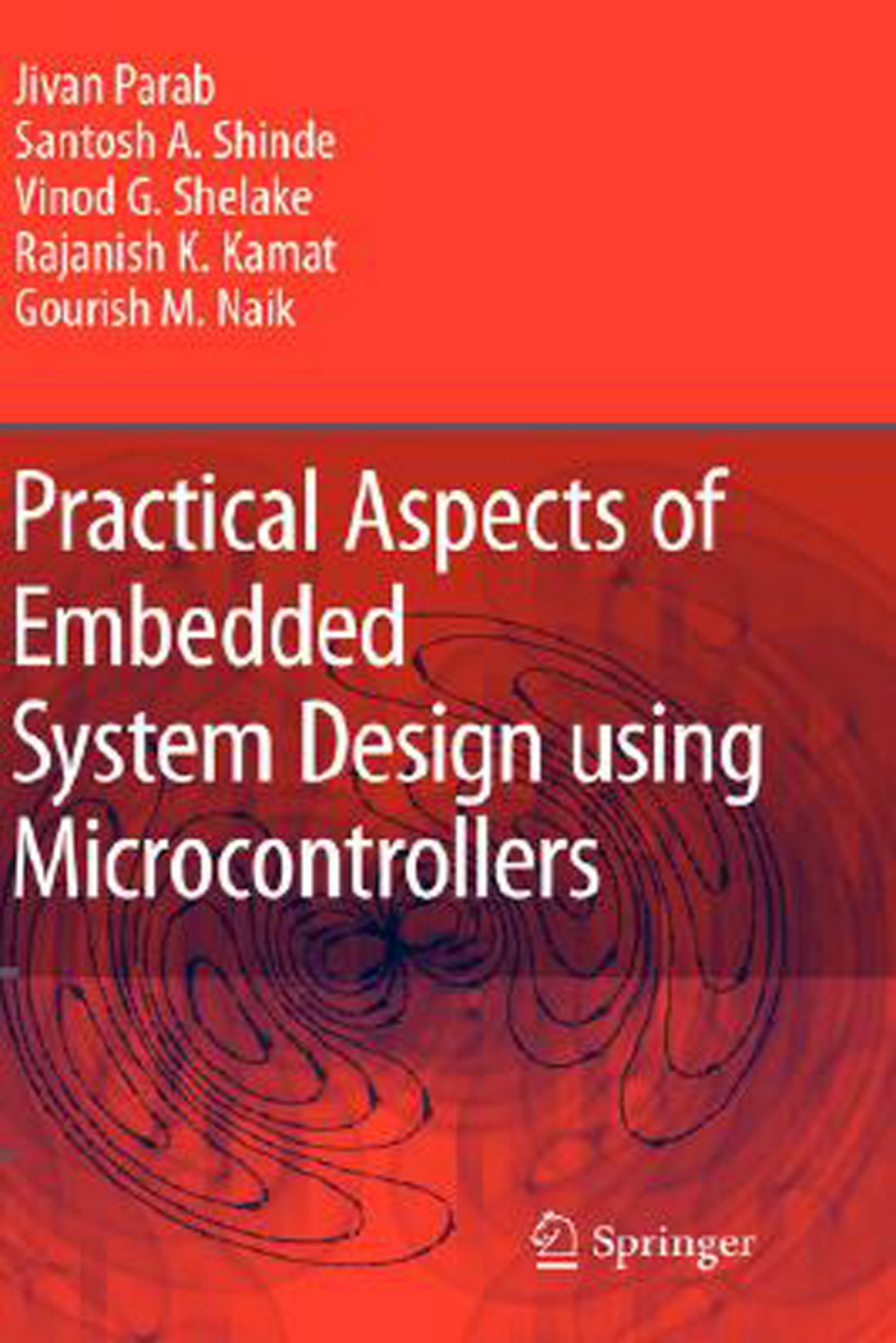
Jivan Parab
Santosh A. Shinde
Vinod G. Shelake
Rajanish K. Kamat
Gourish M. Naik

Practical Aspects of Embedded System Design using Microcontrollers



Springer

Practical Aspects of Embedded System Design using Microcontrollers

Jivan S. Parab • Santosh A. Shinde
Vinod G. Shelake • Rajanish K. Kamat
Gourish M. Naik

Practical Aspects of Embedded System Design using Microcontrollers

Springer

Jivan S. Parab
Goa University
Goa, 403 206
India

Santosh A. Shinde
Shivaji University
Kolhapur, 416 004
India

Vinod G. Shelake
Shivaji University
Kolhapur, 416 004
India

Dr. Rajanish K. Kamat
Shivaji University
Kolhapur, 416 004
India

Dr. Gourish M. Naik
Goa University
Goa, 403 206
India

ISBN 978-1-4020-8392-1 e-ISBN 978-1-4020-8393-8

Library of Congress Control Number: 2008928690

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Foreword

My perception regarding embedded systems goes on the following lines “Embedded Systems are very simple. It just takes a genius to understand its simplicity” and I know that authors of this book are the genius in this subject. With their many years of experience in industry consultancy and academia they posses the arts and science of designing successful, working and useful Embedded Systems. The “Art”, part comes with a mix of knowledge, experience, intuition and creativeness that the readers will witness from the various case studies developed in this book. While their “Science” and “Engineering” foundations are evident from the adopted design methodologies guaranteeing correctness with proper hardware selection and time as well as memory efficient code. In fact this is the second book on this subject by the same team. I have gone through the first one “Exploring C for Microcontrollers: A hands on Approach” published by Springer and found it very informative. I learnt that the book is popular with embedded designers in US and UK. The same approach of “Learning by Doing” as in explored in the first book has also been extended for this second book.

The most significant aspect about embedded systems that I like is its unique synergy between hardware and software. An Embedded Engineer is supposed to be an expert in multiple domains such as microcontrollers, FPGAs, digital logic, C programming, sensors, instrumentation and last but not the least even nuts and bolts i.e. mechatronics. With a continued interaction with some of the authors of this book, I found them to possess expertise in this field having multiple facets. Namely Dr. Gourish Naik has been instrumental since his IISc days to incorporate Embedded Systems aspects in academics. Dr. R.K. Kamat who was offered a position in Motorola in Europe possess great capability in design and the development of Embedded Systems.

Now let me focus on the very need of this book. As all of us are aware since their inception, embedded systems have caused a tremendous change in society, a change that is continuing from last few decades at a pace surpassing every imagination. With their increasing significance in world markets, there is a scarcity of experienced embedded system professionals. I learnt that embedded systems development professionals have handicapped Hong Kong industrialists’ ability to exploit high added value market potentials in embedded systems products. In Europe, the European Commission has recognized the importance of embedded systems by

creating a new unit in the IST Directorate. The visions surrounding the AMI-space (embedded systems everywhere, described in the context of human life as 'ambient intelligence') have considerably influenced the 6th Framework Programme of the IST domain. However, with such growing activities in this field, the scarcity of experienced embedded systems development professionals is quite natural. This has spurred a growing emphasis on embedded systems education in most of the US, UK and Indian universities for nurturing quality human resource in this field of significant importance. While the academics are trying to do their best in inculcating the concepts, there are very few course wares or books that will practically cover the concepts. This book will help in filling up the supply-demand gap in training the Embedded Systems Professionals.

The book covers applications based on two widely used 8 bit microcontrollers viz. PIC series from Microchip and MCS51 series from Atmel. Authors have chosen the right microcontroller for the right application. The latest chips have been used in developing the applications. Self explanatory C code with proper documentation is given for each application. Routine things such as lengthy datasheets have been skipped. Good web resources have been identified so that the readers can simply find the details after going through the Web URLs.

With these few words, I strongly recommend this book for intermediate programmers, electronics, electrical, instrumentation engineers or any individual who is inclined to take up his/ her career in this field. I am sure that reader will welcome this book and gain great concepts by adopting the practical approach taken up throughout the book.

Dr. B. Selvan

Dr. Balakrishnan Selvan obtained a Ph.D. in 1991, from the University of Bradford's Postgraduate School of Studies in Information Systems Engineering. In 1983 he received a M.Sc. degree in Electrical Communication Engineering from the Indian Institute of Science, Bangalore. Between the years of 1984 and 1997 he held various teaching and research appointments, in the field of communications and computing, at universities in Singapore and UK. In 1997 he joined Alcatel Submarine Networks at Greenwich, London, as a Principal Engineer for design and development of DWDM terminal equipment. In 2003 he set up his own consultancy firm, which specialise in providing information technology solutions for small business in and around South East London.

Dr. Selvan is a Chartered Engineer (UK Engineering Council), and a member of the Institution of Engineering and Technology (UK).

Author's Profile

"Website of the research group may be seen at URL: http://www.rkkamat.in".

Jivan S. Parab, Goa University, Goa, India
After graduating from Goa University, Jivan was hired by Masibus Instruments Pvt. Ltd., as a design engineer. After working for a year in Masibus, Jivan shifted to academics and joined Goa University, Goa, as he was concerned about the increasing diabetic patients in India and abroad. He was passionate about development of low cost, portable glucometer for poor people. With his rich experience in designing heterogeneous Embedded Systems comprising of microcontrollers, FPGAs and onboard flash, he has almost completed the project and very soon will be launching the same with his completion of doctorate in the same topic.

Santosh A. Shinde, Shivaji University, Kolhapur, India
Santosh had a stint in Embedded Instrumentation by practically working in Wimson Electronics Pvt. Ltd., as an R&D Engineer in their SMD division. Santosh has worked with many of the popular microcontrollers from Intel, Atmel, Philips and Microchip. He is experienced in programming in C, C++, under LINUX, DOS, and Win9x, WinXP. He is also familiar with many EDA tools such as Handel-C, Modelsim, Gerber, Orcad, Mentor Graphics, Xilinx, and CAD software. He will be submitting his doctorate very soon on FPGA based programmable ASIC for circumventing SPAM.

Vinod G. Shelake, Shivaji University, Kolhapur, India

Vinod is always been fascinated about developing Embedded products for computer network security. In order to gain real life experience, he joined Software Technology Parks of India, an autonomous body under Government of India, who has build and maintains the countrywide backbone of Internet exchanges. As an avid embedded enthusiast, he left STPI to devote more time on R&D in this field. Currently he is busy in development of a FPGA based firewall with lots of novel features than those existing in market. Vinod holds Masters in Electronics specialized in Embedded Systems and soon he will submit his dissertation for Ph.D. in Embedded VLSI systems.

Dr. Rajanish K. Kamat, Shivaji University, Kolhapur, India

Dr. Rajanish K. Kamat loves Electronics, Internet and all the high tech latest things in the world. He's in them all the time. When he is not tapping keys for a research paper or a book like this, he is either teaching for Masters student or guiding research to Ph.D. students. Dr. Kamat is right now working with Shivaji University, Kolhapur where he is involved in teaching, research and consultancy. Besides he is also taking care of Internet gateway of Shivaji University. He has been exposed to almost every variant of mechanical and electronic computing device there is (and has been). This everyday contact with the electronic industry allows Dr. Kamat to bring this real-world experience to the books like this. His expertise has been recognized by the Department of Science and Technology, Government of India by awarding him a major project on Soft IP cores under the Young Scientist Scheme. He is a single point contact for all the authors.

Dr. Gourish M. Naik, Goa University, Goa, India

Embedded devices are not Dr. Gourish Naik's only love. He enjoys to be literally "on the road" to modify Electronics in cars. He's also walked among his share of optical communications too as a part of his Ph.D. work way back in 1987 from the prestigious Indian Institute of Science, Bangalore. Computers, Electronics, Robotics continued to be his hobbies and that's why he has taken up teaching and research as a full time profession. At Goa University, Dr. G.M. Naik is heading the Electronics as well as Instrumentation sections and has earned reputation as a consultant all over in India. He has been instrumental to incorporate the latest in Embedded Systems in the curriculum. University Grants Commission, the nodal body for the universities in India has recognized and appreciated his efforts by granting him "Innovative Program" in Embedded Systems.

Preface

Embedded Systems: A Component Based Software Industry

According to Business Communications Company Inc. (BCC) research report the embedded software business is predicted to grow from about $1.6 billion in 2004 to $3.5 billion by 2009, at an average annual growth rate (AAGR) of 16%. The growth rate for the Embedded hardware will reach $78.7 billion in 2009. The estimated growth rate is propelled by several key themes: namely the penetration of Applications Specific Processors (ASPs) as well as stand-alone chips such as microprocessors and microcontrollers, which has cannibalized their sells as compared to the consumption volume of stand-alone Micro-Processing Units (MPUs), Application Specific Integrated Circuits (ASIC), Field Programmable Gate Arrays (FPGA) and Digital Signal Processors (DSP). In general the growth of system-on-a-chip components has really revitalized the embedded system market. Another report by the Indian Semiconductor Association (ISA) and Frost & Sullivan supports the flourishing growth rate statistics. It states that semiconductor and embedded industry is projected to bloom from $3.25 billion in 2005 to $43.7 billion by 2015. With such an attractive growth statistics, the field of embedded systems now influences many industrial sectors including automotive, aerospace, consumer electronics, communications, medical and manufacturing. Today it is the fastest growing sector in IT and still open with many opportunities. Traditional research in Embedded Systems is in progress in good number of research fields such as software, Real Time Operating System (RTOS), new communication protocols, microcontroller based system, low power design, immunity to Electro-Magnetic Interference EMI, etc. to name a few. We have taken up the design aspects of Microcontroller based Embedded Systems with more emphasis on the software.

Who This Book is For

Last year the ‘IDC’ a premier global market intelligence firm’s analysis revealed that the embedded industry product development is expected to be as high as $75 billion. This entails the industry requirement of trained human resource with mixed skill set

both in hardware and software. Unfortunately, the synergetic demand of hardware and software or some times even referred to as firmware competency has lead to a supply-demand gap of HR in this field. This gap expressed in numerical figures lead to requirement of around 150,000 embedded engineers in the current year and more in years to come to serve the global embedded industry. Our previous as well as the current book published under the realm of Springer are the ultimate solutions to bridge the supply – demand gap of Embedded System professionals. The book is intended for graduate and postgraduate students from the Electrical, Electronics, Computer and Instrumentation Engineering. It is equally beneficial for industry professionals, hobbyists and software people who would like to try their luck with Embedded Systems. Undoubtedly, some people can use this book in laboratory courses. Experience programmers can skip some basic part and get right into the application case studies.

We promise that the potential readers can lessen the steepness of the learning curve for Embedded Systems by using this book. Through this book, we hope for you to be able to switch to Microcontrollers and Embedded Systems in the shortest possible timeframe. Back when we started our career in this field, we weren't lucky enough to have a book like this to learn from! As such, a reader will find lots of information for newcomers, even those who have not programmed much before. On the other end of the scale, we have worked hard to put in this book lot of information on advanced functionality in Embedded Systems such as I^2C. If you are a veteran user looking to take your microcontroller based design skills above and beyond where they are right now, we are hoping you will find there is lots to be had here.

"Hands on Approach"

As Aristotle said: "What we have to learn to do, we learn by doing." The approach adopted by us is "Practical Design" and will definitely inspire the student and design community to learn on their own. A quote from W. McKeachie, "Professors known as outstanding lecturers do two things; they use a simple plan and many examples." Yes!! We have given the bare minimum theoretical aspects and rest all is the practical circuit diagrams and complete C code with 33 case studies so as to enjoy implementing the stuff in laboratory. The book is developed with the main goal of making the task of learning Embedded C something fun that you do not have to worry about. There is a famous quotation by Jim Rohn, "Formal education will make you a living; self education will make you a fortune." With this book we are offering the potential readers an opportunity to learn on their own and enter into the ubiquitous world of Microcontroller based Embedded Systems.

What is Different about this Book?

A.A. Hodge said "He is wise who knows the sources of knowledge – who knows who has written and where it is to be found." True enough! We have skipped the routine theoretical aspects of microcontrollers such as lengthy description of registers,

on –chip memory map, pinout, sinking sourcing current values, etc. (Open any textbook, and these things are right there). Instead we assume that either the potential reader is aware of these things or he will resort to the web references listed at the end of the book.

Some of the salient features of the book are as follows:

- The book is presented so as to refer in whatever order you want. Once you have the prerequisite basics down, we encourage you to flick through the table of contents, find something that interests you most, and start reading from there.
- It covers design based on the representative members of both RISC and CISC architectures.
- The most interesting are the 33 number of case studies. We have undertaken several tasks necessary for building a good source of case study material. A good taxonomy is built, and a large collection of primary sources is presented as web based resources.
- The devices chosen for the applications are from the industry leading vendors such as Atmel, Microchip, Philips, Maxim and so on.
- It is made sure that all the above mentioned devices are available in the market and most of them are cost effective.
- Clear and precise circuit diagrams along with complete listing of C source code per application will enable the reader to experiment the given stuff in his laboratory.
- A lucid flow of the resource material and the participatory style will definitely make you friendly with the subject matter.
- Actual screenshots taken and embedded in the text to illustrate the concepts.
- Another feature is reusability of the code. With little modification the codes developed in this book may fit in your embedded application saving you from the labor of reinventing the wheel.
- Yet another feature is Simulate-ability of the code that will boost the confidence of the readers and enable them to go one step forward towards testing the same on the hardware platform.

How This Book Was Prepared?

The book is a result of author's many years of experience in academics, research and industry. With the overwhelmed response received to the first book "Exploring C for Microcontrollers: A hands on approach" published by Springer in May 2007, authors were more than happy. However, many readers expressed a balanced coverage of RISC and CISC architectures. Authors acted on these suggestions and framed the existing book. Looking at I^2C popularity a chapter was devoted for the same. Similarly the most popular PIC16F877 was chosen for the case studies. For the sake of comparison another equally popular microcontroller from CISC architecture AT89S52 was chosen for a set of case studies covered in last chapter. Thanks to our student community who is now largely placed in reputed industries for identifying the problem statements for the case studies.

Chapter Descriptions

We recommend you to begin by reading through the summary paragraphs of each chapter below, which introduce each section and provide you with a good overall picture of how the book is organized.

Chapter 1 is the "Welcome Speech" for inspiring the potential readers. It focuses on the importance of the subject. In this chapter there are several references of many forecasts, that visualizes the growing importance of embedded systems in years to come. After reading through one gets a realization that the traditional academic courses focusing either mostly on hardware as in many Electronics/ Electrical Engineering programs or mostly on software as in many Computer Science programs will not suffice the expertise in this synergetic domain. Latest trends and statistics from leading marketing and research firms will convenience the reader to kick start their venture in this field. Coverage of MPLAB for PIC will introduce to the IDE environment for PIC. The IDE for the AT89s52 has been skipped as it is already been found its way in the earlier book by the same authors.

Rest of the book proceeds towards a systematic building block approach. Chapters 2–5 are based on PIC16F877 while the Chapter 6 applications use AT89S52 microcontroller.

Chapter 2 covers the fundamental aspects of microcontroller based system design from interaction to ambient environment point of view. It begins with the basic LED interfacing and its variation and moves on to the more complex interfaces such as seven segment LED, LCD, buzzer interfacing, etc. In many situations the embedded device resorts to polling a switch status for intelligent branching of the code. Sometimes in more complicated circumstances the status of a number of signals coming from the switch needs to be sensed. In this chapter we have taken care of both of them. A basic DIP switch interfacing and the thumbwheel switch interfacing is presented in depth.

Analog signals are very common inputs to embedded systems. Transducers and sensors such as temperature, pressure, velocity, humidity are truly analog. Therefore we need to convert these analog signals in to digital so that the PIC can read it. Upon processing in digital domain again the PIC has to enable/disable or control the actuators back in analog domain. This core issue of digitization and control is taken up in Chapter 3. This chapter will further boost your interest as it covers lots of interesting variations such as using onchip ADC of PIC, interfacing external ADC for mutichannel data logging applications. PWM based DAC is certainly more competent with the theoretically infinite resolution. Again the combination of ADC and a port pin of PIC is used for the temperature control application avoiding the power hungry DAC. Temperature being chosen owing to its universality in most of the control systems. Generation of PNR signal and waveforms serves the testing applications for the embedded products.

Although hyper terminal was more used with Win 98, but still in the age of Win XP it has become a serial gateway for group of embedded appliances to be controlled from a PC terminal. Many embedded systems compliant for the PC serial

communication now use their own propriety terminal emulation programs. But with out experience there is nothing robust like a hyper terminal for the serial emulation. In Chapter 4 we have revised a step by step procedure for setting up the hyper terminal for communicating with the embedded board. The case studies developed here comprises of displaying data on Hyper Terminal from the PIC processor, getting sensor output (LM35) on the hyper terminal and actuating a relay. Additionally, we have demonstrated stepper motor control by outputting the speed, direction, etc. from the hyper terminal. A potential developer may take these applications to a greater heights such as domestic appliance control, home light control, home security opening, closing the door with camera interface using a single PC with the hyper terminal. Other intention is to motive the user for writing such a GUI (may be using Visual Basic or Visual C++) for serial communication or even for the upcoming USB.

Embedded world is witnessing incorporation of many new protocols for interconnectivity with each other. I^2C, SPI, CAN, UART are some of the latest protocol suites used with the embedded products. Chapter 5 is all about I^2C and application based on it. Why we have taken this particular suite? There are two reasons. First it is the most popular one. The popularity is realized by the fact that its 7 bit addressing space has been now upgraded to 10 bit to fit more client devices. Second reason being once you understand one protocol, other will follow on the similar lines with few differences here and there. The chapter begin with basic case studies such as I^2C based RTC and serial EPROM interfacing. Then it moves on to the interfacing of two different ADC chips viz. PCF8591 and AD1236. Main difference is the resolution offered by these ADCs. We want to emphasize here that the appropriate device with the desired specifications should be used for the intended application. After all we are embedded developers and we value the specifications more than any other engineering disciplines. An intelligent reader can make out the difference in resolution by comparing the above two interfacing approaches. An embedded application will be incomplete without making its impact in an analog world. Therefore the last application of this chapter In order to generate different wave, we have interface I^2C based DAC (MAX5822) to PIC. Here the values corresponding to respective waveform is sent to DAC serially using SDA line and then subsequently you may view it on the CRO.

The last chapter is an odd man out in the RISC domain. Most of the embedded system applications do not require more than what provided by the AT89S52 microcontroller a popular derivative of the basic 8051. With 8 Kbytes of Flash, 256 bytes of RAM, 32 I/O lines, Watchdog timer, two data pointers, three 16-bit timer/counters, a six-vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator, and clock circuitry you have everything that is required in the world for a successful embedded application. More than this you will feel at home with the support of powerful tools IDEs and webforums of enthusiastic developers working with this device. We have developed many interesting case studies such as a night lamp controller, automation of a nylon rubber stamp making machine, digital IC tester, etc. The tiny BIOS opens yet another window of programming style based on the ISRs. Designers are always been wondered how to partition the things in analog and digital domains. The salinity measurement system evidences the benefits of accomplishing the nonlinearity correction in analog domain that reduces the

computing burden of the microcontroller and helps in getting optimized timing even with CISC architecture. The sensor interfacing being the universal application for microcontrollers, we have gone a step further towards making them fault tolerant and accurate measuring systems with their arrays. The common philosophy of applications developed in this chapter is their inherent computing complexity apart from the conventional stress on I/O and onchip resources for which the microcontroller is best suited. With this chapter a potential reader can compare the performance of PIC16F877 a RISC processor with the CISC AT89S52. Efforts towards interchanging the processors for the given application will give an insight as regards to the choice of a proper microcontroller for appropriate application. With this, we left the decision of the "RISC Vs CISC" debate to the wise reader. Not the least the universality of Embedded C and the almost unchanging program structure will prove its usefulness for the embedded paradigm.

Errors

Warning: The programs given in this book may contain errors. Authors assume no liability for any damage or accidents or any sort of mental harassment of the readers.

This note is not uncommon in these days of legal litigations. However, we promise you that we have taken all the efforts to make the book free of any sort of errors. But "To Err is Human". Should you come across any errors or would like to seek any clarifications regarding the hardware, software, availability of chips, etc. please feel free to give a shout by email to Dr. R.K. Kamat at rkk_eln@unishivaji.ac.in. He is a single contact point for all the authors.

At Last

The goal of the present book is to empower the potential reader having more or less programming or electronics experience, to build embedded systems using microcontrollers around the home, office, store, etc. We have tried our best to overcome the lack of hands on approach with our maturity in this domain The book will serve a good reference for the academic people and overcome the fear of the newbie's in this field. Because after all as teachers we believe in what Linda Conway has said, "It's not what is poured into a student that counts, but what is planted."

We wish you all the best for planting the concepts of embedded systems in your minds that will feel your life with happiness.

Jivan S. Parab
Santosh A. Shinde
Vinod G. Shelake
Dr. R.K. Kamat
Dr. G.M. Naik

Acknowledgement

Several key people helped us to make this project successful. First and foremost Professor M.M. Salunkhe, Vice Chancellor of Shivaji University, Kolhapur, India for encouragement and support. Further Dr. Kamat and Dr. Naik would like to thank their respective wives for their understanding and patience shown when the preparation of the book took time which could have been spent with the family. Our thanks are then to Dr. Kamat's wife Rucha and Dr. Naik's wife Deepa.

Jivan wants to thank his sisters Jyoti and Jagurti and parents for all the support received. Thanks are also due to our friend circle Kunal, Rupesh, Roy, Jesni, Yogan, Jaymala, Mahesh, Mamata and Sapana for giving inputs for the case studies. Mr. Rajendra Gad deserves special thanks for the support received at Goa University.

Mr. Santosh Shinde would like to thank his parents as well as his friends Mr. Abhijeet and Masoom for their support. Mr. Vinod Shelake would like to thanks parents and Mrs. Sharyu for their support.

Particular thanks goes to Shivaji University and Goa University authorities for the support received towards the infrastructure, kits and PCs used while preparing the book.

All the authors would like to express their special appreciation towards Dr. B. Selvan who has readily agreed to review the book and consented for expressing the same in the form of foreword. Thanks are due to Mr. Mark de Jongh, Senior Publishing Editor and Mrs. Cindy Zitter from Springer for prompt communication and online support all the time.

Jivan S. Parab
Santosh A. Shinde
Vinod G. Shelake
Dr. R.K. Kamat
Dr. G.M. Naik

Contents

Chapter 1
Introduction

Welcome to the world of 'Embedded System's dreamland'!
Operational excellence in training, research and consultancy of more than a decade, has resulted in crafting this book. Our aim is to make learning so much more fun than learning from books or traditional classroom setting and as the name indicates more emphasis on practical knowledge. The primary focus on 'application oriented system design' is to bridge the gap between industry requirements and students' skill set. Read through and implement the code presented here for your laboratory experiment and we promise that your employability skills will be significantly increased as you will be closer to the industrial applications described here. Starting from fairly basic experiments such as LED, LCD interfacing this book will show you how to go about realizing bigger systems and complex applications. However, the potential reader of this book should have a basic knowledge of C programming and initial practical experience in compiling and debugging programs. It is ideal for programmers and engineers who already have some understanding of programming and who now wish to gain a solid understanding of the use of C for embedded systems. Even if you do not have any experience of C in an embedded system, you will successively build it with the participating approach of the book. The hands-on training approach and lots of industry oriented real life exercises will take you to a large step forward in your Embedded C-programming. Thus this book is an opportunity to program a test embedded system using industry standard development tools and debugging aids. The importance of embedded systems is illustrated by following that now 94% of the chips/microprocessors produced in the market are for embedded products.

So if you have missed the bus for participating in this rapidly growing field then you can catch up by adopting the hands on approach of this book. Chapter 1 proceeds on following lines:

J.S. Parab, et al., *Practical Aspects of Embedded System Design using Microcontrollers,*

1.1 Defining Embedded Systems

It is little difficult, and somewhat controversial, to formulate a precise definition of Embedded System. Definitions given by various references are as follows:

- An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, sometimes with real-time computing constraints [12].
- Specialized computer system hardware that is used in larger systems or machines to control devices such as automobiles, home appliances, and office equipment [17].
- Any electronic system that uses a CPU chip, but that is not a general-purpose workstation, desktop or laptop computer. Such systems generally use microprocessors, or they may use custom-designed chips or both [13].
- An embedded system is some combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular kind of application device [14].
- An embedded system is a combination of computer circuitry and software that is built into a product for purposes such as control, monitoring and communication without human intervention [15].

All the above definitions of the embedded systems project them as a part pf the computing systems. However, the embedded systems stands very much apart from the computing systems in several respects. Definition given by the Institution of Electrical Engineers (IEE) looks more practical.

IEE defines Embedded Systems [18] as: “the devices used to control, monitor or assist the operation of equipment, machinery or plant. “Embedded” reflects the fact that they are an integral part of the system. In many cases their embeddedness may be such that their presents is far from obvious to the casual observer and even the more technically skilled might need to examine the operations of a piece of equipment for some time before being able to conclude that an embedded control system was involved in its function. At the other extreme, a general-purpose computer may be used to control the operations of a large complex processing plant, and its presence will be obvious.”

From applications point of view [19] Embedded systems are defined as systems in every “intelligent” device that is infiltrating our daily lives: the cell phone in your pocket, and all the wireless infrastructure behind it; the Palm Pilot on your desk; the Internet router your e-mails are channeled through; your big-screen home

theater system; the air traffic control station as well as the delayed aircraft it is monitoring! Software now makes up 90% of the value of these devices.

The controversial aspects in defining an Embedded systems are due to their constant evolution at a rapid pace. For example today's cell phones or personal gadgets have built in intelligence with more and more functionality, so whether they fir in "embedded" arena or migrating towards the "personal computer" domain? On the other hand some embedded products are built with PC motherboard without other peripherals such as keyboards. Again it becomes difficult to classify them under PC domain or solely under Embedded. The situation further poses challenges as these days the embedded system has to run database management systems such as SQL, in addition to their dedicated one and only one task. An interesting aspect of the embedded system seems to be emerging with the vanishing demarcation between them and PC domain as a computer whose end special purpose function is not to be a computer or computer but for non-computer purpose. The most current definition of the Embedded System incorporating most of their functional aspects is as follows:

"A specialized computer system that is part of a larger sys-tem or machine. Typically, an embedded system is housed on a single microprocessor board with the programs stored in ROM. Virtually all appliances that have a digital inter-face like watches, microwaves, VCRs, cars utilize embed-ded systems. Some embedded systems include an operating system, but many are so specialized that the entire logic can be implemented as a single program [25]."

1.2 Essential Attributes of Embedded Systems

The definitions from various sources gives an insight as regards to the essential attributes of the embedded systems. They are as follows:

- Single/dedicated tasking
- Power constrained (requires weight efficiency)
- Memory constrained (requires code size efficiency)
- Real time response (requires run time efficiency)
- Firmware dominated with currency and time efficiency
- Reliability and fault tolerant architecture
- Simplified user interface (generally no GUI)
- Less human interaction (infinite loop approach)
- Very frequent interaction with the ambient physical medium (reactive systems)
- Works with special purpose OS (rather than general purpose such as Linux or Ms Windows)
- Minimum interrupt latency
- Generally mass produced/high volume systems (cost effective)
- Maintainability
- Safety
- Security

The field of Embedded Systems appears to be at the cross-section of many technologies and subject areas. As far as the functionality is concerned, it derives the concepts from Electronics (microprocessors, microcontrollers, etc.) and Computer Science (operating system issues, software engineering, etc.). As the system interacts with the physical environment the key concepts of sensors, control engineering, communication technology such as optical networking, etc. also plays a vital role in increasing the utility of the system. With the growing impact of the Internet and web era, Ethernet interfacing and on chip TCP/IP are being embedded on the embedded board. Growing trend in this area is hardware software co-design and use of FPGA based customized embedded processors from third party vendors to achieve real time response, power, weight and computational efficiency.

1.3 Embedded Systems Historical Aspects

The history of embedded systems goes way back to the sixties. However, the systems developed those days could not penetrate themselves for the common man due to their prohibitively high cost and limited portability. An article from Embedded Technology Journal quotes: “With the attributes mentioned in the previous heading, it is clear that such a system could have been developed with only with the advent of the microprocessors. To briefly trace the history of embedded systems architectures, we have moved rapidly from systems-in-chassis to systems-on-board, then into system-on-chip (SoC) integration over the past decade. Each time we have integrated, our power density has increased as our form factors shrank. Interestingly, today, embedded systems have more in common with supercomputers than with commodity desktop and laptop machines”. It is further analyzed that both supercomputers and embedded computers have hit the wall of diminishing returns on single-thread, Von Neumann processors and have moved into the domain of multicore and alternative architecture processing [22]. It has been reported [23] that, the first embedded system to be produced in large quantities was the Autonetics D-17 guidance computer which was used in the Minuteman missile, released in 1961. It was built from discrete transistor logic and had a hard disk for main memory. When the Minuteman II went into production in 1966, the D-17 was replaced with a new computer that was the first high-volume use of integrated circuits. This process reduced the price of ICs from $1,000 each to $3 each which made it affordable to use them on commercial products [23].

The real era of Embedded dominance took off in 1992, with the foundation of the PC/104 Consortium by Ampro, RTD, and other manufacturers. The group established a format for Intel microprocessors based on a motherboard approximately four inches square, and just under an inch high. The boards were stackable, allowing a very powerful computer to be assembled in a box approximately four inches square, or even less [21]. Today, there are estimated to be well over 100 different companies making PC/104 products. There are PC/104 cards to add

ethernet, FireWire, hard drives, RAM drives, video cards, audio cards, general I/O, flash cards, modems, GPS, cellular telephone, wireless Internet, and more, to the PC/104 motherboard of your choice. References [23] quote that "the title of the first modern embedded system is often given to the Apollo Guidance Computer which was developed by Charles Stark Draper at the MIT Instrumentation Laboratory. Each spaceflight to the moon had two of these computers and they ran the inertial guidance systems of the command module and LEM. When the project began, the computer was considered the riskiest item as it used the new monolithic integrated circuits, to reduce the size and weight." The major events that marked the history of Embedded Systems were [24]:

- In 1968, Bob Noyce and Gordon Moore left Fairchild Semiconductor and formed Integrated Electronics (Intel).
- At Intel in 1971, Federico Faggin, Ted Hoff, and Stan Mazor invented the first single chip microprocessor, the 4004, a 4-bit microprocessor.
- In 1974, the 8008 and 8080, 8-bit microprocessors, were designed at Intel using NMOS technology.
- In 1974, Motorola also released the MC6800, an 8-bit microprocessor.
- During early 1980s, microcontrollers began to be designed optimized towards power and physical size.
- Intel came out with the 8051 microcontroller; while Motorola produced the 6805, 6808, 6811, and 6812.
- In 1999, Motorola shipped its 2 billionth MC68HC05 microcontroller.
- In 2004, Motorola spun off its microcontroller division as Freescale Semiconductor.

1.4 Embedded Solutions Continue to Flood Market

Around a decade ago (in 1995), Mary Ryan, in EEDesign, has wrote "… but embedded chips form the backbone of the electronics driven world in which we live… they are part of almost everything that runs on electricity" and today we are evidencing the same with the growth statistics in this sector. Following reports from various sources emphasizes the same.

- Of the nine billion processors manufactured in 2005, less than 2% became the brains of new PCs, Macs, and Unix workstations. The other 8.8 billion went into embedded systems [6].
- Recently published research by Venture Development Corporation (VDC) concludes that over 4 billion embedded systems/devices were shipped worldwide in 2006. According to VDC's 2007 Embedded Systems Market Statistics report, significant growth in the number of embedded shipments is expected to continue over the coming years [7]. This well known independent technology market research and strategy consulting firm has also predicted that through 2009, the number of embedded devices shipping with a commercial and/or open source

operating system will grow at a faster rate than shipments of devices with an in-house/proprietary operating system or with no formal operating system.

- The prospects for growth of Linux adoption in the mobile and embedded space are significantly promising. According to the Canalys report on Q2 2007 market share, Linux holds 13.3% of the global smartphone market, which puts it ahead of the Windows, BlackBerry, and Palm operating systems. In China, where the smartphone market is huge and growing at an extremely rapid pace, Linux is used on 30% of all smartphone handsets [8]. It is further predicted that the year 2008 won't be the Year of the Linux Desktop, but there will be more rapid growth in the mobile and embedded markets as Linux-based phones and ultraportable products emerge and gain popularity.
- Between 2006 and 2010, the market volume for automotive microcontrollers will expand about 63%, concludes a study from market researcher Frost & Sullivan [10]. The main factor to drive the demand is the proliferation of electronic content in vehicles aiming at reducing human errors as well as the increasing number of safety features such as radars, ultra sonic sensors and multiplexing with all of them requiring increasing amounts of processing power and intelligence. The study forecasts the market to grow from $5.83 billion in 2006 to $9.52 billion in 2010.
- A new comprehensive analysis on the Microcontroller market predicts that 2007 worldwide microcontroller revenue will increase by 10% to nearly $14 billion. The fastest growing segment within microcontrollers is the 32-bit market, which is estimated to be growing at a compound annual growth rate of 16% each year, compared to the overall market for microcontrollers which should garner around 8% growth each year on average [11].
- The worldwide portable flash player market exploded in 2003 and is expected to grow from 12.5 million units in 2003 to over 50 million units in 2008 [20].

1.5 Latest Trends in Embedded Systems

With the ever pervasive requirement, Embedded systems are being influenced by several factors such as interoperability, security, cost and openness. These issues are being discussed in forums such as IEEE for standardization and policy making [26].

- The field of embedded systems is likely to grow by leaps and bounds due to the prevailing need of making the computer transparent and ubiquitous.
- TCP/IP, embedded browsing, and Java will be latest buzz words in this sector.
- A new paradigm of IP-less addressing scheme based on properties or content is going to be developed due to the unsuitability of the traditional IP suite for the embedded nodes.
- Embedded microprocessor oriented towards server I/O, built in networking protocols will be more used.

- A huge potential exists for microelectronic mechanical systems, so that these cheaper and smaller sensors and actuators can be employed to create ubiquitous smart environments.
- With the increasing significance to the ‘connectivity’ theme, the embedded products need to adopt standards at hardware, software and middleware levels. Only this will ensure interoperability between these devices.
- As most of the applications are small and works even with the 8 bit functionality, there is going to be personalized level development in this field which is fuelled by the availability of the software tools in the form of freeware.
- Reconfiguration is a key issue as the state of art embedded applications demands online debugging, self healing and correcting by rebooting the system software.
- Scalability, security, real time response and high availability will be more important issues for the Embedded Systems.
- Software is going to be a deciding factor as there is lot of constraint on the memory. The emerging flash technology will certainly decide the cost of the product.
- Embedded operating systems will be facing the tradeoff of compactness Vs providing full functionality and sophistication.
- Almost 60% of all the processors used are 8-bit technology, because not only they satisfy the requirement but also leads to the cost effectiveness.

1.6 Competition for Processing Cores in Embedded Systems

Microcontrollers were developed out of the need for small, low power systems. They do not have the expandability or performance as compared to the microprocessors which came into the market much before. The main intention behind their development is to use them in domains such as control, consumer applications such as personal electronic devices, defense applications and office appliances such as facsimile machines, printers, etc. where the general purpose architecture of microprocessors turns out to be negative in several respects. Although these days there is a growing trend to use the customized microcontrollers popularly known as flexible microcontrollers from third party vendors such as picoblaze from Xilinx or Nios from Altera on a FPGA platform, the importance of the 8 bit microcontrollers such as PIC or MCS51 series has no way affected. It has been reported that “the next-generation automotive electronic systems need highly specialized, cost-optimized devices to meet market requirements. Considering the dramatic increase in development costs for state-of-the-art process technologies, specialization of traditional microcontrollers no longer makes business sense. Neither do feature-rich devices targeted at broad-base markets, as they are often too expensive. Alternatively, the flexible microcontroller solution offers a process to develop the exact microcontroller for a specific application by implementing it into an FPGA for prototyping [58].” However, for the most of the day to day applications the capabilities and

onchip resources of PIC or MCS51 series devices are no way proved to be bottleneck. That's why we are witnessing the penetration of these tiny chips which are hidden inside a surprising number of products such as a microwave oven, a car engine, home automation, TV, VCR just to name a few. Although the FPGA's have strikingly powerful features such as programming and reprogramming as needed during the design process, rapid prototyping and faster time-to-market, field upgradeability, etc. still these devices have to go a long way as a core of Embedded Systems designed by hobbysists, student and academic community. With this focal view, this book covers most of the needed stuff of the representative member of PIC and MCS51 series microcontrollers so as to inculcate their embedding as a processing core for the intended applications.

1.7 Programming Paradigm for Microcontrollers

Embedded systems programming is the programming of an embedded system in some device using the permitted programming interfaces provided by that system [16].

Although initially the designers were skeptical about the usefulness of 'C' for microcontroller programming paradigm, later they found that there is nothing like it to program in C rather than the traditional assembly. 'C' is by now the most popular and widely used language for programming microcontrollers. Hardware programmers and firmware experts found many features of this language as most promising for effectively using the onchip resources provided by the manufacturers. It is therefore that C has been listed ahead of assembly, C++ and Java in the popularity charts of Embedded Systems. For programming microcontrollers such as MCS51, PIC, AVR C is more useful owing to its closeness to the hardware. On the other hand C++ tends to be used for large programs where the object oriented features can be used to advantage. As far as the Embedded sector is concerned there is a very remote possibility that C++ will replace C in near future. The main competitor of 'C' is the assembly language, which has been outlasted by C in widespread use. However in the mainstream general purpose programming paradigm, languages such as Java are however more intended to replace C++.

Some of the high level features of the traditional 'C' have been revised and customized by the IDEs of the microcontrollers to access the hardware resources effectively.

Pure C lacks in the some of the things such as [9]:

- Impossible to check whether or not there were an overflow after arithmetic operation (in order to check this you have to read overflow flag).
- It is impossible to organize multithreaded operations, because for this you should save register values to save the states.

The above aspects have been incorporated in the Embedded C and most of the IDEs. Most of the underlying concepts regarding the 'C' for microcontroller programming have been covered in our latest book published by Springer [27].

1.8 Our Approach: "Towards a Full Proof 'C' Library for Embedded Systems"

A famous saying about software "If architects built houses the way software engineers built software, the first woodpecker that came along would destroy civilization" is seen more valid for the embedded software. That's why this book gives a complete listing of the 'C' programs keeping aside the theoretical algorithms, flowcharts or pseudo codes. We are adopting this approach for several reasons. First and foremost is there are several texts giving theory and we want to go away from this and present something useful and executable stuff for testing in the laboratory. Secondly, with many years of experience in this field we found that the software is the main culprit in project failure. This view is validated with sound theoretical analysis which entails that the underlying hardware of the embedded product has been evolved and became almost standard. Today many standardization frameworks such as IEEE exists forcing the manufacturers to obey the certifications and incorporate measures to make their product platform independent, fault tolerant to some extent and interactive by following widely accepted communication protocols. However, the software part which keeps on changing per application, has been at the mercy of the programmers.

An interesting article [28], explains why software has been the most crucial part of Embedded Systems. Here the authors deliberate the reasons as:

- Increasing complexity of the software as compared to hardware due to an infinite number of possible execution paths, handling of huge data, etc. The hardware design comparatively has to follow less number of complicated states.
- Software is extremely sensitive to errors. A single incorrect setting or resetting of the flag completely changes the program execution and gives wrong results.
- Software is difficult to test even with today's sophisticated simulators. Sometimes in the Embedded arena we need to see whether the simulated output will really come true or cause a catastrophic failure.
- As contrasted to hardware where one can define the test points clearly, the software is hardly prone to check the correlations between various variables used in the program. No doubt the watch windows are provided to help debug, but practically one can't keep all watch window per variable.
- Lastly unlike the hardware there is lack of professional standards for software.

It is aid that the software project failures have a lot in common with airplane crashes. Just as pilots never intend to crash, software developers don't aim to fail. When a commercial plane crashes, investigators look at many factors, such as the weather, maintenance records, the pilot's disposition and training, and cultural factors within the airline. Similarly, we need to look at the business environment, technical management, project management, and organizational culture to get to the roots of software failures [29].

With the increasing importance to the embedded software, there is need to incorporate certain software failure case studies in academics as well as industry oriented courseware, which is yet to be done. Some interesting case studies related to system failure due to software lacunae are as follows.

The Therac 25 was supposed treat cancer patients and save their lives by zapping tumors with targeted blasts of radiation. Instead, the device delivered massive overdoses that killed three patients and injured several others because of software glitches by a lone programmer whose code was never properly inspected and tested [30]. Ironically, today's most popular programming languages, C and C++, are among the most error prone. That's because C compilers have plenty of latitude to compile and link – without providing any diagnostics – code that can produce serious run-time errors, especially when ported to a new processor. Dan Saks, an author who has documented nearly 40 "gotchas" says that "There are a lot of little goodies in C that programmers are not fully aware of," "The lesson is to understand what you can assume and what you can't [31]."

1.9 Finalizing Hardware

This book describes two types of microcontrollers for designing embedded systems. Chapter 2–5 are based on PIC series 16F877 basically a RISC architecture. Chapter 6 is based on Atmel 89S52 microcontroller designed with CISC philosophy. Almost all the books focus on only a single microcontroller. Then, why two in this book? The reason behind using two types of microcontrollers is to demonstrate their basic capabilities and their proper choice for the intended application. Thus a designer can practically experience the basic difference between them and decide where the square peg fits in the round hole.

The basic bottlenecks [32] in using the CISC microcontrollers for the emebedded applications are as follows:

- Long and unpredictable interrupt latencies (slowing down the entire application)
- Vast instruction sets give slow decoding (complexity of decoder is nore)
- Frequent accesses to Memory which are generally slower devices
- Slow procedure calling
- Strictly one job at a time (dedicated ISR per interrupt or device)
- Software has to be structured to suit architecture (to facilitate fast task switching)
- Redundant instructions and addressing modes (popular quote: 20% instructions used 80% of time)
- Inconsistent instruction sets
- Bus not fully utilized (idle buses most of the time as the decoder is all the while busy due to complex instruction set)

On the other hand the RISC processors has very simple instructions operating at a faster speed, almost one instruction per clock cycle. Hardwired implementation of the control unit in RISC gives them VLSI area saving advantage compared to the micro-code implementation of their counterparts. Few more appealing points of RISC architecture [33] are as follows:

- RISC employs initial design methodology to incorporate only frequently used instructions in the instruction set and thus achieves the same function performed by a much more complex instruction in a CISC.

- RISC processor has a large number of general-purpose registers, largely reducing the frequency of the most time consuming memory access.
- Interms of clock rate, the RISC with its much simpler circuits can have a higher clock rate that again increases the performance of a processor.
- The RISC processor can provide processing power more than three times of a CISC processor in a particular field of application.

The RISC disadvantage sometimes is the difficulty in the design of compilers. Interestingly many embedded computer users have known and taken advantage of the benefits of RISC (reduced instruction set computers) versus ×86 CISC (complex instruction set computer) processors for years. But one of the main advantages has always been one of the main drawbacks – the variety of RISC choices has prevented a standard approach to specifying portions of the RISC design. The choice eventually came down to designing a custom RISC-based single board computer (SBC), or choosing an industry-standard form-factor – even if it didn't quite fit the application's space or I/O requirements [34].

1.10 Exploring PIC16F877 for Embedded Systems

Historically, PIC (Peripheral Interface Controller) is the IC which was developed to control peripheral devices, alleviating the load from the main CPU.
The core features of PIC16F877 as per the data sheet [35, 36] are as follows:

- 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 8 K × 14 words of FLASH Program Memory,
- Up to 368 × 8 bytes of Data Memory (RAM)
- Up to 256 × 8 bytes of EEPROM Data Memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low power, high speed CMOS FLASH/EEPROM technology
- Fully static design
- In-Circuit Serial Programming (ICSP) via two pins
- Single 5 V In-Circuit Serial Programming capability

- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0–5.5 V
- High Sink/Source Current: 25 mA
- Commercial, Industrial and Extended temperature ranges
- Low-power consumption:
  –<0.6 mA typical @ 3 V, 4 MHz
  –20 μA typical @ 3 V, 32 kHz
  –<1 μA typical standby current

PIC16F877 is a popular choice of Embedded community. The foremost reason is that it offers high pin count and minimal cost.

Many developers have come out readymade programmers for PIC16F877. Good ones we found on the Internet are

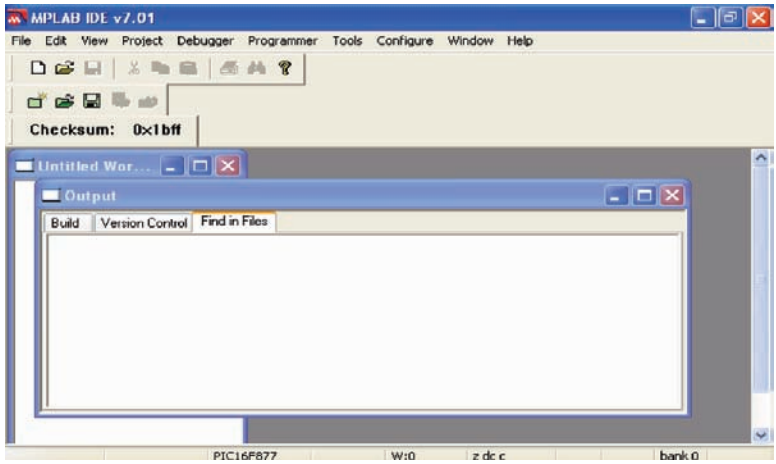
- http://www.angelfire.com/ok3/masterbyte/ A project to home-built a reliable PIC 16F87X based programmer and development system. Moreover, a 32 bit native software has been developed around this programmer. It is written on the webpage that it can also program the PIC16F84 and the I^2C serial EEPROM 24LC16.
- http://feng3.cool.ne.jp/en/pg5v2.html "Multi PIC Programmer 5 Ver.2" is a PIC programmer, which can program to 8-pin to 40-pin devices using single ZIF socket. It is in order to enable it to program 40-pin devices like PIC16F877 with a ZIF socket.
- http://www.ise.pw.edu.pl/~wzab/picprog/picprog.html Although there are many free programmers for PIC16 × 84 microcontrollers, most of them do not support so called "margining" – verifying of programmed contents at different power supply voltages. But the one given here supports margining. The programmer is provided as "free hardware". Author has given permission to reproduce it and use for any purpose (even commercial) without paying any fee, but why not visit his web URL and appreciate his efforts.

1.11 A Word About IDE

In the confines of a software project, "spending $2,000 on tools might save you $100,000 in programming effort," said Stewart of Embedded Research Solutions. True enough! Understanding the IDE plays an important role in simulation and programming. We have used MPLAB for the PIC16F877. MPLAB is an integration of a compiler, an assembler, a project manager, an editor, a debugger, simulator, and an assortment of other tools within one Windows application. A user developing an application can do many things such as code development, compile, debug and test and application with the MPLAB IDE. Most important thing is availability as a freeware and can be downloaded from website [37]. We want to introduce the MPLAB IDE and illustrate working with it in a step by step manner as follows:

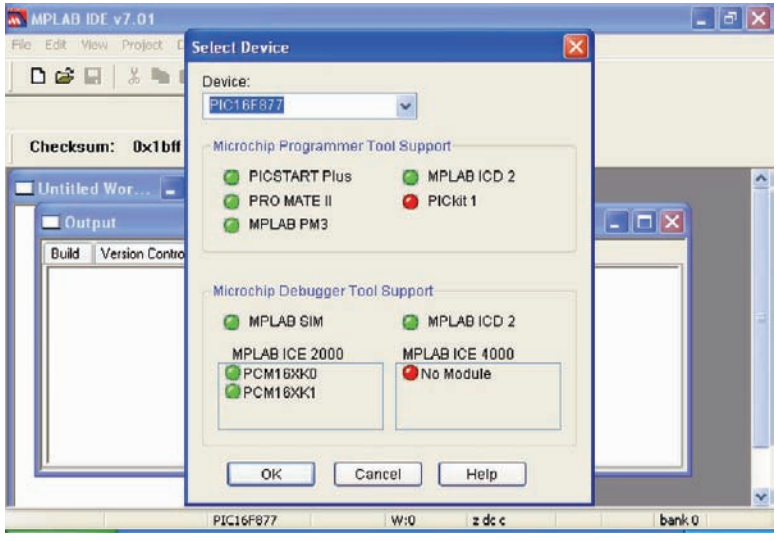
Step 1: Running MPLAB IDE

To start MPLAB IDE, double click on MPLAB IDE the following screen will appear.



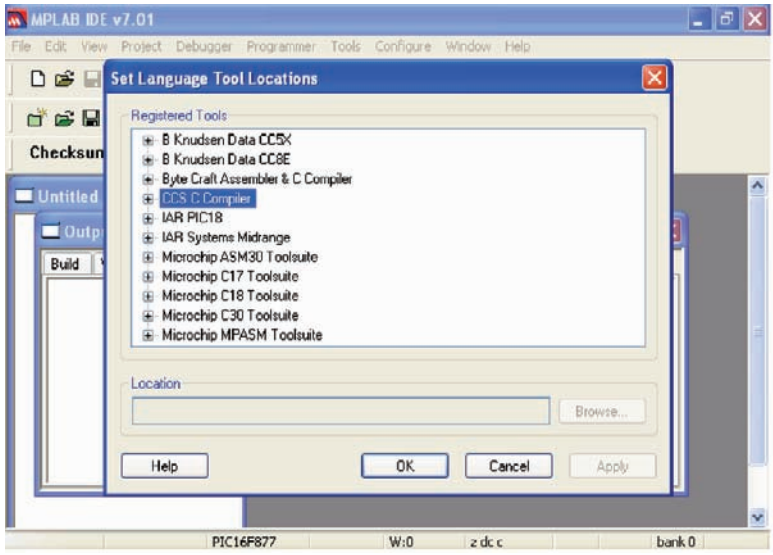
Step 2: Selecting the device

Click on configuration®click on select Device(PIC16F877) then following window will appear and click ok.

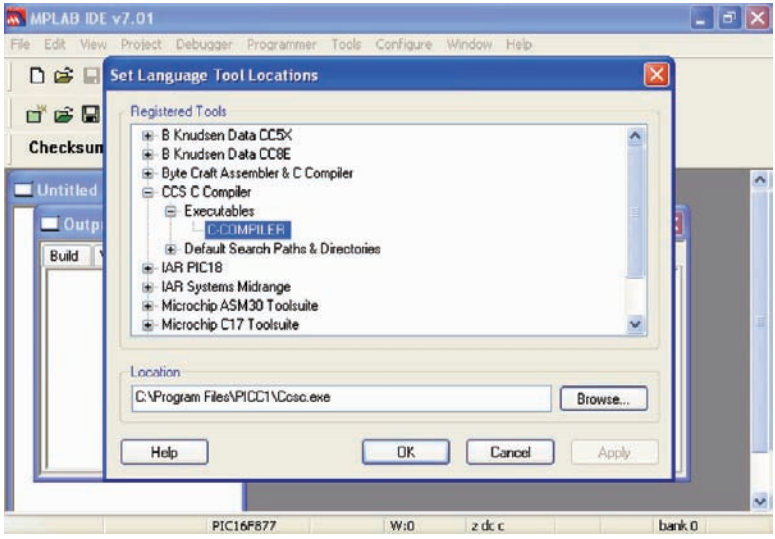


Step 3: Setting up language tool

On tool bar menu select project®set language tool locations, the window shown below will appear.

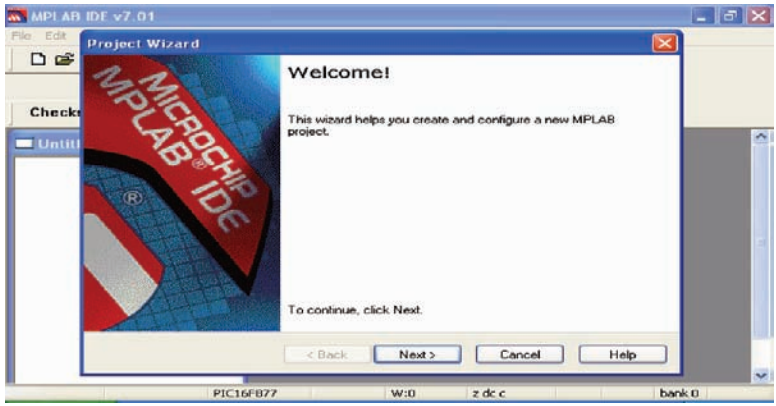


Expand the CCS C compiler and select C compiler under executables and Then press ok.

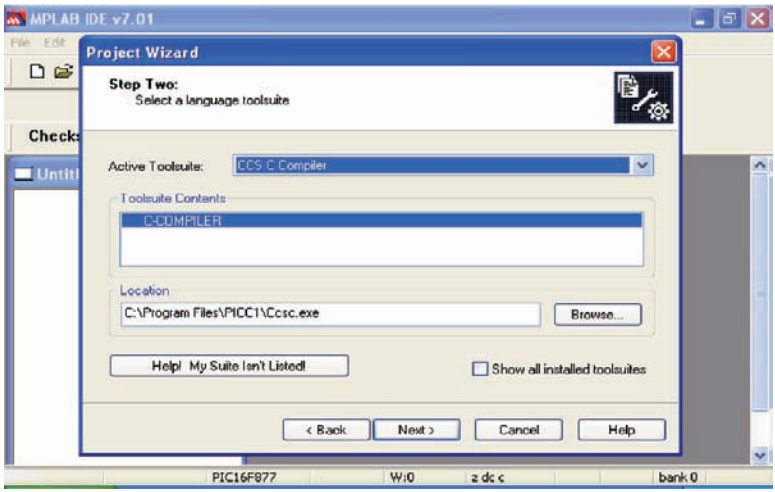


Step 4: Creating the project

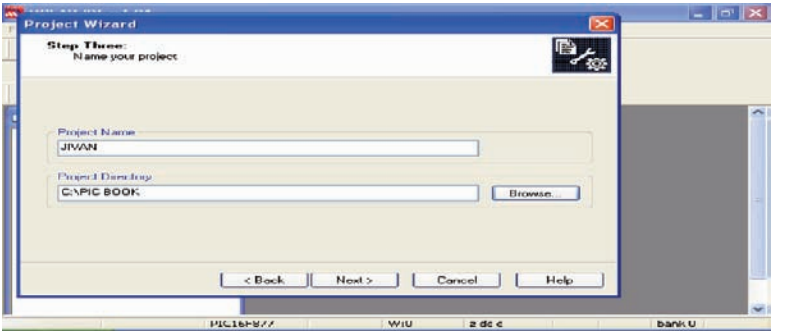
Chose project®Project wizard, the following welocome window will appear.



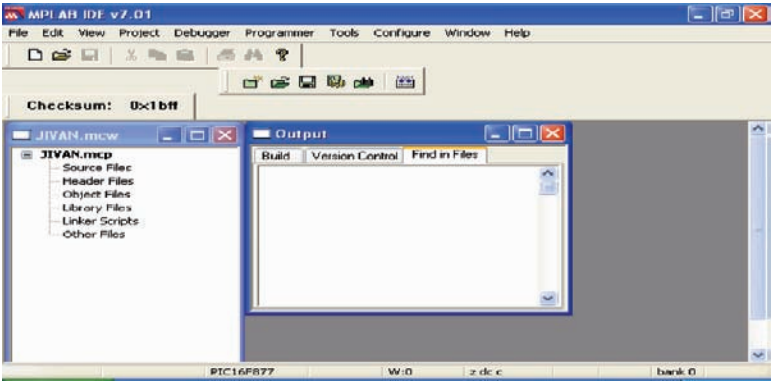
Click on next to continue the next dialog (Step 1) allows you to select the device, which you have already done make sure that it displays PIC16F877. If it does not, select the PIC16F877 from drop down menu. Then click next again it displays following window.



Click on next that will allows you to select project working directory and Name to the project as shown below, then click next®next®finish.

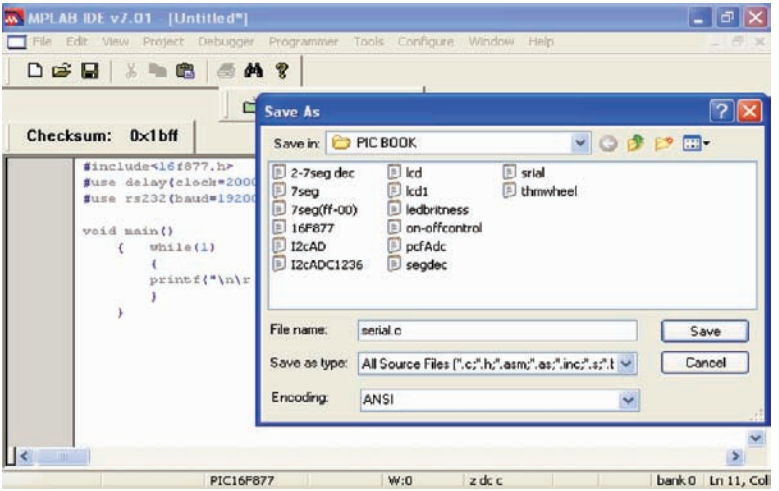


Then click next®next®finish the follow window will appear.



Step 5: Create new file

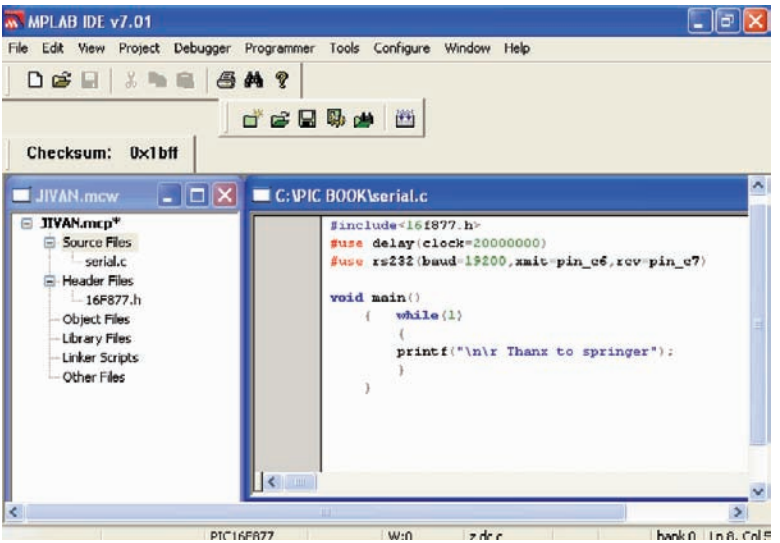
Choose file→new, type the you c code for respective application and save it with C extension as shown below.



Step 6: Adding files to the project

Right click on source file in the project window and then add your c file e.g. serial.c, then right click on header files, then browse the PIC16F877 header file where PIC C compiler is installed, e.g. C:\Program files\PICC\Devices\16F877.h.

After doing above steps the final window appears as shown below.



Step 7: Building project

Click on project®build option-®project, then select PCM option in compiler then click ok.

Step 8: Programming the device with any available programmer or the ones mentioned in previous article.

1.12 Details About the AT89S52 and Its Development Environment

The last chapter of this book is solely on AT89S52. We are not giving here anything regarding the device, its IDE and other stuff. Interested readers may go through our previous book [27]: "Exploring C for Microcontrollers: A Hands on Approach" authored by Jivan S. Parab, Vinod G. Shelake, Rajanish K. Kamat and Gourish M. Naik and published by Springer recently in May 2007. In addition most of the essential details may be explored by referring the references [37–40].

Chapter 2
Interacting with the Outside World Using Simple I/O Devices

The microcontroller interacts with the outside world by means of devices such as LEDs, relays, LCDs extra. This chapter covers all the interfacing aspects of PIC to the outside world. The interfacing aspects in general comprises of solving the current/voltage in compatibility issues, introducing drivers, current protection resistors, port saving measures and satisfying the timing requirements. A good reference to solve these incompatibility issues may be seen in the online URL given at reference [53]. The code developed in this chapter is quite useful and may be treated as a software library in any project design.
Following case studies are developed:

2.1 LED Interfacing

LEDs the tiny semiconductor devices, passes an electric current to in only one direction and produce light as a byproduct of current flow. These days they have challenged the mere existence of the fluorescent lights, as they run cooler and last longer. The lighting industry is being dominated by LEDs and they are now dominating the commercial world, with their optimum energy use and lesser maintenance costs. Recently even the video screens, animated signs, traffic lights and home lights are some of the domains being ruled by the arrays of tiny LEDs.

The role of LED's as a binary indicator for indication and diagnostics is undisputable. Although it is one of the most simplest interfacing, it presents a range of opportunities when used for applications such as advertisement displays, home

J.S. Parab, et al., *Practical Aspects of Embedded System Design using Microcontrollers,*

show pieces, etc. Many of the embedded system products makes use of the SMD LED's (owing to manufacturing ease) although they are less brighter than their through hole counterparts.

The LEDs can be interfaced to the microcontroller in either in 'active high' or 'active low' mode. However the 'active low' mode gives certain advantages as grounding the cathode is much easier and reliable than outputting logic 1 from the port pin. With the active low mode the sinking capacity of the microcontroller is more important than the sourcing. In this module three case studies pertaining to the LED interfacing have been developed useful for different applications in Embedded Systems.

Program 2.1 Blinking of the LED

This application is analogous to the "Hello World" program for the software engineer in making. This simplest LED interfacing (Fig. 2.1) exercise helps to kick start the programming and enhances the confidence level. It also clarifies the basic infinite loop theory of the embedded systems by using while{1} construct. The concept of passing the delay values to the built in delay routine in the PIC C is demonstrated.

Program Source Code

```
*********************************************************************************
// Program to illustrate the blinking LEDs connected to the Port B of PIC16F877.
--------------------------------------------------------------------------------

#include<16f877.h>// Include the header file of the PIC16F877 device.
#use delay(clock=20000000)// Use the clock of 20 MHz for the delay
void main()// Start of the main program.
{
while(1) // loop for the LED ON-Off.
```

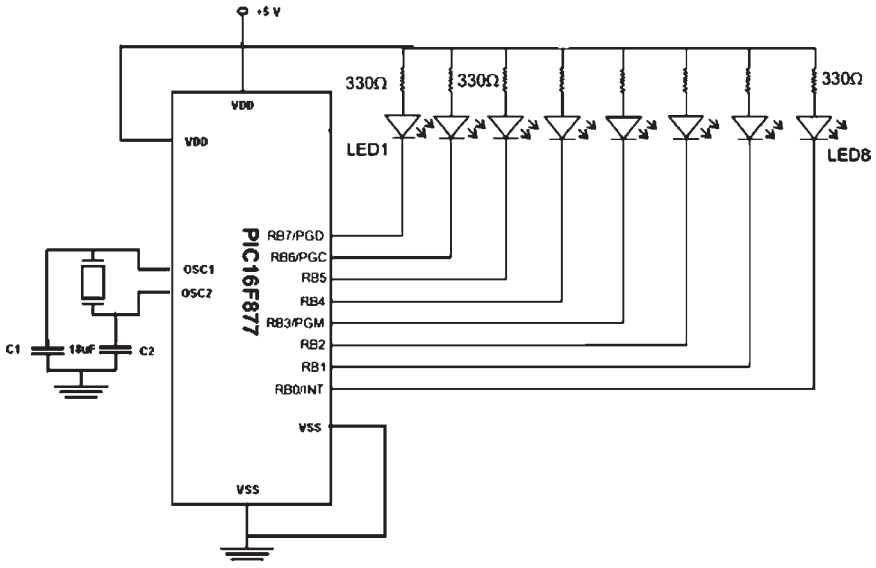


Fig. 2.1 LED interface to PIC16F877

```
   {
      output_b(0×ff);        // Out 0XFF to the Port B.
      delay_ms(500);         // Delay of 500 ms.
      output_b(0×0);         // Out 0X 00 to the port B.
      delay_ms(500);
}// end of for loop.
}
```

****

Blinking frequency variation can be verified by changing the delay count.

Program 2.2 Blink and send the data pattern stored in array

The application is all about making a configurable LED arrays by sending the display data pattern through an array. A slight extension of this application with more LEDs connected to the other ports (which will form the LED matrix) is quite useful for displaying the moving images with limited animation. In the present application, patterns to be displayed are stored in an array and sent to the LEDs interleaving a short blinking.

Program Source Code

****

```
/* Program for the LED interfacing to the PIC16F877. LEDs are connected to the
port B. The LEDs are ON-Off with the predefined software delay. And after ON-
Off 10 times send the data stored in Array.
Program to illustrate the blinking and sending the data pattern stored in array to the
LEDs connected to the PIC16F877.*/
```

```
#include<16f877.h> // Include the header file of the PIC16F877 device.
#use delay(clock=20000000) // Use the clock of 20 MHz for the delay
unsigned int led[]={0×FE,0×FD,0×FB,0×F7,0×EF,0×Df,0×Bf,0×7f};
unsigned int i;     // define the integer.
void main()         // Start of the main program.
{
while(1) // loop for the LED ON-Off.
   {
   for(i=0;i<10;i++)          // switch leds ON-OFF for 10 times
   {
      output_b(0×ff);         // Out 0×FF to the Port B.
      delay_ms(500);          // Delay of 500 ms.
      output_b(0×0);          // Out 0× 00 to the port B.
      delay_ms(500);
   } // end of for loop.
   for(i=0;i<8;i++)           // send the sequence stored in arrray
      {
      output_d(led[i]);       // Send the Data stored in the Array.
      delay_ms(500);          // Delay of 500 ms.
      }
   }       // End of While loop
}          // End of main.
```

****

Program 2.3 LED ON-OFF by using shift operators

This application is one step towards "scrolling marquee LED signs". Such applications make use of the IC shift registers for the purpose of scrolling. However, the same is achieved by using the 'shift' operator that scrolls the data pattern stored in an array in a sequential fashion.

Program Source Code

```
*********************************************************************************
//This Program for LED interface
//Data Lines – PORT B
_____________________________________________________________________________

#include<16f877.h>
#use delay(clock=20000000)
unsigned int i;
char a;
void main()
{
a=0×01;
      while(1)
      {
      for(i=0;i<10;i++){     // switch leds on OFF for 10 times
         output_b(0×ff);
         delay_ms(500);
         output_b(0×0);
         delay_ms(500);
         }
for(i=0;i<8;i++)
         {
         output_b(a);
         delay_ms(500);
   a=a<<1;                  // send the sequence stored in array
   if (a==0×0)
   a = 0×01;
}} }
*********************************************************************************
```

2.2 Switch (DIP) Interfacing

Program 2.4 DIP switch interfacing

A Dual In Line Package (DIP) switch is an electric switch that is packaged in a group in a standard dual in-line package (DIP). DIP switches are primarily used for setting up various configuration options. Some of the applications of the DIP switches are as follows:

- Configuring of defaults on computer boards
- Changing the baud rate on modems

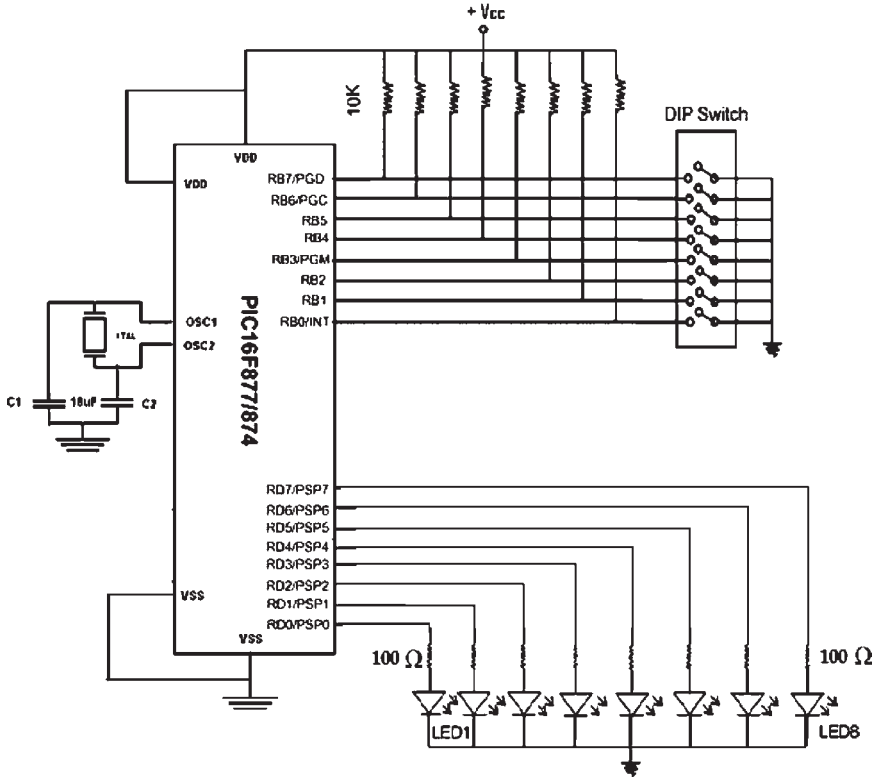


Fig. 2.2 DIP switch interface to PIC16F877

- Setting default options for video cards
- Configuration of pin-outs on cables
- Assigning address on the PC's system bus to an peripheral or I/O bus
- Selecting IRQs in PCs

The DIP switches can be interfaced to the microcontroller either by using a pull-up resistor (i.e. by connecting the switches to Vcc through a current limiting resistor) or in a pull-down mode with a small value resistor between the microcontroller pin and ground. The former method ensures the sinking current limitations of the microcontroller pin although it gives logic 0 for switch closure. The latter method as shown in Fig. 2.2 gives logic 1 for the switch closure, however results in a larger current dissipation during the switch closure. In this application status of the DIP switches is read through port B and the same is sent to the LEDs interfaced to port D.

Program Source Code

```
*********************************************************************************
/* This program will read the DIP switched connected to port B and outputs the
same on LEDS, which are connected to port D */
```

```
//        Data Lines (DIP switched is connected to PORT B
//        LEDs are connected to PORT D
```

```
#include<16F877.h>
#use delay(clock=20000000)
unsigned char read;        // Define the Character Read
void main()                // Start of the main function.
{
   output_d(0×ff);         // Out the 0X FF to the Port D connected to LEDs
while(1)
      {
   read=input_b();         // Read the data from the DIP switch connected to the
                              Port B.
   delay_ms(500);          // Delay of 500 ms
   output_d(read);         // Out the Read data to port D.
     }
}
```

****

2.3 Interfacing Buzzer

Piezoelectric sound components are known for their penetrating tones that are free of harmonics. These devices works on the principle of activating the piezoelectric diaphragm that consists of a piezoelectric ceramic plate with electrodes on both sides and a metal plate either of brass or stainless steel. With their low power requirement, these devices can be driven by the microcontroller port to produce high acoustic output. With the Application of the DC voltage through the port line causes these devices to produce mechanical distortion to the diaphragm. Even by applying an AC voltage/variable pulses using the timer counter of the microcontroller will move the diaphragm in a repeated bending motion, creating different sound effects. The devices such as Murata's PKB24SPC-3601-B0 piezoelectric buzzers [51] are useful for microcontroller interfacing.

Program 2.5 Buzzer interfacing

This application demonstrates a buzzer interfacing to the PIC microcontroller (refer Fig. 2.3).

Program Source Code

****

```
/* Program illustrates the interfacing of the Buzzer connected to the pin 0 of the
Port D. with the predefined software Delay. */
```

```
#include<16F877.h>
#use delay(clock=20000000)
void main()
```

```
{
   while(1)
{
      output_high(PIN_D0); // buzzer is connected to pin D0
      delay_ms(300);
         output_low(PIN_D0);
      delay_ms(300);
   }
}
```

****

Program 2.6 Activating buzzer with pin sense

This application is all about activating the buzzer upon pressing the push button key. It can be modified for many possibilities such as 'quiz buzzer', 'safety alarm', etc. Debouncing has to be done whenever the mechanical switches are involved. This is accomplished by introducing a software delay in this application.

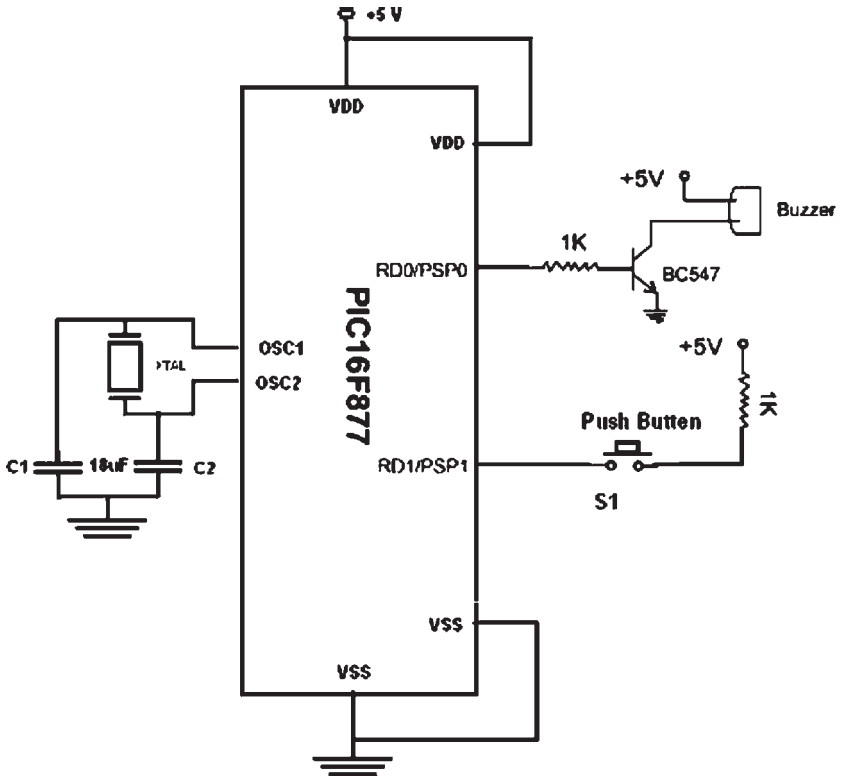


Fig. 2.3 Buzzer interface to PIC16F877

Program Source Code

```
*************************************************************************************
/* This program illustrate the Pin D1 is connected to push button which will act as
an input. The Status is indicated by the Buzzer activation connected to the port pin
D0.       /*
_____________________________________________________________________________
#include<16F877.h>
#use delay(clock=20000000)
void main()
{
      while(1)
{
      if(input(PIN_D1)==0)
      {
      output_high(PIN_D0); // buzzer is connected to pin D0
   delay_ms(50);
         }
         else
         {
      output_low(PIN_D0);
      delay_ms(50);
         }
            }
   }
*************************************************************************************
```

2.4 Keypad Interfacing

Almost all the embedded instruments require a front panel keypad interface to facilitate modifications of the instrument settings. The Series 96 is Grayhill's most economical 4 × 4 keypad family [1,2]. The contact system utilizes conductive rubber to mate the appropriate PC board traces. These keypads are offered in matrix circuitry, with shielded and backlit options. The specifications required for interfacing (refer Fig. 2.4) are reproduced here. More details are available from the company website available at reference 1. The 10 K resistor works as a pull up resistor for the active row input.
Specifications:

- Rating Criteria
- Rating at 12 Vdc: 5 mA for 0.5 s
- Contact Bounce: <12 ms
- Contact Resistance: <100Ω (at stated operating force)
- Voltage Breakdown: 250 Vac between components
- Mechanical Operation Life: 1,000,000 operations per key

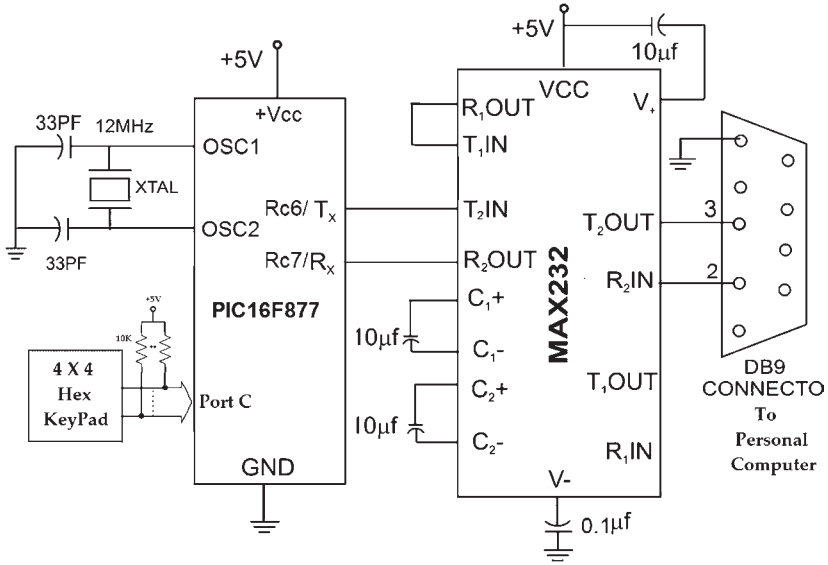


Fig. 2.4 Hex keypad interfacing

Program 2.7 Hex keypad interfacing

The program works on the principle of scanning the rows and columns. The key pressed is displayed on the PC through HyperTerminal.

Program Source Code

```
**************************************************************************************
//          Program for On-Board Key interface
_____________________________________________________________________________________
#include<16F877.H>
#use delay(clock=20000000)
#use rs232(baud=19200,xmit=PIN_C6,rcv=PIN_C7)
unsigned char row1[4]={0×fe,0×fd,0×fb,0×f7},value;
unsigned char row,col,colread,a,b,c,i;
void display(int );
void main()
{
        printf("nr Key Interface");
        while(1)
        {
        row=0×fe;
delay_ms(500);
        for(i=0;i<4;i++)
           {
      output_d(row);
            colread=input_d();
col=colread &0×f0;
```

```
   if(col<0×f0)
      {
      display(colread);
      }
else{
         row=row1[i];
}}}}
void display(int x)
{
   switch(x)
         {
         case 0×ee:
            value =0×00;
      putc('0');
            break;
         case 0×de:
            value =0×01;
      putc('1');
            break;
         case 0×be:
            value =0×02;
      putc('2');
            break;
   case 0×7e:
      value =0×03;
      putc('3');
         break;
         case 0×ed:
      value =0×04;
      putc('4');
         break;
         case 0×dd:
            value =0×05;
      putc('5');
            break;
         case 0×bd:
            value =0×06;
      putc('6');
         break;
         case 0×7d:
            value =0×07;
      putc('7');
         break;
         case 0×eb:
            value =0×08;
```

```
        putc('8');
              break;
    case 0×db:
              value =0×09;
        putc('9');
           break;
    case 0×bb:
              value =0×A;
        putc('A');
              break;
           case 0×7b:
              value =0×0B;
        putc('B');
              break;
           case 0×e7:
              value =0×0C;
        putc('C');
              break;
           case 0×d7:
              value =0×0D;
        putc('D');
              break;
    case 0×b7:
              value =0×0E;
        putc('E');
              break;
           case 0×77:
        value =0×0F;
        putc('F');
        break;
    }
    output_d(value);
}
*****************************************************************************
```

2.5 Thumbwheel Switches Interface

Thumbwheel switches are ideal for inputting data in terms of number or code for a manufacturing, process automation, and measurement systems. These switches are mounted at the front panel so that the data can be inputted by an operator.

Thumbwheel switches are also found in conjunction with the programmable logic controller (PLC) as a standard input device. They are available in three basic varieties viz. octal, binary coded decimal (BCD), and hexadecimal.

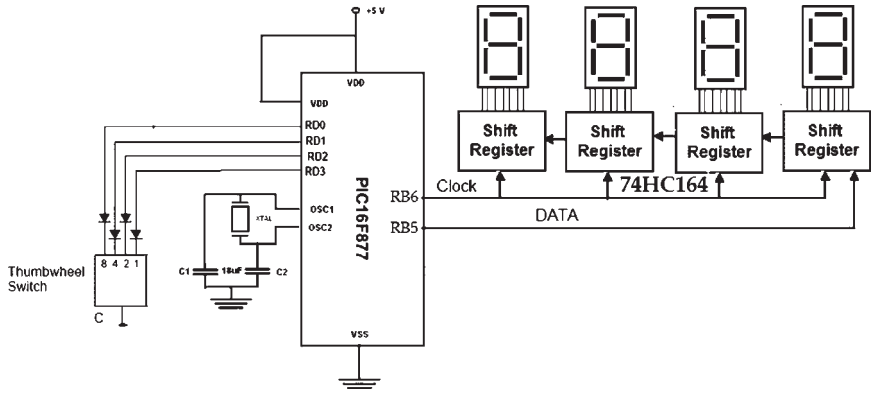


Fig. 2.5 Thumbwheel switch interface to PIC16F877

Standard thumbwheel switches are available from number of manufacturers [3–5]. One of the popular thumbwheel switches is EECO's 1400 series switches [3] that feature snap-in front mounting in a large, easy to use package size. Multiswitch assemblies can be created easily by snapping the switches together, with no extra hardware requirement. The 1400 series offers a choice of thumbwheel, push-set or lock-set actuator methods. The lock-set model eliminates the possibility of accidental setting changes in critical applications. The 1400 series offers several 10 and 16 position binary and decimal output codes. Options such as diode provision termination, stops and F-pins for P.C. board mounting make the 1400 series switch a popular choice among design engineers [3]. The diodes are used on the port lines for isolation purpose (Refer Fig. 2.5).

Program 2.8 Thumbwheel switch interface

Program Source Code

```
**************************************************************************************
//C program to Read thumbwheel switch and display the value on 4 7 segment
// Thumbwheel switch is connected to PORTD
// PB5 —>serial data for segment
//PB6 —> clock
--------------------------------------------------------------------------------------
#include<16F877.H>
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
unsigned int unit,tens,hundred,thousand;
char
seq[16]={0XC0,0XF9,0XA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90,0X88,
0X80,0XC6,0XC0,0X86,0X8E};
void display(char );
void clear(void);
void main (void)
```

```
   {
long int i;
int N;
while (1) {
N=input_d();
      unit =N%10;
         tens=(N/10)%10;
         hundred=(N/100)%10;
         thousand=(N/1000);
clear();
display (seq[thousand]);
delay_ms(100);
display (seq[hundred]);
delay_ms(100);
display (seq[tens]);
delay_ms(100);
display (seq[unit]);
delay_ms(100);
}}
void display(char k)
{
unsigned char mask;
int i;
mask=0×80;
for(i=0;i<0×08;i++)
{
         if(k&mask)
   output_high(PIN_B5);
   else
   output_low(PIN_B5);
   output_low(PIN_B6);
   output_high(PIN_B6);
   output_low(PIN_B6);
   mask=mask>>1;
}}
void clear(void)
{
int i;
for (i=0;i<33;i++)
{
output_high(PIN_B5);;
output_low(PIN_B6);output_high(PIN_B6);output_low(PIN_B6);
}
}
```

****

2.6 Seven Segment Display Interfacing

The seven segment display are still been used in Embedded Applications (where the users do not want compromise with the display intensity) in spite of the dominance of the LCDs. They basically comprises of arrangement of the LEDs in "Eight" (8) passion, and a Dot (.) with a common electrode, lead (Anode or Cathode). Two types of Seven segment displays are available in market. The first one is common anode in which all the anodes of the LEDs are connected together, leaving the cathodes open for connection. The second one is common cathode in which all the cathodes of the LEDs are connected together, leaving the anodes open for connection. The common cathode is more popular than its counterpart since grounding the cathodes is much easy and reliable than powering the common anodes. Few examples of the commercially available seven segment LEDs are as follows:

Part number/ trade name	Common cathode/anode	Company	Features
MAN6960	Common anode	FAIRCHILD	2,200 ucd luminous intensity @ 10 mA
LDS-C516RI	Common cathode	LITEON	2,200 ucd luminous intensity @ 10 mA
LDS-A516RI	Common anode	LITEON	2,200 ucd luminous intensity @ 10 mA
AND-2307SCL	Common cathode	AND Opto-Electronics	GaAsP/GaP–Red; GaP–Green 7 Segment, Large Size; 2.3″
AND-2307GAL	Common anode	AND	Display–green, OptoElectronics face–gray
MSQ6911C	Common anode	FAIRCHILD	2,200 ucd luminous intensity @ 20 mA. High efficiency. Right hand decimal
KW1-2302CS	Common cathode	GTC	Super bright, super big 2.3″ single digit, 42 mcd luminous intensity @ 10 mA

In this application common cathode seven segment displays is interfaced to the PIC on a multiplex basis. There are several advantages of going for a multiplexed display such as:

- Less wiring
- Low power driving electronics
- Above two issues leads to better economics
- Reduced power consumption as only one seven segment display will be operational at a time

The 'persistence of vision' mechanism of the multiplexed display works well for a limited number of displays.

As an illustration only two displays are interfaced (Fig. 2.6). This can be extended to eight displays using all the lines of port B.

Program 2.9 Display the count from 00-99 on two seven segment display

Program Source Code

```
**********************************************************************************
//This Program displays the count from 00-99 on 2- seven segment connected trough binary decoder
//segment are connected to PORTD
```

```
#include<16F877.H>
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
void main()
{
int i;
int uni, ten;
while(1){
for (i=0; i<100; i++)
{
uni = (i%10);
ten = (i/10);
uni = uni << 4;
output_d(ten|uni);
delay_ms(100);
}
}
}
```

****

Program 2.10 Down count from value set on DIP

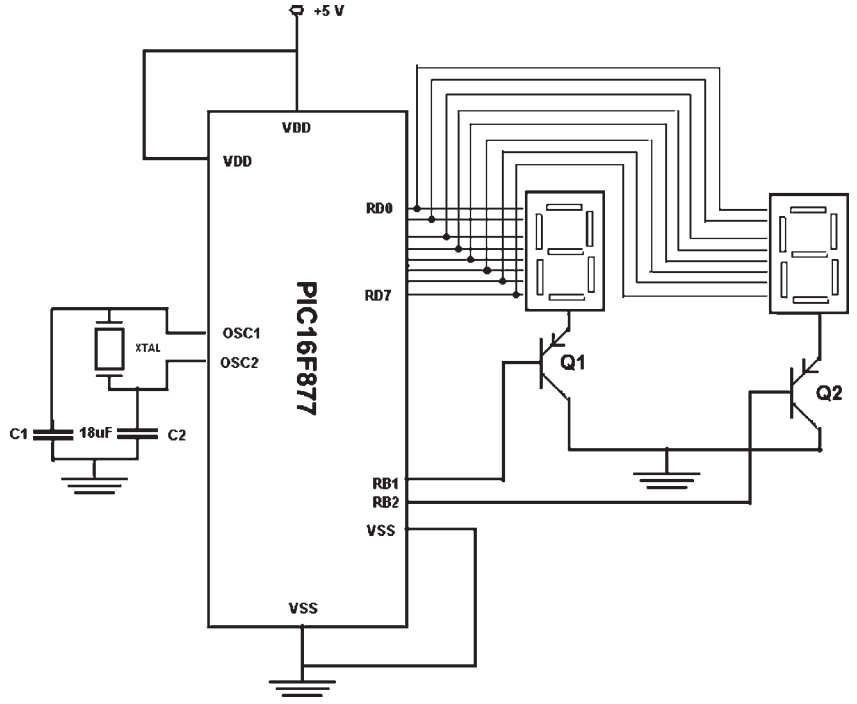


Fig. 2.6 Seven segment display interface to PIC16F877

In this program decoded output are not used but instead character sequences are generated.

Program Source Code

```
**********************************************************************************
// C program to down count from value set on DIP switch to 00 in the first 2 digit (hex).
// DIP switch is connected to PORTD
// PB5 —>serial data
//PB6 —> clock
_____________________________________________________________________________
#include<16F877.H>
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
unsigned int unit,ten,hundred,tenth;
char
seq[16]={0XC0,0XF9,0XA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90,0X88,
0X80,0XC6,0XC0,0X86,0X8E};
void display(char );
void clear(void);
void main (void) {
long int i;
int N;
while (1) {
N=input_d();
for (i=N;i>=0;i–)
   {
      unit =(i%16);
      ten=(i/16);
clear();
display (seq[ten]);
delay_ms(100);
display (seq[unit]);
delay_ms(100);
}
}
}
void display(char k)
{
unsigned char mask;
int i;
mask=0X80;
for(i=0;i<0x08;i++)
{
   if(k&mask)
   output_high(PIN_B5);
   else
```

```
   output_low(PIN_B5);
   output_low(PIN_B6);
   output_high(PIN_B6);
output_low(PIN_B6);
   mask=mask>>1;
}
}
void clear(void)
{
int i;
for (i=0;i<33;i++)
{
output_high(PIN_B5);;
output_low(PIN_B6);output_high(PIN_B6);output_low(PIN_B6);
}}
```

****

In this application, the PIC starts counting up from the binary values set by the DIP switches (refer Fig. 2.7).

Program 2.11 Up count from value set on DIP

Program Source Code

****

```
// C program to up count from value set on DIP switch to 00 in the first 2 digit (binary).
// this program can be easily modified for the 4 digit display.
// DIP switch is connected to PORTD
// PB5 —>serial data
//PB5 —> clock
```

```
#include<16F877.H>
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
unsigned int unit,ten,hundred,tenth;
char
seq[16]={0XC0,0XF9,0XA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90,0X88,
0X80,0XC6,0XC0,0X86, 0X8E};
void display(char );
void clear(void);
void main (void)
{
long int i;
int N;
while (1) {
N=input_d();
for (i=N;i>=0;i–)
      {
         unit =(i%10);
```

```
        ten=(i/10);
clear();
display (seq[ten]);
delay_ms(100);
display (seq[unit]);
delay_ms(100);
}}}
void display(char k)
{
unsigned char mask;
int i;
mask=0X80;
for(i=0;i<0×08;i++)
{
   if(k&mask)
   output_high(PIN_B5);
   else
   output_low(PIN_B5);
   output_low(PIN_B6);
   output_high(PIN_B6);
output_low(PIN_B6);
   mask=mask>>1;
}}
void clear(void)
{
int i;
for (i=0;i<33;i++)
{
output_high(PIN_B5);;
output_low(PIN_B6);output_high(PIN_B6);output_low(PIN_B6);
}
}
```

//Note: The above programs can be modified to display four digits by adding more seven segment displays.

****

2.7 LCD Interface to the PIC

The embedded projects becomes spicy with the addition of alphanumeric LCD that facilitates the user instructions as well as project response in alphanumeric form which makes the application professional and easy to use. It not only enhances the

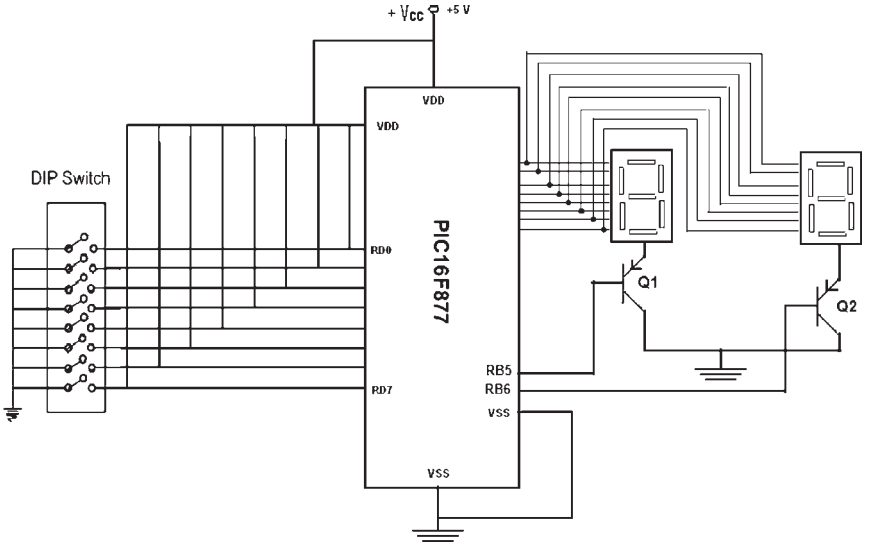


Fig. 2.7 DIP switch and seven segment display interface to PIC16F877

presentation aspects but eases the debugging process by setting single stepping, breakpoints and interrupts wherever required.

HD44780 Character LCD is a popular industry standard liquid crystal display (LCD) display device designed for interfacing with microcontrollers. It has capability to display in 16 × 2 configurations. These LCDs are found in many appliances such as copiers, fax machines, laser printers, industrial test equipment, networking equipment such as routers and storage devices, etc. to name a few.

Manufacture's data sheet [6] and even many web pages [7] covers the commands for the LCD. By adding a shift register a two wire interface for the LCD is also been developed [52]

Pin configuration for LCD HD44780

Pin number	Description
1	Ground
2	Vcc
3	Contrast voltage
4	'R/S' Instruction/register select • By setting the bit, byte at the current LCD "cursor" position can be read or written • Resetting the bit indicates, either an instruction being sent to the LCD or the execution status of the last instruction being read back
5	'R/W' Read/write LCD registers
6	'E' Clock Used to initiate the data transfer within the LCD
7–14	Data I/O Pins

Program 2.12 LCD Interface to PIC (Refer Fig. 2.8)

Program Source Code

```
*********************************************************************************
// This program the string “This is our 2nd book on embedded system” on the
LCD.
_____________________________________________________________________________
#include<16F877.h>
#use delay(clock=20000000)
unsigned int array[]={“Thanks to publishers”};
unsigned char a,i,b,j;
void INITlcd(void);
void ENABLE (void) ;
void LINE(int);
void main()
{
   while(1){
      INITlcd();
      LINE(1);
   for (i=0;i<16;i++)
   {
      if(i==8)
   LINE(2);
   b=array[i];
   output_d(b);
   ENABLE();
}
   output_low(PIN_B0);
   output_d(0×01);
   ENABLE();
   output_high(PIN_B0);
LINE(1);
for (i=16;i<33;i++)
   {
   if(i==24)
   LINE(2);
   b=array[i];
   output_d(b);
   ENABLE();
}
}}
void ENABLE(void) {
output_high(PIN_B2);
delay_ms(10);
output_low(PIN_B2);
delay_ms(10);
```

```
}
void LINE(int j){
   if (j==1) {
   output_low(PIN_B0); output_low(PIN_B1);
   output_d(0×80);
   ENABLE();
   output_high(PIN_B0);
   }
   else
   {
   output_low(PIN_B0);
   output_low(PIN_B1);
   output_d(0×C0);
   ENABLE();
   output_high(PIN_B0);;
   }}
void INITlcd(void)
{ output_low(PIN_B0);//RS
output_low(PIN_B1);//RW
output_low(PIN_B2);//EL
output_d(0×38);
   ENABLE();
ENABLE();
ENABLE();
ENABLE();
output_d(0×06);
ENABLE();
   output_d(0×0E);
   ENABLE();
   output_d(0×01);
   ENABLE();
}
*********************************************************************************
```

2.8 Relay Interface to the PIC

Interfacing electromagnetic relays to the microcontroller port poses several challenges as they require more driving current than that supplied by the port pin. Moreover the relay may cause damage to the port itself. This is more true regarding the reed relays. The back emf issue can be sorted out by connecting a free wheeling diode of appropriate reverse rating across the relay coil. The current requirement of the relay can be met by introducing a driver transistor. For added isolation it is recommended to employ an optocoupler such as CNY17, PC123 between the port

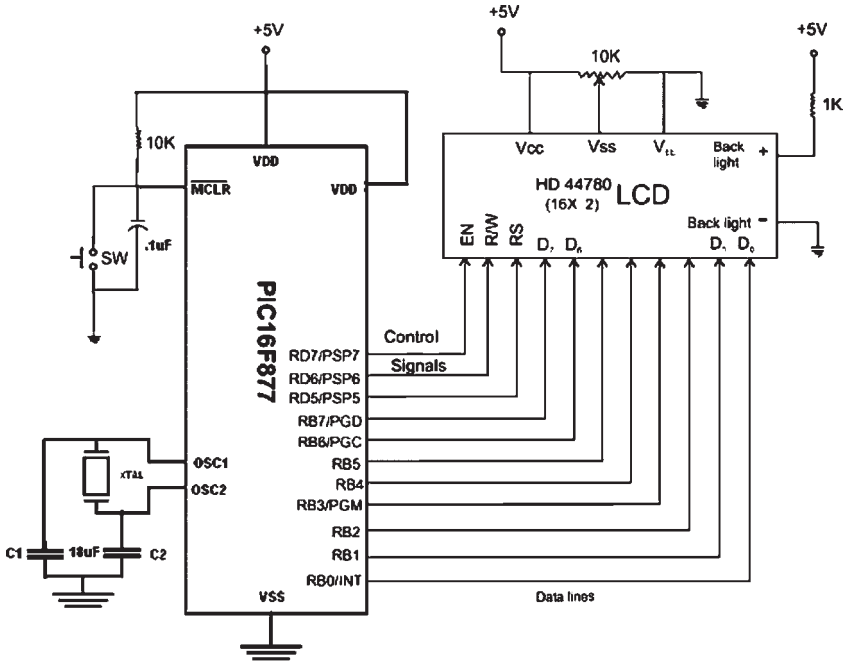


Fig. 2.8 LCD HD44780 interface to PIC16F877

lines and relay driver. In case the current requirement of the relay is not getting sufficed by the driver transistor then a Darlington Pair like TIP122 can be used which can go safely upto 5 A of current. Another good option could be a IC driver such as ULN2003 to drive multiple relays in one go through the microcontroller ports.

Program 2.13 Relay interfacing (Refer Fig. 2.9)

Program Source Code

```
*******************************************************************************
// this program illustrates the switching on-off relay based on pin sense.
_____________________________________________________________________________
#include<16F877.h>
#use delay (clock=20000000)
void main()
{
   while(1)
   {
   if(input(PIN_B3)==1)
      {
      output_high(PIN_B4); // relay is connected to pin B4
      delay_ms(500);
```

```
  }
        else
        {
     output_low(PIN_B4);
     delay_ms(500);
        }
  }}
```

****

Program 2.14 Relay on off based on temperature sense

Program Source Code

****

/* Program Illustrate the Relay on off on the basis of temperature sensed form the Temperature Sensor LM35. */

```
#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
int adcdata,chadress,dataavg=0;
```

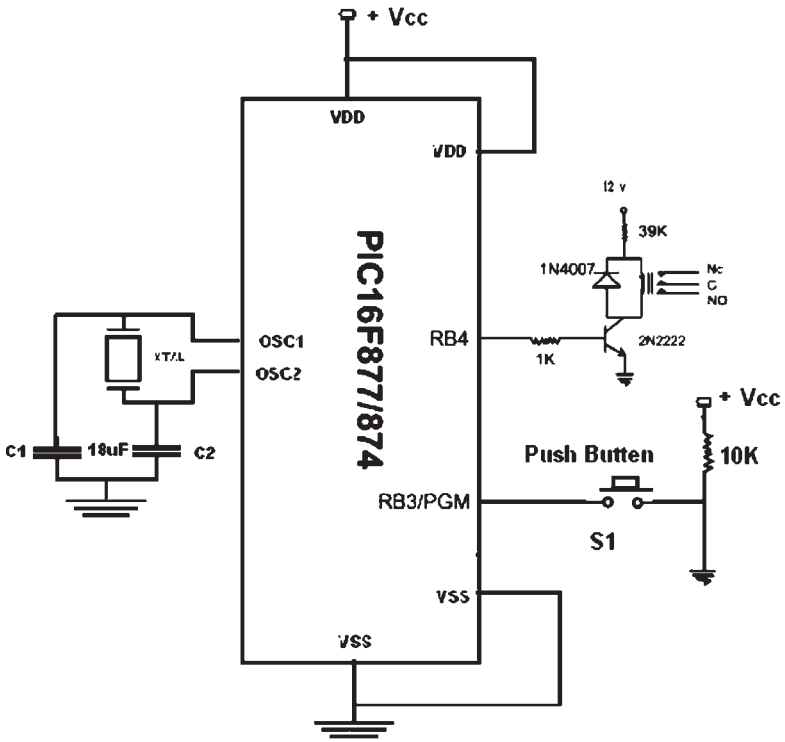


Fig. 2.9 Relay interface to PIC16F877

```
main()
{
   for(chadress=0;chadress<=7;chadress++)
   {
      output_d(chadress);   // send channel address on port D
      delay_ms(50);
      output_high(PIN_B0);            //ale is connected to B0
      delay_ms(50);
      output_low(PIN_B0);
output_low(PIN_B1);       //eoc is connected to B1
output_low(PIN_B2);       //soc is connected to B2
   delay_ms(200);
   output_high(PIN_B2);   /*0 to 1 transition req to clear sar */
   delay_ms(200);
   output_low(PIN_B2);    /*1 to 0 transition req to start conversion */
while(input(PIN_B1)==1)
   {
   adcdata=input_d();
   adcdata=adcdata+0×30;
   dataavg= (dataavg+adcdata);
   }
}
if(dataavg>=50)
{
output_low(PIN_B4); //if tem exceeds 50 deg stitch of the bulb
}
else
{
output_high(PIN_B4);
}}
**********************************************************************
```

Chapter 3
Accessing On-Chip and Off-Chip Peripherals

The PIC microcontroller is designed with on-chip ADC. However, sometimes the system requires a multi-channel digitization. The processed digitized version needs to be conveyed to the analog domain by using DAC. An optocoupler is required many a times for safely driving without causing any damage to the microcontroller itself. The case studies developed in this chapter will serve all the hardware and software requirements for the process industries where the PIC will be the core processor for the algorithmic implementation. Moreover the waveform generation and the pseudo random signal generation through PIC may be used for test signal generation.
Following case studies are developed in this chapter:

3.1 Using On-Chip ADC
3.2 Interfacing ADC0809 to PIC
3.3 Optoisolator Interfacing
3.4 DAC Implementation Using On-Chip PWM
3.5 Waveform Generation Using PIC
3.6 Pseudo-Random Number Generation Through PIC
3.7 ON-OFF Temperature Controller Using On Chip ADC
3.8 Implementing a PID Temperature Controller Using PIC16F877

3.1 Using the On-Chip ADC

One of the capabilities of the PIC16F877 which makes it an ideal device for data acquisition systems is its eight channels, 10 bit ADC. This can be usefully combined with the USART and 256 bytes of EEPROM to make an ideal single chip data acquisition solution. Moreover power consumption can be significantly reduced by using the sleep mode with the aid of the timer 1. In this application (shown in Fig. 3.1) the ambient temperature is measured using LM35 which generates a higher output voltage than thermocouples and exhibit greater linearity as compared to thermistors. The features of LM35 are as follows:

- Output voltage proportional to the Celsius temperature
- Scale factor.01 V/°C

J.S. Parab, et al., *Practical Aspects of Embedded System Design using Microcontrollers*,

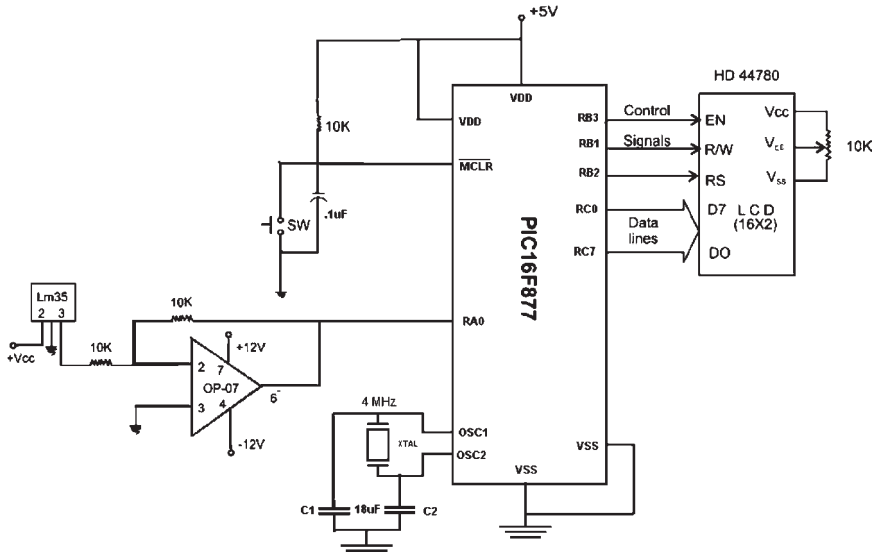


Fig. 3.1 Using on-chip ADC of PIC16F877

- No need of any external calibration or trimming
- Accuracy of ±0.4°C at room temperature and ±0.8°C over a range of 0°C to +100°C.
- LM35DZ draws only 60 μA from its supply
- Low self-heating less than 0.1°C temperature rise in still air

In this application the temperature measured using LM35 is displayed on the LCD.

Program 3.1 Program for data logging using on-chip ADC

Program Source Code

```
*******************************************************************************
// this is c code which makes use of on chip ADC.
-------------------------------------------------------------------------------
#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
#include<lcd.h>
int adc_data;
void main()
{
   setup_adc_ports(ALL_ANALOG);
   setup_adc(ADC_CLOCK_INTERNAL);
   set_adc_channel(0);
   lcdint();
   while(1)
```

```
    {
    adc_data = read_adc();
    lcd_data(adc_data,0×a2);
    }
}
// this is part of above program i.e. code for <lcd.h> header file
#include<string.h>
void lcd_int();
void busycheck();
void lcd_string(unsigned char,unsigned int);
unsigned int array1[]={“ On Chip ADC ”};
unsigned int array2[]={“ADC Value :”};
unsigned char arr[]={“0123456789”};
unsigned char a,i,b;
void lcdint(){
    output_low(PIN_B3);
    output_high(PIN_B1);
    busycheck();
    output_d(0×38);          /* Entry mode set */
    output_high(PIN_B3);
    output_low(PIN_B3);
    busycheck();
    output_d(0×06);          /* function set */
    output_high(PIN_B3);
    output_low(PIN_B3);
    busycheck();
    output_d(0×01);          /* clear display */
    output_high(PIN_B3);
    output_low(PIN_B3);
    busycheck();
    output_d(0×0c);          /* cursor on */
    output_high(PIN_B3);
    output_low(PIN_B3);
       lcd_string(array1,0×82);
       lcd_string(array2,0× c4);
    busycheck();
    output_d(0×d4);
    output_high(PIN_B3);
    output_low(PIN_B3);
    for (i=0;i<20;i++)
    {
          busycheck();
          output_d(0×01);
          output_low(PIN_B1);
          output_high(PIN_B1);
```

```
        output_d('-');
        output_high(PIN_B3);
        output_low(PIN_B3);
    }
}
void lcd_string(unsigned char str,unsigned addr1)
{
    int i,l;
    l = strlen(str);
    busycheck();
    output_d(addr1);          /* starting address */
    output_high(PIN_B3);
    output_low(PIN_B3);
    for (i=0;i<l;i++)
    {
       busycheck();
       output_d(0×01);
       output_low(PIN_B1);
       output_high(PIN_B1);
       b=str[i];
       output_d(b);
       output_high(PIN_B3);
       output_low(PIN_B3);
    }
}
void lcd_data(int8 dat,unsigned addr)
{
    unsigned int d10,d1,i,bcd,temp,d100;
    d100= dat/100;
    temp = dat%100;
    d10 = temp /10;
    d1= temp %10;
       busycheck();
    output_d(addr);
    output_high(PIN_B3);
    output_low(PIN_B3);
    for(i=0;i<0×03;i++)
          {
             if(i==0×00)
             bcd = d100;
             if(i==0×01)
             bcd = d10;
             else if(i==0×02)
             bcd = d1;
             busycheck();
```

```
        output_d(0×01);
        output_low(PIN_B1);
        output_high(PIN_B1);
        output_d(arr[bcd]);
        output_high(PIN_B3);
        output_low(PIN_B3);
            }
}
void busycheck()
{
   output_d(0×02);
   output_low(PIN_B1);
   output_high(PIN_B1);
   delay_ms(2);
busy:
   output_high(PIN_B3);
   output_low(PIN_B3);
   a=input_d();
   if ((a&0×80)==0×80)
      goto busy;
   output_d(0×0);
   output_low(PIN_B1);
   output_high(PIN_B1);
   delay_us(10);
}
```

****

The above program can be slightly modified by using a reciprocal conversion factor i.e. 100°C/V. With the implementation of the following equation to convert output voltage to temperature the measurement can be done from linearly from 0°C to 100°C.

$$\text{Temperature } (^{\circ}\text{C}) = \text{Vout}^{*}(100^{\circ}\text{C/V})$$

This gives Vout as 1 V,when temperature is 100°C.

3.2 Interfacing ADC (0809) to PIC

National Semiconductor ADC0808/ADC0809 is popularly known as "Data Acquisition Devices (DAD)". They have built in elements required for most of the standard data acquisition system. This device comprises of an 8-bit A/D converter, 8-channel multiplexer with an address input latch, and associated control logic. Interfacing with the PIC becomes hassle free with minimum requirements of external components by using these devices. Important feature of these devices along with application note may be seen on the web URL of National Semiconductor [54]. Some notable features are

- Operates ratiometrically or with 5 Vdc or analog span adjusted voltage reference
- No zero or full scale adjust required
- Eight-channel multiplexer with address logic
- 0–5 V input range with single 5 V power supply
- Outputs meet TTL voltage level specifications
- ADC0808 equivalent to MM74C949

ADC0809 comprises of CMOS logic gates which typically have build-in protection diodes. This makes them much less vulnerable to ESD damage than power MOS transistors. However, few precautionary measures are better when handling these devices [55].

- All unused inputs must be connected to GND or +5 V to prevent excessive current consumption and erratic behavior.
- Never connect an input signal to a CMOS device when the power is off.
- Never store unprotected CMOS in non-conductive trays.
- Place CMOS devices pin down in conductive foam when they are not stored in a conductive tray or installed in a circuit.
- Use a battery powered or ESD protected soldering iron to solder MOS chips.
- If not using a ground strap, when in doubt ground yourself! This will avoid nasty static discharges.

A very simple procedure without using the microcontroller can also be adopted for testing this ADC [55]. Connect the ADC0809 in the free-running mode; that is the START input pulse (as well as the ALE) provided by the EOC output. Connect all the address lines LOW and tie all inputs LOW except for IN0 (Pin 26). Use a 555 timer to provide a TTL pulse train of about 200 kHz for the clock. Input an adjustable 0–5 V DC at IN0 (pin 26) using a 5 or 10 KW potentiometer connected between the 5 V supply and ground. For several input voltages, display the 8-bit output using the LEDs of your diagnostic circuitry. Record these readings. Change the frequency of the clock while observing the EOC output. One can also find out the conversion time (100 μS) from these observations.

Program 3.2 Eight channel ADC (0809) interfacing (Refer Fig. 3.2)

Program Source Code

```
*******************************************************************************
// Program to illustrate the external 8 Channel ADC0809 interface to PIC16F877.
-------------------------------------------------------------------------------
#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
int adcdata, chadress;
main()
{
   for(chadress=0;chadress<=7;chadress++)
```

```
   {
        output_d(chadress);          // send channel address on port D
        delay_ms(50);
        output_high(PIN_B0);         //ale is connected to B0
        delay_ms(50);
        output_low(PIN_B0);
output_low(PIN_B1);       //eoc is connected to B1
output_low(PIN_B2);       //soc is connected to B2
      delay_ms(200);
      output_high(PIN_B2);          /*0 to 1 transition required to clear SAR */
      delay_ms(200);
      output_low(PIN_B2);           /*1 to 0 transition required to start conversion */
while(input(PIN_B1)==1)
   {
   adcdata=input_d();
   adcdata=adcdata+0×30;
   printf( "\n adc value is =",adcdata);
}}}
```

****

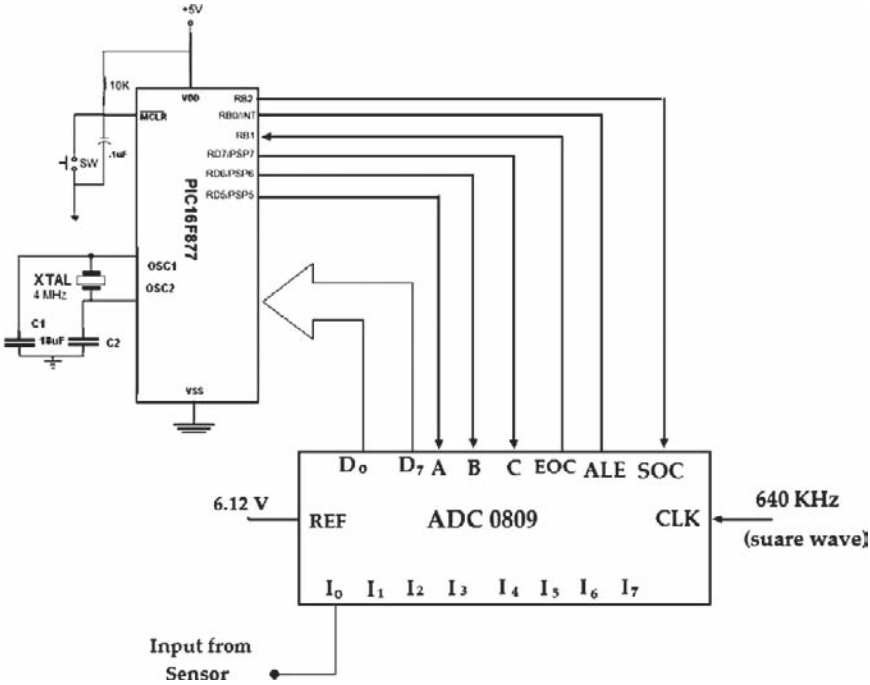


Fig. 3.2 Interfacing ADC 0809 to PIC

3.3 Opto-Isolator Interfacing

Optoisolators, are the devices having at least one emitter, which is optically coupled to a photo-detector through some sort of an insulating medium. Also popularly known as optical coupler or optocoupler, it is a semiconductor device that allows signals to be transferred between two circuits, with complete electrical isolation. These devices typically find their applications in circumstances where a low voltage circuit such as microcontroller port or logic gates, is driving high voltage circuits having inductive loads. The optical isolation prevents the back emf damaging the low voltage circuit and thus ensures complete safety.

Optocouplers are available in several configurations; the basic one is with a LED and a phototransistor. The main competitors of optocouplers are the isolation transformers, but the former allows DC coupling between source and load, even in the severe over voltage conditions on the load side, while the later might fail to do so in such conditions. Many variations exist as far as the optical is concerned. There are optocouplers with air or a dielectric waveguide as the optical transmission medium.

The CNY17 series optocouplers from Fairchild Semiconductors consists of a phototransistor optically coupled to a gallium arsenide infrared-emitting diode in a six-lead plastic dual inline package. The elements are mounted on one leadframe using a coplanar technique, providing a fixed distance between input and output for highest safety requirements. These devices are recommended for safe protective separation against electrical shock according to safety class II (reinforced isolation):

- For application class I–IV at mains voltage 3,300 V
- For application class I–III at mains voltage 3,600 V according to VDE 0884 (Table 2)

They are suitable for applications in Switch-mode power supplies, computer peripheral interface, microprocessor system interface, line receiver, etc.[55].
As per the Temic Semiconductor [55] application note these standards are as follows:

- VDE0884: Optocoupler providing protective separation
- VDE0804: Telecommunication apparatus and data processing
- VDE0805/IEC 950/EN6095: Office machines (applied for reinforced isolation for mains voltages 3,400 VRMS)
- VDE0860/lEC 65: Safety for mains operated electronic and related household apparatus

Complete datasheet is available on the web [56].

Program 3.3 Opto-isolator interfacing (Fig. 3.3 and Fig. 3.4)

Program Source Code

```
**********************************************************************************
/* Program to illustrate the stepper motor driving using the Opto-Isolator CNY17. the
sequence of characters is send to activate the coils of the stepper motor (Fig. 3.4)
interface to PIC. */
```

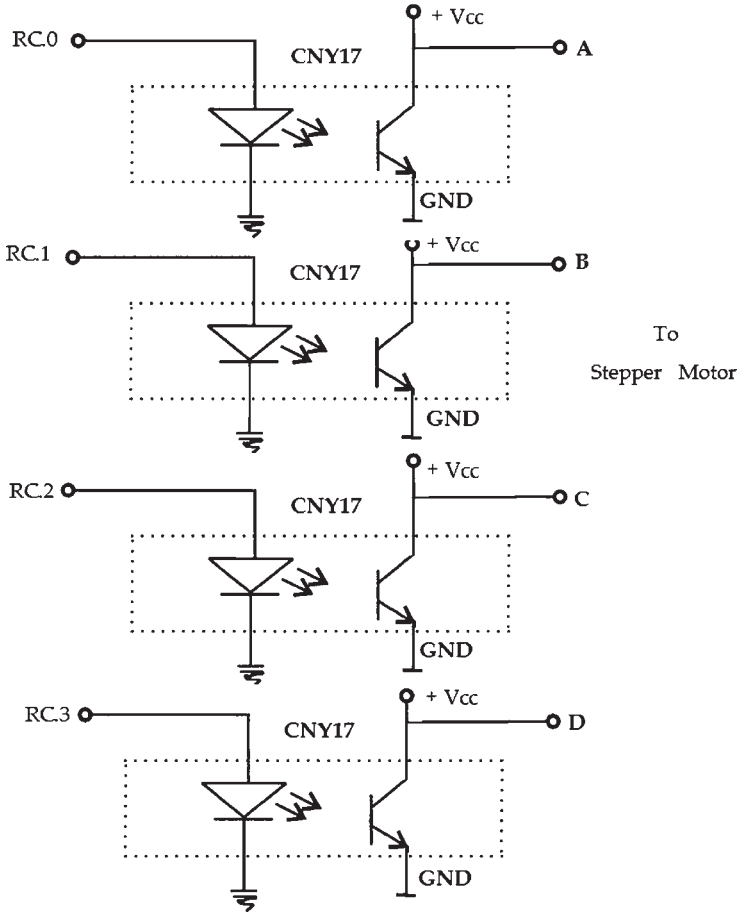


Fig. 3.3 Opto-isolator interfacing to PIC16F877

```
#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
unsigned char a;
void main()
{
   output_c(0×f0);
while(1)
{
   output_c(0×f9);
   delay_ms(10);
   output_c(0×f5);
   delay_ms(10);
   output_c(0×f6);
```

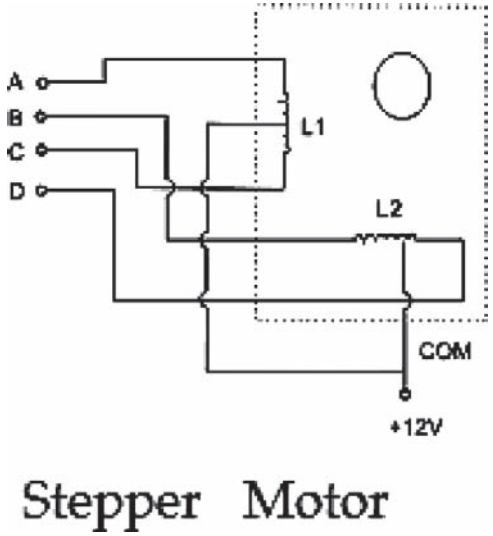


Fig. 3.4 Stepper motor

```
    delay_ms(10);
    output_c(0×fa);
    delay_ms(10);
}
}
```

****

3.4 DAC Implementation Using On-Chip PWM

A good number of embedded microcontroller applications require generation of analog signals for actuating the final control element or simply for communicating with the analog world. Many designers go for an integrated or stand-alone digital-to-analog converter (DAC). However, in lieu of the external DAC, PWM signals provided as a part of the on-chip resources can be used for generating the required analog signals (as shown in Fig. 3.5). They scan be used to create both dc and ac analog signals. One difficulty with the built-in digital to analog converter (DAC) provided by many microcontrollers is lack of sufficient resolution. Higher end, and more expensive, microcontrollers provide either an parallel port or an I^2C based interface for connecting to an external DAC chip. This leads not only to higher price of these microcontrollers, but combined with the expense of an external DAC chip, pushes up component count, total cost and reliability.

In order to solve the above mentioned problems, many low cost microcontrollers today include pulse width modulation (PWM) as a regular feature. PWM based

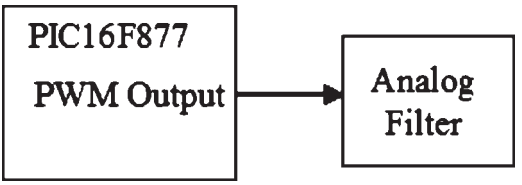


Fig. 3.5 Block schematic of DAC implementation using on-chip PWM

DAC finds lot of applications in motor speed control, light dimmer applications, controlling a SMPS and even in digital audio. PWM offers several advantages over an analog control. For example, using PWM to control the brightness of a lamp, the heat dissipated from the lamp is less than the heat generated from an analog control that converts the current to heat. Hence, less power is delivered to the load (light), which will prolong the life cycle of the load [57].

The basic mechanism of PWM based DAC is as follows. PWM signal is a digital signal with fixed frequency but varying duty cycle. For implementation of the DAC, the duty cycle of the PWM signal is varied with time, and subsequently the variable duty cycle signal is filtered using analog filter.

The resolution of a PWM DAC is equivalent to the resolution of the PWM signal used to create the DAC. The resolution of the PWM signal depends on the length of the counter and the smallest possible duty-cycle change in the PWM counter. Mathematically this is expressed as

$$\text{Required in counts} = (\text{Length 0 counter in counts})/\text{Smallest duty cycle change in counts}$$

On the same lines, the frequency required for the PWM output signal can be calculated. It is equivalent to the update rate of the DAC, since each change in PWM duty cycle is the equivalent of one DAC sample. The frequency required for the PWM timer will depend on the required PWM signal frequency and the desired resolution. Mathematical foundation and microcontroller based implementation has been covered by an application note by Texas Instruments [56].

$$\text{Required PWM Timer Frequency} = \text{PWM Signal Frequency} \times 2^n$$

Where n is the desired resolution.

Program 3.4 DAC implementation using on-chip PWM of PIC16F877 (Refer Fig. 3.6)

Program Source Code

```
*******************************************************************************
// Program to illustrate the DAC implementation using on-chip PWM
-------------------------------------------------------------------------------
#include <16f877.h>
#use delay(clock=10000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
```

```
unsigned int value;
void main()
{
setup_ccp1(CCP_PWM);
setup_timer_2(T2_DIV_BY_1, 127, 1);
while(TRUE)
        {
   value=read_adc();
   set_pwm1_duty(0×7f);
   } }
```

****

3.5 Waveform Generation Using PIC

The present application is all about generation of waveforms using PIC16F877. In general microcontroller based waveform generators are preferred in many applications as they can be used to generate sequential bit patterns with predictable timing combined with frequency. Programmable waveform generation is used in many test applications such load susceptibility testing, dynamic response evaluation, repetitive stimulus application, etc.

This application generates waveforms to port B of PIC16F877.

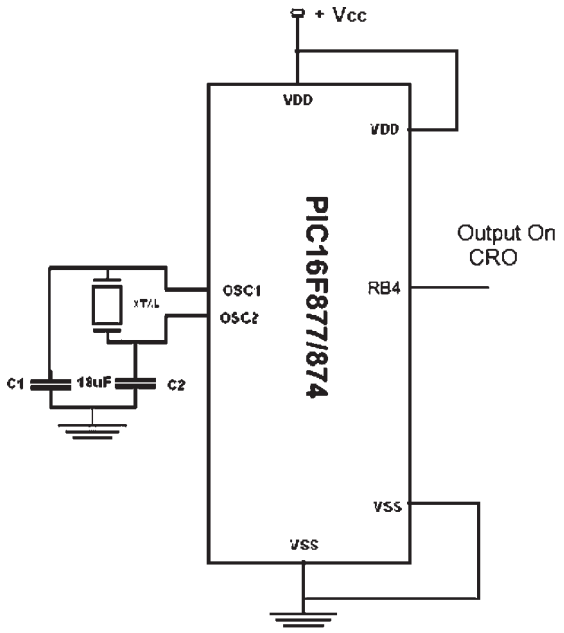


Fig. 3.6 DAC implementation using on-chip PWM of PIC16F877

Program 3.5 Square wave generation

Program Source Code

```
**********************************************************************************
// Program to illustrate the Square Wave generation
_____________________________________________________________________
#include<16f877.h>
#use delay(clock=20000000)
unsigned char a;
unsigned char b,c;
void main()
{
while(1){
   output_high(PIN_A0);
   output_b(0×00);
   delay_ms(5);
   output_b(0×ff);
   delay_ms(5);
   }
}
**********************************************************************************
```

Program 3.6 Triangular wave generation

Program Source Code

```
**********************************************************************************
// Program to illustrate the Triangular Wave generation
_____________________________________________________________________
#include<16f877.h>
#use delay(clock=20000000)
unsigned char a;
unsigned char i;
void main()
{
      output_a(0×02);
while(1)
      {
      for(i=0×00;i<0×ff;i++)
         {
   output_b(i);
   }
      for(i=0×ff;i>0;i–)
         {
   output_b(i);
   }
      }
}
**********************************************************************************
```

Program 3.7 Negative RAMP generation

Program Source Code

```
**************************************************************************************
// Program to illustrate the Negative Ramp generation
--------------------------------------------------------------------------------------
#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
unsigned char a;
unsigned char i;
void main()
{
      output_a(0×08);
   while(1)
      {
      for(i=0×ff;i>0;i–)
         {
      output_b(i);
      }
      }
}
**************************************************************************************
```

Program 3.8 Sine wave generation

Program Source Code

```
**************************************************************************************
// Program to illustrate the Sine Wave generation
--------------------------------------------------------------------------------------
#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=19200,xmit=PIN_c6,rcv=PIN_c7)
void change(void);
unsigned int i;
unsigned int16 ab[]={9,2047,2403,2747,3071,3363,3615,3820,3971,4063,4095,
4063,3971,3820,3615,3363,3071,2747,2403,2047,1692,1347,1024,731,479,274,1
23,31,0,31,123,274,479,731,1024,1347,1692,7899};
void main()
{
   output_a(0×10);
      while(1)
      {
      for(i=1;i < 37;i++)
         {
               output_b(ab[i]/100);
            }          } }
**************************************************************************************
```

3.6 Pseudo-Random Number Generation Through PIC

Program 3.9 Pseudo random number generation (Refer Fig. 3.7)

Program Source Code

```
**************************************************************************************
//This program illustrate to generate random number by using inbuilt ADC
--------------------------------------------------------------------------------------
#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
#include<lcd.h>
int Randomdata;
void main()
{
   setup_adc_ports(ALL_ANALOG);
   setup_adc(ADC_CLOCK_INTERNAL);
   set_adc_channel(0);
   lcdint();
   while(1)
      {
      Randomdata = read_adc();
      lcd_data(Randomdata,0×a2);
      }
}
**************************************************************************************
```

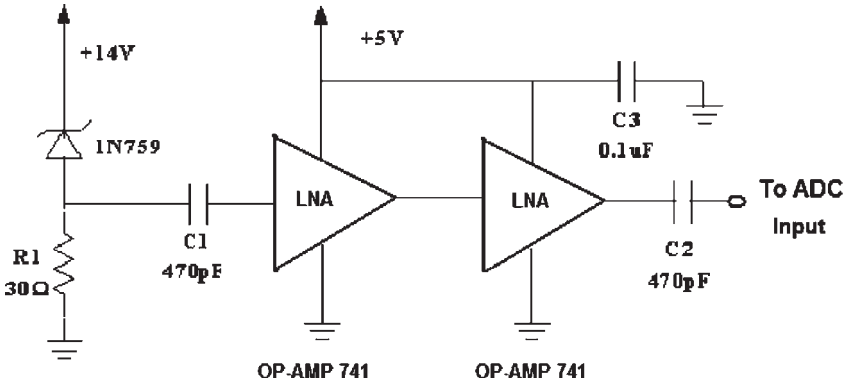


Fig. 3.7 Analog random voltage generation using Zener and Op-amps

Program 3.10 Generate random number by using external ADC0808 interface to PIC (Refer Fig. 3.8)

Program Source Code

```
**************************************************************************
/* Program illustrate the random number generation using PIC16F877 and the
external ADC 0808 connected. */
--------------------------------------------------------------------------
#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
#include<lcd.h>
int randomdata,chadress;
main()
{
   for(chadress=0;chadress<=7;chadress++)
   {
        output_d(chadress);          // send channel address on port D
        delay_ms(50);
        output_high(PIN_B0);         //ale is connected to B0
        delay_ms(50);
        output_low(PIN_B0);
output_low(PIN_B1);       //eoc is connected to B1
output_low(PIN_B2);       //soc is connected to B2
      delay_ms(200);
      output_high(PIN_B2);           /*0 to 1 transition req to clear sar */
      delay_ms(200);
      output_low(PIN_B2);/*1 to 0 transition req to start convertion */
```

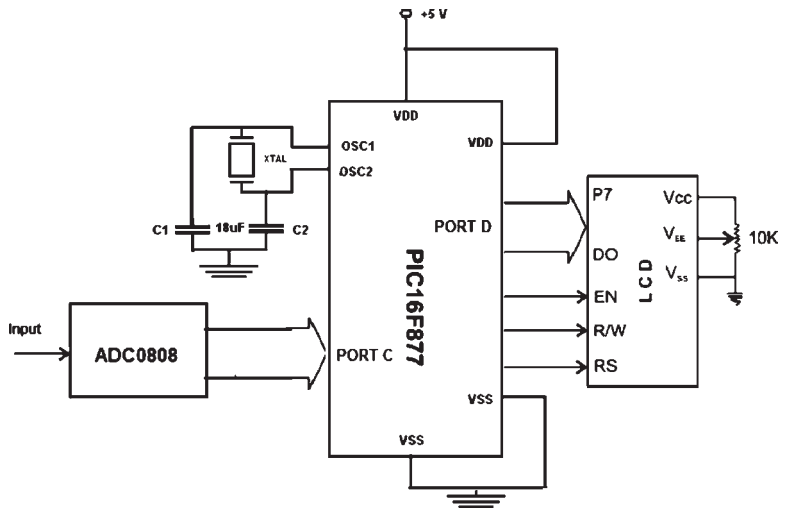


Fig. 3.8 Generate random number by using external ADC0808 interface to PIC

```
while(input(PIN_B1)==1)
   {
   randomdata=input_d();
   randomdata=adcdata+0×30;
lcd_data(randomdata,0×a2);
   printf( "\n adc value is =",randomdata);
   }
}}
```

****

3.7 On-Off Temperature Controller Using On-Chip ADC

Program 3.11 In the case-study a simple ON-OFF temperature controller is designed using LM35 as sensor, the output of which is digitized using the onchip ADC. The control action is implemented using the relay driven through port line B3. The program code is self explanatory.

Program Source Code

****

```
// Program illustrate ON-OFF temperature controller using on-chip ADC.
```

```
#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
#include<lcd.h>
int adc_data;
void main()
{
      setup_adc_ports(ALL_ANALOG);
      setup_adc(ADC_CLOCK_INTERNAL);
      set_adc_channel(0);
      lcdint();
      while(1)
         {
         adc_data = read_adc();
         lcd_data(adc_data,0×a2);
         }}
// this is part of above program i.e. code for <lcd.h> header file
#include<string.h>
void lcd_int();
void busycheck();
void lcd_string(unsigned char,unsigned int);
unsigned int array1[]={" On Chip ADC "};
unsigned int array2[]={"ADC Value :"};
unsigned char arr[]={"0123456789"};
```

```
unsigned char a,i,b;
void lcdint(){
   output_low(PIN_B3);
   output_high(PIN_B1);
   busycheck();
   output_d(0×38);          /* Entry mode set */
   output_high(PIN_B3);
   output_low(PIN_B3);
   busycheck();
   output_d(0×06);          /* function set */
   output_high(PIN_B3);
   output_low(PIN_B3);
   busycheck();
   output_d(0×01);          /* clear display */
   output_high(PIN_B3);
   output_low(PIN_B3);
   busycheck();
   output_d(0×0c);          /* cursor on */
   output_high(PIN_B3);
   output_low(PIN_B3);
            lcd_string(array1,0×82);
            lcd_string(array2,0× c4);
   busycheck();
   output_d(0×d4);
   output_high(PIN_B3);
   output_low(PIN_B3);
   for (i=0;i<20;i++)
   {
         busycheck();
         output_d(0×01);
         output_low(PIN_B1);
         output_high(PIN_B1);
         output_d('−');
         output_high(PIN_B3);
         output_low(PIN_B3);
   }
}
void lcd_string(unsigned char str,unsigned addr1)
{
      int i,l;
      l = strlen(str);
            busycheck();
   output_d(addr1);         /* starting address */
   output_high(PIN_B3);
   output_low(PIN_B3);
```

```
   for (i=0;i<l;i++)
   {
         busycheck();
         output_d(0×01);
         output_low(PIN_B1);
         output_high(PIN_B1);
         b=str[i];
         output_d(b);
         output_high(PIN_B3);
         output_low(PIN_B3);
   }
}
void lcd_data(int8 dat,unsigned addr)
{
      unsigned int d10,d1,i,bcd,temp,d100;
      d100= dat/100;
      temp = dat%100;
      d10 = temp /10;
      d1  = temp %10;
            busycheck();
   output_d(addr);
   output_high(PIN_B3);
      output_low(PIN_B3);
            for(i=0;i<0×03;i++)
               {
                  if(i==0×00)
                  bcd = d100;
                  if(i==0×01)
                  bcd = d10;
                  else if(i==0×02)
                  bcd = d1;
                  busycheck();
         output_d(0×01);
         output_low(PIN_B1);
         output_high(PIN_B1);
         output_d(arr[bcd]);
         output_high(PIN_B3);
         output_low(PIN_B3);
               }
}
void busycheck()
{
   output_d(0×02);
   output_low(PIN_B1);
   output_high(PIN_B1);
```

```
   delay_ms(2);
busy:
   output_high(PIN_B3);
   output_low(PIN_B3);
   a=input_d();
   if ((a&0×80)==0×80)
goto busy;
   output_d(0×0);
   output_low(PIN_B1);
   output_high(PIN_B1);
   delay_us(10);
}
*******************************************************************************
```

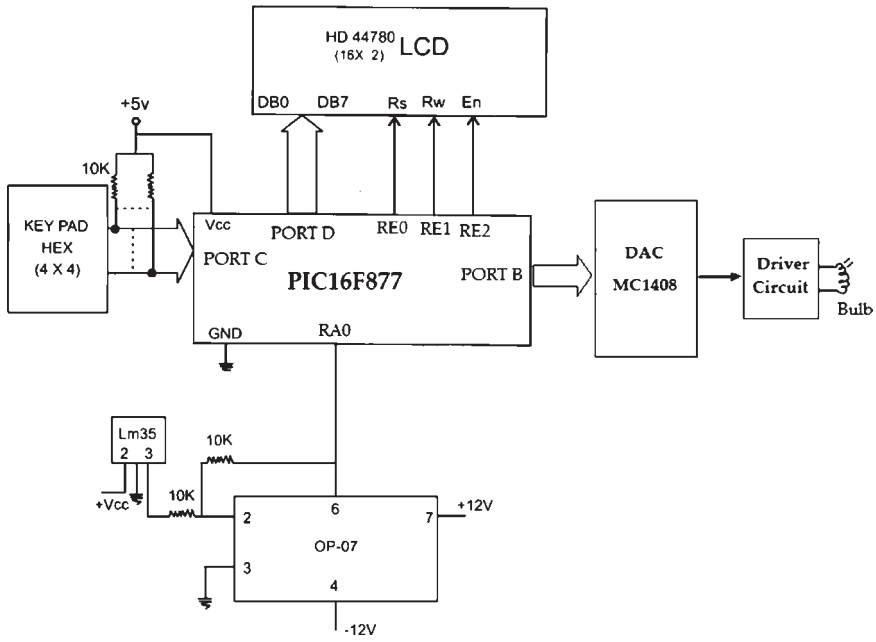


Fig. 3.9 Schematic of PID temperature controller using PIC16F877

3.8 Implementing a PID Temperature Controller Using PIC16F877

Over the last 50 years the PID control algorithm has gained wide popularity in process industries. PID is a feedback control system in which the controller output is proportional to the error and also depending on the integral and derivative of the error.

PID control is superior to the other control strategies in several respects. For instance, the proportional controller exhibits a relatively high overshoot, a long settling time as well as a steady-state error and a offset which remains even after the control is accomplished. The integral controller has still higher overshoot than the proportional controller due to the inherently slower starting of integration process, but with elimination of the offset of the proportional control. PI controller combines the properties of the P and I and exhibits no offset but with the same problems of higher overshoot and settling time. The PD controller almost solves the problems of overshoot owing to the high dynamic response of the derivative action; a small value of offset still persists due to the proportional action. However, the PID control combines all the plus points of the P, I and D strategies and thus exhibits a overshoot than the PD controller and no offset due to the I action.

Due to the above mentioned advantages PID has almost become a universal control strategy. Three levels of tuning are required in this control strategy – Proportional, Integral, and Derivative.

Program 3.12 PID temperature controller (Refer Fig. 3.9) using PIC16F877

Program Source Code

```
*******************************************************************************
// Temperature controller based on PID
_____________________________________________________________________________

#include<16F877.h>
#use delay(clock=20000000)
unsigned char a,i,b,j;
void INITlcd(void);
void ENABLE (void) ;
void LINE(int);
int keyb(void);
void dis(int);
int setpt= 60;
int kp= 12, ki=1,kd=11,C=1;
void main(void)
{
char test[]=“PID”;
char *p;
int j,newt,error1,error2, rate =1,sum,currenttemp1;
setup_adc_ports(ALL_ANALOG);
```

```
   setup_adc(ADC_CLOCK_INTERNAL);
   set_adc_channel(0);
INITlcd();
LINE(1);
p=&test;
for(j=0;j<17;j++)
{
if(j==0)LINE(1);
if(j==8)LINE(2);
output_d(*p++);
ENABLE();
delay_ms(10);
while(1)
{
j=0;
j=keyb();
if(j>8)j=0;
if(j==1){
setpt++;
dis(j);
C=6;}
if(j==1){
setpt–;
dis(j);
C=7;
if(j==3){
kp++;
dis(j);
C=6;
}
if(j==4){
kp–;
dis(j);
C=7;
}
if(j==5){
ki++;
dis(j);
C=6;
}
if(j==6){
ki–;
dis(j);
C=7;
}
```

```
if(j==7){
kd++;
dis(j);
C=6;
}
if(j==8){
kd–;
dis(j);
C=7;
}
currenttemp1= read_adc();
currenttemp1=currenttemp1*2;
error1=setpt-currenttemp1;
newt=currenttemp1+kp*error1;
sum=sum+error1;
newt=(newt+(kd*sum));
newt=(newt+(kd*sum*(error2-error1)));
error2=error1;
if(newt>255)
{
newt=255;
sum=sum-error1;
}
if(newt<0)
{
newt=0;
sum=sum-error1;
}
output_b(newt);   // port B is connected to DAC0808
if((j==0)&&(C<1))
dis(0×09);
delay_ms(10);
if(C>=1)
C–;
}
}}}
void ENABLE(void)
{
   output_high(PIN_E2);
delay_ms(10);
   output_low(PIN_E2);
delay_ms(10);
}
void LINE(int j){
       if (j==1) {
```

```
        output_low(PIN_E0); output_low(PIN_E1);
        output_d(0×80);
        ENABLE();
        output_high(PIN_E0);
        }
        else
        {
        output_low(PIN_E0); output_low(PIN_E1);
        output_d(0×C0);        // PORT D LCD data lines
        ENABLE();
        output_high(PIN_E0);;
        }}
void INITlcd(void)
{ output_low(PIN_E0);//RS
output_low(PIN_E1);//RW
output_low(PIN_E2);//EL
output_d(0×38);
        ENABLE();
ENABLE();
ENABLE();
ENABLE();
output_d(0×06);
ENABLE();
        output_d(0×0E);
        ENABLE();
    output_d(0×01);
        ENABLE();
}
int keyb(void){
int key=0;
output_c(0×FE)
delay_ms(10);
readkey=input_c();
if (readkey==0×EE) key = 1;
if (readkey==0×DE) key = 2;
if (readkey==0×BE) key = 3;
if (readkey==0×7E) key = 4;
return(key);
}
void dis(int j)
{
char code currenttemp[]=“ Tem = ”;
char code inS[]=“ increment setpt ”;
char code dec[]=“ decrement setpt”;
char code ikp[]=“ increment kp ”;
```

```
char code dkp[]=“ decrement kp ”;
char code iki[]=“ increment ki ”;
char code dki[]=“ decrement ki ”;
char code ikd[]=“ increment kd ”;
char code dkd[]=“ decrement kd ”;
char *p;
int k,currenttemp1 ;
if(j !=9){
INITlcd();
if(j==1) p=&inS ;
if(j==2) p=&dec ;
if(j==3) p=&ikp ;
if(j==4) p=&dkp ;
if(j==5) p=&iki ;
if(j==6) p=&dki ;
if(j==7) p=&ikd ;
if(j==8) p=&dkd ;
for(k=0;k<17;k++)
{
if(k==0) line(1);
if(k==8) line(2);
output_d(*p++);
ENABLE();
}
delay_ms(50);
INITlcd();
LINE(1);
output_d(((setpt/100)+0×30));
ENABLE();
output_d((((setpt/10)%10)+0×30));
ENABLE();
output_d(((setpt%10)+0×30));
ENABLE();
delay_ms(500);
INITlcd();
LINE(1);
output_d((((kp/10))+0×30));
ENABLE();
output_d(((kp%10)+0×30));
ENABLE();
delay_ms(50);
INITlcd();
LINE(2);
output_d(((((ki/10))+0×30)));
ENABLE();
```

```
output_d(((ki%10)+0×30));
ENABLE();
delay_ms(50);
INITlcd();
LINE(1);
output_d(((kd/10))+0×30));
ENABLE();
output_d((kd%10)+0×30));
ENABLE();
delay_ms(50);
}
if(j==7)
{
p=&currenttemp;
INITlcd();
for(k=0;k<8;k++)
{
if(k==0)
LINE(1);
output_d(*p++);
ENABLE();
}
currenttemp1= read_adc();
LINE(2);
output_d((currenttemp1/1000)+0×30));
ENABLE();
output_d(((currenttemp1/100)%10)+0×30));
ENABLE();
output_d((((currenttemp1/10))%10)+0×30));
ENABLE();
output_d(((currenttemp1)%10)+0×30));
ENABLE();
}
}
*****************************************************************************
```

Chapter 4
Serial Interface to PIC

This chapter discusses case studies developed for interfacing PIC through HyperTerminal. Following applications are developed:

4.1 Configuring HyperTerminal
4.2 Setting up HyperTerminal
4.3 Displaying Data on HyperTerminal
4.4 HyperTerminal Interface: Getting Sensor Signal on HyperTerminal
4.5 HyperTerminal Based Control: Controlling an Actuator Such as Relay from PC HyperTerminal
4.6 Controlling a Stepper Motor from HyperTerminal: HyperTerminal Keyboard Provides Direction

HyperTerminal is a data communications utility program that has been distributed with many Microsoft Windows® operating systems, including Windows 95, Windows 98, Windows Me, Windows 2000, and Windows XP. It is a program for connecting a computer to other computers. It can also be set for accessing Telnet sites, bulletin board systems (BBSs), online services, using the connectivity options such as a modem or a null modem cable or even through Ethernet connection. With the advent of world wide web where sophisticated protocols are taking care of the online communication, the use of HyperTerminal has become almost rare. However for an embedded engineer, it is still a great debugging tool for testing the modems or send the commands to the embedded appliance connected to PC. It can be as well used for sending bulky files without setting up the network for the PC. Moreover, this is really useful for the old PCs where the setting up of the network is difficult.

By default the 'HyperTerminal' application resides in the communications option. It can be activated by following the following steps (refer Fig. 4.1):

- Click Start on the desktop
- Go to All Programs
- Click Accessories
- Click Communications
- Clicking HyperTerminal to start the program

J.S. Parab, et al., *Practical Aspects of Embedded System Design using Microcontrollers*,

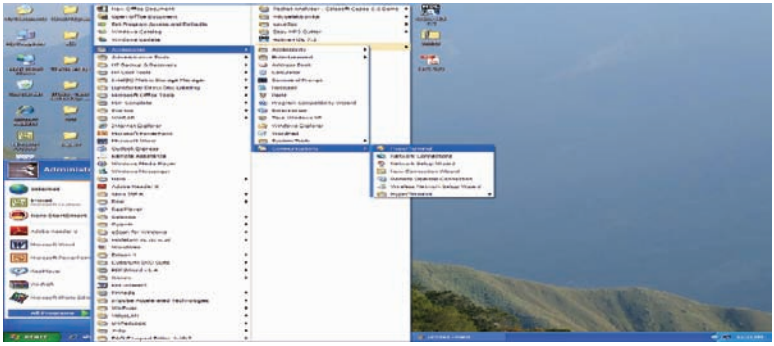


Fig. 4.1 Locating the "HyperTerminal" in WINDOWS

If the HyperTerminal is bring used for the first time, it will seek information to configure. This includes "Location Information" such as country, area code, modem or TCP/IP and tone/pulse dialing options.

In case the user wants to use the 'HyperTerminal' program other than the MS Windows, then there are couple of sharewares which is doing the same things. The download URLs are given in the references [41, 42]. In fact they have more capabilities than that included in MS Windows [43].

The main advantage of the HyperTerminal is importing the data directly in other applications such as Excel to plot the graph or to visualize the trends online.

4.1 Configuring HyperTerminal

The step by step procedure given below will help you to configure the HyperTerminal for executing the case studies developed here.

1. Configuring HyperTerminal

4.2 Setting Up HyperTerminal

Following step by step procedure will guide you to setup the HyperTerminal for the applications developed in this chapter.

1. Click the HyperTerminal icon and through the file option create "new Connection" "File:NewConnection" Name the connection (for example, PIC 1). (refer Fig. 4.2)
2. Select the appropriate "COM" port. Care has to be taken to select the appropriate COM port and its cousin. This means if a modem is connected to COM2 then its cousin COM4 should be avoided for the data logging application given here (because the cousin port also shares the same interrupt). (refer Fig. 4.3)

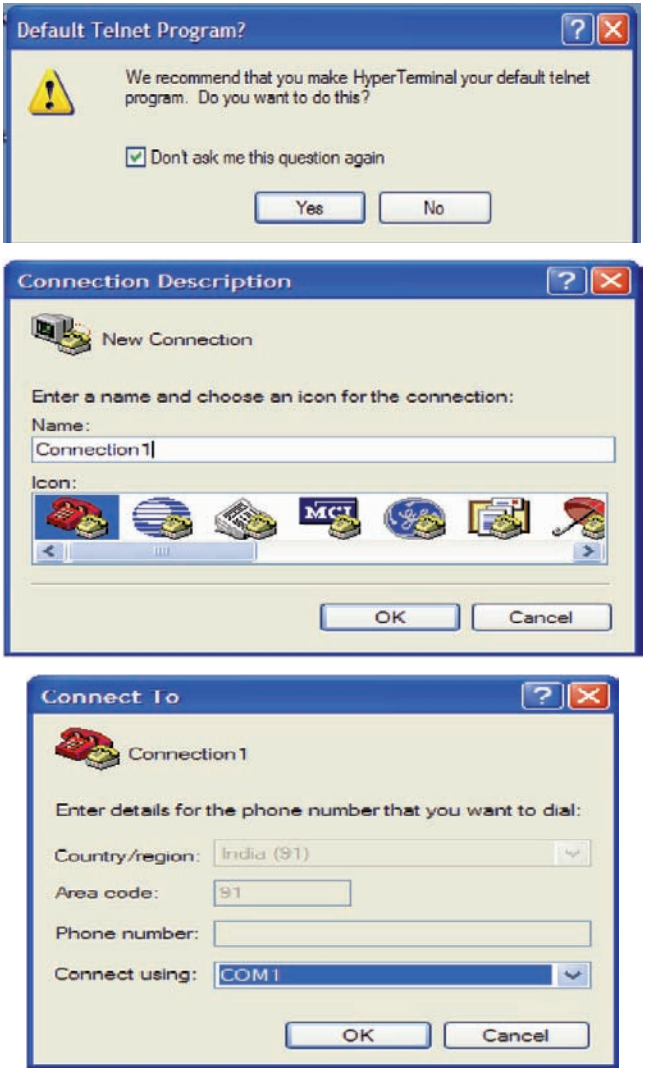


Fig. 4.2 Setting the necessary parameters of the HyperTerminal

3. Set “Bits per second” to 9,600.
4. Set “Data Bits” to 8.
5. Set “parity” to none.
6. Set “Stop Bits” to 1.
7. Set “Flow control” to none.
8. Click OK – the HyperTerminal screen should appear
9. The configuration can be saved by selecting the “File” menu select “Save As” that saves the configuration file to any convenient folder

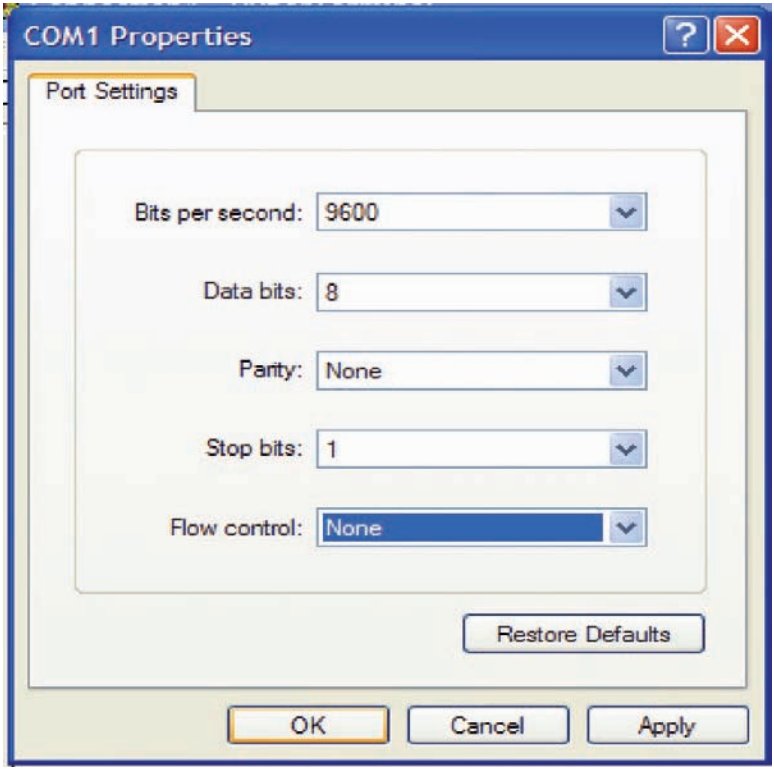


Fig. 4.2 (continued)

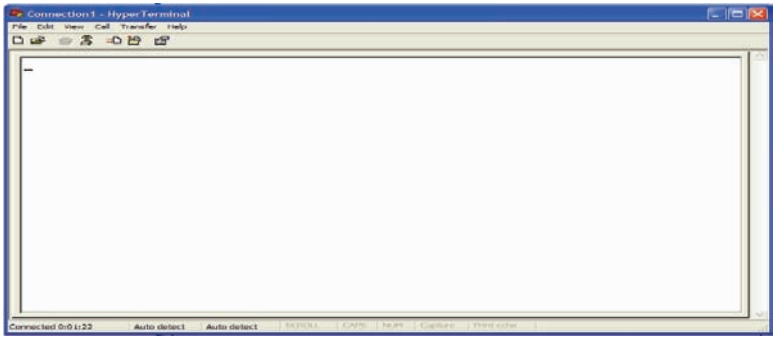


Fig. 4.3 HyperTerminal is now ready interaction with the selected COM port

A shortcut to the configuration file can be created and placed on the desktop for easy reference. HyperTerminal setup procedure is described in many technical manuals on web and may be referred for in depth information [44–46].

4.3 Displaying Data on HyperTerminal

RS232 (single-ended) communication with PC was introduced way back in 1962. Till date it has marked its successful existence and has been widely used through the embedded industry for various data logging and other applications. The main reason attributed to the popularity is the simplicity and reliability for data communication below 256 kbps as most of the embedded system applications do not violate this limit.

In RS-232, there is a provision of independent channels for full-duplex communications. The signals are represented by voltage levels with respect to a system common i.e. wither a power or logic ground. The "idle" state (MARK) corresponds to negative signal level with respect to common, while the "active" state (SPACE) corresponds to positive signal level with respect to common. There are good numbers of handshaking lines which can be used for specifying communications protocol. A common ground between the DTE and DCE is required for communication through RS-232.

In case of RS232 standard the data is bi-polar with the voltage levels as follows:

- +3 to +12 V corresponds to an "ON" or 0-state (SPACE) condition
- −3 to −12 V corresponds to an "OFF" 1-state (MARK) condition
- +3 to −3 V corresponds to the "dead area" kept to absorb line noise

A standard serial interfacing for PC, RS232C, requires negative logic, i.e., logic '1' is −3 to −12 V and logic '0' is +3 to +12 V. While the microcontroller follows the TTL compatibility. In order to achieve the compatibility a popular converter chip used is MAX232. The MAX-232 has 2-channel RS232C port and requires external 10 uF capacitors. Another possibility to sort out the incompatibility issue is by using the DS275 chip, which is smaller in size and requires no external capacitors. The applications developed in this chapter resorts to MAX-232 for achieving compatibility.

Online tutorials for indepth information regarding the RS-232 and MAX-232 are available on Internet [47, 48].

The DB-9 (also called as DE-9 D-sub 9-pin connector) connector is used for the connections as shown in Fig. 4.4. One caution while connections are made is that the pin numbers for transmit and receive (3 and 2) are opposite of those of the DB-25 connector (2 and 3). This decides the device as DTE or DCE.

Program 4.1 Serial transmission only

This program transmits the serial string though HyperTerminal. The clock is specified for fixing the baud rate.

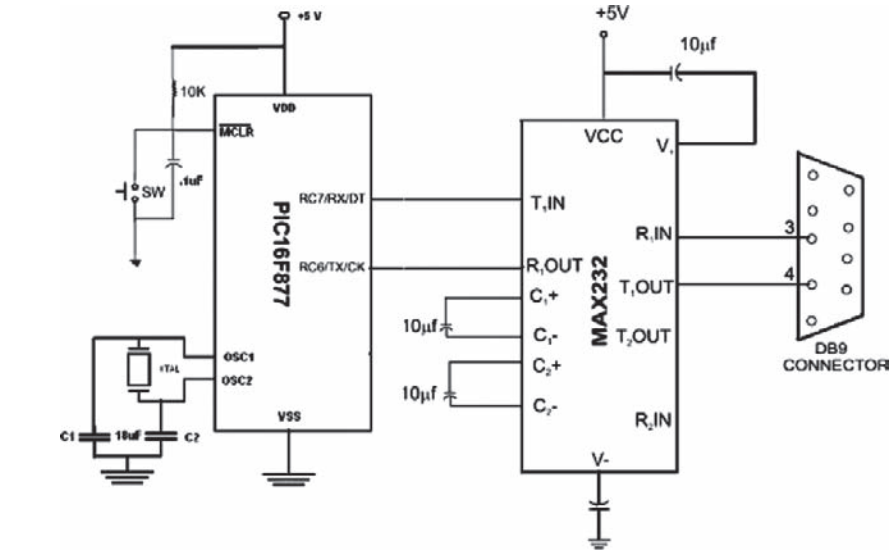


Fig. 4.4 HyperTerminal interface to PIC16F877 through MAX232

Program Source Code

```
**************************************************************
// Program to transmit the serial string through HyperTerminal.
_____________________________________________________________

#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
void main()
  {
    while(1)
      {
      printf("\n\r Welcome…");    // String to transmit.
      }
  }
*************************************************************
```

Program 4.2 Serial reception and transmission

This program facilitates both transmission and reception through HyperTerminal.

Program Source Code

```
**************************************************************
// Program to both Serial Reception and Transmission through HyperTerminal.
_____________________________________________________________

#include<16f877.h>
#use delay(clock=20000000)
```

```
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
void main()
  {
    printf("\n\r Serial Reception and Transmission");
    printf("\n\r Press Key On the Keyboard");
    printf("\n\r");
    while(1)
      {
      putc(getc());
      }
  }
```

****

4.4 HyperTerminal Interface: Getting Sensor Signal on HyperTerminal

This temperature monitoring application senses and logs the temperature value onto the HyperTerminal. LM35 precision integrated-circuit temperature sensors, is used for temperature sening. It offers linearly output proportional to the Celsius (Centigrade) temperature at no extra requirement of external calibration or trimming. It is available at fairly low cost owing to the trimming and calibration at the wafer level. Moreover the low output impedance of LM35, in addition to the higly linear output, and precise inherent calibration facilitates easy interfacing to the outside world. As shown in

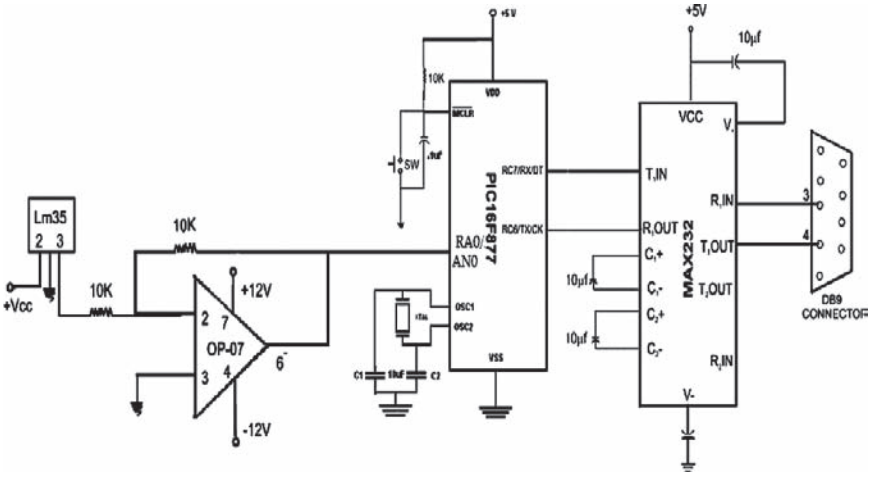


Fig. 4.5 Receiving sensor data on the HyperTerminal

Fig. 4.5 only a unity gain amplifer serves the prupose that too required for isolation. readout or control circuitry especially easy. In the present pplication it is used with single power supply of +5 V. The self heating is assured minimum as it draws a merely 60 μA of current from the supply. Onchip ADC of the PIC16F877 is used for digitization of the analog signal corresponding to the temperature data.

The datasheets of LM35 and OP-07 are available at online URL mentioned in the references at [49, 50].

With the execution of the following program the temperature value os displayed on the HyperTerminal.

Program 4.3 HyperTerminal interface: getting sensor signal on HyperTerminal

Program Source Code

****

\\ Program to Read the signal from the temperature sensor LM35 on HyperTerminal.

```
#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
int adc_data;
void main()
{
   setup_adc_ports(ALL_ANALOG);
   setup_adc(ADC_CLOCK_INTERNAL);
   set_adc_channel(0);
   while(1)
      {
      adc_data = read_adc();
      printf("\n\r Temperature %d",adc_data);
      }
}
```

****

4.5 HyperTerminal Based Control: Controlling an Actuator such as Relay from PC HyperTerminal

With this application, the relay connected to the port line RB4 of PIC16F877 microcontroller using HyperTerminal. This case study can be modified of course by addition of few more hardware components and using port lines to build a home automation project.

Program 4.4 Controlling an actuator such as relay from PC HyperTerminal

Program Source Code

****

```
* Program to control the Relay connected to the Microcontroller through PC
HyperTerminal */
// Relay connected to Pin RB4 of the PIC microcontroller.
```

```
#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
char a;
void main()
   {
         printf("\n\r Press O to turn on the Relay");
         printf("\n\r Press S to turn off the Relay");
         while(1)
            {
            a=putc(getc());
   if(a=='O')       // If Key pressed is 'O'
   {
      output_high(PIN_B4);            // turn on relay is connected to pin RB4
      delay_ms(500);
         }
         if(a=='S')// If Key pressed is 'S'
         {
      output_low(PIN_B4);//turnoff the relay
      delay_ms(500);
            }
   }}
**********************************************************
```

4.6 Controlling a Stepper Motor from HyperTerminal: HyperTerminal Keyboard Provides Direction

Program 4.5 Stepper motor control through HyperTerminal

Program Source Code

```
**********************************************************
\\ Program to control the stepper motor connected to PIC using PC
HyperTerminal.
/* Port D is connected to the stepper motor driver circuit.
Modes selection through HyperTerminal: - When
Key 'R' Run the Motor.
Key 'S' Stop the Motor.
Key 'C' Run the Motor in clockwise direction.
Key 'A' Run the Motor in Anti-clockwise direction.
```

```
#include<16f877.h>
#use delay(clock=20000000)
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
char a;
unsigned char mot[4] = {0×0a,0×06,0×05,0×09};
unsigned char run,clkwise;
int i;
void init(void);
void init()
{
unsigned char y,z;
y=putc(getc() );
if (y== 'R')
      run=1;
      if (y == 'S')
   run=0;
      if (y== 'C')
   clkwise=1;
      if (y == 'A')
   clkwise=0;
   }
void main()
      {
         printf("\n\r Press R to start the motor");
         printf("\n\r Press S to stop the motor");
         printf("\n\r Press C to rotate motor in clockwise direction");
         printf("\n\r Press A to rotate motor in Anti-clockwise direction");
{
init();
run=1;
clkwise=1;
while(1)
{
while(run==1)
{if(clkwise==1){
   for ( i=0;i<4 ; i++)
      { output_d(mot [i]);
      delay_ms( 100);
      }}
         else{for ( i=3;i<=0 ; i−)
      { output_d(mot [i]);
      delay_ms( 100);
}
}
}}}}
**********************************************************
```

Chapter 5
PIC Interfaced to I²C Compatible Devices

I²C stands for Inter-Integrated Circuit is a multi-master serial computer bus developed by Philips in 1980s. It is an ideal interface widely used for attaching low-speed peripherals to a embedded systems. Some of the popular basic applications of the I²C interface include EEPROMS, DDR2 SDRAM or NVRAM memory module interfacing, facilitating systems management for PCI cards commonly through an SMBus 2.0 connection, low speed ADC, DAC interfacing. The interface has also been widely used for changing contrast, hue, and color balance settings in display monitors, adjusting the volume settings in intelligent sound systems, data logging from sensors and adjusting fan speed. The popularity of the I²C interface has grown at a rpid pace in the audio and video application domains. The I²C protocol has now been adopted by several leading chip manufacturers like Xicor, SGS-Thomson, Siemens, Intel, TI, Maxim, Atmel, and Analog Devices. Following applications are developed in this chapter which will give you an overview of the interface aspects of this popular standard.

5.1 Details of I²C Interface

5.1.1 Basic Features

The I²C bus comprises of a bi-directional Serial Clock Line [SCL] and Serial Data Lines [SDA]. Both the SCL and SDA lines are pulled high via an Rp resistor as shown in Fig. 5.1. As per the datasheet the resistor Rs is optional, and used for ESD protection for 'Hot-Swap' devices.

J.S. Parab, et al., *Practical Aspects of Embedded System Design using Microcontrollers,*

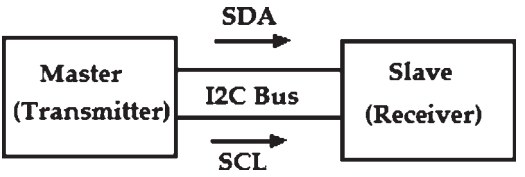


Fig. 5.1 Bidirectional interface for the I²C

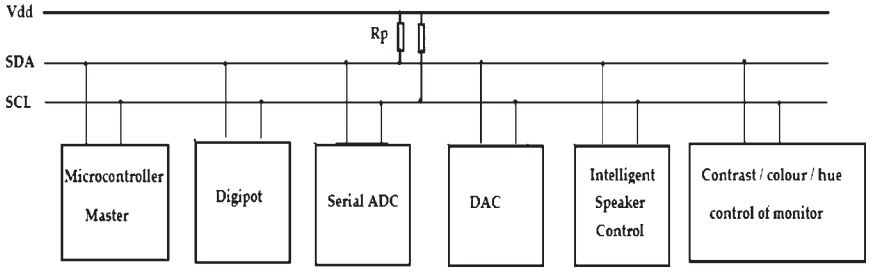


Fig. 5.2 Devices connected master slave mode using I²C bus

The physical topology of the I²C bus is two active wires and a ground connection. The active wires, called SDA and SCL, are both bi-directional (as shown in Fig. 5.1). SDA is the Serial Data line, and SCL is the Serial Clock line. Every device connected to the bus is assigned an unique address. Then these devices can act as a receiver and/or transmitter, depending on the configuration. Devices such as an LCD driver can act as only a receiver, while a memory, I/O chip, ASIC. Microcontroller can be both transmitter and receiver. The I²C bus is a multi-master bus in which more than one device is capable of initiating a data transfer to others connected on the bus. Once such a device goes into 'Bus Master' mode all the other devices acts as Bus Slaves (as shown in Fig. 5.2). Generally all the bus masters are microcontroller.

5.1.2 Sequence of Events in I²C Suite

I²C works as per the master/slave protocol. Master initiates the communication and then following sequence of events takes place for the communication purpose [59]:

1. The Master device issues a start condition. This condition conveys all the slave devices to listen on the serial data line for further instructions.
2. The Master device then sends the address of the target slave device along with a read/write flag.
3. The Slave device with the matching address responds with an acknowledgement signal to confirm the readiness for communication.
4. Communication then starts between the Master and the corresponding Slave on the data bus. Both the master and slave can receive or transmit data depending

Fig. 5.3 I²C communication protocol sequences of events

on whether the communication is a read or write. The transmitter sends 8-bits of data to the receiver which replies with a 1-bit acknowledgement.

5. When the communication is complete, the master issues a stop condition indicating that everything is done. The I²C frame is shown in Fig. 5.3.

5.1.3 Modes Supported by I²C

I²C protocol; supports the following modes [60]:

1. Master Transmitter Mode – Serial data output through SDA while SCL outputs the serial clock
2. Master Receiver Mode – Serial data is received via SDA while SCL outputs the serial clock
3. Slave Receiver Mode – Serial data and the serial clock are received through SDA and SCL
4. Slave Transmitter Mode – Serial data is transmitted via SDA while the serial clock is input through SCL
5. Data transfers up to 100 Kbps in standard mode and up to 400 Kbps in fast-mode.
6. Bi-directional data transfer
7. Own address and General Call address detection
8. Seven-bit addressing format
9. Fixed data width of 8 bits
10. Data transfer in multiples of bytes
11. One-byte write and read buffer

5.1.4 Synchronization and Arbitration in the I²C Bus

There are couple of situations in which care is to be taken for synchronizing between the devices. The situation arises due to the fact that each master has to generate its own clock signal. The varying clock signal rates causes synchronization problems. The arbitration issue arises to prevent more than one master to be active on the bus on any given moment. I²C masters must have the synchronization and arbitration logic to counter the above mentioned issues.

The internal logic goes on following lines. I²C master has two internal counters for counting the length of a high value and low value on the SLC line. With single

master present on the bus, these two counters define the clock frequency. With two or more masters connected in parallel, these counters fail to run at the same speed. Once this situation arises, it can be handled by synchronization and arbitration logic.

The multi-master situation is detected as the SLC clock line is pulled up to the supply voltage with an external pull-up resistor and has to pull low by the master to initiate the communication. Switching low of the SLC line by one of the masters, makes the other masters aware of the situation and inturn they reset their counter, regardless of the current count. Subsequently they all internally switch to the low state and their low-period counter starts counting. The situation reverses when the I²C master decides to switch the SLC line again to the high state after some time. The contention situation is automatically taken care of by the basic fact that the SLC clock line will not switch to the high state as long as there is at least one I²C master still counting the low period. Should the last I²C master decide to switch SLC to the high state; the clock line immediately changes state. All the masters with faster counters will be just waiting for this moment and starts counting the high line as son as the line switches state. This entails that the SLC clock line is defined by the fastest I²C master attached to the bus, and the length of the low period by the slowest participant.

5.1.5 Evolving Specifications of I²C Bus

After its development in 1982, the I²C protocol suite is constantly undergoing improvements. The improvements are updates regularly on many websites [61].

5.1.5.1 Version 1.0 – 1992

This version of the 1992 I²C-bus specification included the following modifications:

- Omission of the programming of a slave address by software. However, soon they realized the complications in and it was never implemented
- Omission of “low-speed mode”
- Addition of the Fast-mode which allows a fourfold increase of the bit rate up to 400 kbit/s still maintaining the downward compatibility
- Addition of 10-bit addressing which increased the additional slave addresses upto 1024
- Slope control and input filtering FOR FAST-MODE DEVICES was specified to make them EMC compliant

5.1.5.2 Version 2.0 – 1998

By this time the bus had penetrated in over 1,000 different ICs and licensed to more than 50 companies. The version 2.0 met those requirements and included the following modifications:

- Addition of the high-speed mode (Hs-mode). Inturn the bit rate goes up to 3.4 Mbit/s
- The low output level and hysteresis of devices with a supply voltage of 2 V and below has been adapted to meet the required noise margins and to maintain the compatibility with higher supply voltage devices
- Omission of the 0.6 V at 6 mA requirement for the output stages of Fast-mode devices
- The fixed input levels for new devices were replaced by bus voltage-related levels
- Application information for bi-directional level shifter was added

5.1.5.3 Version 2.1 – 2000

The Version 2.1 of the I²C-bus specification is the most current one. Following minor modifications are done:

- After a repeated START condition in Hs-mode, it is possible to stretch the clock signal SCLH
- Some timing parameters in Hs-mode have been relaxed

Details of the latest version may be obtained from Internet [61].

5.2 I²C Based Real Time Clock

The PCF8583 used in this application is a clock/calendar circuit based on a 2048-bit static CMOS RAM organized as 256 words by 8 bits. The mechanism of I²C bus has been used to pass addresses and data. Built-in word address register is automatically after each written or read data byte. Address pin A0 is used for programming the hardware address, allowing the connection of two devices to the bus without additional hardware.

The built-in 32.768 kHz oscillator circuit and the first 8 bytes of the RAM are used for the clock/calendar and counter functions. The next 8 bytes may be programmed as alarm registers or used as free RAM space. The remaining 240 bytes are free RAM locations.

The features are listed on website of NXP which is a top 10 semiconductor company founded by Philips [62]. The website further lists the features:

- I²C-bus interface operating supply voltage: 2.5–6 V? Clock operating supply voltage (0–+70°C): 1.0–6.0 V
- 240 × 8-bit low-voltage RAM
- Data retention voltage: 1.0–6 V
- Operating current (at fSCL = 0 Hz): max. 50 µA
- Clock function with four year calendar
- Universal timer with alarm and overflow indication
- 24 or 12 h format

- 32.768 kHz or 50 Hz time base
- Serial input/output bus (I^2C)
- Automatic word address incrementing
- Programmable alarm, timer and interrupt function
- Slave address:

– READ: A1 or A3
– WRITE: A0 or A2.

The pins and their designated functions are as follows:

Pin	Designated function
A0	Address inputs
INT	Comparator or an interrupt
SDA, SCL	I^2C bus
OSCIN, OSCOUT	Connect to 32,768 Hz watch crystal derived from PIC
Vcc	+5 V DC
GND	Ground

The registers available in the device are as follows:

00	Control/status
01	Hundredth of a second
02	Seconds
03	Minutes
04	Hours
05	Year/date
06	Weekdays/months
07	Timer
08	Alarm control
09–0f	Alarm registers or RAM
0f–ff	RAM

Complete datasheet may be downloaded from the web URL [63].

Program 5.1 I^2C Based real time clock (Refer Fig. 5.4)

Program Source Code

```
**************************************************************************
// I2C based Real time Clock
   DEVICE ADDRESS – 0XA0 – 0XA1
_________________________________________________________________________

#include <16F877.H>
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
#use I2C(MASTER,sda=PIN_C4,scl=PIN_C3)
unsigned int time[]={0×30,0×57,0×12};
unsigned int readtime[0×03];
unsigned long int hour,second,minute;
int i,j;
void set_rtc_time()
{
   for (i=2;i<=4;i++)
   {
   i2c_start();
```

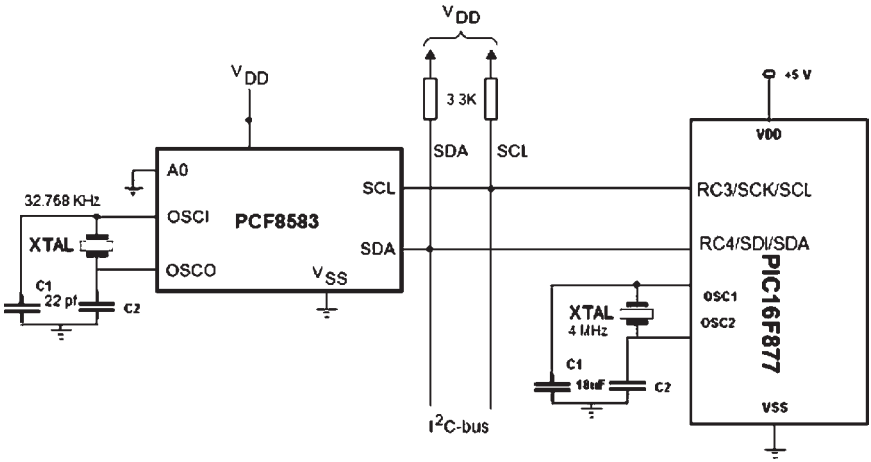


Fig. 5.4 I²C based real time clock

```
   i2c_write(0xa0 | 0x00);
   i2c_write(i);
   i2c_write(time[(i-2)]);
   i2c_stop();
   }
}
void get_rtc_time()
{
   for (i=2;i<=4;i++)
   {
      i2c_start();
      i2c_write(0xa0);
      i2c_write(i);
      i2c_start();
      i2c_write(0×a0 | 0×01);
   readtime[(i-2)]=i2c_read(0);
      i2c_stop();
   }
}
void main()
{
   set_rtc_time();
while(1)
   {
      get_rtc_time();
      hour = readtime[2];
      minute = readtime[1];
      second=readtime[0];
```

```
      printf(" Time : %× : %× : %× ≠n≠r",readtime[2],readtime[1],readtime[0]);
   }
}
**************************************************************************
```

The same code can be modified to display the time on LCD. The format of Time you will observe is 00:00:00(Hours:Min:Sec). In order ot generate this timing, the clock given to RTC chip is 32.768 kHz

5.3 Serial I²C Based EPROM 24AA256 Interface to PIC16F877

5.3.1 Where EPROM Fits in Embedded Systems?

An EEPROM or Electrically Erasable Programmable Read-Only Memory, comprises of arrays of floating-gate transistors on chip used for non-volatile storage of data. EPROM's are heavily used in Embedded Sensor Systems and Embedded Instrumentation. The sensor system to be pretested before deployment of the sensor for field applications can rely on the EPROM test readings for comparison and to decide whether it is working properly. The above mentioned testing can be carried out without removing the sensor from socket. Another application of the EPROM in Embedded Sensor System is storing the look up tables for calibration purpose. Even the nonlinearity of the sensors such as thermistor can be corrected using this method.

The serial EEPROM typically operates in three phases: OP-Code Phase, Address Phase and Data Phase. The OP-Code is usually the first 8-bits input to the serial input pin of the EEPROM device (or with most I²C devices, is implicit); followed by 8–24 bits of addressing depending on the depth of the device, then data to be read or written.

5.3.2 Advantages of Serial EPROM

Advantages of serial EPROM interfacing are covered in one of the application note by microchip [64]. They include:

1. Requirement of minimum number of I/Os typically two to four port lines to accomplish complete interfacing.
2. Small footprint. The available package size ranges for densities of 256–16 K bits, is space-saving 8 pin PDIP and 150 mil wide SOIC packaging.
3. Less current consumption typically of the order of 3 mA due to limited I/O ports.
4. Offers Byte programmability i.e. ability to erase and program one byte at a time without affecting the contents of the other memory locations in the array.
5. Supports clock rates of up to 6 MHz, however many devices are now I²C compliant and supports data rate upto 100 kHz and 400 kHz.

On the other hand the competitors pf serial EPROM's, the parallel ones offers some comparatively positive features interms of memory density and AC performance. However, in the age of memory constrained code development, the low memory space is the standard metrics of the Embedded Systems.

5.3.3 Serial EPROM Execution Cycle

Each EEPROM device comes with its own set of OP-Code instructions to map to different functions. In general the microcontroller and EPROMs with the serial interfaces such as I²C or SPI goes in a handshake manner with timely exchange of the control signals. Following signal are almost standard ones for the serial EPROMs:

- Write Enable (WREN)
- Write Disable (WRDI)
- Read Status Register (RDSR)
- Write Status Register (WRSR)
- Read Data (READ)
- Write Data (WRITE)

The above mentioned EEPROM devices supports some standard modes such as

- Program
- Sector Erase
- Chip Erase commands
- Byte programming

The reason behind the presentation of the I²C based serial EPROM interfacing to PIC microcontroller will be justified if one goes through Fig. 5.5.

In this application I²C compliant serial EPROM 24AA256 is interfaced to the PIC16F877.

5.3.4 Features of EPROM 24AA256

The Microchip Technology Inc. 24AA256/24LC256/ 24FC256 (24XX256*) is a 32 K × 8 (256 Kbit) Serial Electrically Erasable PROM, capable of operation across a broad voltage range (1.8–5.5 V). The thrust application areas for this low power device are personal communications and data acquisition. Microchip applications note on this 256 K I²C CMOS Serial EEPROM lists the features of this device [65].

- Constructed with Low-power CMOS technology:
  – Maximum write current 3 mA at 5.5 V
  – Maximum read current 400 μA at 5.5 V
  – Standby current 100 nA typical at 5.5 V

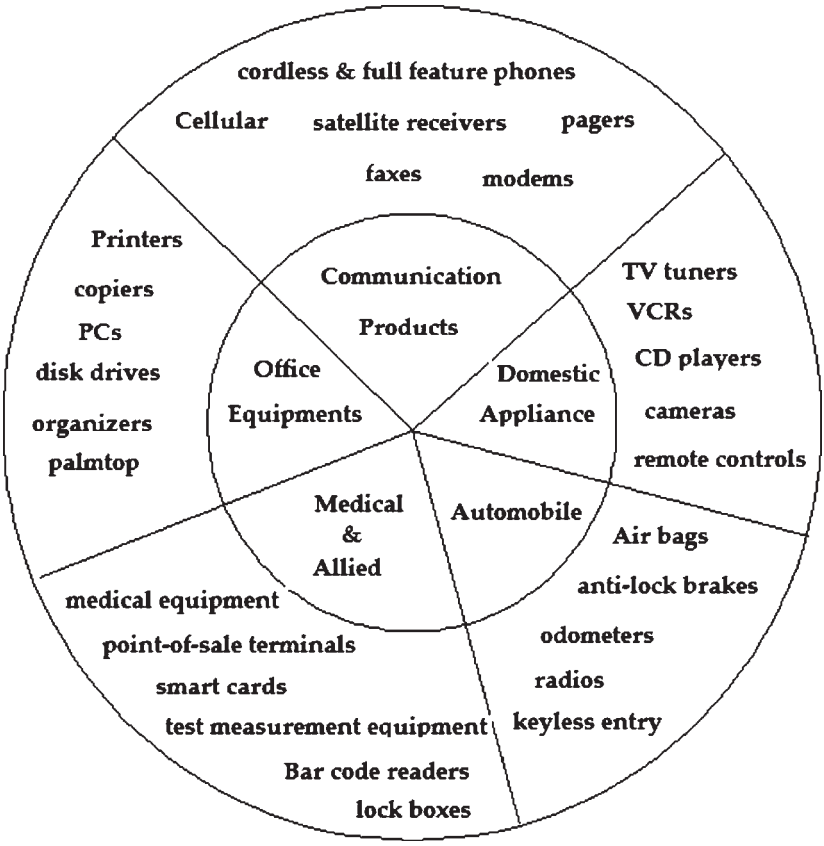


Fig. 5.5 Various applications of serial EPROM in day to day life

- Two-wire serial interface bus, I²C compatible
- Cascadable for up to eight devices
- Self-timed erase/write cycle
- Sixty-four-byte Page Write mode available
- Five milliseconds maximum write cycle time
- Hardware write-protect for entire array
- Output slope control to eliminate ground bounce
- Schmitt Trigger inputs for noise suppression
- One million erase/write cycles
- Electrostatic discharge protection > 4,000 V
- Data retention > 200 years
- Eight-pin PDIP, SOIC, TSSOP, MSOP and DFN packages, 14-lead TSSOP package
- Standard and Pb-free finishes available

5.3.5 Interfacing Aspects

The A0, A1 and A2 inputs are designated as “User Configurable Chip Select” signals.

Multiple devices can be connected by setting appropriate logic levels on these pins. With three select lines maximum eight devices can be connected. The logic levels on these pins are compared with the corresponding bits in the slave address and accordingly the chip selection is done. In the present application, (shown in Fig. 5.6) the MSOP package of 24AA256 is used for which the pins A0 and A1 are not connected as per instructions in the data sheet.

Program 5.2 Read and write data to/from I²C based EPROM

Program Source Code

```
*************************************************************************
// Program to Read /Write the Data to/from serial EEPROM
// Address - ‘1010 0010’
_____________________________________________________________________
#include <16F877.H>
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
#use I2C(MASTER,sda=PIN_C4,scl=PIN_C3)
int address;
int data;
void main()
{
printf(“≠n≠r EEPROM Write Data is 0×55”);
      i2c_start();
   i2c_write(0×a2);
      i2c_write(0×00);
      i2c_write(0×00);
      i2c_write(0×55);
      i2c_stop();
while(1)
   {
      i2c_start();
      i2c_write(0xa2);
      i2c_write(0);
      i2c_write(0);
      i2c_start();
      i2c_write(0xa3);
      data=i2c_read(0);
      i2c_stop();
      printf(“≠n≠r %× ”,data);
         output_d(data);      // Port D s connected to LEDS
   }
```

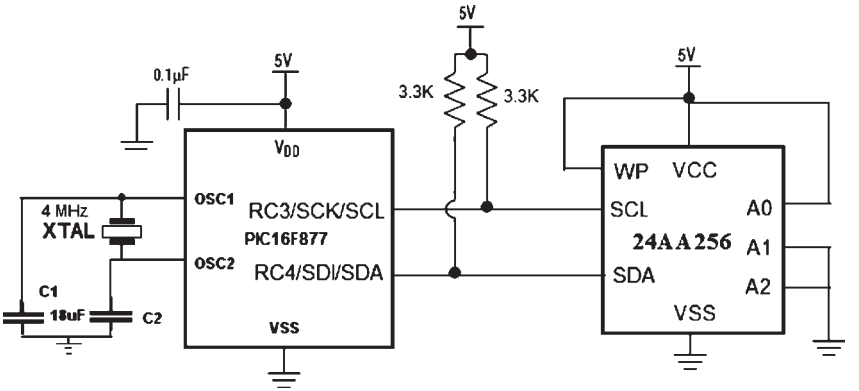


Fig. 5.6 Serial EPROM connected to PIC16F877

```
}
**************************************************************************
```

Here you will observe that, the pattern/data stored in serial EPROM, is read and displayed again both on the HyperTerminal and LEDs connected to port D of the PIC.

5.4 I²C Based PCF 8591ADC Interface

5.4.1 Advantages of Serial ADC Interface

This application demonstrates a serial ADC interfacing to the PIC microcontroller. At the outset, it is worthwhile to take a review of the pros and cons of the serial ADC. A conventional parallel ADC is generally connected directly or possibly through buffers to the data bus of the microcontroller that takes care of the interfacing needs. The advantage of this scheme is reading the digitized output at a time, however at the expense of more hardware and built in multiplexer in case of multi-channel versions of such ADCs. Serial ADC accomplishes the post digitization transfer of the signal in a serial fashion over a single line. A common clock in a typical I²C or SPI interface dictates the transfer rate and in multichannel versions, additional bits corresponding to the channel IDs are incorporated. The key advantage of the serial ADCs and for that matter any other serial peripherals can be appreciated from a space saving point of view. Due to the above mentioned advantage manufacturers offer these devices in an 8-pin DIP or SO package and designers get the benefit of simpler PCB deign with few tracks for their embedded product.

5.4.2 PCF 8591 I²C Compliant Serial ADC

The PCF8591 is a single-chip, single-supply low power 8-bit CMOS data acquisition device with four analog inputs, one analog output and a serial I²C-bus interface manufactured by Philips. In compliance with the I²C protocol, three address pins A0, A1 and A2 are used for configuring the hardware address. Eight such devices can be connected to the I²C-bus without additional hardware.

PCF8591 is widely used in applications such as analog input multiplexing, on-chip track and hold function, 8-bit analog-to-digital conversion and an 8-bit digital-to-analog conversion. The maximum conversion rate is only limited by the maximum data transfer rate of the I²C bus.

5.4.3 PCF 8591 Features

The features of this device with a detailed data sheet can be downloaded from the Internet [66]. The features are as follows:

- Single power supply
- Operating supply voltage 2.5–6 V
- Low standby current
- Serial input/output via I²C-bus
- Address by three hardware address pins
- Sampling rate given by I²C-bus speed
- Four analog inputs programmable as single-ended or differential inputs
- Auto-incremented channel selection
- Analog voltage range from VSS to VDD
- On-chip track and hold circuit
- Eight-bit successive approximation A/D conversion
- Multiplying DAC with one analog output

Application notes in the following domain may be seen on web [66]:

- Closed loop control systems
- Low power converter for remote data acquisition
- Battery operated equipment
- Acquisition of analog values in automotive, audio and TV applications

5.4.4 A/D Conversion of PCF8591

The A/D converter of PCF8591 is based on successive approximation conversion technique. Conversion cycle is initiated at the trailing edge of the acknowledge clock pulse and is carried on simultaneously with the transmission of the result

of the previous conversion. After initiation of the conversion cycle, the input voltage of the selected channel is sampled, stored on chip and then digitized in 8 bit binary equivalent. The ADC has on chip data registers for storage of results and an auto increment flag to indicate selection of the next channel. A self testing of the device can be done by doing the power on reset and checking the first byte as 80 H.

As a CMOS device certain precautions are to be taken care of such as connecting the inputs either to to VSS or VDD when not in use. Also connecting the analog inputs AGND or VREF when they are not required. The excessive ground and supply noise and cross talk can be minimized by careful PCB design. Decoupling capacitors (>10 µF) are recommended for power supply and reference voltage inputs [67].

Program 5.3 I²C based PCF8591 ADC interface to PIC16F877 (Refer Fig. 5.7)

Program Source Code

```
**************************************************************************
/* Program illustrates reading the data from the I2C Based ADC PCF8591 to the
PIC16F877. */
```

```
#include <16f877.H>
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
#use I2C(MASTER,sda=PIN_C4,scl=PIN_C3)
#include<lcd1.h>
int data;
void main()
{
```

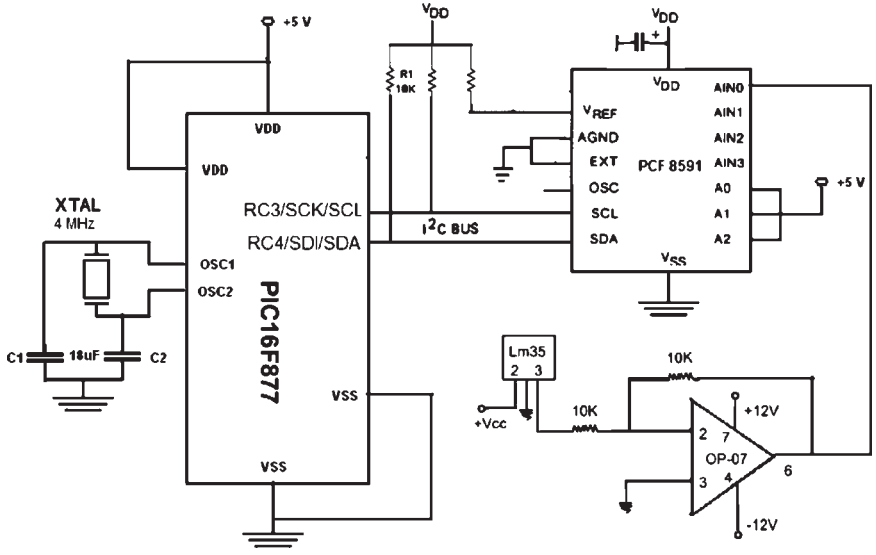


Fig. 5.7 I²C Based PCF 8591 ADC interface to PIC16F877

```
        i2c_start();
        i2c_write(0×9e);        //Device Address
        i2c_write(0×41);        //Analog input with channel 1
        lcdint();
        while(1){
        i2c_start();
        i2c_write(0×9f);        //Device Address
    data=i2c_read(0);
        lcd_data(data,0×a2);
        printf("≠n≠r adc Data : %x ",data);
        i2c_stop();
    }
}
*********************************************************************
```

Here the temperature sensor LM35 is connected to channel 0 (AIN0,), The current temperature is displayed on LCD and also on HyperTerminal.

5.5 I²C Based ADC – AD1236

5.5.1 AD1236 from Maxim

In case your embedded system product needs more bit capability and resolution than the PCF8591, then this application will definitely help you. It is based on the MAX1236 a low-power, 12-bit, multi-channel analog-to-digital converters (ADCs) having additional features such as internal track/hold (T/H), voltage reference, clock, and most important an I²C-compatible 2-wire serial interface. The supply requirement is as low as a single supply of 4.5–5.5 V with a current consumption of only 670 µA at the maximum sampling rate of 94.4 ksps. In case, the application demands further lowering down the rated current then the ADC can be operated at less than 230 µA for sampling rates under 46 ksps. The ADC uses a unique "Auto Shutdown" technology that powers down the devices between conversions, reducing supply current to less than 1 µA at low throughput rates. The MAX1236 comes with four analog input channels, software configurable fully differential analog inputs for unipolar or bipolar, and single-ended or differential operation. It features a 4.096 V internal reference is available in an 8-pin µMAX® package.

5.5.2 Features of MAX1236

The key features of this device are listed on the web URL of Maxim [68]. Some of them are as follows:

- High-Speed I²C-Compatible Serial Interface
  - 400 kHz Fast Mode
  - 1.7 MHz High-Speed Mode
- Single-Supply 4.5–5.5 V
- Internal Reference 4.096 V
- External Reference: 1 V to VDD
- Internal Clock
- Four-Channel Single-Ended or 2-Channel Fully Differential input
- Internal FIFO with Channel-Scan Mode
- Low Power consumption at different conversion rates.
  - 670 µA at 94.4 ksps
  - 230 µA at 40 ksps
  - 60 µA at 10 ksps
  - 6 µA at 1 ksps
  - 0.5 µA in Power-Down Mode
- Software-Configurable Unipolar/Bipolar
- Small Packages 8-Pin µMAX

5.5.3 Conversion Technique and Other Details

The MAX1236ADCs uses successive-approximation conversion techniques along with built in fully differential input track/hold (T/H) circuitry to capture and convert an analog signal to a serial 12-bit digital output. Owing to its low power requirement and adaptability to different conversion rates this ADC finds lots of applications in the following embedded domains:

- Battery-Powered Test Equipment
- Handheld Portable Applications
- Medical Instruments
- Receive Signal Strength Indicators
- Solar-Powered Remote Systems
- System Supervision

Application notes on these domains are available on Internet [69].

Program 5.4 Read the serial value from I²C based ADC MAX1236 (Refer Fig. 5.8)

Program Source Code

```
**********************************************************************
/* This program is used to read the adc value from I2C BASED ADC - MAX1236
having ADDRESS - '01101000' i.e. 68 or 69 in hexadecimal. */
```

```
#include <16F877.h>
#use delay (clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6,rcv=PIN_C7)
#use I2C(MASTER, sda=PIN_C4,scl=PIN_C3)
```

```
#include<lcd1.h>
unsigned int adc_datahi,adc_datalw;
unsigned int16 adc_data;
Void main ()
{
   lcdint ();
   While (1)
      {
      i2c_start();
      i2c_write(0×68);       // device address
      i2c_write(0×fa);
      i2c_write(0×01);
      i2c_stop();
      delay_ms(10);
      i2c_start();
      i2c_write(0×69);
      adc_datahi = i2c_read();
      adc_datalw = i2c_read();
      adc_data=make16(adc_datahi,adc_datalw);
      i2c_stop();
      adc_datahi = adc_datahi &0×0f;
      lcd_data(adc_data,0×a2);
      printf("≠n adc %x%x ",adc_datahi,adc_datalw);
   }
      }
```

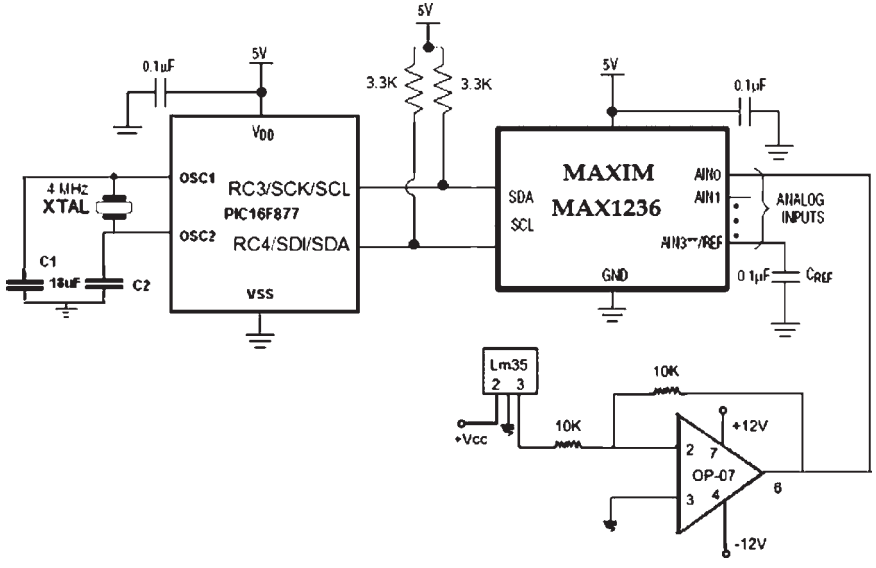


Fig. 5.8 I²C based ADC MAX1236 interface to PIC16F877

```
**********************************************************************
```

Here the temperature sensor LM35 is connected to channel 0 (AIN0,). The current Temperature sensed is displayed on LCD and also on HyperTerminal.

5.6 MAX5822 DAC Interfaced to PIC

The MAX5822 is a dual, 12-bit, voltage-output, I²C compliant digital-to-analog converter with clock frequency upto 400 kHz and single supply voltage from 2.7 to 5.5 V. The device is an ideal match for battery powered instrumentation owing to its low current consumption of the order of 115 µA at 3.6 V supply voltage supported by a power-down mode which lowers the current consumption to less than 1 µA. Few more features that makes it value addition for the embedded instrumentation are CMOS compatible inputs buffered with Schmitt triggers, facilitating direct interfacing to opto-coupled and transformer-isolated interfaces. The digital noise arising due to address mismatch in I²C network is minimized by feedthrough network that disconnects the clock (SCL) signal from the rest of the network upon detection of address mismatch. The miniature 8-pin µMAX® package further helps in portability aspects.

5.6.1 Features

- Features listed on the Maxim website [70] are as follows:
- Ultra-Low Supply Current
  - 115 µA at VDD = 3.6 V
  - 135 µA at VDD = 5.5 V
- 300 nA Low-Power Power-Down Mode
- Single 2.7–5.5 V Supply Voltage
- Fast 400 kHz I²C-Compatible 2-Wire Serial Interface
- Schmitt-Trigger Inputs for Direct Interfacing to Optocouplers
- Rail-to-Rail Output Buffer Amplifiers
- Three Software-Selectable Power-Down Output Impedances
  - 100 kΩ, 1 kΩ, and High Impedance
- Read-Back Mode for Bus and Data Checking
- Power-On Reset to Zero
- Eight-Pin µMAX Package

The application domains [70] mentioned for this DAC are

- Battery-Powered Instrumentation
- Digital Gain and Offset Adjustments
- Low-Cost Instrumentation
- Programmable Attenuators
- Programmable Voltage and Current Sources
- VCO/Varactor Diode Control

5.6.2 Equation for Output Voltage

The MAX5822's input coding is binary. The output voltage is given by the following equation:

$$Vout = (Vref \times D) / 2^N$$

where,

N = 12 (bits)

D = the decimal value of the input code ranging from 0 to 4095

Program 5.5 Sine wave generation by using Max5822 DAC

Program Source Code

```
**********************************************************************
/* program illustrate generation of sine wave by using Max5822 DAC */
_____________________________________________________________________

#include <16f877.h>
#use delay(clock=20000000)
#use I2C(MASTER,sda=PIN_C4,scl=PIN_C3)
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
//#use I2C(MASTER,sda=pin_c4,scl=pin_c3)
int i,j;
unsigned int16 ab[]={2047,2403,2747,3071,3363,3615,3820,3971,4063,4095,4063,
3971,3820,3615,3363,3071,2747,2403,2047,1692,1347,1024,731,479,274,123,31,
0,31,123,274,479,731,1024,1347,1692,7899};
void main()
{
   i2c_start();
   i2c_write(0×72);
   i2c_write(0×F0);
   i2c_write(0×04);
   i2c_stop();
while(1)
{
   i2c_start();
   i2c_write(0×72);
   for (i=0;i<37;i++)
   {
      i2c_write(ab[i]/0×100);
      i2c_write(ab[i]%0×100);
   }
   i2c_stop();
}
}
**********************************************************************
```

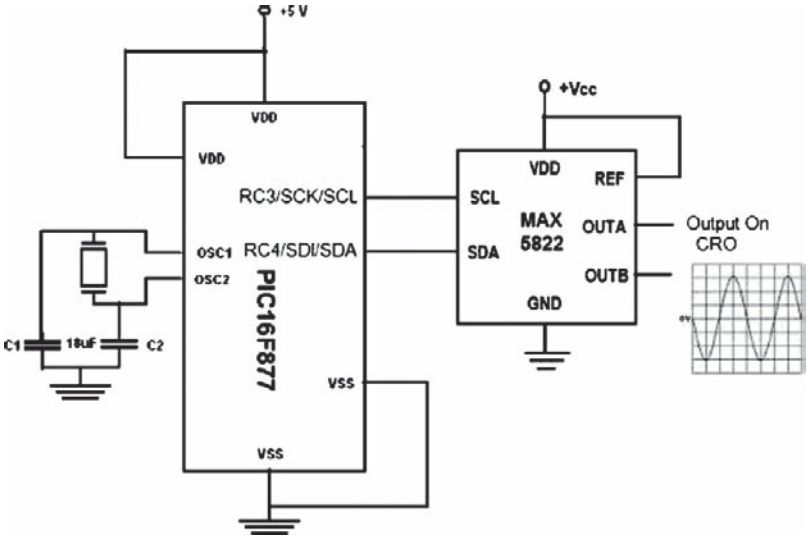


Fig. 5.9 DAC MAX5822 interfacing to PIC16F877

Program 5.6 Square wave generation by using Max5822 DAC (Refer Fig. 5.9)

Program Source Code

```
**************************************************************************
// this program to generate square wave by using Max5822 DAC
```

```
#include <16f877.h>
#use delay(clock=20000000)
#use I2C(MASTER,sda=PIN_C4,scl=PIN_C3,slow)
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
int i,j;
void main()
{
   i2c_start();
   i2c_write(0×72);
   i2c_write(0×F0);
   i2c_write(0×04);
   i2c_stop();
while(1)
{
   i2c_start();
   i2c_write(0×72);
   i2c_write(0×0f);
   i2c_write(0×ff);
   i2c_stop();
   i2c_start();
```

```
   i2c_write(0×72);
   i2c_write(0×00);
   i2c_write(0×0);
   i2c_stop();
}
}
```

****

Program 5.7 Triangular wave generation by using Max5822 DAC (Refer Fig. 5.9)

Program Source Code

****

```
// program illustrate generation of Triangular wave by using Max5822 DAC
```

```
#include <16f877.h>
#use delay(clock=20000000)
#use I2C(MASTER,sda=PIN_C4,scl=PIN_C3,slow)
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
int i,j;
void main()
{
   i2c_start();
   i2c_write(0×72);
   i2c_write(0×F0);
   i2c_write(0×04);
   i2c_stop();
while(1)
{
   for(i=0×0;i<0×0f;i++)
      {
      for(j=0×0;j<0×ff;j++)
         {
         i2c_start();
         i2c_write(0×72);
         i2c_write(i);
         i2c_write(j);
         i2c_stop();
         }
      }
   for(i=0×0f;i>0×0;i–)
      {
      for(j=0×ff;j>0×0;j–)
         {
         i2c_start();
         i2c_write(0×72);
         i2c_write(i);
         i2c_write(j);
```

```
        i2c_stop();
        }
    }
}
}
```

****

Program 5.8 Negative ramp wave generation by using Max5822 DAC (Refer Fig. 5.9)

Program Source Code

****

```
/* program illustrate generation of negative Ramp wave by using Max5822 DAC */
-------------------------------------------------------------------------------
#include <16f877.h>
#use delay(clock=20000000)
#use I2C(MASTER,sda=PIN_C4,scl=PIN_C3,slow)
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
int i,j;
void main()
{
   i2c_start();
   i2c_write(0×72);
   i2c_write(0×F0);
   i2c_write(0×04);
   i2c_stop();
while(1)
{
   for(i=0;i<0×0f;i++)
      {
      for(j=0;j<0×ff;j++)
         {
         i2c_start();
         i2c_write(0×72);
         i2c_write(i);
         i2c_write(j);
         i2c_stop();
         }
      }
}
}
```

****

Program 5.9 Positive ramp wave generation by using Max5822 DAC (Refer Fig. 5.9)

Program Source Code

****

```
/* program illustrate generation of positive going ramp wave by using Max5822 DAC
-------------------------------------------------------------------------------
#include <16f877.h>
```

```
#use delay(clock=20000000)
#use I2C(MASTER,sda=PIN_C4,scl=PIN_C3,slow)
#use rs232(baud=19200,xmit=pin_c6,rcv=pin_c7)
int i,j;
void main()
{
   i2c_start();
   i2c_write(0×72);
   i2c_write(0×F0);
   i2c_write(0×04);
   i2c_stop();
while(1)
{
   for(i=0×0f;i>0×0;i–)
      {
      for(j=0×ff;j>0×0;j–)
         {
         i2c_start();
         i2c_write(0×72);
         i2c_write(i);
         i2c_write(j);
         i2c_stop();
         }
      }
}
}
**********************************************************************
```

Chapter 6
Embedded Control Applications Using AT89S52

This chapter is a complete diversion from the PIC16F877. Here we are back to a very popular CISC architecture i.e. Atmel 89S52. Let us justify the reasons for the departure from what follows right away since Chapter 2. At the outset, we submit that the talk of the town "RISC versus CISC debate" has almost come to an end with the state of art complex microprocessor architectures, ASICs and commercial off-the-shelf (COTS) architectures. However, the market statistics won't forgive us if we ignore the fact that today the MCS51 series accounts for the more than 25% of the 8 bit Microcontroller worldwide market, with part volumes exceeding 100,000,000 or more units per annum. We must give justice to the investment to the tune of billions of dollars and most importantly programming expertise that comes from millions of programmer hours apart from the efforts taken to launch full proof IDEs, compliers, debuggers and other tools. Therefore we have taken the most popular representative sample of MCS51 series i.e. Atmel 89S52 for the applications chosen in this chapter. Following case studies are developed in this chapter:

6.1 Night Lamp Controller
6.2 Microcontroller Based Control for Nylon Rubber Stamp Making Machine Control
6.3 A Tiny BIOS or Diagnostic Interface with MCS51
6.4 Simple Digital IC Tester
6.5 Microcontroller Based Salinity Measurement System
6.6 Fault Tolerant Sensor Interface
6.7 Sensor Matrix Interface
6.8 Design Micricontroller Based Servo Controller

6.1 Night Lamp Controller

In this application automatic switching of night lamp is implemented using the on-chip timer of the AT89S52 microcontroller (as shown in Fig. 6.1). The time elapsed is displayed on the LCD. This application will not only relieve the user from the turning on and off the lamp but also lessens the electricity consumption and sometimes also useful to give an illusion of someone's presence for the thieves.

J.S. Parab, et al., *Practical Aspects of Embedded System Design using Microcontrollers*,

A slight modification by introducing a light sensing module will make this circuit fully automatic.

Program 6.1 Night lamp controller

Program Source Code

```
**********************************************************************
// Program illustrates the Night lamp controller using the microcontroller.
----------------------------------------------------------------------
#include <REG52.H>                  /* special function register declarations */
                                    /* for the intended 8051 derivative */
sbit RS = P2^0;                     // LCD Initialization signals
sbit RW = P2^1;
sbit EL = P2^2;
sbit Relay = P2^3;                  // Relay connected to P2.3 pin of the
                                      controller
sbit R1 = P3^0;                     //Key board signals
sbit R2 = P3^1;
sbit R3 = P3^2;
```

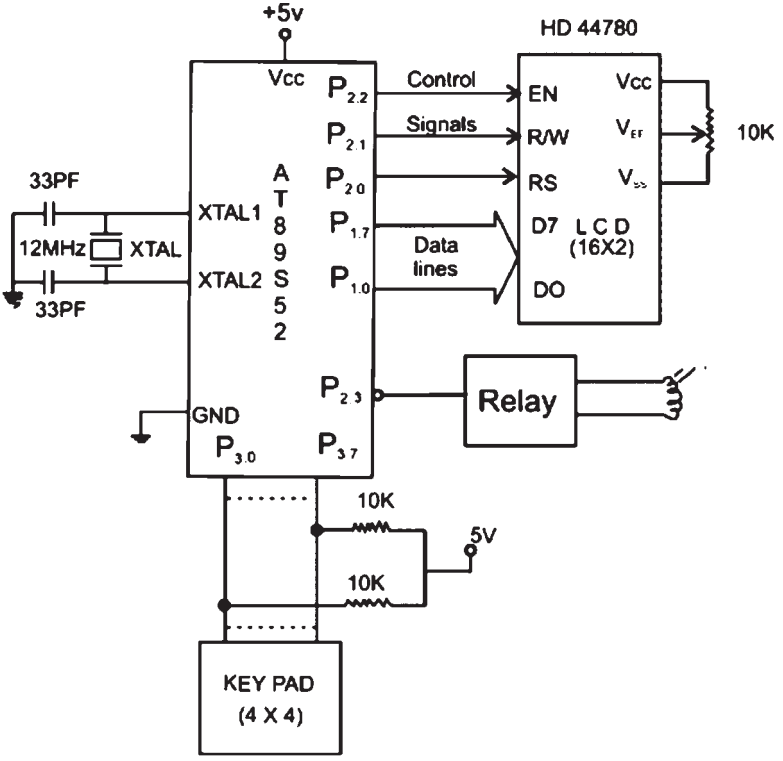


Fig. 6.1 Circuit diagram for night light controller using 89S52 microcontroller

```
sbit R4= P3^3;
sbit C1 = P3^4;
sbit C2 = P3^5;
sbit C3 = P3^6;
sbit C4 = P3^7;
void Delay(int);
void INIT(void);
void ENABLE(void);
void LINE(int);
int keyb(void);
void settime(void);
void starttimer(void);
void check_timeout(int);
int settime1=10;                    // Default time
int set_temp;
void settime(void)                  // Routine to set time to keep the lamp on
{
int ten_pos=0, unit_pos=0;
while(!(ten_pos=keyb()));
while(!(unit_pos=keyb()));
P1=(((ten_pos))+0×30);
ENABLE();
P1=((unit_pos)+0×30);
ENABLE();
Delay(2000);
settime1= ten_pos*10 + unit_pos;
}
void starttimer()
{
Relay=1;                            // Turn on the Lamp
set_temp= settime1;
check_timeout(set_temp);
set_temp−;
P1=(((set_temp)/10)+0×30);          // display the remaining time on LCD
ENABLE();
P1=((set_temp/10)%10+0×30);         // display the remaining time on LCD
ENABLE();
Delay(2000);
}
void check_timeout(g)               // check if count is zero and according
                                       switch off the Lamp
{
if(g == 0)
{
Relay=0;                            // turn off the Lamp
```

```
}
}
void main(void)
{
char test[]="Night lamp controller";
char code set_msg[]=" 1-> set time";
char code start_msg[]=" 2-> start timer ";
char *p;
int j;
INIT();                                  // Initaialization routine for LCD
LINE(1);
p=&test;
for(j=0;j<17;j++)
{
if(j==0)LINE(1);
if(j==8)LINE(2);
P1=*p++;
ENABLE();
Delay(200);
}
while(1){
RS = 1;
p=&set_msg;
for(j=0;j<17;j++)
{
if(j==0)LINE(1);
if(j==8)LINE(2);
P1=*p++;
ENABLE();
Delay(200);
}
RS = 1;
p=&start_msg;
for(j=0;j<17;j++)
{
if(j==0)LINE(1);
if(j==8)LINE(2);
P1=*p++;
ENABLE();
Delay(200);
}
j=0;
j=keyb();
if(j>0);
```

```
if(j==1){                    // if key 1 is pressed then set the timer to
                                keep the Lamp oN
settime();
}
if(j==2){                    // if key 2 is pressed then start the timer to
                                count in down mode
starttimer();
}
}
}
void Delay(int k)            // Dealay routine
{
int i,j;
for (j=0;j<k+1;j++){
for (i=0;i<100;i++);
}}
void ENABLE(void)
{
EL=1;
Delay(1);
EL=0;
Delay(1);
}
void LINE(int i){
   if (i==1) {
   RS=0;
   RW=0;
   P1=0×80;
   ENABLE();
   RS=1;
   }
   else
   {
   RS=0;
   RW=0;
   P1=0×C0;
   ENABLE();
   RS=1;
   }
}
void INIT(void)
{
   RS=0;
   RW=0;
   EL=0;
```

```
   P1 = 0×38;
   ENABLE();
   ENABLE();
   ENABLE();
   ENABLE();
   P1 = 0×06;
   ENABLE();
   P1 = 0×0E;
   ENABLE();
   P1 = 0×01;
   ENABLE();
}
int keyb(void){
int key=0;
C1=1;
C2=1;C3=1;C4=1;R1=0;R2=1;R3=1;R4=1;
if (C1==0) key = 1;
if (C2==0) key = 2;
if (C3==0) key = 3;
if (C4==0) key = 4;
return(key);
}
**********************************************************************
```

6.2 Microcontroller Based Control for Nylon Rubber Stamp Making Machine

Rubber stamping is a craft in which some type of ink made of dye or pigment is applied to an image or pattern that has been carved, molded, laser engraved or vulcanized, onto a sheet of rubber. Spanish explorers were pioneers in developing this "sticky substance" that bounced, used by South American Indians. In 1736, Charles Marie de la Condamine, a French scientist studying the Amazon, sent a piece of "India Rubber" back to France [71]. With the widespread use of the rubber stamps, there are good number of manufacturers making the machines for rubber stamps [72]. Most of these machines are based on the vulcanizing press mechanism. It is the most cost effective methods of making rubber stamps and leads to cost effectiveness especially in case of mass produced stamps. In this process a batch of stamps is produced using a mould. For making the mould a standard process is adopted. The first step is making a master plate manufactured from metal or polymer. The master plate comprises of the necessary artwork/text to be used for making an impression in the mould. Thus made master plate is then pressed into the matrix board which further acts on the rubber. A definite cycle of heat and pressure is applied to the master plate and matrix board inside a Vulcanizing press. With this

the Matrix follows the shape of the artwork/text provided by the master plate, which subsequently hardens on cooling. Once the mould is ready as per the above process, the raw rubber stamp gum is placed on top of the mould and then placed inside the stamp press. Hydraulic pressure is placed upon the rubber and the mould from within the stamp press causing the rubber to melt into the areas of the mould that contain the images and text, curing and hardening takes about 10 min. Once cured the sheet of rubber is pulled away from the mould and cut up into individual stamps to be affixed to mounts [73].

Most of the rubber stamp making machines has a built in mechanism to accomplish the manufacturing process described above. However, for following this process an experienced human operator is required which will decide the pressure and temperature cycles to be applied? In this application we have automated the entire process by incorporating the 89S52 microcontroller based system design that will suffice most of the manual rubber stamp making machines.

The microcontroller based automation as shown in Fig. 6.2, leads to certain definite advantages such as consistency, less manufacturing time and simplified operation. The consistency is achieved with the exact application of downward pressure and dwell-time set by the thumbwheel switches. The hot-stamping operation is also completed within a very short amount of time with exact application of the power through the relay mechanism. The LCD display not only ensures the ease of operation, but also provides some valuable information such as the number of imprints completed, process temperature and pressure. The operation commences with the application of paddle operated switch.

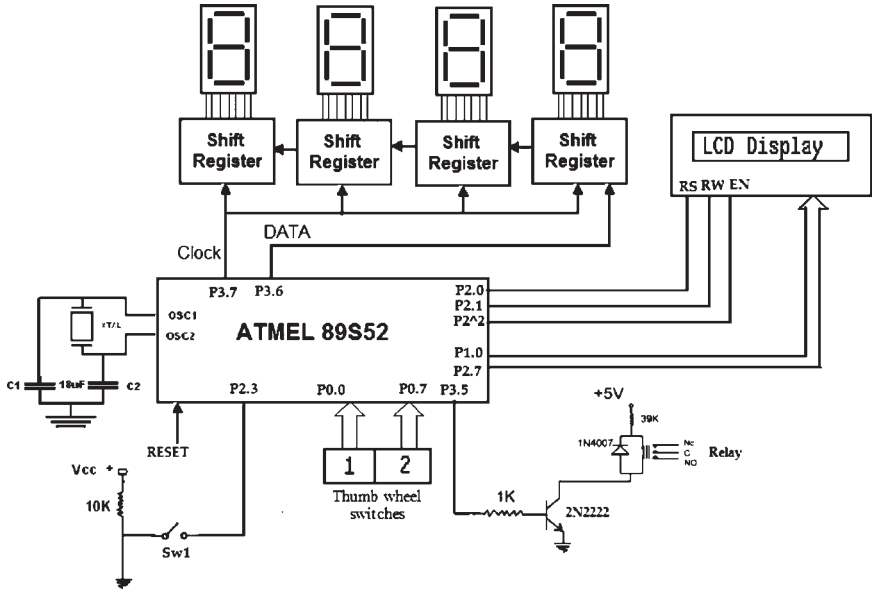


Fig. 6.2 Microcontroller based control for nylon rubber stamp making machine

Program 6.2 Microcontroller based control for nylon rubber stamp making machine

Program Source Code

```
**********************************************************************
/* Program Illustrate the Microcontroller 89S52 based control for Nylon Rubber
Stamp Making Machine.*/
-----------------------------------------------------------------------
#include <REG52.H>                  /* special function register declarations */
                                    /* for the intended 8051 derivative */
#include <stdio.h>                  /* prototype declarations for I/O functions */
sbit RS = P2^0;
sbit RW = P2^1;
sbit EN = P2^2;
sbit Sw1= P2^3;
sbit RL=P3^5;
sbit dat=P3^6;
sbit clk= P3^7;
unsigned int unit,ten,hundred,tenth,tens,tenthou,thousand;
void delay(int);                    /* Stop Exection with Serial Intr. */
void INIT(void);
void ENABLE(void);
void LINE(int);
void display(char );
void clear(void);
void delay(void);
char
seq[16]={0XC0,0XF9,0XA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90,0X88,
0X80,0XC6,0XC0,0X86,0X8E};
void main (void) {
int i,N;
while (1) {
N=P1;
if (Sw1==0)
   {
        unit =(N%10);
        ten=(N/10)%10;
        hundred=(N/100)%10;
tenth=(N/1000);
clear();
display (seq[tenth]);
display (seq[hundred]);
display (seq[ten]);
display (seq[unit]);
INIT();
LINE(1);
```

```
P1=tenthou;
ENABLE();
P1=thousand;
ENABLE();
P1=hundred;
ENABLE();
P1=tens;
ENABLE();
P1=unit;
ENABLE();
RL=1;
delay ();
clear ();
}
else
{
for (i=N; i>=0;i–)
      { unit =(N%10);
        ten=(N/10)%10;
        hundred=(N/100)%10;
tenth=(N/1000);
clear();
display (seq[ten]);
display (seq[unit]);
delay();
INIT();
LINE(1);
P1=tenthou;
ENABLE();
P1=thousand;
ENABLE();
P1=hundred;
ENABLE();
P1=tens;
ENABLE();
P1=unit;
ENABLE();
}
RL=0;
   }
}
}
void LINE(int i){
      if (i==1) {
      RS=0;
```

```
        RW=0;
        P1=0×80;
        ENABLE();
        RS=1;
        }
        else
        {
        RS=0;
        RW=0;
        P1=0×C0;
        ENABLE();
        RS=1;
        }
}
void delay()
{
int i,j;
for (j=0;j<10;j++){
for (i=0;i<100;i++);
}}
void ENABLE(void)
{
EN=1;
delay();
EN=0;
delay();
}
void INIT(void)
{
        RS=0;                          /*initialization of LCD display*/
        RW=0;
        EN=0;
        P1 = 0×38;
        ENABLE();
        ENABLE();
        ENABLE();
        ENABLE();
        P1 = 0×06;
        ENABLE();
        P1 = 0×0E;
        ENABLE();
        P1 = 0×01;
        ENABLE();
}
void display(char k)
```

```
{
unsigned char mask;
int i;
mask=0X80;
for(i=0;i<8;i++)
{
      if(k&mask)
      dat=1;
      else
      dat=0;
      clk=0;
      clk=1;
   clk=0;
      mask=mask>>1;
}
}
void clear(void)
{
int i;
for (i=0;i<33;i++)
{
dat=1;
clk=0;clk=1;clk=0;
}
}
```

****

6.3 A Tiny BIOS or Diagnostic Interface with MCS51

BIOS, in computing, stands for Basic Input/Output System that refers to the software code run by a computer after powered on. Its primary function is to prepare the machine so other software programs stored on various media (such as hard drives, floppies, and CDs) can load, execute, and assume control of the computer.

In this case study a program is developed in which few I/O routines like LCD interfacing, keyboard interfacing, etc. have been developed as per the BIOS style of coding. It is organized in such a way that the information such as port addresses, characters to be displayed are passed as parameters to the interrupt service routine.

Program 6.3 A tiny BIOS or diagnostic interface with MCS51 (Refer Fig. 6.3)

Program Source Code

****

```
/* Program to illustrate the Tiny BIOS interface with the Microcontroller 89S52.*/
```

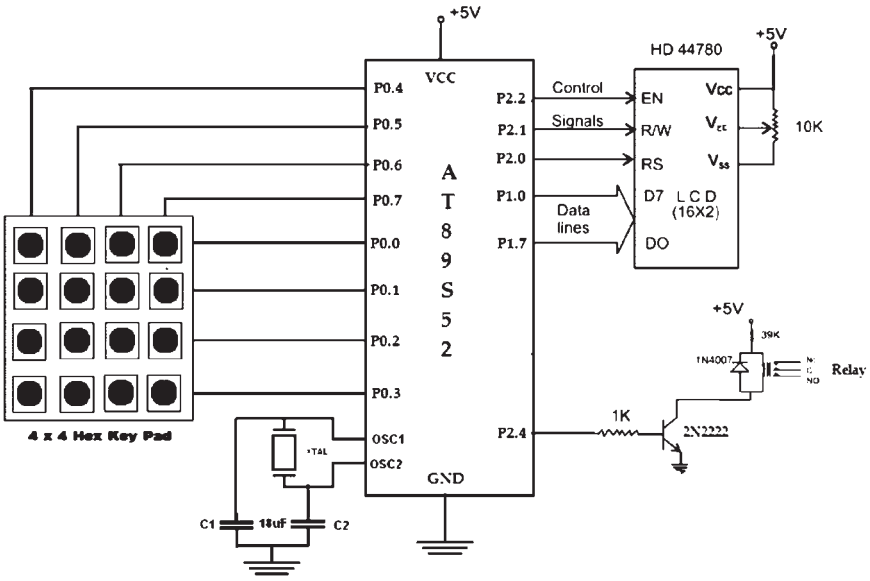


Fig. 6.3 A tiny BIOS or diagnostic interface with MCS51

```
#include <REG52.H>          /* special function register declarations */
                            /* for the intended 8051 derivative */
#include <stdio.h>          /* prototype declarations for I/O functions */
sbit RS = P2^0;
sbit RW = P2^1;
sbit EL = P2^2;
sbit Relay = P2^4;
sbit R1 = P0^0;
sbit R2 = P0^1;
sbit R3 = P0^2;
sbit R4 = P0^3;
sbit C1 = P0^4;
sbit C2 = P0^5;
sbit C3 = P0^6;
sbit C4 = P0^7;
char array[]="EMBEDDED C PROGRAMMING IS INTERRESTING ";
void Delay(int);
void INIT(void);
void ENABLE(void);
void LINE(int);
int keyb(void);
char getascii(int);
```

```
char getascii(int k)
{
   int ascii;
   if(k<10)
   ascii=k+0×30;
   if (k==10) ascii=0×30;
return (ascii);
   }
int keyb(void){
int key=0;
C1=1;
C2=1;
C3=1;
C4=1;
R1=0;
R2=1;
R3=1;
R4=1;
if (C1==0) key = 1;
if (C2==0) key = 2;
if (C3==0) key = 3;
if (C4==0) key = 4;
R1=1;
R2=0;
R3=1;
R4=1;
if (C1==0) key = 5;
if (C2==0) key = 6;
if (C3==0) key = 7;
if (C4==0) key = 8;
R1=1;
R2=1;
R3=0;
R4=1;
if (C1==0) key = 9;
if (C2==0) key = 10;
if (C3==0) key = 11;
if (C4==0) key = 12;
R1=1;
R2=1;
R3=1;
R4=0;
if (C1==0) key = 13;
if (C2==0) key = 14;
if (C3==0) key = 15;
```

```
if (C4==0) key = 16;
return(key);
}
void LINE(int i)
   {
   if (i==1)
      {
      RS=0;
      RW=0;
      P1=0×80;
      ENABLE();
      RS=1;
      }
   else
      {
      RS=0;
      RW=0;
      P1=0×C0;
      ENABLE();
      RS=1;
      }
   }
void Delay(int k)
   {
   int i,j;
   for (j=0;j<k+1;j++)
      {
   for (i=0;i<10000;i++);
      }
   }
void ENABLE(void)
   {
   EL=1;
   Delay(1);
   EL=0;
   Delay(1);
   }
void INIT(void)                    /*initialization of LCD display*/
   {
   RS=0;
   RW=0;
   EL=0;
   P1 = 0×38;
   ENABLE();
```

```
   ENABLE();
   ENABLE();
   ENABLE();
   P1 = 0×06;
   ENABLE();
   P1 = 0×0E;
   ENABLE();
   P1 = 0×01;
   ENABLE();
   }
void main (void)
   {
   char b;
   int j;
while (1)
   {
j=0;
j = keyb();
if (j!=0){
if (j==1)                          // display on LCD is enabled
{
INIT();
      LINE (1);
      for (b=0;b<30;b++)
         {
            if (b==8)
               LINE(2);
               P1=array[b];
               ENABLE();
         }
RS=0;
P1=0×01;
ENABLE();
RS=1;
LINE(1);
for (b=16;b<33;b++)
      {
   if (b==24)
      LINE(2);
      P1=array[b];
      ENABLE();
      }
}
else if (j==2)
```

```
{
Relay=1;                          //‘switch on the relay
Delay(10000);
Relay=0;                          // switch off the relay after 10 s
}
//else if(j==3)
//{
//activate some other peripherals like LEDs, HyperTerminal communication, serial
7 SEG, etc.
//}
} }
}
**********************************************************************
```

Note: Many monitor routine may be added on similar lines.

6.4 Simple Digital IC Tester for 74XX Series

Digital IC testing is by itself a completely different philosophy. In this case study we have employed AT89S52 for the testing purpose. The basic approach is based on table lookup of the truth tables which are invoked based on the IC the type of the IC inserted in the socket. User has to specify the IC number by entering through the keyboard such as key1 corresponds to 7400, key2 to 7408, etc.

By interfacing additional external memory one can modify the application for testing more digital ICs. Fig. 6.4 shows the block schematic while the circuit diagram is as per Fig. 6.5.

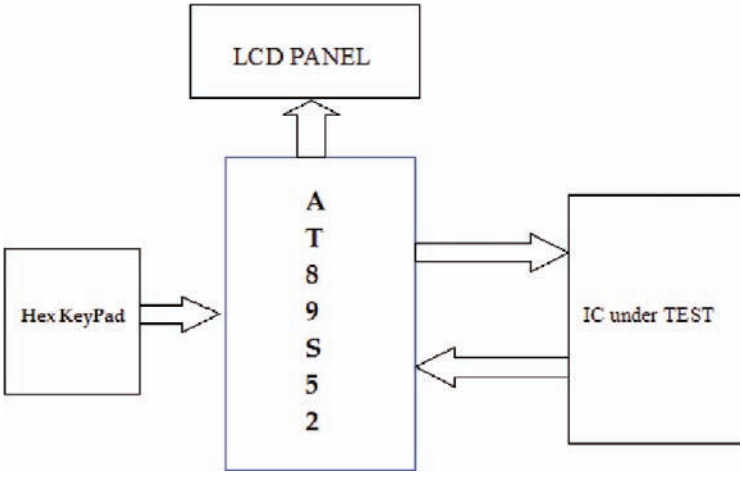


Fig. 6.4 Block diagram of simple digital IC tester for the 74XX series

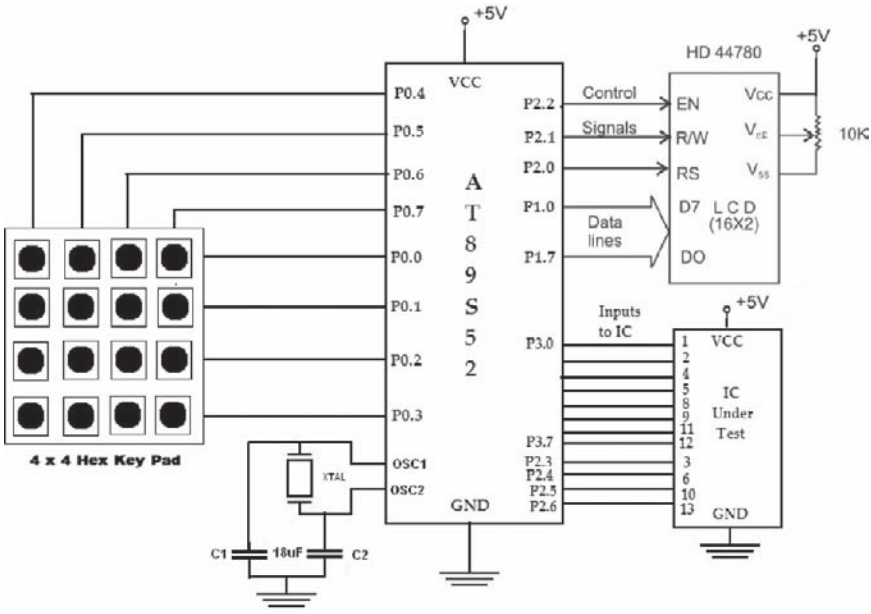


Fig. 6.5 Schematic of simple digital IC tester for 74XX series

Program 6.4 Simple digital IC tester for the 74XX series

Program Source Code

```
**************************************************************************
/* Program illustrates the Simple Digital IC tester for 74XX series using 89S52
microcontroller*/
-------------------------------------------------------------------------
#include <REG52.H>                  /* special function register declarations */
#include <stdio.h>                  /* for the intended 8051 derivative */
sbit RS = P2^0;
sbit RW = P2^1;
sbit EL = P2^2;
sbit R1 = P0^0;
sbit R2 = P0^1;
sbit R3 = P0^2;
sbit R4 = P0^3;
sbit C1 = P0^4;
sbit C2 = P0^5;
sbit C3 = P0^6;
sbit C4 = P0^7;
sbit a = P2^3;
sbit b= P2^4;
sbit c = P2^5;
```

```
sbit d = P2^6;
void Delay(int);                    /* Stop Exection with Serial Intr. */
void INIT(void);
void ENABLE(void);
void LINE(int);
int keyb(void);
void LINE(int i){
   if (i==1) {
   RS=0;
   RW=0;
   P1=0×80;
   ENABLE();
   RS=1;
   }
   else
   {
   RS=0;
   RW=0;
   P1=0×C0;
   ENABLE();
   RS=1;
   }
}
void delay(int k)
{
int i,j;
for (j=0;j<k+1;j++){
for (i=0;i<100;i++);
} }
void ENABLE(void)
{
EL=1;
delay(1);
EL=0;
delay(1);
}
void INIT(void)
{
   RS=0;
   RW=0;
   EL=0;
   P1 = 0×38;
   ENABLE();
   ENABLE();
   ENABLE();
```

```
    ENABLE();
    P1 = 0×06;
    ENABLE();
    P1 = 0×0E;
    ENABLE();
    P1 = 0×01;
    ENABLE();
}
int keyb(void){
int key=0;
C1=1;
C2=1;
C3=1;
C4=1;
R1=0;
R2=1;
R3=1;
R4=1;
if (C1==0) key = 1;
if (C2==0) key = 2;
if (C3==0) key = 3;
if (C4==0) key = 4;
R1=1;
R2=0;
R3=1;
R4=1;
if (C1==0) key = 5;
if (C2==0) key = 6;
if (C3==0) key = 7;
if (C4==0) key = 8;
R1=1;
R2=1;
R3=0;
R4=1;
if (C1==0) key = 9;
if (C2==0) key = 10;
if (C3==0) key = 11;
if (C4==0) key = 12;
R1=1;
R2=1;
R3=1;
R4=0;
if (C1==0) key = 13;
if (C2==0) key = 14;
if (C3==0) key = 15;
```

```
if (C4==0) key = 16;
return(key);
}
void main(void) {
int j,i,Read;
char *p;
char status[]="Digital Ic tester";
char status1[]="gate is faulty";
char status2[]="Gate is ok";
INIT();
LINE(1);
p=&status;
for (i=0;i<17;i++)
{
if(i==0) LINE(1);
if(i==8) LINE(2);
P1=*p++;
ENABLE();
Delay(200);
}
while(1){
j=0;
j = keyb();
if (j==1){
                                    //it is 7400 NAND gate
P3=0;                               // input to all gate of 7400 (only one pattern)
Read=P2;                            //output of gate is read by 89C52
if (a=b=c=d=1)
{
P3=0×55;
Read=P2;
if (a=b=c=d=0)
{
P3=0×aa;
Read=P2;
if (a=b=d=c=0)
{
P3=0×ff;
Read=P2;
if (a=b=c=d=0)
{
p=&status2;
for(i=0;i<17;i++)
{
if(i==0)LINE(1);
```

```
if(i==8)LINE(2);
P1=*p++;
ENABLE();
Delay(200);
} } } } } }
else
{
p=&status1;
for(i=0;i<17;i++)
{
if(i==0)LINE(1);
if(i==8)LINE(2);
P1=*p++;
ENABLE();
Delay(200);
} }
}
}
//if (j=2)
//{
//similar routine for other gate for e.g. 7404
//}
//if (j=3)
//{
//similar routine for other gate for e.g. 7408
//}
*****************************************************************************
```

6.5 Microcontroller Based Salinity Measurement System

Conductivity and salinity are the two most common water related parameters measured by Scientists and Engineers for assessment and research purpose [74]. Salinity is the presence of soluble salts in soils or waters. It is a general term used to describe the presence of elevated levels of different salts such as sodium chloride, magnesium and calcium sulfates and bicarbonates, in soil and water. It usually results from water tables rising to, or close to, the ground surface [75]. There are many good reasons why salinity measurement is treated as the most important in several processes. For instance one of the main reason for the growth of unwanted algae is a low salinity of the water in a tank. It is also instrumental in and deciding factor in saltwater fish's osmoregulatory behavior. Soil salinity a yet another important parameter is a measure of the total amount of soluble salt in soil. It is reported that [76], as salinity levels increase, plants extract water less easily from soil, aggravating water stress conditions. High soil salinity can also cause nutrient imbalances, result in the

accumulation of elements toxic to plants, and reduce water infiltration if the level of one salt element sodium is high.

The sensing module used in this application is based on the Debye-Hückel [77] that starts from the assumption that the electrical potential fluctuations due to the ions in an electrolyte are small. At each point, a potential shift f will give rise to a charge density opposing the fluctuation:

$$\mathrm{r} = (e^2 \mathrm{f} / kT)\mathrm{S}c_i z_i^2$$

where the *c*'s are the concentrations in ions m^{-3}.

A conductivity probe based on the above theory has been designed [78] and usefully used for many applications. The main intention behind including this application is to demonstrate the complementary advantages of the analog signal conditioning when it is usefully combined with the microcontrollers. The conductivity modulating probe embedded in an oscillator circuit exhibit extra sensitivity at increased resolution. The non-linearity is corrected using the linearization circuit in the analog domain itself. Such an approach definitely offloads the linearization burden of the microcontroller and enables quick sampling especially required in dynamic salinity changing conditions. The block schematic and the analog signal diagrams are shown in Figs. 6.6–6.8.

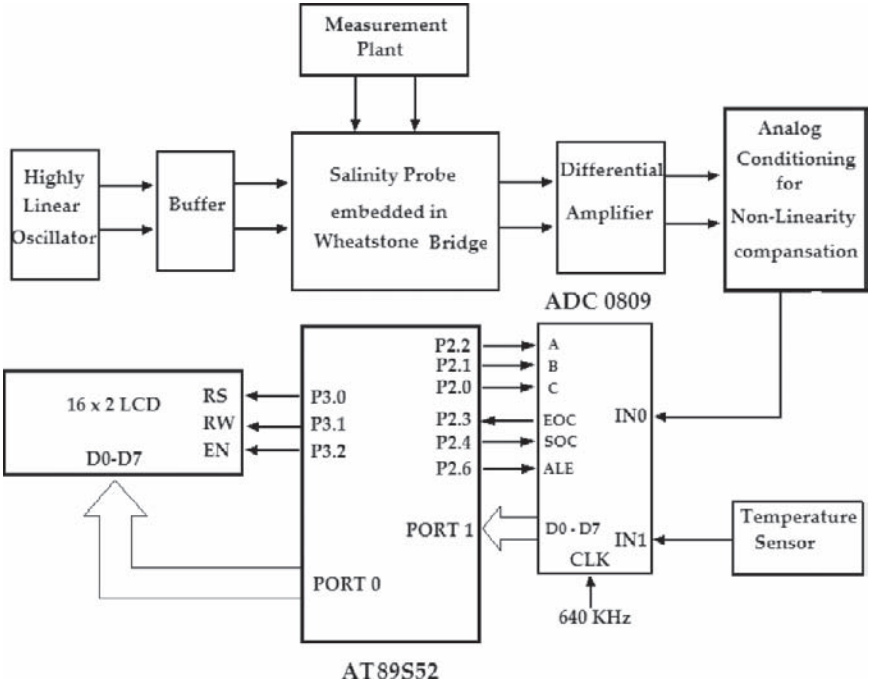


Fig. 6.6 Microcontroller based smart salinity measurement systems

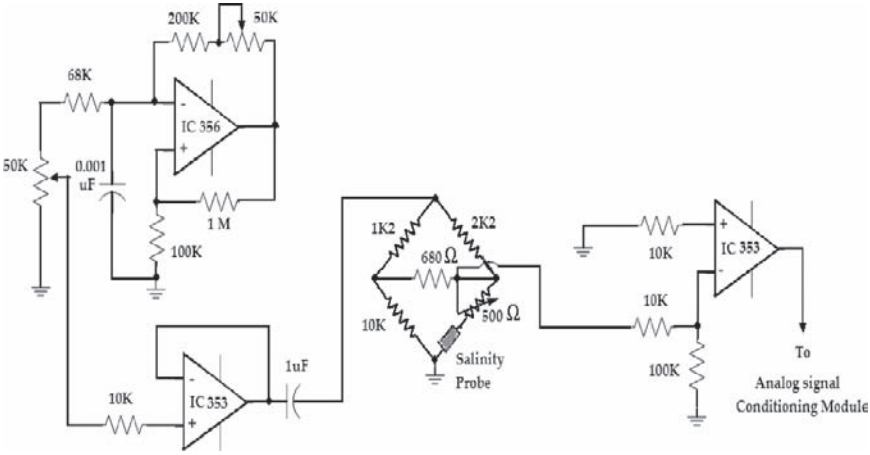


Fig. 6.7 Oscillator driven salinity probe

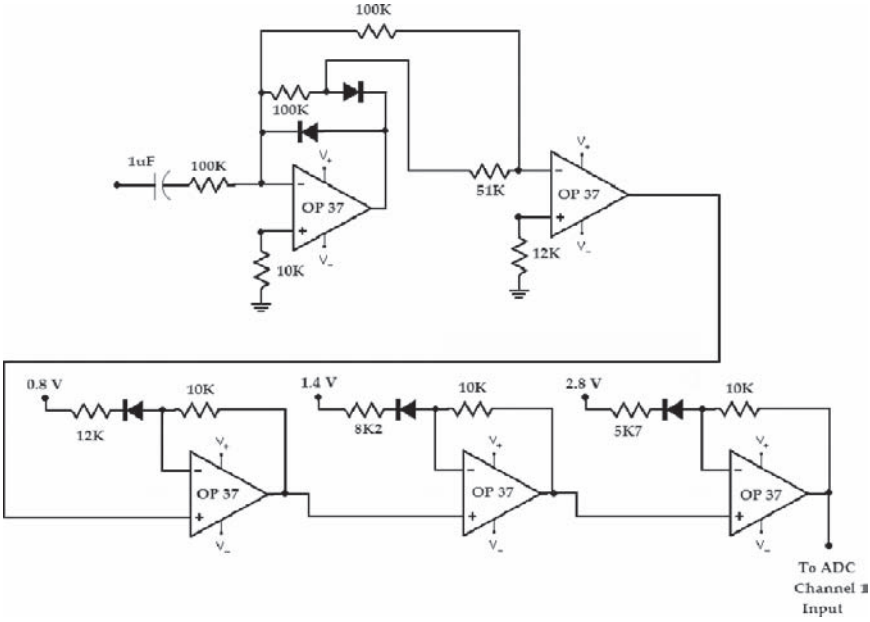


Fig. 6.8 Analog signal conditioning

Program 6.5 Microcontroller based salinity measurement

Program Source Code

```
**************************************************************************
/* Program illustrates the Microcontroller based Salinity Measurement.
_____________________________________________________________________
C program:
#include <REG52.H> /* special function register declarations */
#include <stdio.h>
/* for the intended 8051 derivative */
sbit RS = P3^0;
sbit RW = P3^1;
sbit EL = P3^2;
sbit SOC= P2^4;
sbit EOC= P2^3;
sbit ALE= P2^6;

sbit a =P2^2;
sbit b = P2^1;
sbit c =P2^0;
void delay(int);
void INIT(void);
void ENABLE(void);
void LINE(int);
void LINE(int i)
{ if(i==1)
{ RS=0;
RW=0;
P0=0×80;
ENABLE();
RS=1;
}else
{RS=0;
RW=0;
P0=0×C0;
ENABLE();
RS=1;
}
}
void delay(int k)
{int i,j;
for (j=0;j<k+1;j++)
{for (i=0;i<10000;i++);
} }
void ENABLE (void)
–
```

```
   EL=1;
delay(1);
EL=0;
delay(1);
}
void INIT(void)        // Initialization of the LCD by giving the proper commands.
{ RS=0;
RW=0;
EL=0;
P0 = 0×38;             // 2 lines and 5*7 matrix LCD.
ENABLE();
ENABLE();
ENABLE();
ENABLE();
P0 = 0×06;             //Shift cursor to left
ENABLE();
P0 = 0×0E;             // Display ON, Cursor Blinking
ENABLE();
P0 = 0×01;             // Clear display Screen
ENABLE();
}
void main (void)
{int unit, tens, hundred, unit1, tens1, hundred1;
unsigned char d;
a=0;
b=0;
c=0;
while(1)
{
P1=0×ff;
SOC=1;
delay(8);
SOC=0;
delay(8);
SOC=1;
delay(8);
d=P1;
}
unit = (d%10);
tens = (d/10)%10;
hundred = (d/100);
unit = (unit + 0×30);
unit1=unit+unit;
tens = (tens + 0×30);
tens1=tens+tens;
```

```
hundred = (hundred + 0×30);
hundred1=hundred+hundred;
INIT();
LINE(1);
P0=hundred1;
ENABLE();
P0=tens1;
ENABLE();
P0='.';
ENABLE();
P0=unit1;
ENABLE();
P0='c';
ENABLE();
} }
*****************************************************************************
```

6.6 Fault Tolerant Sensor Interface

Program 6.6 Fault tolerant sensor interface

A System having sensors that can adjust in any environment, the system consists of substitute or standby sensor. At the start 89C52 controller read all sensors for a stable value A. in read the sensor's one at a time to check for large variations, if large variation found in the readings and switch it to the standby sensors

Program Source Code (Refer Fig. 6.9)

```
*****************************************************************************
/* Program illustrates the fault tolerant sensor interface */
_____________________________________________________________________________

#include<reg52.h>
sbit ale=P2^3;
sbit row_soc=P2^2;
sbit eoc=P2^1;
sbit RS = P2^4;
sbit RW = P2^5;
sbit EL = P2^6;
int temp=27;
void line(int);
void ENABLE(void);
void INIT(void);
void delay(int);
void main(void)
   {
   int col,address,data_read[7];       /*try with char*/
```

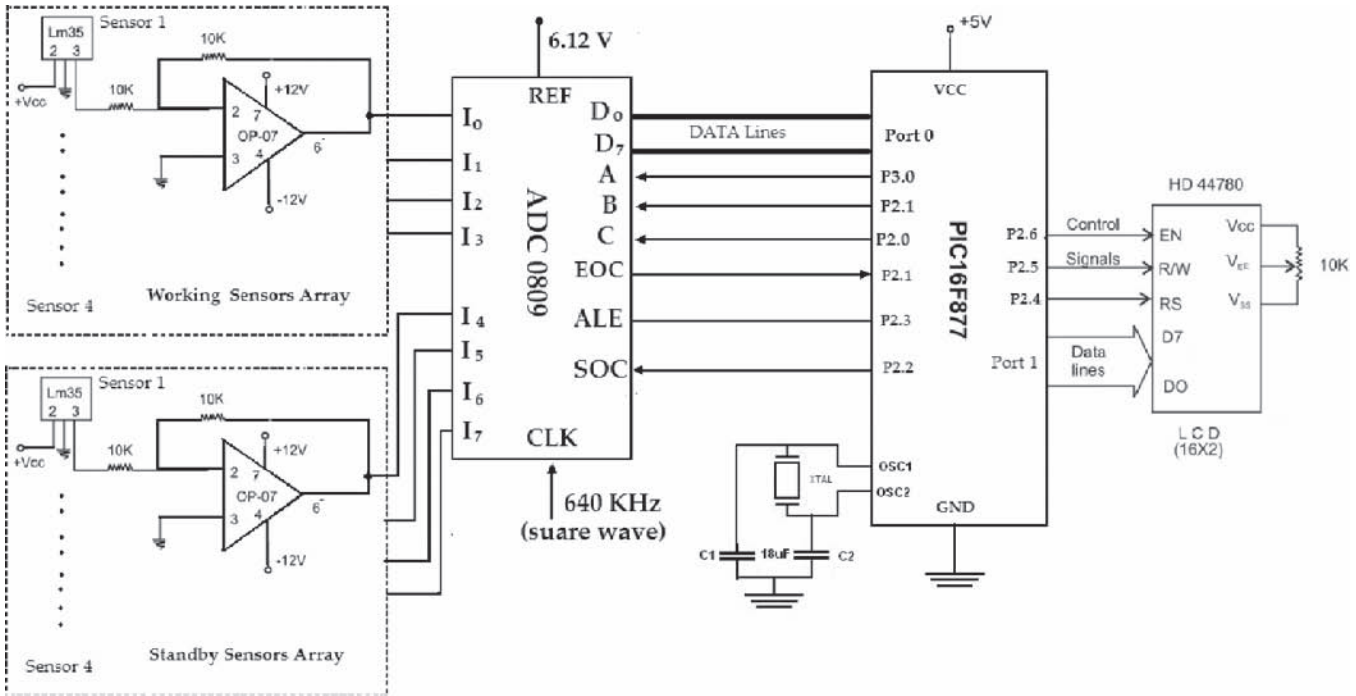


Fig. 6.9 Fault tolerant sensor interface

```
int temp=27;
char *p;
int k,h,j;
char nofault[]="There is no fault";
char fault[]="System is faulty";
   row_soc=1;
   col=0;
   while(1)
   {
for(h=0;h<3;h++)
{
   row_soc=~row_soc;
      for(col=0;col<=3;col++)
      {
address=(col & 0×07); /*allow only lower 3bits*/
         P2=address;
delay(10);          /*allow address to stabilise*/
         ale=1;     /*ale P2~3 intially it's zero above*/
         delay(10);
         ale=0;
         eoc=0;
         row_soc=0;
         delay(200);
         row_soc=1;          /*0 to 1 transition req to clear sar */
         delay(200);
         row_soc=0;          /*1 to 0 transition req to start convertion */
   while(eoc==1);/* conversion complete,does eoc need to b config input before
//reading*/
data_read[address]=P0;
INIT();
   line(1);
   P1=((data_read[address] /1000)+0×30);
   ENABLE();
   P1=(((data_read[address] / 100)%10)+0×30);
   ENABLE();
   P1=(((data_read[address] /10)%10)+0×30);
   ENABLE();
   P1=((data_read[address] %10)+0×30);
   ENABLE();
   P1='o';
ENABLE();
P1='C';
ENABLE();
}
for (j=0;j<=7;j++)
```

```
{
if( (temp <=(data_read[j]-5))||(temp >=(data_read[j]+5)) )
{
   INIT();
p=&nofault;
for (k=0;k<17;k++)
{
if (k==0) line(1);
if (k==8) line(2);
P1=*p++;
ENABLE();
} }
   else    // switch to another matrix
{
INIT();
p=&fault;
for (k=0;k<17;k++)
{
if (k==0) line(1);
if (k==8) line(2);
P1=*p++;
ENABLE();
//(Routine for another sensor matrix which may be connected to port 0)
//repeat a program to generate to send address and read data and display the data on
LCD)
}
   }
   }
} } }
void line(int i)
      {
      if (i==1)
         {
         RS=0;
         RW=0;
         P1=0×80;
         ENABLE();
         RS=1;
         }
      else
         {
         RS=0;
         RW=0;
         P1=0×C0;
         ENABLE();
```

```
        RS=1;
        }
    }
void enable(void)
    {
    EL=1;
    delay(2);
    EL=0;
    delay(2);
    }
void INIT(void)
    {
    RS=0;
    RW=0;
    EL=0;
    P1 = 0×38;
    enable();
    enable();
    enable();
    enable();
    P1 = 0×06;
    enable();
    P1 = 0×0E;
    enable();
    P1 = 0×01;
    enable();
    }
void delay(int k)
{
int i,j;
  for(i=0;i<k;i++)
  {
  for(j=0;j<1000;j++);
    }
  }
*****************************************************************************
```

6.7 Sensor Matrix Interface

Program to read the data from temperature sensor which is connected to ADC 0808(8 bit) and display the temperature on LCD panel.

Program 6.7 Sensor matrix interface (Refer Fig. 6.10)

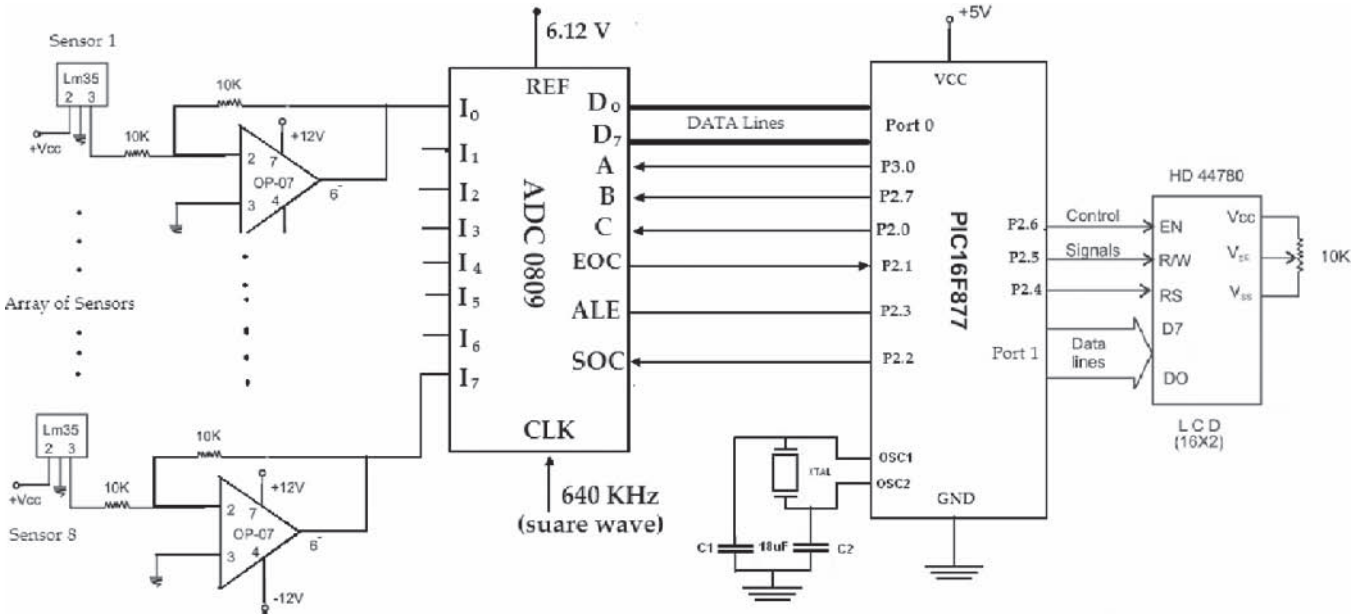


Fig. 6.10 Schematic of sensor matrix interface to AT89S52

Program Source Code

```
**************************************************************************
/ * Program to read the data from temperature sensor which is connected to ADC
0808(8 bit) and display the temperature on LCD panel*/
/* 1. Put address on lines
   2. Delay (stabilise address)
   3. Put ale high
   4. Rising edge on soc clear's reg n falling edge starts conversion
   5. Wait for eoc high
   6. Read data lines and display.inc address and continue*/
/*lcd data on port1, control on port 2high
adc data on port0,address,ale on port 2low*/
//ale n soc can b on single line (delay)
/*latch address n use same lines as soc eoc*/
_____________________________________________________________________
#include<reg52.h>
sbit ale=P2^3;
sbit row_soc=P2^2;
sbit eoc=P2^1;
sbit RS = P2^4;
sbit RW = P2^5;
sbit EL = P2^6;
void line();
void enable(void);
void INIT(void);
void delay(int);
void main(void)
{
int col,address,data_read[7];          /*try with char*/
row_soc=1;
col=0;
while(1)
{
   row_soc=~row_soc;
      for(col=0;col<=3;col++)
      {
         address=(col & 0×07);         /*allow only lower 3bits*/
P2=address;
         delay(10);                    /*allow address to stabilize*/
ale=1;                                 /*ale P2~3 initially it's zero above*/
delay(10);
      ale=0;
eoc=0;
      row_soc=0;
      delay(200);
```

```
      row_soc=1;                    /*0 to 1 transition require to clear sar */
         delay(200);
      row_soc=0;                    /*1 to 0 transition require to start conversion */
while(eoc==1);                      /*conversion complete, does eoc need to be
                                      configure input before reading*/
data_read[address]=P0;
INIT();
line();
P1=((data_read[address] /1000)+0×30);
   enable();
   P1=(((data_read[address] / 100)%10)+0×30);
   enable();
   P1=(((data_read[address] /10)%10)+0×30);
   enable();
   P1=((data_read[address] %10)+0×30);
   enable();
P1='o';
enable();
P1='C';
enable();
}
   }
   // if(address>1)address=0;
   }
   void line(void)
         {
         //if (i==1)
            {
            RS=0;
            RW=0;
            P1=0×80;
            enable();
            RS=1;
            }
/* else
            {
            RS=0;
               RW=0;
            P1=0×C0;
            ENABLE();
            RS=1;
            } */
            }
void enable(void)
            {
```

```
            EL=1;
            delay(2);
            EL=0;
            delay(2);
            }
void INIT(void)
            {
            RS=0;
            RW=0;
            EL=0;
            P1 = 0×38;
            enable();
            enable();
            enable();
            enable();
            P1 = 0×06;
            enable();
            P1 = 0×0E;
            enable();
            P1 = 0×01;
            enable();
            }
void delay(int k)
{
int i,j;
for(i=0;i<k;i++)
{
for(j=0;j<1000;j++);
            }
}
*****************************************************************************
```

6.8 Design Microcontroller Based Servo Controller

Servo controllers are popularly used in motion control industries. The underlying concepts of servo control have not changed significantly in the last 50 years, but definitely there has been evolution in this field. A closed loop servo control system offers improved accuracy, good transient response besides reduction in the steady state errors and exhibits limited sensitivity to load parameters.

The application developed here is keeping in view the interested mechatronics professionals who are now relying on the microcontroller for increased portability, ease of programming, accuracy and flexibility. The main thrust in these applications is fixing the exact positions for the motors used for driving the actuators.

Apart from the load calculations and other mechanical aspects such as vibration analysis, overshoot, undershoot, etc. programming plays an important role to tailor the setup as per the requirement. In the following application a motor control is illustrated by using microcontroller. The setpoint is simulated using the potentiometer. The block schematic is shown in Fig. 6.11.

Program 6.8 Microcontroller based servo controller

Program Source Code

```
*************************************************************************
/* Program to illustrate the Servo motor controller using 89S52 microcontroller. */
_________________________________________________________________________

#include <REG52.H>                /* special function register declarations */
/* for the intended 8051 derivative */
sbit RS = P2^0;
sbit RW = P2^1;
sbit EL = P2^2;
sbit soc = P1^0;
sbit eoc = P1^1;
sbit a = P1^4;
sbit b = P1^5;
sbit c = P1^6;
sbit d = P1^7;
sbit R1 = P3^0;
sbit R2 = P3^1;
sbit R3 = P3^2;
sbit R4= P3^3;
sbit C1 = P3^4;
```

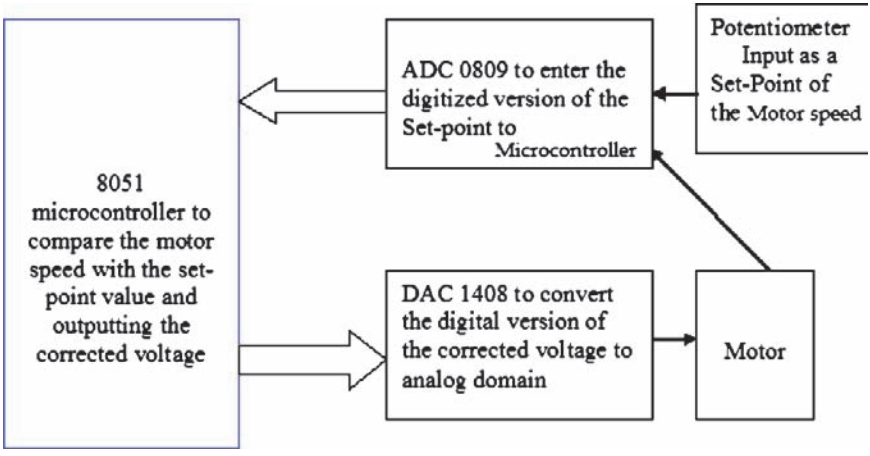


Fig. 6.11 Block diagram of the microcontroller based servo controller

```
sbit C2 = P3^5;
sbit C3 = P3^6;
sbit C4 = P3^7;
void Delay(int);
void INIT(void);
void ENABLE(void);
void LINE(int);
int keyb(void);
int setpt=30;
void dis(int);
void main(void)
{
char test[]=“servo motor”;
char *p;
int j,currentspeed,currentvolt,corrected ;
INIT();
LINE(1);
p=&test;
for(j=0;j<17;j++)
{
if(j==0)
LINE(1);
if(j==8)
LINE(2);
P1=*p++;
ENABLE();
Delay(200);
while(1)
{
j=0;
j=keyb();
if(j>8)j=0;
if(j==1){
setpt++;
dis(j);
c=6;}
if(j==1){
setpt–;
dis(j);
c=7;
}
currentvolt=P2;
currentspeed =currentvolt*2;
corrected=setpt-currentvolt;
P0=corrected;                    // output the corrected value to motor
```

```
}}}
void Delay(int k)
{
int i,j;
for (j=0;j<k+1;j++){
for (i=0;i<100;i++);
}}
void ENABLE(void)
{
EL=1;
Delay(1);
EL=0;
Delay(1);
}
void LINE(int i){
   if (i==1) {
   RS=0;
   RW=0;
   P1=0×80;
   ENABLE();
   RS=1;
   }
   else
   {
   RS=0;
   RW=0;
   P1=0×C0;
   ENABLE();
   RS=1;
   }
}
void INIT(void)
{
   RS=0;
   RW=0;
   EL=0;
   P1 = 0×38;
   ENABLE();
   ENABLE();
   ENABLE();
   ENABLE();
   P1 = 0×06;
   ENABLE();
   P1 = 0×0E;
   ENABLE();
```

```
   P1 = 0×01;
   ENABLE();
}
int keyb(void){
int key=0;
C1=1;
C2=1;C3=1;C4=1;R1=0;R2=1;R3=1;R4=1;
if (C1==0) key = 1;
if (C2==0) key = 2;
if (C3==0) key = 3;
if (C4==0) key = 4;
return(key);
}
void dis(int j)
{
char code volt[]=“ voltage = ”;
char code inS[]=“ increment setpt ”;
char code dec[]=“ decrement setpt”;
char *p;
int k ,currentspeed,currentvolt;
if(j !=9){
INIT() ;
if(j==1) p=&inS ;
if(j==2) p=&dec ;
for(k=0;k<17;k++)
{
if(k==0) LINE(1);
if(k==8) LINE(2);
P1=*p++;
ENABLE();
}
Delay(500);
INIT();
LINE(1);
P1=((setpt/100)+0×30);
ENABLE();
P1=(((setpt/10)%10)+0×30);
ENABLE();
P1=((setpt%10)+0×30);
ENABLE();
Delay(2000);
}
if(j==7)
{
p=& volt;
```

```
INIT();
for(k=0;k<8;k++)
{
if(k==0)LINE(1);
P1=*p++;
ENABLE();
}
Delay(400);
soc=0;
Delay(200);
soc=1;
Delay(200);
soc=0;
Delay(200);
eoc=0;
while(eoc==1);
currentvolt=P2;
LINE(2);
P1=((currentspeed/1000)+0×30);
ENABLE();
P1=(((currentspeed/100)%10)+0×30);
ENABLE();
P1=((((currentspeed/10))%10)+0×30);
ENABLE();
P1=((currentspeed%10)+0×30);
ENABLE();
}}
*****************************************************************************
```

References

1. Keypad from JK Microsystems http://www.jkmicro.com/products/accessories/keypad.html
2. Keypad data manual: www.grayhill.com or http://www.jkmicro.com
3. EECO Pushwheel and Thumbwheel Switches http://www.eecoswitch.com/Catalog%20Files/cat_thum.htm
4. Thumbwheel switches by purdy Electronics Corporation http://www.purdyelectronics.com/products/switches/thumb.cfm
5. Thumbwheel switchon Global Spec http://electronic-components.globalspec. com/Industrial-Directory/thumbwheel_switch
6. (http://www.netrino.com/Embedded-Systems/Glossary Netrino: The Embedded Systems Expert)
7. http://edageek.com/2007/12/31/vdc-embedded-systems/ VDC Publishes 2007 Embedded Systems Market Statistics Report
8. http://64.233.183.104/search?q=cache:y0zFFo7vIJ8J:arstechnica.com/news.ars/post/20080104-evaluating-prospects-for-linux-growth-in-2008.html+Embedded+systems+popularity+Statistics+2008&hl=en&ct=clnk&cd=5&gl=in Evaluating prospects for Linux growth in 2008 by Ryan Paul | Published: January 04, 2008
9. www.scienceprog.com/microcontroller-c-programming Microcontroller C programming
10. http://eetimes.eu/germany/202101848 Automotive applications to drive microcontroller market, Christoph Hammerschmidt, EE Times Europe
11. http://www.electronics.ca/presscenter/articles/580/1/New-Study-Predicts-10-percent-Growth-for-Microcontrollers/Page1.html New Study Predicts 10 percent Growth for Microcontrollers, By Electronics.ca Research Network Published 06/28/2007
12. Michael Barr. Embedded Systems Glossary. Netrino Technical Library. Retrieved on 2007-04-21
13. http://www.pcmag.com/encyclopedia_term/0,2542,t=embedded+system&i=42554,00.asp Definition of: embedded system
14. http://searchenterpriselinux.techtarget.com/sDefinition/0,,sid39_gci837507,00.html Search EnterpriseLinux.com Definitions
15. http://www.linfo.org/embedded_system.html Embedded System Definition
16. http://whatis.techtarget.com/definition/0,,sid9_gci212053,00.html# embedded systems programming
17. http://www.bitpipe.com/tlist/Embedded-Systems-Hardware.html Embedded Systems Hardware
18. http://www.ssiembedded.com/embedded_systems_definition.html Definition of Embedded Systems
19. www.ibm.com/developerworks/rational/library/459.html Testing embedded systems: Do you have the GuTs for it? Vincent Encontre, November 2004
20. www.itfacts.biz

21. http://www.ad-mkt-review.com/public_html/air/ai200205.html Embedded Systems Take Off, by Glen Emerson Morris
22. http://www.embeddedtechjournal.com/articles_2005/20051122_sc05.htm Supercomputing To Go HPEC Raises its Head at SC|05 by Kevin Morris, Embedded Technology Journal, November 22, 2005
23. http://computer-engineering.science-tips.org/computer-organization/embedded-systems/introduction-to-embedded-systems.html Introduction to Embedded Systems, Saturday, 19 January 2008
24. www.async.elen.utah.edu/ myers/ece5780/lectures/lec2-2×3.pdf Lecture notes on Embedded System Design
25. www.webopedia.com/TERM/E/embedded_system.html Definition of Embedded System
26. www.hpl.hp.com/personal/Dejan_Milojicic/embedded.pdf Embedded Systems trends war by Dejan Milojicic, Hewlett Packard Laboratories
27. "Exploring C for Microcontrollers: A Hands on Approach," Jivan S. Parab, Vinod G. Shelake, Rajanish K. Kamat and Gourish M. Naik, London: Springer, 2007
28. http://www.embedded.com/2000/0011/0011feat1.htm Safety First: Avoiding Software Mishaps Charles Knutson and Sam Carmichael
29. http://spectrum.ieee.org/sep05/1685 IEEE Spectrum Online Why Software Fails, by Robert N. Charette, First Published September 2005
30. http://www.informationweek.com/research/showArticle.jhtml?articleID=185300011&pgno=1 Embedded Experts: Fix Code Bugs Or Cost Lives Attendees at last week's Embedded Systems Conference got an earful on the high price to be paid by poorly written or tested code, by Rick Merritt EE Times, April 10, 2006
31. http://www.informationweek.com/research/showArticle.jhtml?articleID=185300011&pgno=2&queryText= Embedded Experts: Fix Code Bugs Or Cost Lives
32. http://www.hitex.co.uk/c166/risc.html RISC Architectures For Embedded Applications Introduction
33. www.electronicsforu.com/electronicsforu/articles/hits.asp?id=1106 ANUPAMA PROCESSOR A Boon for Embedded System Design and Realisation
34. http://www.cotsjournalonline.com/home/article.php?id=100267 Defining a Standard for Embedded RISC Systems, February 2005
35. ww1.microchip.com/downloads/en/DeviceDoc/30292c.pdf: PIC datasheet
36. web.mit.edu/rec/datasheets/PIC16F84.pdf PIC Data sheet
37. www.atmel.com/dyn/resources/prod_documents/doc1919.pdf AT 89S52 datasheet
38. www.keil.com/dd/docs/datashts/atmel/at89s52_ds.pdf AT 89S52 datasheet
39. www.keil.com/ Keil IDE for 89S52
40. http://www.ikalogic.com/isp.php In System Programming (ISP) for ATMEL chips, A step by step construction guide, by Ibrahim Kamal
41. http://www.soft32.com/download_139.html Download the HyperTerminal Private Edition 6.3: A simple terminal communications program used for accessing Telnet sites, terminal-to-host communications, and file transfer
42. http://www.brothersoft.com/downloads/hyper-terminal.html Hyper Terminal Free Download
43. http://www.freedownloadscenter.com/Network_and_Internet/Terminals_and_Telnet_Clients/HyperTerminal_Private_Edition.html HyperTerminal Private Edition 3.0
44. www.shortridge.com/pdf/hyperterminalprocedure10703.pdf RS232 Download using HyperTerminal™, Application note by Shortridge Instruments, Inc.
45. www.ozitronics.com/download/hyperterminal.pdf How to set up and use Windows 'Hyperterminal'
46. www.cyq.com/htdocs/hyperterminal.htm hyper terminal setup
47. http://www.arcelect.com/rs232.htm RS 232 Tutorial
48. http://www.kmitl.ac.th/ kswichit%20/MAX232/MAX232.htm RS232C Level Converter by Wichit Sirichote, kswichit@kmitl.ac.th
49. http://www.national.com/mpf/LM/LM35.html LM35 - Precision Centigrade Temperature Sensor
50. http://www.analog.com/en/prod/0,2877,OP07,00.html OP07 Ultralow Offset Voltage Operational Amplifier

51. http://www.murata.com/ Document P15E6.pdf 03.8.20 PIEZOELECTRIC SOUND COMPONENTS, APPLICATION MANUAL
52. http://www.rentron.com/Myke1.htm Build your own “2-Wire LCD Interface” using the PIC16C84 microcontroller by Myke Predko
53. http://mic.unn.ac.uk/miclearning/modules/micros/ch5/micro05notes.html Tutorial 5 – Interfacing to the Outside World
54. www.national.com/pf/DC/ADC0809.html Data sheet of ADC 0809
55. http://www.mines.edu/Academic/courses/physics/phgn317/lab7/Lab7_02.htm PH317 Lab 7 – Digital-Analog/Analog-Digital Converters
56. http://focus.ti.com/lit/an/slaa116/slaa116.pdf Using PWM Timer_B as a DAC, Application Report SLAA116 – December 2000
57. www.agilent.com Agilent PWM Waveform Generation Using U1252A DMM Application Note
58. http://www.electronicsweekly.com/Articles/2006/06/09/38623/Choosing+a+microcontroller.htm Choosing a microcontroller by John Anderson, Cyan Technology, Friday, 9 June 2006
59. http://www.totalphase.com/support/articles/article01/ I2C Background
60. http://www.cast-inc.com/cores/i2c/index.shtml I2C Philips Serial Bus Interface Core
61. www.esacademy.com/faq/i2c/general/I2C-Bus_Specification.pdf THE I2C-BUS SPECIFICATION, VERSION 2.1, JANUARY 2000
62. http://www.nxp.com/#/pip/pip=[pip=PCF8583_5]|pp=[v=d,t=pip,i=PCF8583_5,fi=53497,ps=0][0] Clock/calendar with 240 × 8-bit RAM
63. dhost.info/ky3orr/funkcje/download.php?dzial=pliki&link=dokumentacje/pcf8583.pdf PCF8583 Datasheet
64. www.electroscheme.ru/datasheet/Microchip/00551%20-%20Serial%20vs%20Parallel%20EEPROM.pdf Serial EEPROM Solutions vs. Parallel Solutions Application note 551
65. www.parallax.com/Portals/0/Downloads/docs/prod/oem/24LC256.pdf 24AA256/24LC256/24FC256 Datasheet
66. http://www.nxp.com/#/pip/pip=[pip=PCF8591_6]|pp=[v=d,t=pip,i=PCF8591_6,fi=43674,ps=0][0] PCF8591 8-bit A/D and D/A converter datasheet
67. PCF8591, 8-bit A/D and D/A converter datasheet 2003 Jan 27 retrieved from http://www.nxp.com/#/pip/pip=[pip=PCF8591_6]|pp=[v=d,t=pip,i=PCF8591_6,fi=43674,ps=0][0]
68. http://www.maxim-ic.com/quick_view2.cfm/qv_pk/3271 MAX 1236 product information
69. MAX1236–MAX1239 2.7 V to 3.6 V and 4.5 V to 5.5 V, Low-Power, 4-/12-Channel, 2-Wire Serial, 12-Bit ADCs datasheet retrieved from http://www.maxim-ic.com/quick_view2.cfm?t=qv&qv_pk=3271#Applications%2FUses
70. http://www.maxim-ic.com/quick_view2.cfm/qv_pk/3290 MAX5822 Dual, 12-Bit, Low-Power, 2-Wire, Serial Voltage-Output DAC
71. http://www.rubberstampingfun.com/rsepage.html Rubber Stamping & Maya Indians: A Brief History
72. Rubber stamp making machine manufacturing company. http://www.polydiam.com/shop/ www.kivicoatings.com www.prestostamps.com www.instastamp.com.au www.markstamps.com
73. http://ezinearticles.com/?How-to-Make-Rubber-Stamps:-Comparing-Methods-of-Manufacture&id=198663 How to Make Rubber Stamps: Comparing Methods of Manufacture by Robert De Rooy
74. http://www.advmnc.com/greenspan/ec.htm Webpage of Electrical conductivity Sensors
75. http://www.nrw.qld.gov.au/salinity/whatis.html What is Salinity?
76. http://extension.usu.edu/files/agpubs/salini.htm SALINITY AND PLANT TOLERANCE, by Jan Kotuby-Amacher, Director, Utah State University Analytical Labs, Rich Koenig, Extension Soils Specialist and Boyd Kitchen, Uintah County Extension Agent
77. http://www.cartage.org.lb/en/themes/sciences/chemistry/Electrochemis/TheoryElectrolytes/Debye/Debye.htm The Debye-Hückel theory
78. “Smart Salinity Measurement System with Analog Pre-linearizer”, R.K. Kamat et al. International Symposium on Smart Materials and Systems, ISSMS – 2004, India

Index