&nBytesRead,NULL))
       {
           AfxMessageBox ("Error");
       }

   if(inbuff !='*')
       buff[i++]=inbuff;
 while(inbuff!='*');

uff[i++] = '\n' ;
uff[i] = 0 ; // null terminated
/str.Format("%s",buff);
/for(int z=0; z<20;z++) buff[z]=0;

/flag_barcode = 0;
/m_data = buff ;
/AfxMessageBox (m_data);
ostMessage(hwnd,WM_POSTMESSAGE,0,0) ; // a way to call class function  from global function
```

```
f(gflag==0) // ensure open only once
   ptr = fopen("data_from_remote.csv","w") ;

flag = 1 ; // set flag to idicate file has been opened
printf(ptr,"%s,\n",&buff[1]) ;
/fprintf(ptr,"test\n,") ;
count++ ;
f(gcount==50) // if 50 location reached close the file and reset gflag
              // here we assume the PIC only send 100 data only ( hmidity + temperature)
   fclose(ptr) ;
   gflag  = 0 ;
   gcount=0 ;

/gcount++ ;



/((CTest_serialDlg *)AfxGetMainWnd())->OnPostMessage() ;



send_char(BYTE inbuff)

VORD nBytesRead;

f(!WriteFile(hComm,&inbuff,1,&nBytesRead,NULL))

   AfxMessageBox ("Error");




CTest_serialDlg::OnButtonHumidity()

/ TODO: Add your control notification handler code here
end_char('H') ;



CTest_serialDlg::OnButtonTemp()

/ TODO: Add your control notification handler code here
end_char('T') ;



Ascii2hex( char temp, char  * hex_data)

f(   ( (temp <= 0x39) && (temp >=0x30) )  ||  ( (temp <= 0x46) && (temp >=0x41) )     )

   *hex_data = temp & 0xf ;
   if(temp >= 0x41)
   {
       *hex_data = *hex_data + 9 ;

   }
   return 0 ;
```



Comment	Description	Designator	Footprint	LibRef	Quantity
Battery	Multicell Battery	BT1	BAT-2	Battery	1
Cap	Capacitor	C1, C2, C3, C4, C5, C6	RAD-0.3	Cap	6
Diode 1N4002	1 Amp General Purpose Rectifier	D1	DIO10.46-5.3x2.8	Diode 1N4002	1
Header 6	Header, 6-Pin	DEBUG	HDR1X6	Header 6	1
Header 16	Header, 16-Pin	P1C	HDR1X16	Header 16	1
Sensor	Header, 5-Pin	P2C	HDR1X5	Header 5	1
Transmiter	Header, 4-Pin	P3C	HDR1X4	Header 4	1
Header 3	Header, 3-Pin	P4C	HDR1X3	Header 3	1
Res1	Resistor	R1, R2, R3	AXIAL-0.3	Res1	3
SW-PB	Switch	S1	SPST-2	SW-PB	1
PIC16F877A		U1	DIP40	PIC16F877A	1
EEPROM 24LC256		U2	DIP8	EEPROM 24LC256	1
XTAL	Crystal Oscillator	Y1	BCY-W2/D3.1	XTAL	1



Comment	Description	Designator	Footprint	LibRef	Quantity
Battery	Multicell Battery	BT1	BAT-2	Battery	1
Cap	Capacitor	C1, C2, C3	RAD-0.3	Cap	3
Cap2	Capacitor	C4, C5, C6, C7	CAPR5-4X5	Cap2	4
Header 4	Header, 4-Pin	P1C	HDR1X4	Header 4	1
Header 3	Header, 3-Pin	P2C	HDR1X3	Header 3	1
MAX232		U3	DIP16	MAX232	1

APPENDIX E

DATA SHEETS

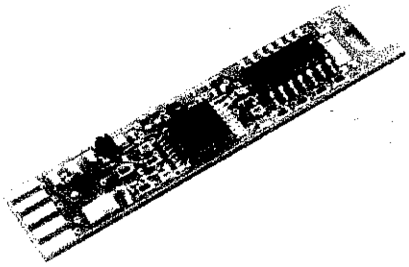
DigiPicco™ Basic I2C Capacitive Humidity Module Digital (I^2C)

Product

Within the markets Measurement, HVAC, Building and Control, and Home Appliances/White Goods, humidity modules are required which are capable to translate the signals of the robust IST humidity sensors into commonly used standards and provide a calibrated sensor signal. Contrary to existing humidity modules or fully integrated solutions the DigiPicco series unifies advantages of both worlds, avoiding their disadvantages: The high precision measurement of humidity with discrete sensors (high stability due to wide active sensor area) combined with calibrated and linearized output signal and fully digital output of both humidity and temperature.

Advantages

- Excellent response time
- Calibration free
- Ready to use
- Precise humidity measurement
- Drift stable thanks to wide sensor area
- With temperature sensor PT1000
- Smallest dimensions
- Mechanical robust and easy to integrate
- Calibrated humidity and temperatures signal on one single bus.
- RoHs conform



Technical data

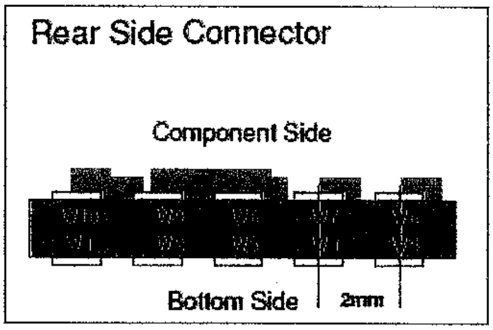
Sensor Type:	P14 SMD
Measurement principle:	Capacitive humidity sensor
Mechanical dimensions:	W=10 x L=47 x T=2.8mm
Humidity measurement range:	0 ... 100 % RH (max. dew point = 85 deg C)
Operating temperature range:	- 25 ... +85 deg C
Supply voltage:	5 Volts DC
Current consumption:	< 3 mA
Output signal:	0x0...0x7FFF (0...100% RH), 0x0...0x7FFF (-40....+125 deg C)
Temperature sensor:	PT1000
Storage temperature:	-40...+100 deg C / at max. 95% RH non condensing
Accuracy:	< ±3 %RH (15 ... 85 % RH @ 23 deg C) < ±0.5 deg C (-25...+85 deg C)
Response time T_{63}:	< 5 sec
Output terminals:	Soldering pads for VCC, clock and data (I^2C), GND

IUMIDITY

DigiPicco™ Basic I2C Capacitive Humidity Module Digital (I^2C)

Terminal Pinout

W1	Reserved
W2	Reserved
W3	Clock SCL (I^2C)
W4	Data SDA (I^2C)
W5	Reserved
W6	Reserved
W7	Signal GND
W8	GND
W9	Reserved
W10	Vcc +



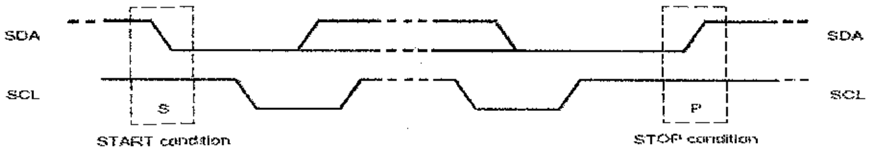
Description I^2C

First of all the external microcontroller (master) sends the start condition to the slave (DigiPicco). Then the master transmits the standard 7 Bit address (0x78) or a factory customizable address. The eight bit (LSB) determines the direction of data flow and has to be set during this operation. Following, the slave (DigiPicco) acknowledges the receipt of data with the acknowledge condition (SDA kept low during a positive clock cycle). After that, the slave (DigiPicco) outputs the data values. After each data byte the master has to acknowledge the receipt of the data values by the acknowledge condition, except before the stop condition has been sent by the master itself.

The humidity and the temperature values exist of two bytes each. The first two bytes are the humidity values and the second two bytes are the temperature values, 15 bit each. This sequence is repeated indefinitely until the stop condition has been sent (also refer to diagram below).

Start Condition:
SDA changes from high to low during SCL is in high condition.

Stop Condition:
SDA changes from low to high during SCL is in high condition.



Start- und Stop Condition

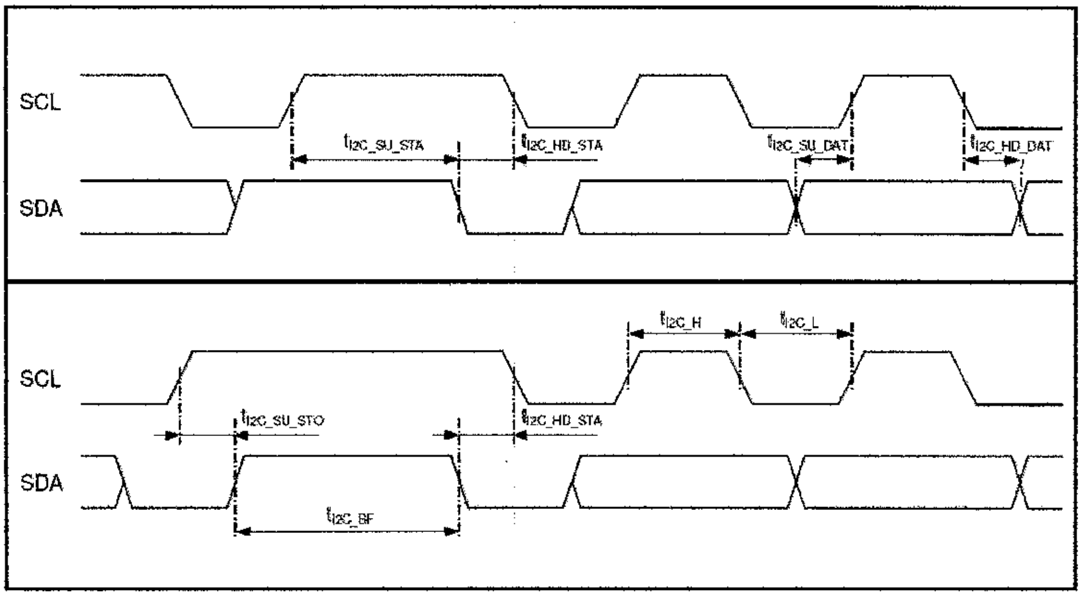
optional

	start condition	slave address	R/W 1	A	1st data byte	A	2nd data byte	A	... nth data byte	stop condition
sent by	master			slave	slave	master	slave	master	slave	Master

Typical read operation timing sequence

DigiPicco™ Basic I2C Capacitive Humidity Module Digital (I^{2}C)

Slave-address:	0x78 or factory definable customer specific address
SCL clock-frequency:	Max. 400kHz
Bus free time between start- and stop condition t_{I2C_BF}:	Min. 1.3µs
Hold delay start condition $t_{I2C\_HD\text{-}STA}$:	Min. 0.6µs
Setup time start condition $t_{I2C\_SU\_STA}$:	Min. 0.6µs
Setup time stop condition $t_{I2C\_SU\_STO}$:	Min. 0.6µs
Data hold time (trigger=data) $t_{I2C\_HD\_DAT}$:	0µs
Data setup time $t_{I2C\_SU\_DAT}$:	Min. 0.1µs
Low period SDA/SCL t_{I2C_L}:	Min. 1.3µs
High period SDA/SCL t_{I2C_H}:	Min. 0.6µs
Input-high-level:	2.4...3V
Input-low-level:	0.0...0.6V
External pull- up resistor:	Min. 2kΩ
Maximum load capacitance:	Max. 2nF



General timing diagram

All mechanical dimensions are valid at 25°C ambient temperature, if not differently indicated. ■ All data except the mechanical dimensions only have information purposes and are not to be understood as assured characteristics. ■ Technical changes without previous announcement as well as mistakes reserve. ■ The information on this data sheet was examined carefully and will be accepted as correct; No liability in case of mistakes.■ Load with extreme values during a longer period can affect the reliability

V4.1-06/2008

INNOVATIVE SENSOR TECHNOLOGY

MICROCHIP

PIC16F87XA Data Sheet

28/40/44-Pin Enhanced Flash Microcontrollers

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, MPLAB, PIC, PICmicro, PICSTART, PRO MATE and PowerSmart are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, microID, MXDEV, MXLAB, PICMASTER, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Application Maestro, dsPICDEM, dsPICDEM.net, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICkit, PICDEM, PICDEM.net, PowerCal, PowerInfo, PowerMate, PowerTool, rfLAB, rfPIC, Select Mode, SmartSensor, SmartShunt, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

Printed on recycled paper.

DNV Certification, Inc.
USA

ANSI-RAB
QMS
ACCREDITED

DNV MSC
The Netherlands
Accredited by the RvA

DNV

ISO 9001 / QS-9000
REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999 and Mountain View, California in March 2002. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.

MICROCHIP

PIC16F87XA

28/40/44-Pin Enhanced Flash Microcontrollers

Devices Included in this Data Sheet:

- PIC16F873A
- PIC16F874A
- PIC16F876A
- PIC16F877A

High-Performance RISC CPU:

- Only 35 single-word instructions to learn
- All single-cycle instructions except for program branches, which are two-cycle
- Operating speed: DC – 20 MHz clock input
DC – 200 ns instruction cycle
- Up to 8K x 14 words of Flash Program Memory, Up to 368 x 8 bytes of Data Memory (RAM), Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to other 28-pin or 40/44-pin PIC16CXXX and PIC16FXXX microcontrollers

Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during Sleep via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I^{2}C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) – 8 bits wide with external $\overline{\text{RD}}$, $\overline{\text{WR}}$ and $\overline{\text{CS}}$ controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

Analog Features:

- 10-bit, up to 8-channel Analog-to-Digital Converter (A/D)
- Brown-out Reset (BOR)
- Analog Comparator module with:
  - Two analog comparators
  - Programmable on-chip voltage reference (VREF) module
  - Programmable input multiplexing from device inputs and internal voltage reference
  - Comparator outputs are externally accessible

Special Microcontroller Features:

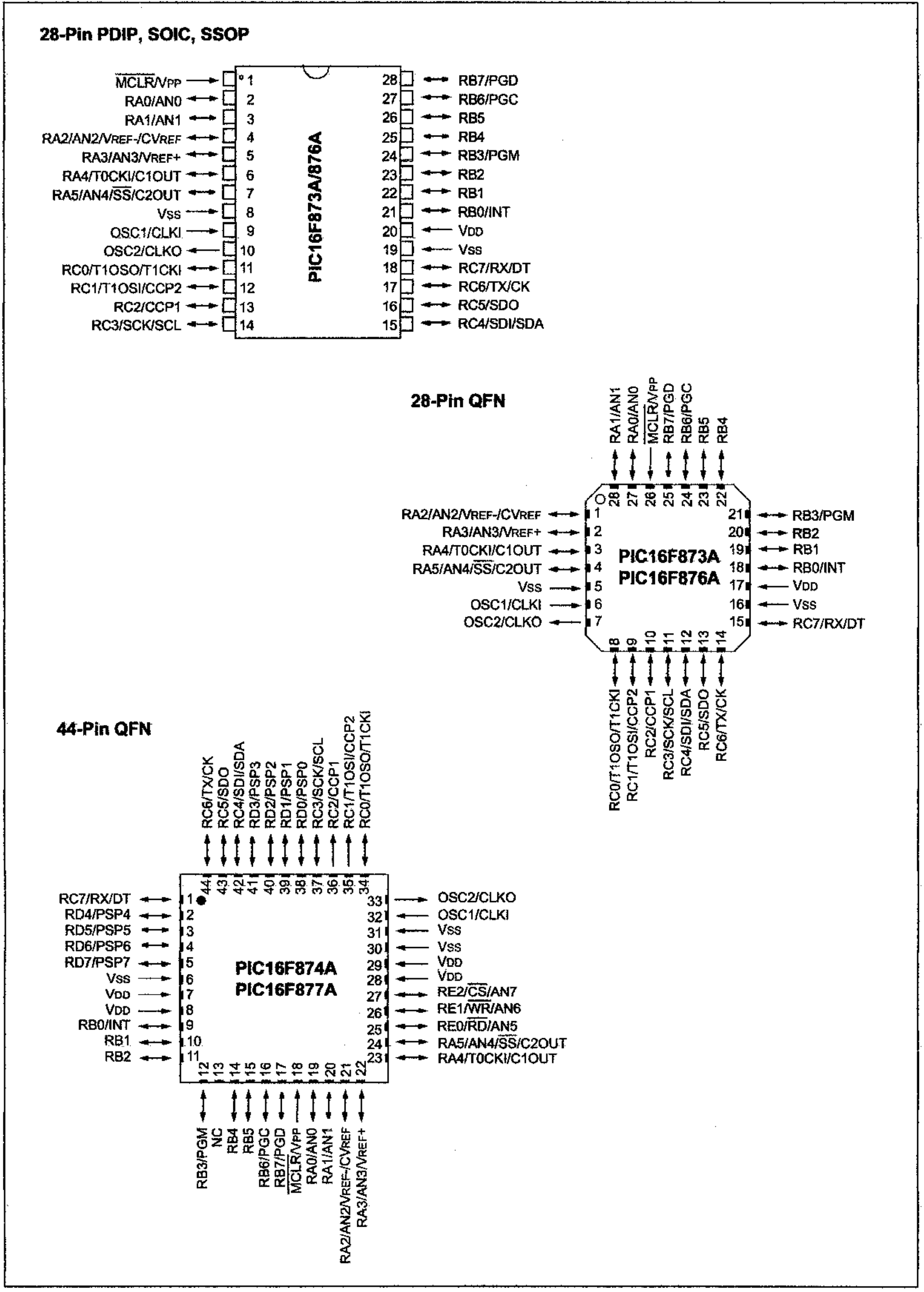
- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Data EEPROM Retention > 40 years
- Self-reprogrammable under software control
- In-Circuit Serial Programming™ (ICSP™) via two pins
- Single-supply 5V In-Circuit Serial Programming
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving Sleep mode
- Selectable oscillator options
- In-Circuit Debug (ICD) via two pins

CMOS Technology:

- Low-power, high-speed Flash/EEPROM technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Commercial and Industrial temperature ranges
- Low-power consumption

Device	Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
	Bytes	# Single Word Instructions						SPI	Master I^{2}C			
PIC16F873A	7.2K	4096	192	128	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F874A	7.2K	4096	192	128	33	8	2	Yes	Yes	Yes	2/1	2
PIC16F876A	14.3K	8192	368	256	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F877A	14.3K	8192	368	256	33	8	2	Yes	Yes	Yes	2/1	2

Pin Diagrams



Pin Diagrams (Continued)

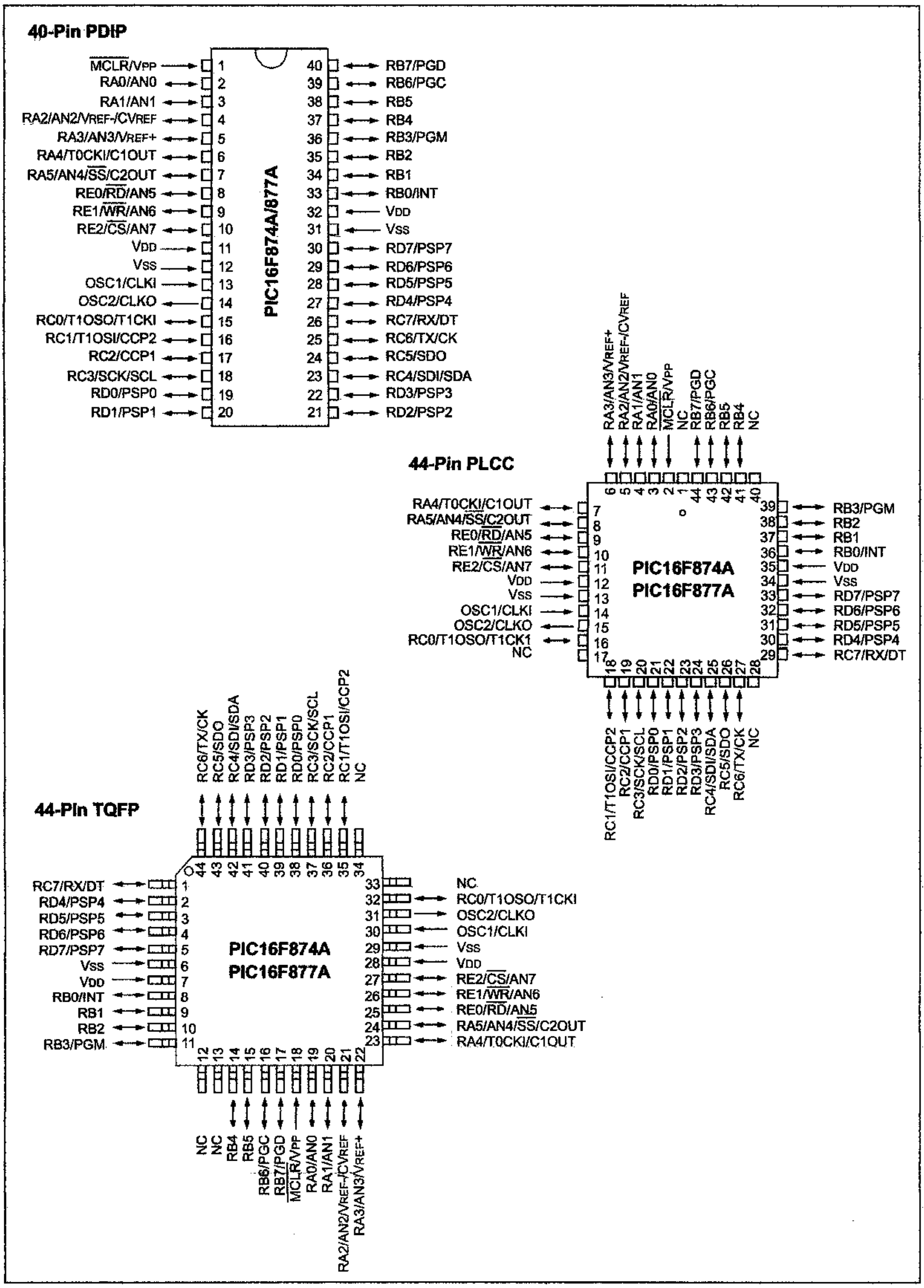


Table of Contents

TO OUR VALUED CUSTOMERS

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip products. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department via E-mail at **docerrors@mail.microchip.com** or fax the **Reader Response Form** in the back of this data sheet to (480) 792-4150. We welcome your feedback.

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please register at our Worldwide Web site at:

http://www.microchip.com

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number, (e.g., DS30000A is version A of document DS30000).

Errata

An errata sheet, describing minor operational differences from the data sheet and recommended workarounds, may exist for current devices. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Web site; http://www.microchip.com
- Your local Microchip sales office (see last page)
- The Microchip Corporate Literature Center; U.S. FAX: (480) 792-7277

When contacting a sales office or the literature center, please specify which device, revision of silicon and data sheet (include literature number) you are using.

Customer Notification System

Register on our Web site at **www.microchip.com/cn** to receive the most current information on all of our products.

1.0 DEVICE OVERVIEW

This document contains device specific information about the following devices:

- PIC16F873A
- PIC16F874A
- PIC16F876A
- PIC16F877A

PIC16F873A/876A devices are available only in 28-pin packages, while PIC16F874A/877A devices are available in 40-pin and 44-pin packages. All devices in the PIC16F87XA family share common architecture with the following differences:

- The PIC16F873A and PIC16F874A have one-half of the total on-chip memory of the PIC16F876A and PIC16F877A
- The 28-pin devices have three I/O ports, while the 40/44-pin devices have five
- The 28-pin devices have fourteen interrupts, while the 40/44-pin devices have fifteen
- The 28-pin devices have five A/D input channels, while the 40/44-pin devices have eight
- The Parallel Slave Port is implemented only on the 40/44-pin devices

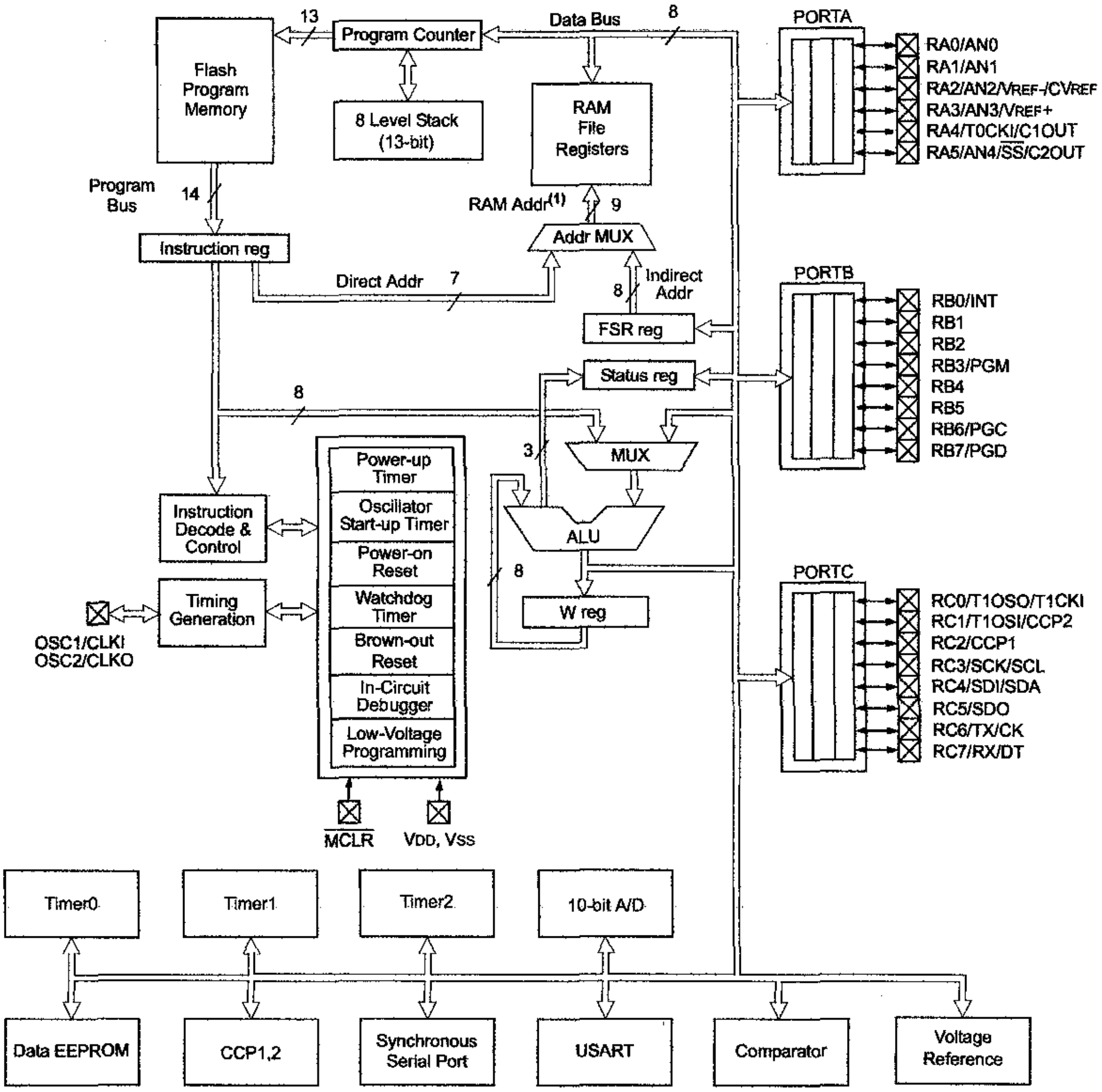
The available features are summarized in Table 1-1. Block diagrams of the PIC16F873A/876A and PIC16F874A/877A devices are provided in Figure 1-1 and Figure 1-2, respectively. The pinouts for these device families are listed in Table 1-2 and Table 1-3.

Additional information may be found in the PICmicro® Mid-Range Reference Manual (DS33023), which may be obtained from your local Microchip Sales Representative or downloaded from the Microchip web site. The Reference Manual should be considered a complementary document to this data sheet and is highly recommended reading for a better understanding of the device architecture and operation of the peripheral modules.

TABLE 1-1: PIC16F87XA DEVICE FEATURES

Key Features	PIC16F873A	PIC16F874A	PIC16F876A	PIC16F877A
Operating Frequency	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz
Resets (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
Flash Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory (bytes)	128	128	256	256
Interrupts	14	15	14	15
I/O Ports	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C	Ports A, B, C, D, E
Timers	3	3	3	3
Capture/Compare/PWM modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Analog Comparators	2	2	2	2
Instruction Set	35 Instructions	35 Instructions	35 Instructions	35 Instructions
Packages	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 44-pin PLCC 44-pin TQFP 44-pin QFN	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 44-pin PLCC 44-pin TQFP 44-pin QFN

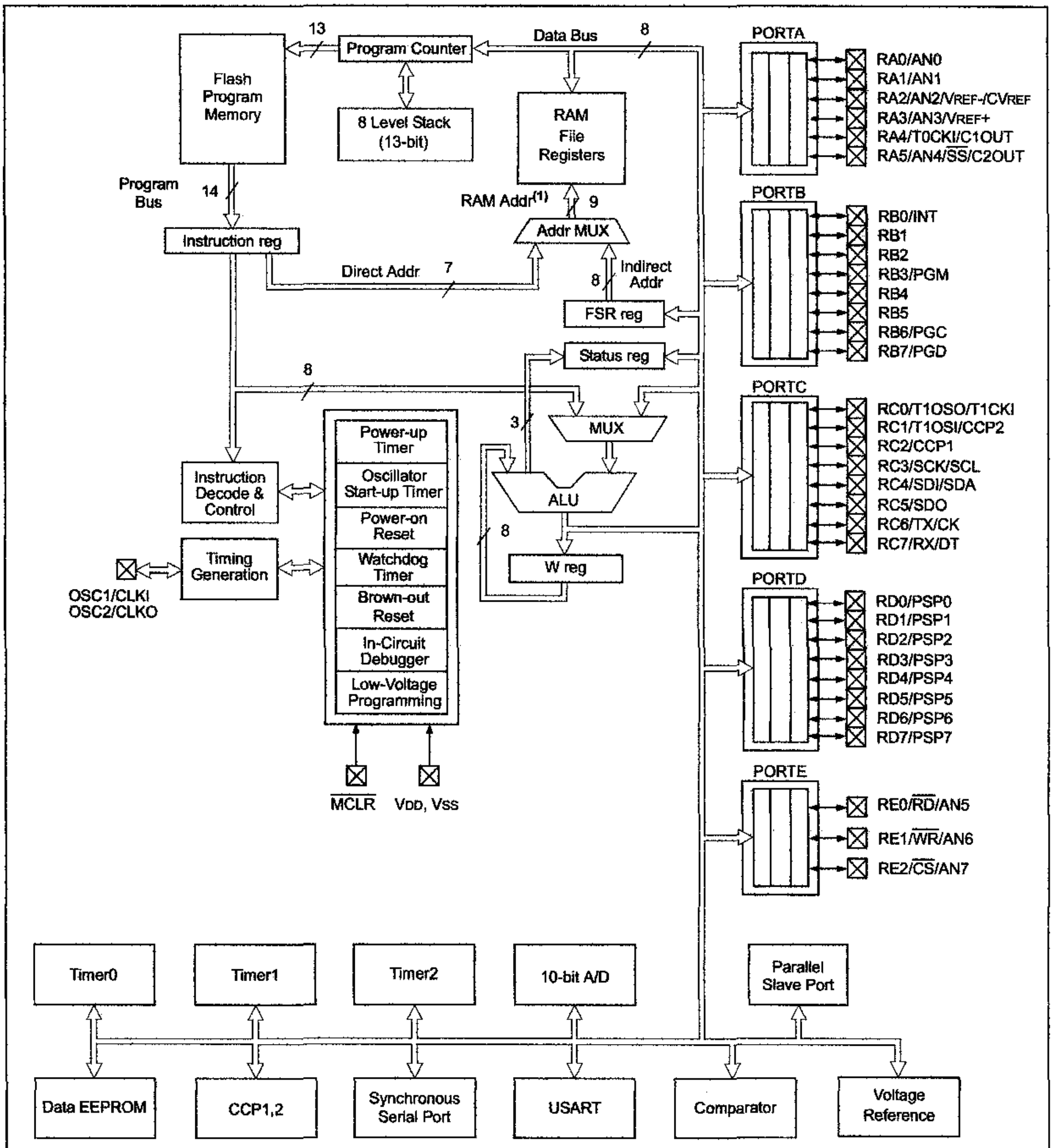
FIGURE 1-1: PIC16F873A/876A BLOCK DIAGRAM



Device	Program Flash	Data Memory	Data EEPROM
PIC16F873A	4K words	192 Bytes	128 Bytes
PIC16F876A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the Status register.

FIGURE 1-2: PIC16F874A/877A BLOCK DIAGRAM



Device	Program Flash	Data Memory	Data EEPROM
PIC16F874A	4K words	192 Bytes	128 Bytes
PIC16F877A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the Status register.

TABLE 1-2: PIC16F873A/876A PINOUT DESCRIPTION

Pin Name	PDIP, SOIC, SSOP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKI	9	6		ST/CMOS[3]	Oscillator crystal or external clock input.
OSC1			I		Oscillator crystal input or external clock source input. ST buffer when configured in RC mode; otherwise CMOS.
CLKI			I		External clock source input. Always associated with pin function OSC1 (see OSC1/CLKI, OSC2/CLKO pins).
OSC2/CLKO	10	7		—	Oscillator crystal or clock output.
OSC2			O		Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
CLKO			O		In RC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
$\overline{\text{MCLR}}$/VPP	1	26		ST	Master Clear (input) or programming voltage (output).
$\overline{\text{MCLR}}$			I		Master Clear (Reset) input. This pin is an active low Reset to the device.
VPP			P		Programming voltage input.
					PORTA is a bidirectional I/O port.
RA0/AN0	2	27		TTL	
RA0			I/O		Digital I/O.
AN0			I		Analog input 0.
RA1/AN1	3	28		TTL	
RA1			I/O		Digital I/O.
AN1			I		Analog input 1.
RA2/AN2/VREF-/	4	1		TTL	
CVREF			I/O		Digital I/O.
RA2			I		Analog input 2.
AN2			I		A/D reference voltage (Low) input.
VREF-			O		Comparator VREF output.
CVREF					
RA3/AN3/VREF+	5	2		TTL	
RA3			I/O		Digital I/O.
AN3			I		Analog input 3.
VREF+			I		A/D reference voltage (High) input.
RA4/T0CKI/C1OUT	6	3		ST	
RA4			I/O		Digital I/O – Open-drain when configured as output.
T0CKI			I		Timer0 external clock input.
C1OUT			O		Comparator 1 output.
RA5/AN4/$\overline{\text{SS}}$/C2OUT	7	4		TTL	
RA5			I/O		Digital I/O.
AN4			I		Analog input 4.
$\overline{\text{SS}}$			I		SPI slave select input.
C2OUT			O		Comparator 2 output.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

TABLE 1-2: PIC16F873A/876A PINOUT DESCRIPTION (CONTINUED)

Pin Name	PDIP, SOIC, SSOP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
					PORTB is a bidirectional I/O port. PORTB can be software programmed for internal weak pull-ups on all inputs.
RB0/INT	21	18		TTL/ST(1)	
RB0			I/O		Digital I/O.
INT			I		External interrupt.
RB1	22	19	I/O	TTL	Digital I/O.
RB2	23	20	I/O	TTL	Digital I/O.
RB3/PGM	24	21		TTL	
RB3			I/O		Digital I/O.
PGM			I		Low-voltage (single-supply) ICSP programming enable pin.
RB4	25	22	I/O	TTL	Digital I/O.
RB5	26	23	I/O	TTL	Digital I/O.
RB6/PGC	27	24		TTL/ST(2)	
RB6			I/O		Digital I/O.
PGC			I		In-circuit debugger and ICSP programming clock.
RB7/PGD	28	25		TTL/ST(2)	
RB7			I/O		Digital I/O.
PGD			I/O		In-circuit debugger and ICSP programming data.
					PORTC is a bidirectional I/O port.
RC0/T1OSO/T1CKI	11	8		ST	
RC0			I/O		Digital I/O.
T1OSO			O		Timer1 oscillator output.
T1CKI			I		Timer1 external clock input.
RC1/T1OSI/CCP2	12	9		ST	
RC1			I/O		Digital I/O.
T1OSI			I		Timer1 oscillator input.
CCP2			I/O		Capture2 input, Compare2 output, PWM2 output.
RC2/CCP1	13	10		ST	
RC2			I/O		Digital I/O.
CCP1			I/O		Capture1 input, Compare1 output, PWM1 output.
RC3/SCK/SCL	14	11		ST	
RC3			I/O		Digital I/O.
SCK			I/O		Synchronous serial clock input/output for SPI mode.
SCL			I/O		Synchronous serial clock input/output for I^2C mode.
RC4/SDI/SDA	15	12		ST	
RC4			I/O		Digital I/O.
SDI			I		SPI data in.
SDA			I/O		I^2C data I/O.
RC5/SDO	16	13		ST	
RC5			I/O		Digital I/O.
SDO			O		SPI data out.
RC6/TX/CK	17	14		ST	
RC6			I/O		Digital I/O.
TX			O		USART asynchronous transmit.
CK			I/O		USART1 synchronous clock.
RC7/RX/DT	18	15		ST	
RC7			I/O		Digital I/O.
RX			I		USART asynchronous receive.
DT			I/O		USART synchronous data.
VSS	8, 19	5, 6	P	—	Ground reference for logic and I/O pins.
VDD	20	17	P	—	Positive supply for logic and I/O pins.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

TABLE 1-3: PIC16F874A/877A PINOUT DESCRIPTION

Pin Name	PDIP Pin#	PLCC Pin#	TQFP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKI	13	14	30	32		ST/CMOS[4]	Oscillator crystal or external clock input.
OSC1					I		Oscillator crystal input or external clock source input. ST buffer when configured in RC mode; otherwise CMOS.
CLKI					I		External clock source input. Always associated with pin function OSC1 (see OSC1/CLKI, OSC2/CLKO pins).
OSC2/CLKO	14	15	31	33		—	Oscillator crystal or clock output.
OSC2					O		Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
CLKO					O		In RC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
$\overline{\text{MCLR}}$/VPP	1	2	18	18		ST	Master Clear (input) or programming voltage (output).
$\overline{\text{MCLR}}$					I		Master Clear (Reset) input. This pin is an active low Reset to the device.
VPP					P		Programming voltage input.
							PORTA is a bidirectional I/O port.
RA0/AN0	2	3	19	19		TTL	
RA0					I/O		Digital I/O.
AN0					I		Analog input 0.
RA1/AN1	3	4	20	20		TTL	
RA1					I/O		Digital I/O.
AN1					I		Analog input 1.
RA2/AN2/VREF-/CVREF	4	5	21	21		TTL	
RA2					I/O		Digital I/O.
AN2					I		Analog input 2.
VREF-					I		A/D reference voltage (Low) input.
CVREF					O		Comparator VREF output.
RA3/AN3/VREF+	5	6	22	22		TTL	
RA3					I/O		Digital I/O.
AN3					I		Analog input 3.
VREF+					I		A/D reference voltage (High) input.
RA4/T0CKI/C1OUT	6	7	23	23		ST	
RA4					I/O		Digital I/O – Open-drain when configured as output.
T0CKI					I		Timer0 external clock input.
C1OUT					O		Comparator 1 output.
RA5/AN4/$\overline{\text{SS}}$/C2OUT	7	8	24	24		TTL	
RA5					I/O		Digital I/O.
AN4					I		Analog input 4.
$\overline{\text{SS}}$					I		SPI slave select input.
C2OUT					O		Comparator 2 output.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.

2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.

3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

TABLE 1-3: PIC16F874A/877A PINOUT DESCRIPTION (CONTINUED)

Pin Name	PDIP Pin#	PLCC Pin#	TQFP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
							PORTB is a bidirectional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	33	36	8	9		TTL/ST(1)	
RB0					I/O		Digital I/O.
INT					I		External interrupt.
RB1	34	37	9	10	I/O	TTL	Digital I/O.
RB2	35	38	10	11	I/O	TTL	Digital I/O.
RB3/PGM	36	39	11	12		TTL	
RB3					I/O		Digital I/O.
PGM					I		Low-voltage ICSP programming enable pin.
RB4	37	41	14	14	I/O	TTL	Digital I/O.
RB5	38	42	15	15	I/O	TTL	Digital I/O.
RB6/PGC	39	43	16	16		TTL/ST(2)	
RB6					I/O		Digital I/O.
PGC					I		In-circuit debugger and ICSP programming clock.
RB7/PGD	40	44	17	17		TTL/ST(2)	
RB7					I/O		Digital I/O.
PGD					I/O		In-circuit debugger and ICSP programming data.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

TABLE 1-3: PIC16F874A/877A PINOUT DESCRIPTION (CONTINUED)

Pin Name	PDIP Pin#	PLCC Pin#	TQFP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
							PORTC is a bidirectional I/O port.
RC0/T1OSO/T1CKI	15	16	32	34		ST	
RC0					I/O		Digital I/O.
T1OSO					O		Timer1 oscillator output.
T1CKI					I		Timer1 external clock input.
RC1/T1OSI/CCP2	16	18	35	35		ST	
RC1					I/O		Digital I/O.
T1OSI					I		Timer1 oscillator input.
CCP2					I/O		Capture2 input, Compare2 output, PWM2 output.
RC2/CCP1	17	19	36	36		ST	
RC2					I/O		Digital I/O.
CCP1					I/O		Capture1 input, Compare1 output, PWM1 output.
RC3/SCK/SCL	18	20	37	37		ST	
RC3					I/O		Digital I/O.
SCK					I/O		Synchronous serial clock input/output for SPI mode.
SCL					I/O		Synchronous serial clock input/output for I^{2}C mode.
RC4/SDI/SDA	23	25	42	42		ST	
RC4					I/O		Digital I/O.
SDI					I		SPI data in.
SDA					I/O		I^{2}C data I/O.
RC5/SDO	24	26	43	43		ST	
RC5					I/O		Digital I/O.
SDO					O		SPI data out.
RC6/TX/CK	25	27	44	44		ST	
RC6					I/O		Digital I/O.
TX					O		USART asynchronous transmit.
CK					I/O		USART1 synchronous clock.
RC7/RX/DT	26	29	1	1		ST	
RC7					I/O		Digital I/O.
RX					I		USART asynchronous receive.
DT					I/O		USART synchronous data.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

TABLE 1-3: PIC16F874A/877A PINOUT DESCRIPTION (CONTINUED)

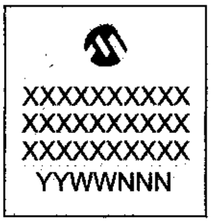
Pin Name	PDIP Pin#	PLCC Pin#	TQFP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
							PORTD is a bidirectional I/O port or Parallel Slave Port when interfacing to a microprocessor bus.
RD0/PSP0	19	21	38	38		ST/TTL$^{(3)}$	
RD0					I/O		Digital I/O.
PSP0					I/O		Parallel Slave Port data.
RD1/PSP1	20	22	39	39		ST/TTL$^{(3)}$	
RD1					I/O		Digital I/O.
PSP1					I/O		Parallel Slave Port data.
RD2/PSP2	21	23	40	40		ST/TTL$^{(3)}$	
RD2					I/O		Digital I/O.
PSP2					I/O		Parallel Slave Port data.
RD3/PSP3	22	24	41	41		ST/TTL$^{(3)}$	
RD3					I/O		Digital I/O.
PSP3					I/O		Parallel Slave Port data.
RD4/PSP4	27	30	2	2		ST/TTL$^{(3)}$	
RD4					I/O		Digital I/O.
PSP4					I/O		Parallel Slave Port data.
RD5/PSP5	28	31	3	3		ST/TTL$^{(3)}$	
RD5					I/O		Digital I/O.
PSP5					I/O		Parallel Slave Port data.
RD6/PSP6	29	32	4	4		ST/TTL$^{(3)}$	
RD6					I/O		Digital I/O.
PSP6					I/O		Parallel Slave Port data.
RD7/PSP7	30	33	5	5		ST/TTL$^{(3)}$	
RD7					I/O		Digital I/O.
PSP7					I/O		Parallel Slave Port data.
							PORTE is a bidirectional I/O port.
RE0/$\overline{\text{RD}}$/AN5	8	9	25	25		ST/TTL$^{(3)}$	
RE0					I/O		Digital I/O.
$\overline{\text{RD}}$					I		Read control for Parallel Slave Port.
AN5					I		Analog input 5.
RE1/$\overline{\text{WR}}$/AN6	9	10	26	26		ST/TTL$^{(3)}$	
RE1					I/O		Digital I/O.
$\overline{\text{WR}}$					I		Write control for Parallel Slave Port.
AN6					I		Analog input 6.
RE2/$\overline{\text{CS}}$/AN7	10	11	27	27		ST/TTL$^{(3)}$	
RE2					I/O		Digital I/O.
$\overline{\text{CS}}$					I		Chip select control for Parallel Slave Port.
AN7					I		Analog input 7.
VSS	12, 31	13, 34	6, 29	6, 30, 31	P	—	Ground reference for logic and I/O pins.
VDD	11, 32	12, 35	7, 28	7, 8, 28, 29	P	—	Positive supply for logic and I/O pins.
NC	—	1, 17, 28, 40	12,13, 33, 34	13	—	—	These pins are not internally connected. These pins should be left unconnected.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

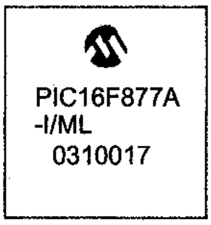
Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

Package Marking Information (Cont'd)

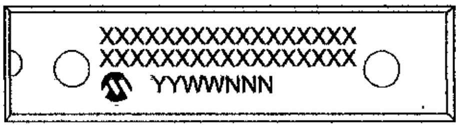
44-Lead QFN



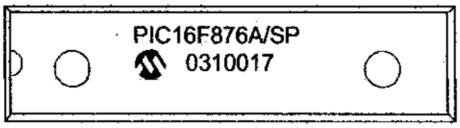
Example



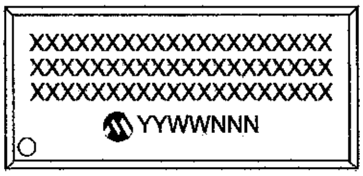
28-Lead PDIP (Skinny DIP)



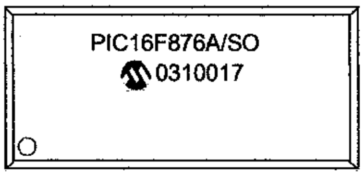
Example



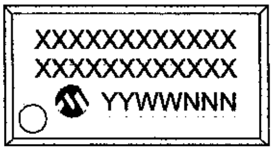
28-Lead SOIC



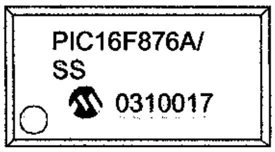
Example



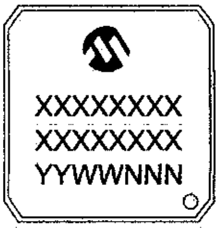
28-Lead SSOP



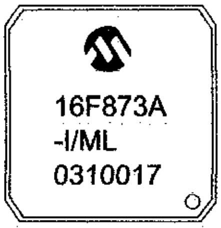
Example



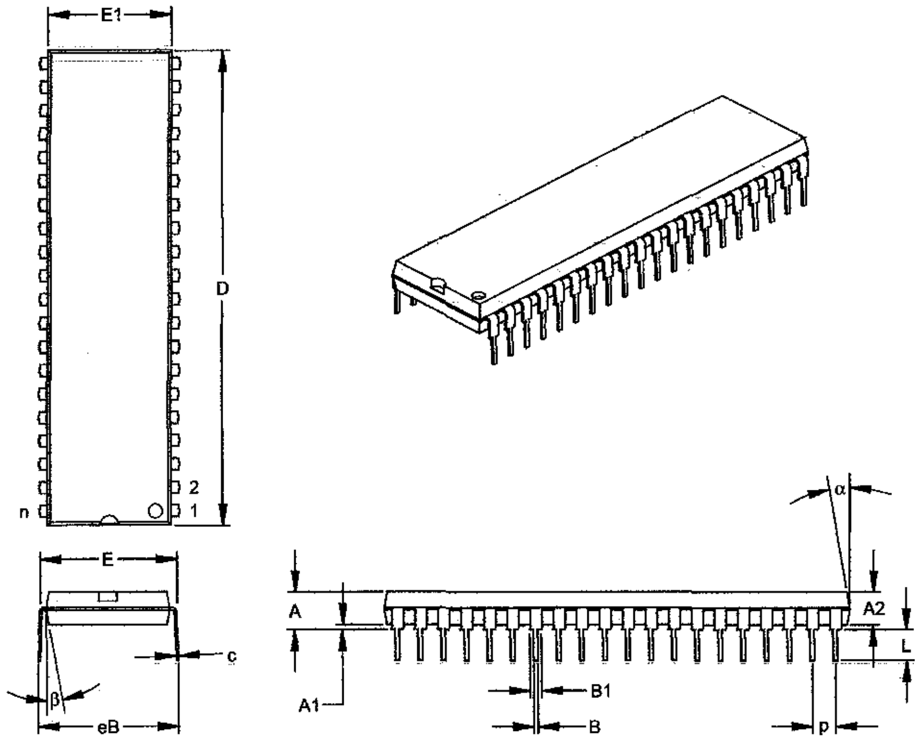
28-Lead QFN



Example



40-Lead Plastic Dual In-line (P) – 600 mil (PDIP)



	Units	INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		40			40	
Pitch	p		.100			2.54	
Top to Seating Plane	A	.160	.175	.190	4.06	4.45	4.83
Molded Package Thickness	A2	.140	.150	.160	3.56	3.81	4.06
Base to Seating Plane	A1	.015			0.38		
Shoulder to Shoulder Width	E	.595	.600	.625	15.11	15.24	15.88
Molded Package Width	E1	.530	.545	.560	13.46	13.84	14.22
Overall Length	D	2.045	2.058	2.065	51.94	52.26	52.45
Tip to Seating Plane	L	.120	.130	.135	3.05	3.30	3.43
Lead Thickness	c	.008	.012	.015	0.20	0.29	0.38
Upper Lead Width	B1	.030	.050	.070	0.76	1.27	1.78
Lower Lead Width	B	.014	.018	.022	0.36	0.46	0.56
Overall Row Spacing §	eB	.620	.650	.680	15.75	16.51	17.27
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	15

* Controlling Parameter

§ Significant Characteristic

Notes:

Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" (0.254mm) per side.

JEDEC Equivalent: MO-011

Drawing No. C04-016

MICROCHIP 24AA256/24LC256/24FC256

256K I^{2}C™ CMOS Serial EEPROM

Device Selection Table

Part Number	Vcc Range	Max. Clock Frequency	Temp. Ranges
24AA256	1.8-5.5V	400 kHz[(1)]	I
24LC256	2.5-5.5V	400 kHz	I, E
24FC256	1.8-5.5V	1 MHz[(2)]	I

Note 1: 100 kHz for Vcc < 2.5V.
2: 400 kHz for Vcc < 2.5V.

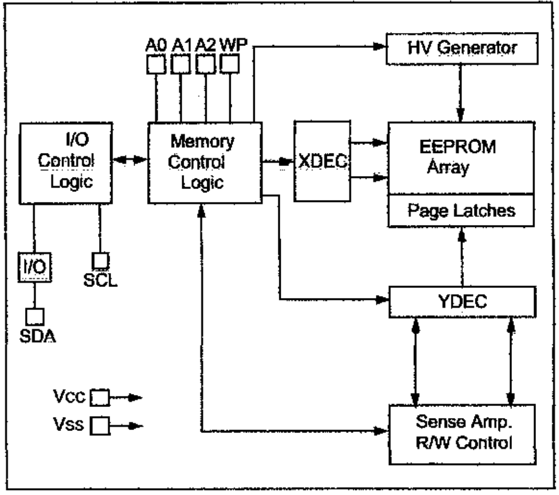
Features

- Low-power CMOS technology:
  - Maximum write current 3 mA at 5.5V
  - Maximum read current 400 µA at 5.5V
  - Standby current 100 nA typical at 5.5V
- 2-wire serial interface bus, I^{2}C™ compatible
- Cascadable for up to eight devices
- Self-timed erase/write cycle
- 64-byte Page Write mode available
- 5 ms max. write cycle time
- Hardware write-protect for entire array
- Output slope control to eliminate ground bounce
- Schmitt Trigger inputs for noise suppression
- 1,000,000 erase/write cycles
- Electrostatic discharge protection > 4000V
- Data retention > 200 years
- 8-pin PDIP, SOIC, TSSOP, MSOP and DFN packages, 14-lead TSSOP package
- Standard and Pb-free finishes available
- Temperature ranges:
  - Industrial (I): -40°C to +85°C
  - Automotive (E): -40°C to +125°C

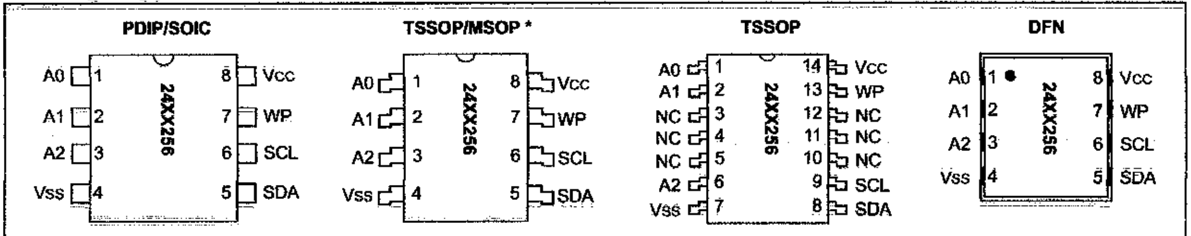
Description

The Microchip Technology Inc. 24AA256/24LC256/24FC256 (24XX256*) is a 32K x 8 (256 Kbit) Serial Electrically Erasable PROM, capable of operation across a broad voltage range (1.8V to 5.5V). It has been developed for advanced, low-power applications such as personal communications or data acquisition. This device also has a page write capability of up to 64 bytes of data. This device is capable of both random and sequential reads up to the 256K boundary. Functional address lines allow up to eight devices on the same bus, for up to 2 Mbit address space. This device is available in the standard 8-pin plastic DIP, SOIC, TSSOP, MSOP, DFN and 14-lead TSSOP packages.

Block Diagram



Package Types



Note: * Pins A0 and A1 are no connects for the MSOP package only.

*24XX256 is used in this document as a generic part number for the 24AA256/24LC256/24FC256 devices.

1.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings(†)

VCC	6.5V
All inputs and outputs w.r.t. VSS	-0.6V to VCC +1.0V
Storage temperature	-65°C to +150°C
Ambient temperature with power applied	-40°C to +125°C
ESD protection on all pins	≥ 4 kV

† NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational listings of this specification is not implied. Exposure to Absolute Maximum Rating conditions for extended periods may affect device reliability.

TABLE 1-1: DC CHARACTERISTICS

DC CHARACTERISTICS			**Electrical Characteristics:** Industrial (I): VCC = +1.8V to 5.5V TA = -40°C to +85°C Automotive (E): VCC = +2.5V to 5.5V TA = -40°C to +125°C			
Param. No.	**Sym**	**Characteristic**	**Min**	**Max**	**Units**	**Conditions**
D1	—	A0, A1, A2, SCL, SDA and WP pins:	—	—	—	—
D2	VIH	High-level input voltage	0.7 VCC	—	V	—
D3	VIL	Low-level input voltage	—	0.3 VCC 0.2 VCC	V V	VCC ≥ 2.5V VCC < 2.5V
D4	VHYS	Hysteresis of Schmitt Trigger inputs (SDA, SCL pins)	0.05 VCC	—	V	VCC ≥ 2.5V **(Note)**
D5	VOL	Low-level output voltage	—	0.40	V	IOL = 3.0 ma @ VCC = 4.5V IOL = 2.1 ma @ VCC = 2.5V
D6	ILI	Input leakage current	—	±1	µA	VIN = VSS or VCC, WP = VSS VIN = VSS or VCC, WP = VCC
D7	ILO	Output leakage current	—	±1	µA	VOUT = VSS or VCC
D8	CIN, COUT	Pin capacitance (all inputs/outputs)	—	10	pF	VCC = 5.0V **(Note)** TA = 25°C, fc = 1 MHz
D9	ICC Read	Operating current	—	400	µA	VCC = 5.5V, SCL = 400 kHz
	ICC Write		—	3	mA	VCC = 5.5V
D10	ICCS	Standby current	— —	1 5	µA µA	TA = -40°C to +85°C SCL = SDA = VCC = 5.5V A0, A1, A2, WP = VSS TA = -40°C to +125°C SCL = SDA = VCC = 5.5V A0, A1, A2, WP = VSS

Note: This parameter is periodically sampled and not 100% tested.

TABLE 1-2: AC CHARACTERISTICS

AC CHARACTERISTICS			Electrical Characteristics: Industrial (I): VCC = +1.8V to 5.5V TA = -40°C to +85°C Automotive (E): VCC = +2.5V to 5.5V TA = -40°C to +125°C			
Param. No.	**Sym**	**Characteristic**	**Min**	**Max**	**Units**	**Conditions**
1	FCLK	Clock frequency	—	100	kHz	1.8V ≤ Vcc < 2.5V
			—	400		2.5V ≤ Vcc ≤ 5.5V
			—	400		1.8V ≤ Vcc < 2.5V 24FC256
			—	1000		2.5V ≤ Vcc ≤ 5.5V 24FC256
2	THIGH	Clock high time	4000	—	ns	1.8V ≤ Vcc < 2.5V
			600	—		2.5V ≤ Vcc ≤ 5.5V
			600	—		1.8V ≤ Vcc < 2.5V 24FC256
			500	—		2.5V ≤ Vcc ≤ 5.5V 24FC256
3	TLOW	Clock low time	4700	—	ns	1.8V ≤ Vcc < 2.5V
			1300	—		2.5V ≤ Vcc ≤ 5.5V
			1300	—		1.8V ≤ Vcc < 2.5V 24FC256
			500	—		2.5V ≤ Vcc ≤ 5.5V 24FC256
4	TR	SDA and SCL rise time	—	1000	ns	1.8V ≤ Vcc < 2.5V
		(Note 1)	—	300		2.5V ≤ Vcc ≤ 5.5V
			—	300		1.8V ≤ Vcc ≤ 5.5V 24FC256
5	TF	SDA and SCL fall time	—	300	ns	All except, 24FC256
		(Note 1)	—	100		1.8V ≤ Vcc ≤ 5.5V 24FC256
6	THD:STA	Start condition hold time	4000	—	ns	1.8V ≤ Vcc < 2.5V
			600	—		2.5V ≤ Vcc ≤ 5.5V
			600	—		1.8V ≤ Vcc < 2.5V 24FC256
			250	—		2.5V ≤ Vcc ≤ 5.5V 24FC256
7	TSU:STA	Start condition setup time	4700	—	ns	1.8V ≤ Vcc < 2.5V
			600	—		2.5V ≤ Vcc ≤ 5.5V
			600	—		1.8V ≤ Vcc < 2.5V 24FC256
			250	—		2.5V ≤ Vcc ≤ 5.5V 24FC256
8	THD:DAT	Data input hold time	0	—	ns	**(Note 2)**
9	TSU:DAT	Data input setup time	250	—	ns	1.8V ≤ Vcc < 2.5V
			100	—		2.5V ≤ Vcc ≤ 5.5V
			100	—		1.8V ≤ Vcc ≤ 5.5V 24FC256
10	TSU:STO	Stop condition setup time	4000	—	ns	1.8V ≤ Vcc < 2.5V
			600	—		2.5V ≤ Vcc ≤ 5.5V
			600	—		1.8V ≤ Vcc < 2.5V 24FC256
			250	—		2.5V ≤ Vcc ≤ 5.5V 24FC256
11	TSU:WP	WP setup time	4000	—	ns	1.8V ≤ Vcc < 2.5V
			600	—		2.5V ≤ Vcc ≤ 5.5V
			600	—		1.8V ≤ Vcc ≤ 5.5V 24FC256
12	THD:WP	WP hold time	4700	—	ns	1.8V ≤ Vcc < 2.5V
			1300	—		2.5V ≤ Vcc ≤ 5.5V
			1300	—		1.8V ≤ Vcc ≤ 5.5V 24FC256

Note 1: Not 100% tested. CB = total capacitance of one bus line in pF.

2: As a transmitter, the device must provide an internal minimum delay time to bridge the undefined region (minimum 300 ns) of the falling edge of SCL to avoid unintended generation of Start or Stop conditions.

3: The combined TSP and VHYS specifications are due to new Schmitt Trigger inputs, which provide improved noise spike suppression. This eliminates the need for a TI specification for standard operation.

4: This parameter is not tested but ensured by characterization. For endurance estimates in a specific application, please consult the Total Endurance™ Model, which can be obtained from Microchip's web site: www.microchip.com.

AC CHARACTERISTICS (Continued)			Electrical Characteristics: Industrial (I): VCC = +1.8V to 5.5V TA = -40°C to +85°C Automotive (E): VCC = +2.5V to 5.5V TA = -40°C to +125°C			
Param. No.	Sym	Characteristic	Min	Max	Units	Conditions
13	TAA	Output valid from clock **(Note 2)**	— — — —	3500 900 900 400	ns	1.8 V ≤ VCC < 2.5V 2.5 V ≤ VCC ≤ 5.5V 1.8V ≤ VCC < 2.5V 24FC256 2.5 V ≤ VCC ≤ 5.5V 24FC256
14	TBUF	Bus free time: Time the bus must be free before a new transmission can start	4700 1300 1300 500	— — — —	ns	1.8V ≤ VCC < 2.5V 2.5V ≤ VCC ≤ 5.5V 1.8V ≤ VCC < 2.5V 24FC256 2.5V ≤ VCC ≤ 5.5V 24FC256
15	TOF	Output fall time from VIH minimum to VIL maximum CB ≤ 100 pF	10 + 0.1CB	250 250	ns	All except, 24FC256 **(Note 1)**
16	TSP	Input filter spike suppression (SDA and SCL pins)	—	50	ns	All except, 24FC256 **(Notes 1 and 3)**
17	TWC	Write cycle time (byte or page)	—	5	ms	—
18	—	Endurance	1,000,000	—	cycles	25°C **(Note 4)**

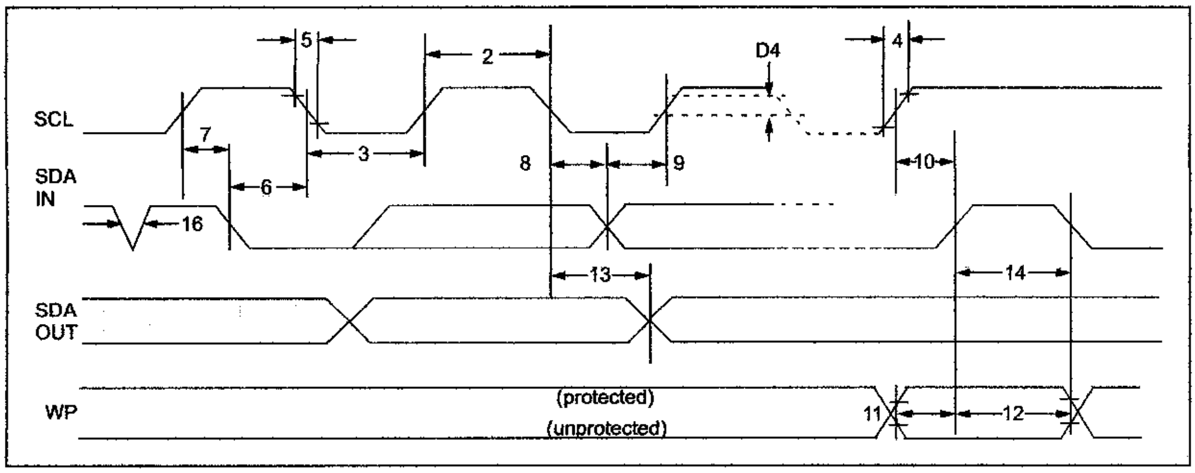
Note 1: Not 100% tested. CB = total capacitance of one bus line in pF.

2: As a transmitter, the device must provide an internal minimum delay time to bridge the undefined region (minimum 300 ns) of the falling edge of SCL to avoid unintended generation of Start or Stop conditions.

3: The combined TSP and VHYS specifications are due to new Schmitt Trigger inputs, which provide improved noise spike suppression. This eliminates the need for a TI specification for standard operation.

4: *This parameter is not tested but ensured by characterization. For endurance estimates in a specific application, please consult the Total Endurance™ Model, which can be obtained from Microchip's web site: www.microchip.com.*

FIGURE 1-1: BUS TIMING DATA



2.0 PIN DESCRIPTIONS

The descriptions of the pins are listed in Table 2-1.

TABLE 2-1: PIN FUNCTION TABLE

Name	8-pin PDIP	8-pin SOIC	8-pin TSSOP	14-pin TSSOP	8-pin MSOP	8-pin DFN	Function
A0	1	1	1	1	—	1	User Configurable Chip Select
A1	2	2	2	2	—	2	User Configurable Chip Select
(NC)	—	—	—	3, 4, 5	1, 2	—	Not Connected
A2	3	3	3	6	3	3	User Configurable Chip Select
Vss	4	4	4	7	4	4	Ground
SDA	5	5	5	8	5	5	Serial Data
SCL	6	6	6	9	6	6	Serial Clock
(NC)	—	—	—	10, 11, 12	—	—	Not Connected
WP	7	7	7	13	7	7	Write-Protect Input
Vcc	8	8	8	14	8	8	+1.8V to 5.5V (24AA256) +2.5V to 5.5V (24LC256) +1.8V to 5.5V (24FC256)

2.1 A0, A1, A2 Chip Address Inputs

The A0, A1 and A2 inputs are used by the 24XX256 for multiple device operations. The levels on these inputs are compared with the corresponding bits in the slave address. The chip is selected if the compare is true.

For the MSOP package only, pins A0 and A1 are not connected.

Up to eight devices (two for the MSOP package) may be connected to the same bus by using different Chip Select bit combinations. If these pins are left unconnected, the inputs will be pulled down internally to Vss. If they are tied to Vcc or driven high, the internal pull-down circuitry is disabled.

In most applications, the chip address inputs A0, A1 and A2 are hard-wired to logic '0' or logic '1'. For applications in which these pins are controlled by a microcontroller or other programmable device, the chip address pins must be driven to logic '0' or logic '1' before normal device operation can proceed.

2.2 Serial Data (SDA)

This is a bidirectional pin used to transfer addresses and data into and out of the device. It is an open drain terminal. Therefore, the SDA bus requires a pull-up resistor to Vcc (typical 10 kΩ for 100 kHz, 2 kΩ for 400 kHz and 1 MHz).

For normal data transfer, SDA is allowed to change only during SCL low. Changes during SCL high are reserved for indicating the Start and Stop conditions.

2.3 Serial Clock (SCL)

This input is used to synchronize the data transfer to and from the device.

2.4 Write-Protect (WP)

This pin can be connected to either Vss, Vcc or left floating. Internal pull-down circuitry on this pin will keep the device in the unprotected state if left floating. If tied to Vss or left floating, normal memory operation is enabled (read/write the entire memory 0000-7FFF).

If tied to Vcc, write operations are inhibited. Read operations are not affected.

3.0 FUNCTIONAL DESCRIPTION

The 24XX256 supports a bidirectional 2-wire bus and data transmission protocol. A device that sends data onto the bus is defined as a transmitter and a device receiving data as a receiver. The bus must be controlled by a master device which generates the serial clock (SCL), controls the bus access, and generates the Start and Stop conditions while the 24XX256 works as a slave. Both master and slave can operate as a transmitter or receiver, but the master device determines which mode is activated.

4.0 BUS CHARACTERISTICS

The following **bus protocol** has been defined:

- Data transfer may be initiated only when the bus is not busy.
- During data transfer, the data line must remain stable whenever the clock line is high. Changes in the data line, while the clock line is high, will be interpreted as a Start or Stop condition.

Accordingly, the following bus conditions have been defined (Figure 4-1).

4.1 Bus not Busy (A)

Both data and clock lines remain high.

4.2 Start Data Transfer (B)

A high-to-low transition of the SDA line while the clock (SCL) is high, determines a Start condition. All commands must be preceded by a Start condition.

4.3 Stop Data Transfer (C)

A low-to-high transition of the SDA line, while the clock (SCL) is high, determines a Stop condition. All operations must end with a Stop condition.

4.4 Data Valid (D)

The state of the data line represents valid data when, after a Start condition, the data line is stable for the duration of the high period of the clock signal.

The data on the line must be changed during the low period of the clock signal. There is one bit of data per clock pulse.

Each data transfer is initiated with a Start condition and terminated with a Stop condition. The number of the data bytes transferred between the Start and Stop conditions is determined by the master device.

4.5 Acknowledge

Each receiving device, when addressed, is obliged to generate an Acknowledge signal after the reception of each byte. The master device must generate an extra clock pulse which is associated with this Acknowledge bit.

Note:	The 24XX256 does not generate any Acknowledge bits if an internal programming cycle is in progress.

A device that acknowledges must pull down the SDA line during the acknowledge clock pulse in such a way that the SDA line is stable low during the high period of the acknowledge related clock pulse. Of course, setup and hold times must be taken into account. During reads, a master must signal an end of data to the slave by NOT generating an Acknowledge bit on the last byte that has been clocked out of the slave. In this case, the slave (24XX256) will leave the data line high to enable the master to generate the Stop condition.

FIGURE 4-1: DATA TRANSFER SEQUENCE ON THE SERIAL BUS

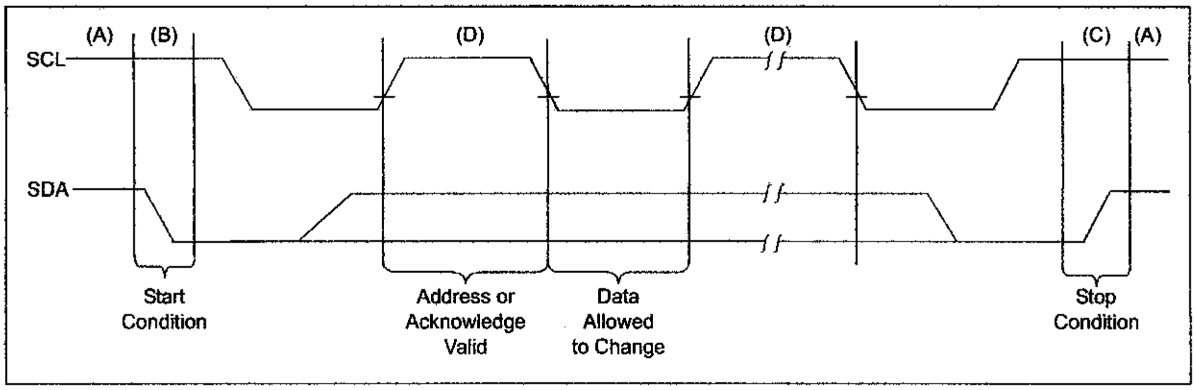
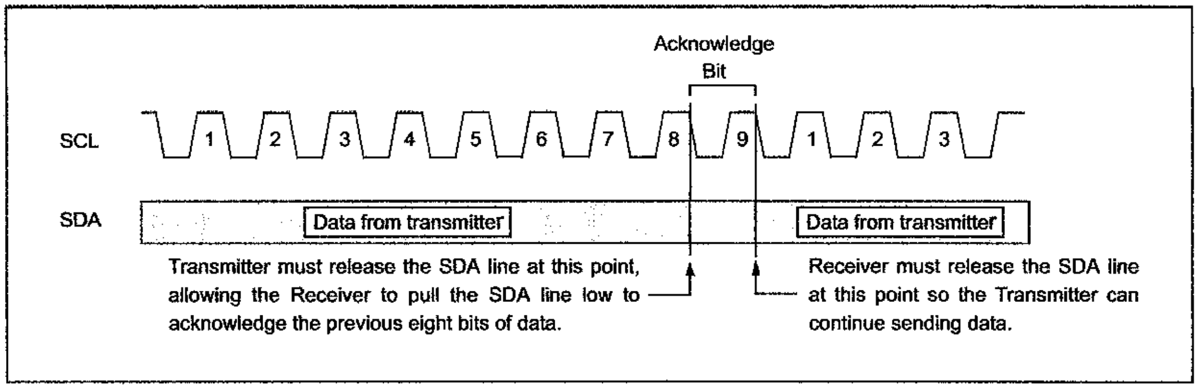


FIGURE 4-2: ACKNOWLEDGE TIMING


Acknowledge Bit
SCL
1
2
3
4
5
6
7
8
9
1
2
3
SDA
Data from transmitter
Data from transmitter
Transmitter must release the SDA line at this point, allowing the Receiver to pull the SDA line low to acknowledge the previous eight bits of data.
Receiver must release the SDA line at this point so the Transmitter can continue sending data.

5.0 DEVICE ADDRESSING

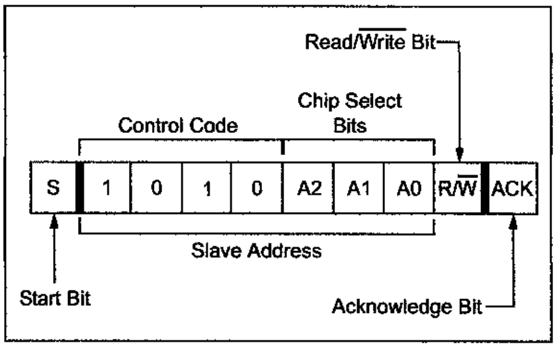
A control byte is the first byte received following the Start condition from the master device (Figure 5-1). The control byte consists of a 4-bit control code. For the 24XX256, this is set as '`1010`' binary for read and write operations. The next three bits of the control byte are the Chip Select bits (A2, A1, A0). The Chip Select bits allow the use of up to eight 24XX256 devices on the same bus and are used to select which device is accessed. The Chip Select bits in the control byte must correspond to the logic levels on the corresponding A2, A1 and A0 pins for the device to respond. These bits are, in effect, the three Most Significant bits of the word address.

For the MSOP package, the A0 and A1 pins are not connected. During device addressing, the A0 and A1 Chip Select bits (Figures 5-1 and 5-2) should be set to '`0`'. Only two 24XX256 MSOP packages can be connected to the same bus.

The last bit of the control byte defines the operation to be performed. When set to a one, a read operation is selected. When set to a zero, a write operation is selected. The next two bytes received define the address of the first data byte (Figure 5-2). Because only A14...A0 are used, the upper address bits are a "don't care." The upper address bits are transferred first, followed by the less significant bits.

Following the Start condition, the 24XX256 monitors the SDA bus checking the device type identifier being transmitted. Upon receiving a '`1010`' code and appropriate device select bits, the slave device outputs an Acknowledge signal on the SDA line. Depending on the state of the R/$\overline{W}$ bit, the 24XX256 will select a read or write operation.

FIGURE 5-1: CONTROL BYTE FORMAT

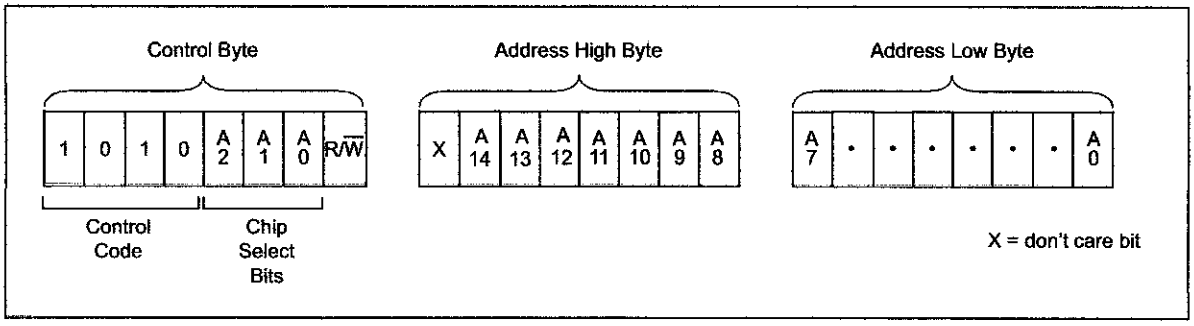


5.1 Contiguous Addressing Across Multiple Devices

The Chip Select bits A2, A1 and A0 can be used to expand the contiguous address space for up to 2 Mbit by adding up to eight 24XX256s on the same bus. In this case, software can use A0 of the **control byte** as address bit A15; A1 as address bit A16; and A2 as address bit A17. It is not possible to sequentially read across device boundaries.

For the MSOP package, up to two 24XX256 devices can be added for up to 512 Kbit of address space. In this case, software can use A2 of the control byte as address bit A17. Bits A0 (A15) and A1 (A16) of the **control byte** must always be set to a logic '`0`' for the MSOP.

FIGURE 5-2: ADDRESS SEQUENCE BIT ASSIGNMENTS



6.0 WRITE OPERATIONS

6.1 Byte Write

Following the Start condition from the master, the control code (four bits), the Chip Select (three bits) and the R/W̅ bit (which is a logic low) are clocked onto the bus by the master transmitter. This indicates to the addressed slave receiver that the address high byte will follow after it has generated an Acknowledge bit during the ninth clock cycle. Therefore, the next byte transmitted by the master is the high-order byte of the word address and will be written into the address pointer of the 24XX256. The next byte is the Least Significant Address Byte. After receiving another Acknowledge signal from the 24XX256, the master device will transmit the data word to be written into the addressed memory location. The 24XX256 acknowledges again and the master generates a Stop condition. This initiates the internal write cycle and during this time, the 24XX256 will not generate Acknowledge signals (Figure 6-1). If an attempt is made to write to the array with the WP pin held high, the device will acknowledge the command but no write cycle will occur, no data will be written, and the device will immediately accept a new command. After a byte Write command, the internal address counter will point to the address location following the one that was just written.

6.2 Page Write

The write control byte, word address and the first data byte are transmitted to the 24XX256 in much the same way as in a byte write. The exception is that instead of generating a Stop condition, the master transmits up to 63 additional bytes, which are temporarily stored in the on-chip page buffer, and will be written into memory once the master has transmitted a Stop condition. Upon receipt of each word, the six lower address pointer bits are internally incremented by one. If the master should transmit more than 64 bytes prior to generating the Stop condition, the address counter will roll over and the previously received data will be overwritten. As with the byte write operation, once the Stop condition is received, an internal write cycle will begin (Figure 6-2). If an attempt is made to write to the array with the WP pin held high, the device will acknowledge the command, but no write cycle will occur, no data will be written and the device will immediately accept a new command.

6.3 Write-Protection

The WP pin allows the user to write-protect the entire array (0000-7FFF) when the pin is tied to VCC. If tied to VSS or left floating, the write protection is disabled. The WP pin is sampled at the Stop bit for every Write command (Figure 1-1). Toggling the WP pin after the Stop bit will have no effect on the execution of the write cycle.

Note: Page write operations are limited to writing bytes within a single physical page, **regardless** of the number of bytes actually being written. Physical page boundaries start at addresses that are integer multiples of the page buffer size (or 'page size') and end at addresses that are integer multiples of [page size - 1]. If a Page Write command attempts to write across a physical page boundary, the result is that the data wraps around to the beginning of the current page (overwriting data previously stored there), instead of being written to the next page, as might be expected. It is, therefore, necessary for the application software to prevent page write operations that would attempt to cross a page boundary.

FIGURE 6-1: BYTE WRITE

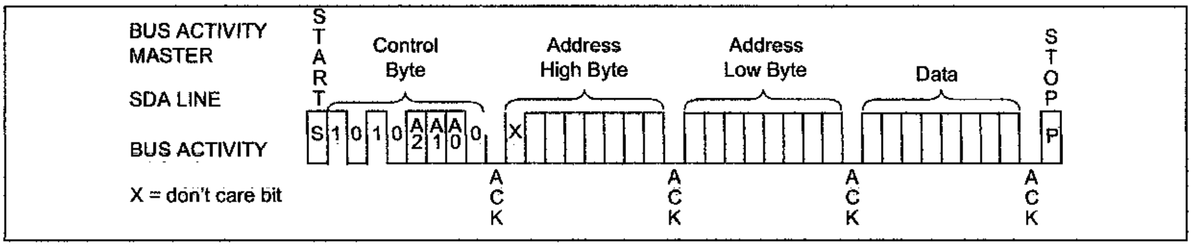
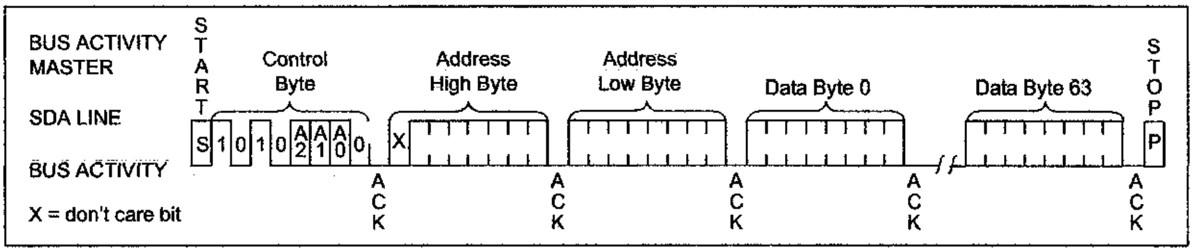


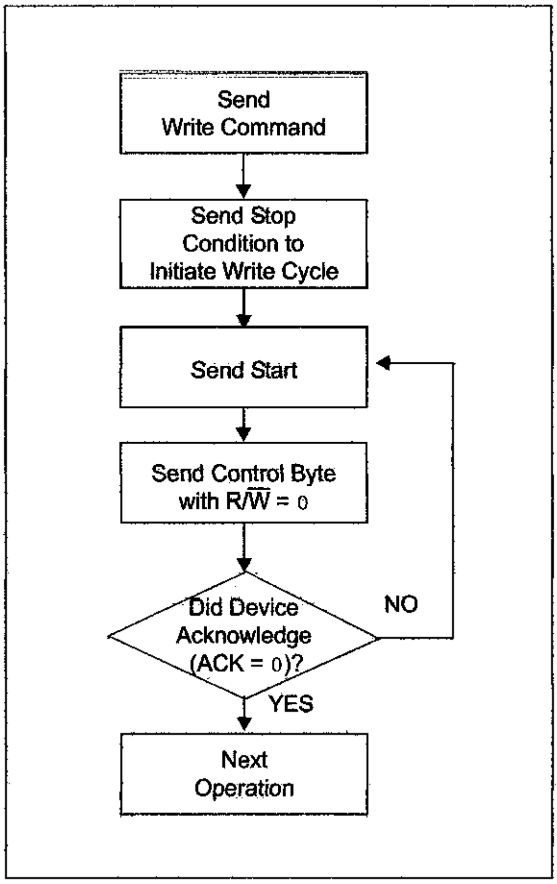
FIGURE 6-2: PAGE WRITE



7.0 ACKNOWLEDGE POLLING

Since the device will not acknowledge during a write cycle, this can be used to determine when the cycle is complete (This feature can be used to maximize bus throughput). Once the Stop condition for a Write command has been issued from the master, the device initiates the internally timed write cycle. ACK polling can be initiated immediately. This involves the master sending a Start condition, followed by the control byte for a Write command ($R/\overline{W}$ = 0). If the device is still busy with the write cycle, then no ACK will be returned. If no ACK is returned, the Start bit and control byte must be resent. If the cycle is complete, then the device will return the ACK and the master can then proceed with the next Read or Write command. See Figure 7-1 for flow diagram.

FIGURE 7-1: ACKNOWLEDGE POLLING FLOW


Send Write Command
Send Stop Condition to Initiate Write Cycle
Send Start
Send Control Byte with R/W = 0
Did Device Acknowledge (ACK = 0)?
NO
YES
Next Operation

8.0 READ OPERATION

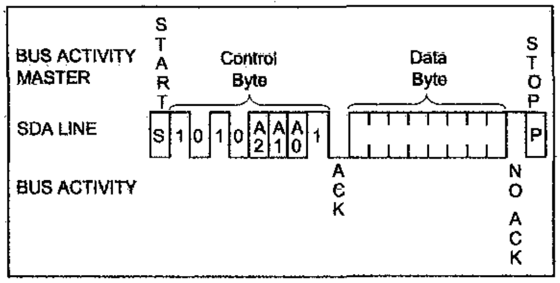
Read operations are initiated in much the same way as write operations, with the exception that the R/W̄ bit of the control byte is set to '1'. There are three basic types of read operations: current address read, random read and sequential read.

8.1 Current Address Read

The 24XX256 contains an address counter that maintains the address of the last word accessed, internally incremented by '1'. Therefore, if the previous read access was to address n (n is any legal address), the next current address read operation would access data from address n + 1.

Upon receipt of the control byte with R/W̄ bit set to '1', the 24XX256 issues an acknowledge and transmits the 8-bit data word. The master will not acknowledge the transfer, but does generate a Stop condition and the 24XX256 discontinues transmission (Figure 8-1).

FIGURE 8-1: CURRENT ADDRESS READ



8.2 Random Read

Random read operations allow the master to access any memory location in a random manner. To perform this type of read operation, the word address must first be set. This is done by sending the word address to the 24XX256 as part of a write operation (R/W̄ bit set to '0'). Once the word address is sent, the master generates a Start condition following the acknowledge. This terminates the write operation, but not before the internal address pointer is set. The master then issues the control byte again, but with the R/W̄ bit set to a one. The 24XX256 will then issue an acknowledge and transmit the 8-bit data word. The master will not acknowledge the transfer, though it does generate a Stop condition, which causes the 24XX256 to discontinue transmission (Figure 8-2). After a random Read command, the internal address counter will point to the address location following the one that was just read.

8.3 Sequential Read

Sequential reads are initiated in the same way as a random read except that after the 24XX256 transmits the first data byte, the master issues an acknowledge as opposed to the Stop condition used in a random read. This acknowledge directs the 24XX256 to transmit the next sequentially addressed 8-bit word (Figure 8-3). Following the final byte transmitted to the master, the master will NOT generate an acknowledge, but will generate a Stop condition. To provide sequential reads, the 24XX256 contains an internal address pointer which is incremented by one at the completion of each operation. This address pointer allows the entire memory contents to be serially read during one operation. The internal address pointer will automatically roll over from address 7FFF to address 0000 if the master acknowledges the byte received from the array address 7FFF.

FIGURE 8-2: RANDOM READ

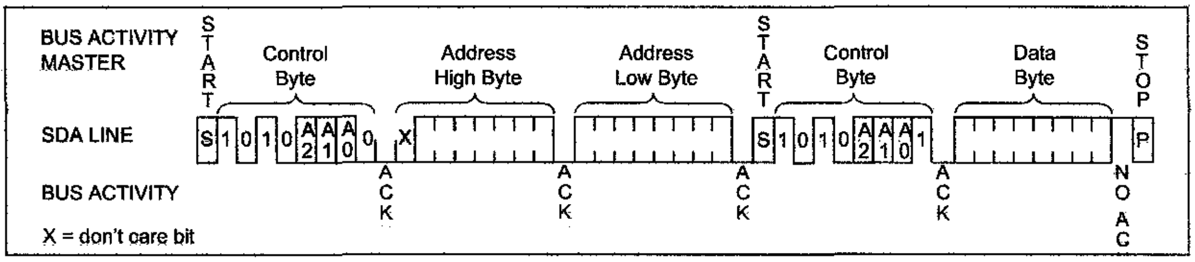
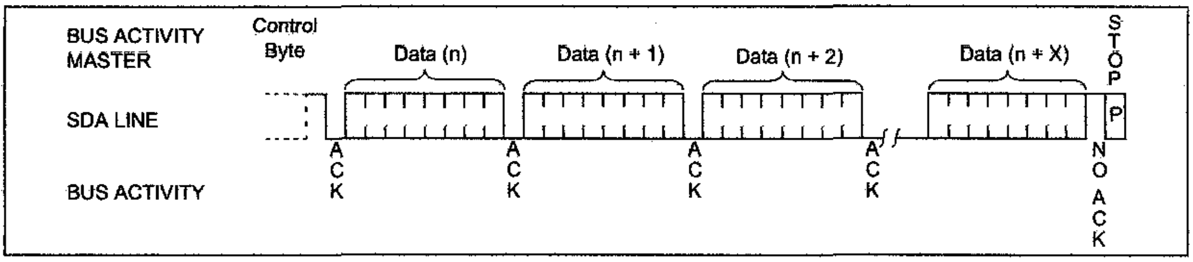
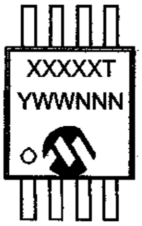


FIGURE 8-3: SEQUENTIAL READ

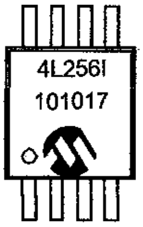


Package Marking Information (Continued)

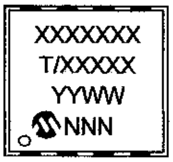
8-Lead MSOP



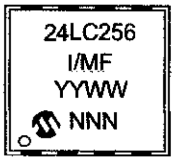
Example:



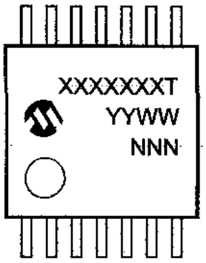
8-Lead DFN-S



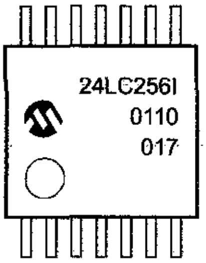
Example:



14-Lead TSSOP

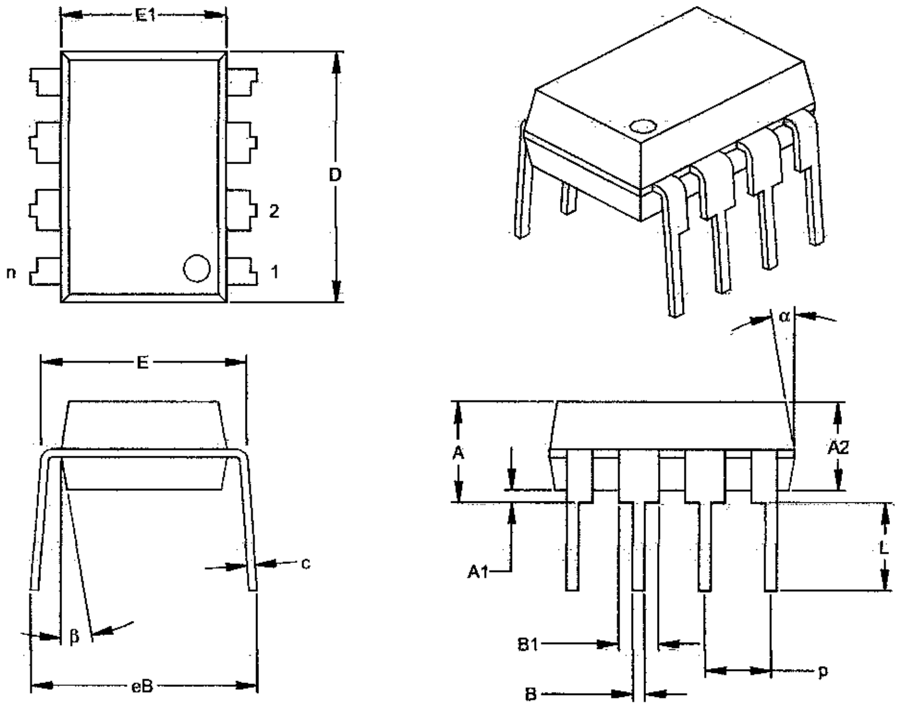


Example:



	TSSOP Package Codes		MSOP Package Codes	
Part No.	**STD**	**Pb-free**	**STD**	**Pb-free**
24AA256	4AD	G4AD	4A256	G4AD
24LC256	4LD	G4LD	4L256	G4LD
24FC256	4FD	G4FD	4F256	G4FD

8-Lead Plastic Dual In-line (P) – 300 mil (PDIP)



Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		8			8	
Pitch	p		.100			2.54	
Top to Seating Plane	A	.140	.155	.170	3.56	3.94	4.32
Molded Package Thickness	A2	.115	.130	.145	2.92	3.30	3.68
Base to Seating Plane	A1	.015			0.38		
Shoulder to Shoulder Width	E	.300	.313	.325	7.62	7.94	8.26
Molded Package Width	E1	.240	.250	.260	6.10	6.35	6.60
Overall Length	D	.360	.373	.385	9.14	9.46	9.78
Tip to Seating Plane	L	.125	.130	.135	3.18	3.30	3.43
Lead Thickness	c	.008	.012	.015	0.20	0.29	0.38
Upper Lead Width	B1	.045	.058	.070	1.14	1.46	1.78
Lower Lead Width	B	.014	.018	.022	0.36	0.46	0.56
Overall Row Spacing §	eB	.310	.370	.430	7.87	9.40	10.92
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	15

* Controlling Parameter

§ Significant Characteristic

Notes:

Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" (0.254mm) per side.

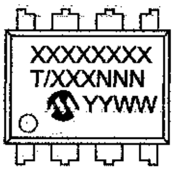
JEDEC Equivalent: MS-001

Drawing No. C04-018

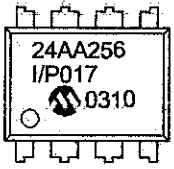
9.0 PACKAGING INFORMATION

9.1 Package Marking Information

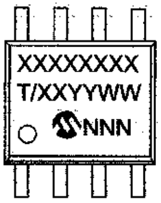
8-Lead PDIP (300 mil)



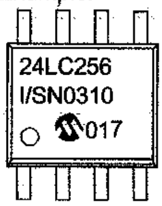
Example:



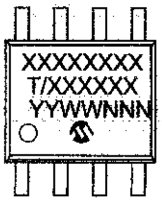
8-Lead SOIC (150 mil)



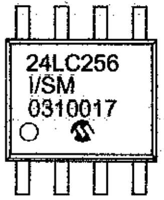
Example:



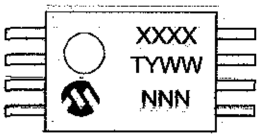
8-Lead SOIC (208 mil)



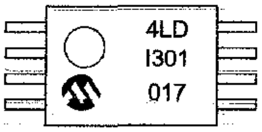
Example:



8-Lead TSSOP



Example:



Legend:	XX...X	Customer specific information*
	T	Temperature grade (I, E)
	Y	Year code (last digit of calendar year)
	YY	Year code (last 2 digits of calendar year)
	WW	Week code (week of January 1 is week '01')
	NNN	Alphanumeric traceability code

Note: In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line thus limiting the number of available characters for customer specific information.

*Standard device marking consists of Microchip part number, year code, week code, and traceability code. For device marking beyond this, certain price adders apply. Please check with your Microchip Sales Office.

RX433

The RF module's frequency is from UHF ASK 300MHz to 434MHz

High sensitivity passive design
4800 B/S baseboard data rate
simple to apply with low external parts count
Low supply voltage Vcc = 5VDC
ASK Data Shaping Comparator Included

DC Characteristics

	Parameter	Conditions	Min	Typ	Max	Unit
Vcc	Operating Supply Voltage		4,9	5	5,1	V
I tot	Operating Supply Voltage		-	4,5	-	mA
V data	Data Out	I Data = -200 µAi	Vcc - 0,5		Vcc	V
		I Data = -10 µA (low)	-		0,3	V

Electrical Characteristics

Characteristics	SYM	Min	Typ	Max	Unit
Operating Radio Frequency	Fc	300-434			MHz
Sensitivity	Pref.			-108	dBm
Channel Width		=+/- 500			kHz
Noise Equivalent BW	NEB.		5	4	kHz
Baseboard data rate				3	kb/s
Receiver turn on time				5	ms

TX433

The RF module's frequency is from UHF ASK 300MHz to 434MHz
The RF module's frequency depends on the quartz surface acoustic wave (saw) of different frequency . The following space is suitable for any frequency.

Electrical Characteristics

Symbol	Characteristics	Conditions	Min	Typ	Max	Unit
Vcc	Operating Supply Voltage		1,5		12	V
Icc	Peak Current			5	9	mA
Vih	Input Low Voltage	I Data = 100µA (High)	Vcc-0,5		Vcc	V
Vil	Absolute Frequency	I Data = 0µA (Low)			0,3	V
Fo	Relative to 433.92MHz		314,8	315	315,2	MHz
Dfo	RF Out power into 500ohm			=+/-150	=+/-200	kHz
Po	Modulation bandwidth	External Cording	-3	0	2	dBm
				5		kHz
Tr	Modulation Rise tme				100	µs
Tf	Modulation Fall time				100	µs

Notes : (case temperature = /25°C, test load impedance = 50ohm
and modulation input is at logic high Low unless noted outerwise)
The RF module not included digital encoder.

Caution : Electrostatic sensitive device. Observe precautions for handling.

HD44780U (LCD-II)

(Dot Matrix Liquid Crystal Display Controller/Driver)

HITACHI

ADE-207-272(Z)
'99.9
Rev. 0.0

Description

The HD44780U dot-matrix liquid crystal display controller and driver LSI displays alphanumerics, Japanese kana characters, and symbols. It can be configured to drive a dot-matrix liquid crystal display under the control of a 4- or 8-bit microprocessor. Since all the functions such as display RAM, character generator, and liquid crystal driver, required for driving a dot-matrix liquid crystal display are internally provided on one chip, a minimal system can be interfaced with this controller/driver.

A single HD44780U can display up to one 8-character line or two 8-character lines.

The HD44780U has pin function compatibility with the HD44780S which allows the user to easily replace an LCD-II with an HD44780U. The HD44780U character generator ROM is extended to generate 208 5 × 8 dot character fonts and 32 5 × 10 dot character fonts for a total of 240 different character fonts.

The low power supply (2.7V to 5.5V) of the HD44780U is suitable for any portable battery-driven product requiring low power dissipation.

Features

- 5 × 8 and 5 × 10 dot matrix possible
- Low power operation support:
  — 2.7 to 5.5V
- Wide range of liquid crystal display driver power
  — 3.0 to 11V
- Liquid crystal drive waveform
  — A (One line frequency AC waveform)
- Correspond to high speed MPU bus interface
  — 2 MHz (when V_{CC} = 5V)
- 4-bit or 8-bit MPU interface enabled
- 80 × 8-bit display RAM (80 characters max.)
- 9,920-bit character generator ROM for a total of 240 character fonts
  — 208 character fonts (5 × 8 dot)
  — 32 character fonts (5 × 10 dot)

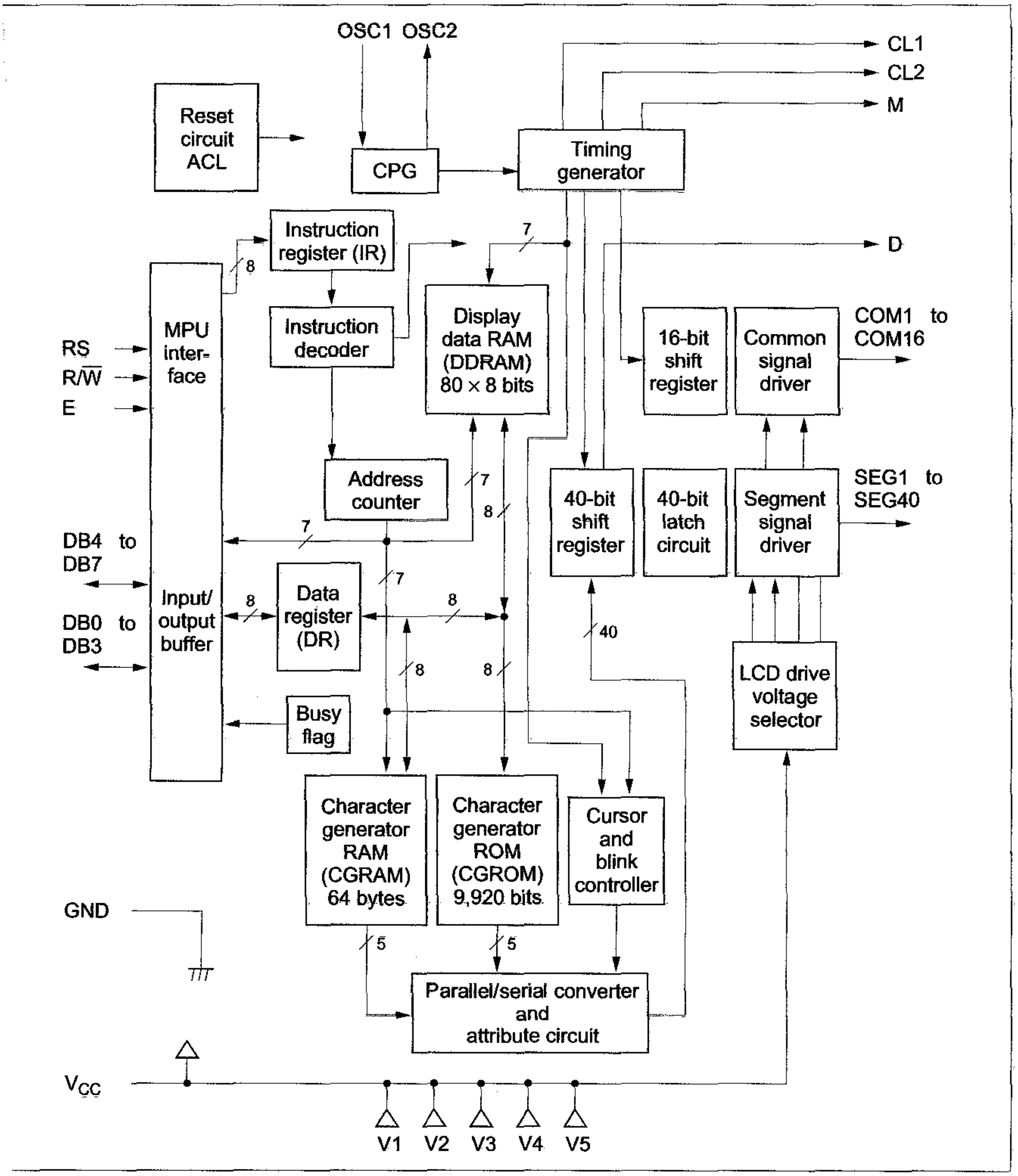
- 64 × 8-bit character generator RAM
  - — 8 character fonts (5 × 8 dot)
  - — 4 character fonts (5 × 10 dot)
- 16-common × 40-segment liquid crystal display driver
- Programmable duty cycles
  - — 1/8 for one line of 5 × 8 dots with cursor
  - — 1/11 for one line of 5 × 10 dots with cursor
  - — 1/16 for two lines of 5 × 8 dots with cursor
- Wide range of instruction functions:
  - — Display clear, cursor home, display on/off, cursor on/off, display character blink, cursor shift, display shift
- Pin function compatibility with HD44780S
- Automatic reset circuit that initializes the controller/driver after power on
- Internal oscillator with external resistors
- Low power consumption

Ordering Information

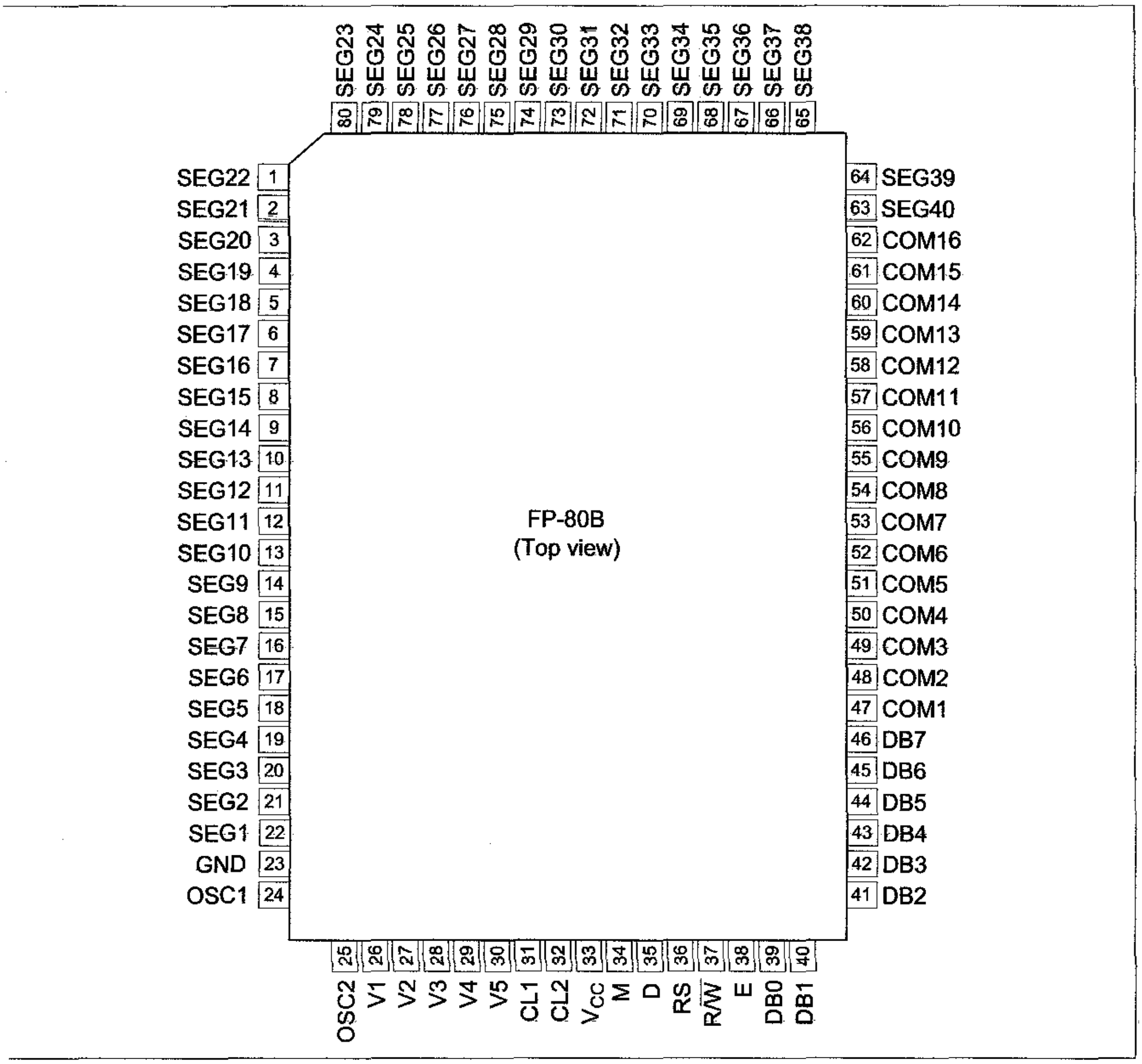
Type No.	Package	CGROM
HD44780UA00FS	FP-80B	Japanese standard font
HCD44780UA00	Chip	
HD44780UA00TF	TFP-80F	
HD44780UA02FS	FP-80B	European standard font
HCD44780UA02	Chip	
HD44780UA02TF	TFP-80F	
HD44780UBxxFS	FP-80B	Custom font
HCD44780UBxx	Chip	
HD44780UBxxTF	TFP-80F	

Note: xx: ROM code No.

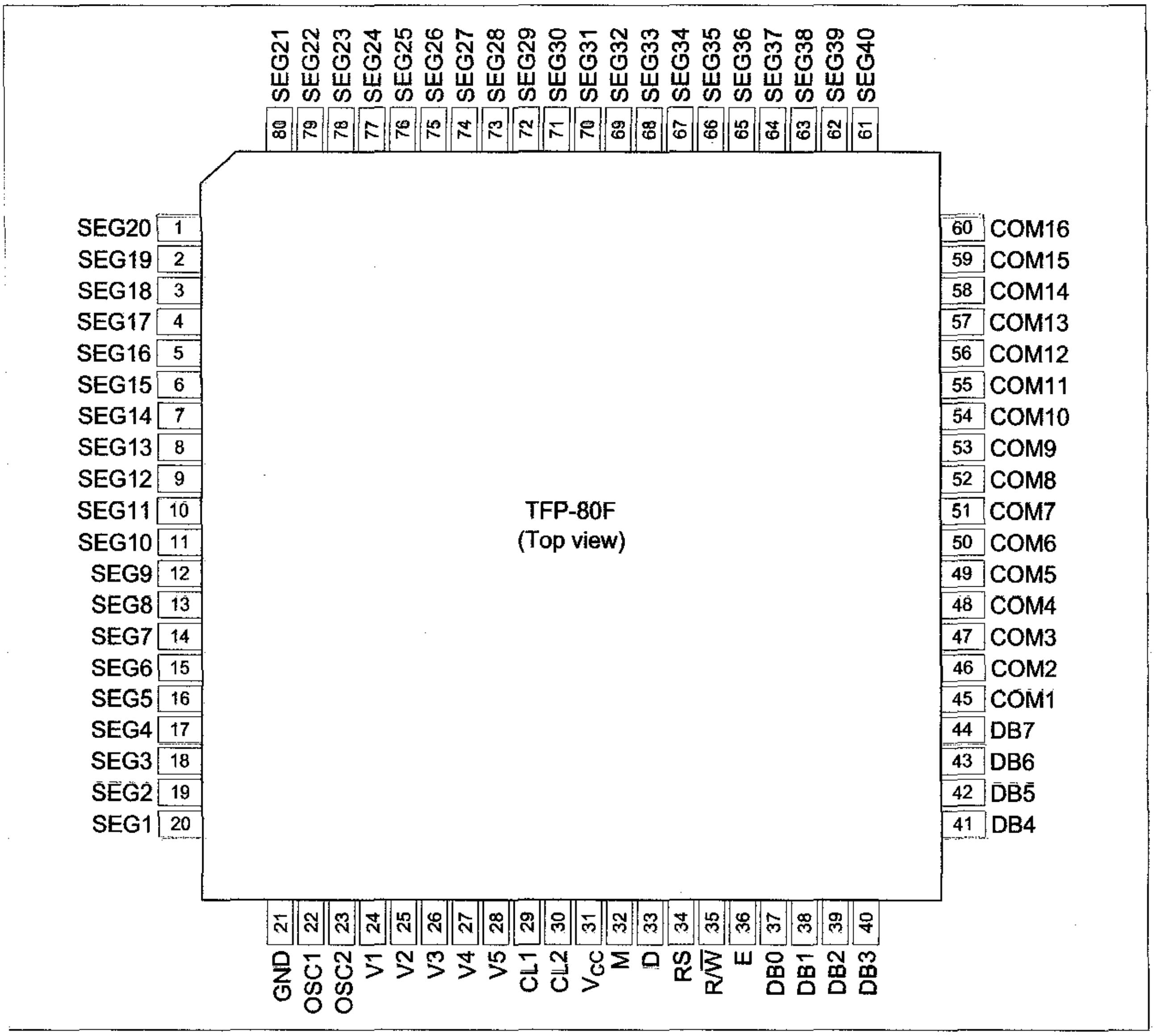
HD44780U Block Diagram



HD44780U Pin Arrangement (FP-80B)

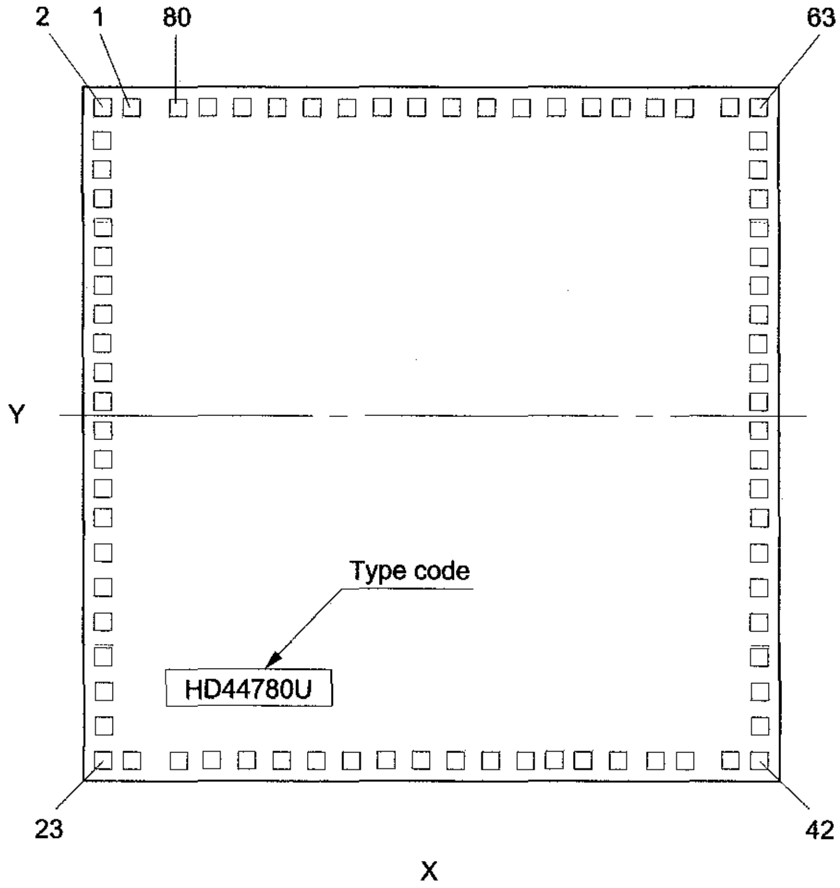


HD44780U Pin Arrangement (TFP-80F)



HD44780U Pad Arrangement

Chip size:	4.90 × 4.90 mm^2
Coordinate:	Pad center (µm)
Origin:	Chip center
Pad size:	114 × 114 µm^2



HCD44780U Pad Location Coordinates

		Coordinate				Coordinate	
Pad No.	Function	X (um)	Y (um)	Pad No.	Function	X (um)	Y (um)
1	SEG22	–2100	2313	41	DB2	2070	–2290
2	SEG21	–2280	2313	42	DB3	2260	–2290
3	SEG20	–2313	2089	43	DB4	2290	–2099
4	SEG19	–2313	1833	44	DB5	2290	–1883
5	SEG18	–2313	1617	45	DB6	2290	–1667
6	SEG17	–2313	1401	46	DB7	2290	–1452
7	SEG16	–2313	1186	47	COM1	2313	–1186
8	SEG15	–2313	970	48	COM2	2313	–970
9	SEG14	–2313	755	49	COM3	2313	–755
10	SEG13	–2313	539	50	COM4	2313	–539
11	SEG12	–2313	323	51	COM5	2313	–323
12	SEG11	–2313	108	52	COM6	2313	–108
13	SEG10	–2313	–108	53	COM7	2313	108
14	SEG9	–2313	–323	54	COM8	2313	323
15	SEG8	–2313	–539	55	COM9	2313	539
16	SEG7	–2313	–755	56	COM10	2313	755
17	SEG6	–2313	–970	57	COM11	2313	970
18	SEG5	–2313	–1186	58	COM12	2313	1186
19	SEG4	–2313	–1401	59	COM13	2313	1401
20	SEG3	–2313	–1617	60	COM14	2313	1617
21	SEG2	–2313	–1833	61	COM15	2313	1833
22	SEG1	–2313	–2073	62	COM16	2313	2095
23	GND	–2280	–2290	63	SEG40	2296	2313
24	OSC1	–2080	–2290	64	SEG39	2100	2313
25	OSC2	–1749	–2290	65	SEG38	1617	2313
26	V1	–1550	–2290	66	SEG37	1401	2313
27	V2	–1268	–2290	67	SEG36	1186	2313
28	V3	–941	–2290	68	SEG35	970	2313
29	V4	–623	–2290	69	SEG34	755	2313
30	V5	–304	–2290	70	SEG33	539	2313
31	CL1	–48	–2290	71	SEG32	323	2313
32	CL2	142	–2290	72	SEG31	108	2313
33	V_{CC}	309	–2290	73	SEG30	–108	2313
34	M	475	–2290	74	SEG29	–323	2313
35	D	665	–2290	75	SEG28	–539	2313
36	RS	832	–2290	76	SEG27	–755	2313
37	R/$\overline{W}$	1022	–2290	77	SEG26	–970	2313
38	E	1204	–2290	78	SEG25	–1186	2313
39	DB0	1454	–2290	79	SEG24	–1401	2313
40	DB1	1684	–2290	80	SEG23	–1617	2313

Pin Functions

Signal	No. of Lines	I/O	Device Interfaced with	Function
RS	1	I	MPU	Selects registers. 0: Instruction register (for write) Busy flag: address counter (for read) 1: Data register (for write and read)
R/$\overline{W}$	1	I	MPU	Selects read or write. 0: Write 1: Read
E	1	I	MPU	Starts data read/write.
DB4 to DB7	4	I/O	MPU	Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. DB7 can be used as a busy flag.
DB0 to DB3	4	I/O	MPU	Four low order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. These pins are not used during 4-bit operation.
CL1	1	O	Extension driver	Clock to latch serial data D sent to the extension driver
CL2	1	O	Extension driver	Clock to shift serial data D
M	1	O	Extension driver	Switch signal for converting the liquid crystal drive waveform to AC
D	1	O	Extension driver	Character pattern data corresponding to each segment signal
COM1 to COM16	16	O	LCD	Common signals that are not used are changed to non-selection waveforms. COM9 to COM16 are non-selection waveforms at 1/8 duty factor and COM12 to COM16 are non-selection waveforms at 1/11 duty factor.
SEG1 to SEG40	40	O	LCD	Segment signals
V1 to V5	5	—	Power supply	Power supply for LCD drive V_{CC} –V5 = 11 V (max)
V_{CC}, GND	2	—	Power supply	V_{CC}: 2.7V to 5.5V, GND: 0V
OSC1, OSC2	2	—	Oscillation resistor clock	When crystal oscillation is performed, a resistor must be connected externally. When the pin input is an external clock, it must be input to OSC1.

Function Description

Registers

The HD44780U has two 8-bit registers, an instruction register (IR) and a data register (DR).

The IR stores instruction codes, such as display clear and cursor shift, and address information for display data RAM (DDRAM) and character generator RAM (CGRAM). The IR can only be written from the MPU.

The DR temporarily stores data to be written into DDRAM or CGRAM and temporarily stores data to be read from DDRAM or CGRAM. Data written into the DR from the MPU is automatically written into DDRAM or CGRAM by an internal operation. The DR is also used for data storage when reading data from DDRAM or CGRAM. When address information is written into the IR, data is read and then stored into the DR from DDRAM or CGRAM by an internal operation. Data transfer between the MPU is then completed when the MPU reads the DR. After the read, data in DDRAM or CGRAM at the next address is sent to the DR for the next read from the MPU. By the register selector (RS) signal, these two registers can be selected (Table 1).

Busy Flag (BF)

When the busy flag is 1, the HD44780U is in the internal operation mode, and the next instruction will not be accepted. When RS = 0 and R/$\overline{W}$ = 1 (Table 1), the busy flag is output to DB7. The next instruction must be written after ensuring that the busy flag is 0.

Address Counter (AC)

The address counter (AC) assigns addresses to both DDRAM and CGRAM. When an address of an instruction is written into the IR, the address information is sent from the IR to the AC. Selection of either DDRAM or CGRAM is also determined concurrently by the instruction.

After writing into (reading from) DDRAM or CGRAM, the AC is automatically incremented by 1 (decremented by 1). The AC contents are then output to DB0 to DB6 when RS = 0 and R/$\overline{W}$ = 1 (Table 1).

Table 1 Register Selection

RS	R/$\overline{W}$	Operation
0	0	IR write as an internal operation (display clear, etc.)
0	1	Read busy flag (DB7) and address counter (DB0 to DB6)
1	0	DR write as an internal operation (DR to DDRAM or CGRAM)
1	1	DR read as an internal operation (DDRAM or CGRAM to DR)

Display Data RAM (DDRAM)

Display data RAM (DDRAM) stores display data represented in 8-bit character codes. Its extended capacity is 80 × 8 bits, or 80 characters. The area in display data RAM (DDRAM) that is not used for display can be used as general data RAM. See Figure 1 for the relationships between DDRAM addresses and positions on the liquid crystal display.

The DDRAM address (A_{DD}) is set in the address counter (AC) as hexadecimal.

- 1-line display (N = 0) (Figure 2)
  - — When there are fewer than 80 display characters, the display begins at the head position. For example, if using only the HD44780, 8 characters are displayed. See Figure 3.

    When the display shift operation is performed, the DDRAM address shifts. See Figure 3.

	High order bits			Low order bits			
AC (hexadecimal)	AC6	AC5	AC4	AC3	AC2	AC1	AC0

Example: DDRAM address 4E

1	0	0	1	1	1	0

Figure 1 DDRAM Address

Display position (digit)	1	2	3	4	5		79	80
DDRAM address (hexadecimal)	00	01	02	03	04	··················	4E	4F

Figure 2 1-Line Display

Display position	1	2	3	4	5	6	7	8
DDRAM address	00	01	02	03	04	05	06	07
For shift left	01	02	03	04	05	06	07	08
For shift right	4F	00	01	02	03	04	05	06

Figure 3 1-Line by 8-Character Display Example

- 2-line display (N = 1) (Figure 4)

— Case 1: When the number of display characters is less than 40 × 2 lines, the two lines are displayed from the head. Note that the first line end address and the second line start address are not consecutive. For example, when just the HD44780 is used, 8 characters × 2 lines are displayed. See Figure 5.

When display shift operation is performed, the DDRAM address shifts. See Figure 5.

Display position	1	2	3	4	5		39	40
DDRAM address (hexadecimal)	00	01	02	03	04	....	26	27
	40	41	42	43	44	....	66	67

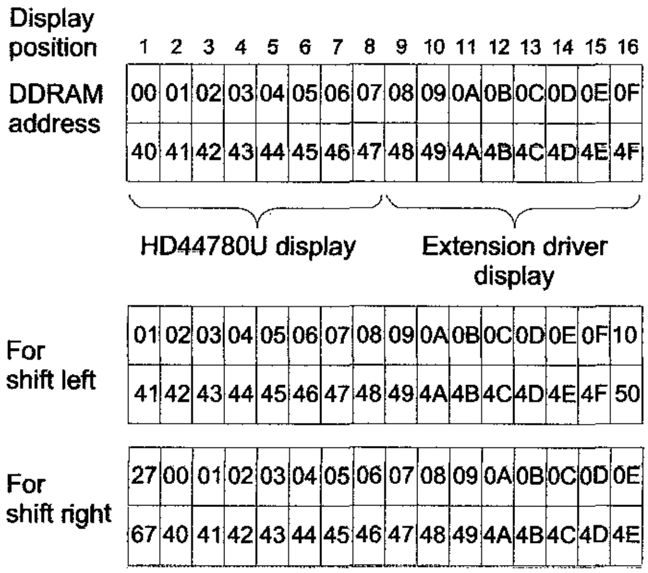
Figure 4 2-Line Display

Display position	1	2	3	4	5	6	7	8
DDRAM address	00	01	02	03	04	05	06	07
	40	41	42	43	44	45	46	47
For shift left	01	02	03	04	05	06	07	08
	41	42	43	44	45	46	47	48
For shift right	27	00	01	02	03	04	05	06
	67	40	41	42	43	44	45	46

Figure 5 2-Line by 8-Character Display Example

— Case 2: For a 16-character × 2-line display, the HD44780 can be extended using one 40-output extension driver. See Figure 6.

When display shift operation is performed, the DDRAM address shifts. See Figure 6.



Display position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

For shift left	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50

For shift right	27	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E
	67	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E

Figure 6 2-Line by 16-Character Display Example

Character Generator ROM (CGROM)

The character generator ROM generates 5 × 8 dot or 5 × 10 dot character patterns from 8-bit character codes (Table 4). It can generate 208 5 × 8 dot character patterns and 32 5 × 10 dot character patterns. User-defined character patterns are also available by mask-programmed ROM.

Character Generator RAM (CGRAM)

In the character generator RAM, the user can rewrite character patterns by program. For 5 × 8 dots, eight character patterns can be written, and for 5 × 10 dots, four character patterns can be written.

Write into DDRAM the character codes at the addresses shown as the left column of Table 4 to show the character patterns stored in CGRAM.

See Table 5 for the relationship between CGRAM addresses and data and display patterns.

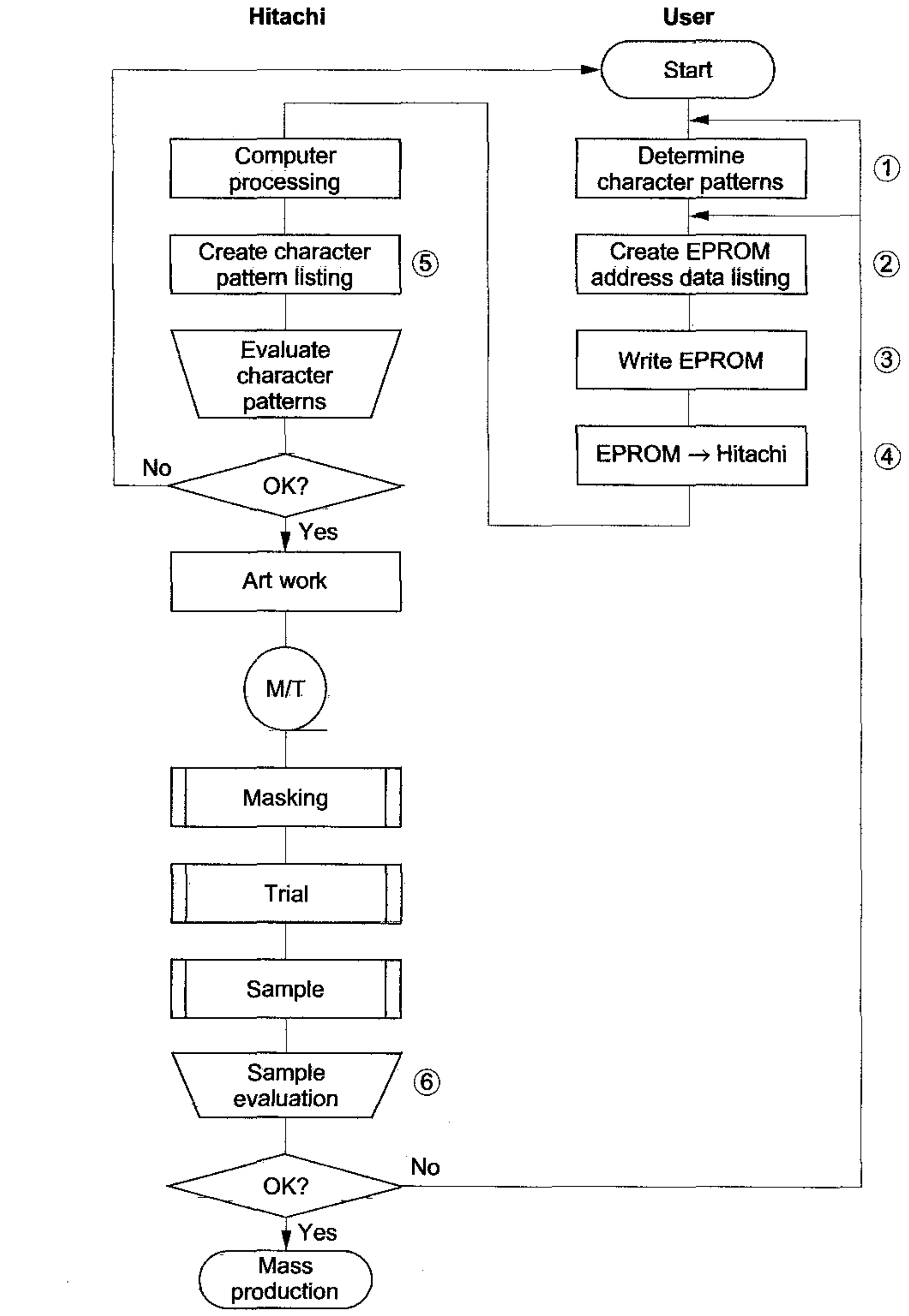
Areas that are not used for display can be used as general data RAM.

Modifying Character Patterns

- Character pattern development procedure

The following operations correspond to the numbers listed in Figure 7:

1. Determine the correspondence between character codes and character patterns.
2. Create a listing indicating the correspondence between EPROM addresses and data.
3. Program the character patterns into the EPROM.
4. Send the EPROM to Hitachi.
5. Computer processing on the EPROM is performed at Hitachi to create a character pattern listing, which is sent to the user.
6. If there are no problems within the character pattern listing, a trial LSI is created at Hitachi and samples are sent to the user for evaluation. When it is confirmed by the user that the character patterns are correctly written, mass production of the LSI proceeds at Hitachi.



Note: For a description of the numbers used in this figure, refer to the preceding page.

Figure 7 Character Pattern Development Procedure

- Programming character patterns

  This section explains the correspondence between addresses and data used to program character patterns in EPROM. The HD44780U character generator ROM can generate 208 5 × 8 dot character patterns and 32 5 × 10 dot character patterns for a total of 240 different character patterns.

  — Character patterns

  EPROM address data and character pattern data correspond with each other to form a 5 × 8 or 5 × 10 dot character pattern (Tables 2 and 3).

Table 2 Example of Correspondence between EPROM Address Data and Character Pattern (5 × 8 Dots)

EPROM Address												Data					
																LSB	
A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	O4	O3	O2	O1	O0	
0	1	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	
								0	0	0	1	1	0	0	0	0	
								0	0	1	0	1	0	1	1	0	
								0	0	1	1	1	1	0	0	1	
								0	1	0	0	1	0	0	0	1	
								0	1	0	1	1	0	0	0	1	
								0	1	1	0	1	1	1	1	0	
								0	1	1	1	0	0	0	0	0	← Cursor position
								1	0	0	0	0	0	0	0	0	
								1	0	0	1	0	0	0	0	0	
								1	0	1	0	0	0	0	0	0	
								1	0	1	1	0	0	0	0	0	
								1	1	0	0	0	0	0	0	0	
								1	1	0	1	0	0	0	0	0	
								1	1	1	0	0	0	0	0	0	
								1	1	1	1	0	0	0	0	0	

Character code

Line position

Notes: 1. EPROM addresses A11 to A4 correspond to a character code.
2. EPROM addresses A3 to A0 specify a line position of the character pattern.
3. EPROM data O4 to O0 correspond to character pattern data.
4. EPROM data O5 to O7 must be specified as 0.
5. A lit display position (black) corresponds to a 1.
6. Line 9 and the following lines must be blanked with 0s for a 5 × 8 dot character fonts.

— Handling unused character patterns

1. EPROM data outside the character pattern area: Always input 0s.
2. EPROM data in CGRAM area: Always input 0s. (Input 0s to EPROM addresses 00H to FFH.)
3. EPROM data used when the user does not use any HD44780U character pattern: According to the user application, handled in one of the two ways listed as follows.
   a. When unused character patterns are not programmed: If an unused character code is written into DDRAM, all its dots are lit. By not programing a character pattern, all of its bits become lit. (This is due to the EPROM being filled with 1s after it is erased.)
   b. When unused character patterns are programmed as 0s: Nothing is displayed even if unused character codes are written into DDRAM. (This is equivalent to a space.)

Table 3 Example of Correspondence between EPROM Address Data and Character Pattern (5 × 10 Dots)

EPROM Address												Data					
																LSB	
A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	O4	O3	O2	O1	O0	
0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	
								0	0	0	1	0	0	0	0	0	
								0	0	1	0	0	1	1	0	1	
								0	0	1	1	1	0	0	1	1	
								0	1	0	0	1	0	0	0	1	
								0	1	0	1	1	0	0	0	1	
								0	1	1	0	0	1	1	1	1	
								0	1	1	1	0	0	0	0	1	
								1	0	0	0	0	0	0	0	1	
								1	0	0	1	0	0	0	0	1	
								1	0	1	0	0	0	0	0	0	← Cursor position
								1	0	1	1	0	0	0	0	0	
								1	1	0	0	0	0	0	0	0	
								1	1	0	1	0	0	0	0	0	
								1	1	1	0	0	0	0	0	0	
								1	1	1	1	0	0	0	0	0	
Character code								Line position									

Notes: 1. EPROM addresses A11 to A3 correspond to a character code.
2. EPROM addresses A3 to A0 specify a line position of the character pattern.
3. EPROM data O4 to O0 correspond to character pattern data.
4. EPROM data O5 to O7 must be specified as 0.
5. A lit display position (black) corresponds to a 1.
6. Line 11 and the following lines must be blanked with 0s for a 5 × 10 dot character fonts.

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A00)

Upper 4 Bits Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	β	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	ˣ	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		-	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	゜	ö	█

Note: The user can specify any pattern for character-generator RAM.

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A02)

Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)	▶		0	@	P	`	p	Б	α	▓	°	À	Ð	à	ð
xxxx0001	(2)	◀	!	1	A	Q	a	q	Д	♪	¡	±	Á	Ñ	á	ñ
xxxx0010	(3)	“	"	2	B	R	b	r	Ж	Γ	¢	²	Â	Ò	â	ò
xxxx0011	(4)	”	#	3	C	S	c	s	З	π	£	³	Ã	Ó	ã	ó
xxxx0100	(5)	⏫	$	4	D	T	d	t	И	Σ	¤	₧	Ä	Ô	ä	ô
xxxx0101	(6)	⏬	%	5	E	U	e	u	Й	σ	¥	µ	Å	Õ	å	õ
xxxx0110	(7)	●	&	6	F	V	f	v	Л	♬	¦	¶	Æ	Ö	æ	ö
xxxx0111	(8)	↵	'	7	G	W	g	w	П	τ	§	·	Ç	×	ç	÷
xxxx1000	(1)	↑	(	8	H	X	h	x	У	🔔	ƒ	ω	È	Φ	è	φ
xxxx1001	(2)	↓	)	9	I	Y	i	y	Ц	Θ	©	¹	É	Ù	é	ù
xxxx1010	(3)	→	*	:	J	Z	j	z	Ч	Ω	ª	º	Ê	Ú	ê	ú
xxxx1011	(4)	←	+	;	K	[	k	{	Ш	δ	«	»	Ë	Û	ë	û
xxxx1100	(5)	≤	,	<	L	\	l	|	Щ	∞	Ю	¼	Ì	Ü	ì	ü
xxxx1101	(6)	≥	-	=	M	]	m	}	Ъ	♥	Я	½	Í	Ý	í	ý
xxxx1110	(7)	▲	.	>	N	^	n	~	Ы	ε	®	¾	Î	Þ	î	þ
xxxx1111	(8)	▼	/	?	O	_	o	⌂	Э	∩	‘	¿	Ï	ß	ï	ÿ