2nd Edition

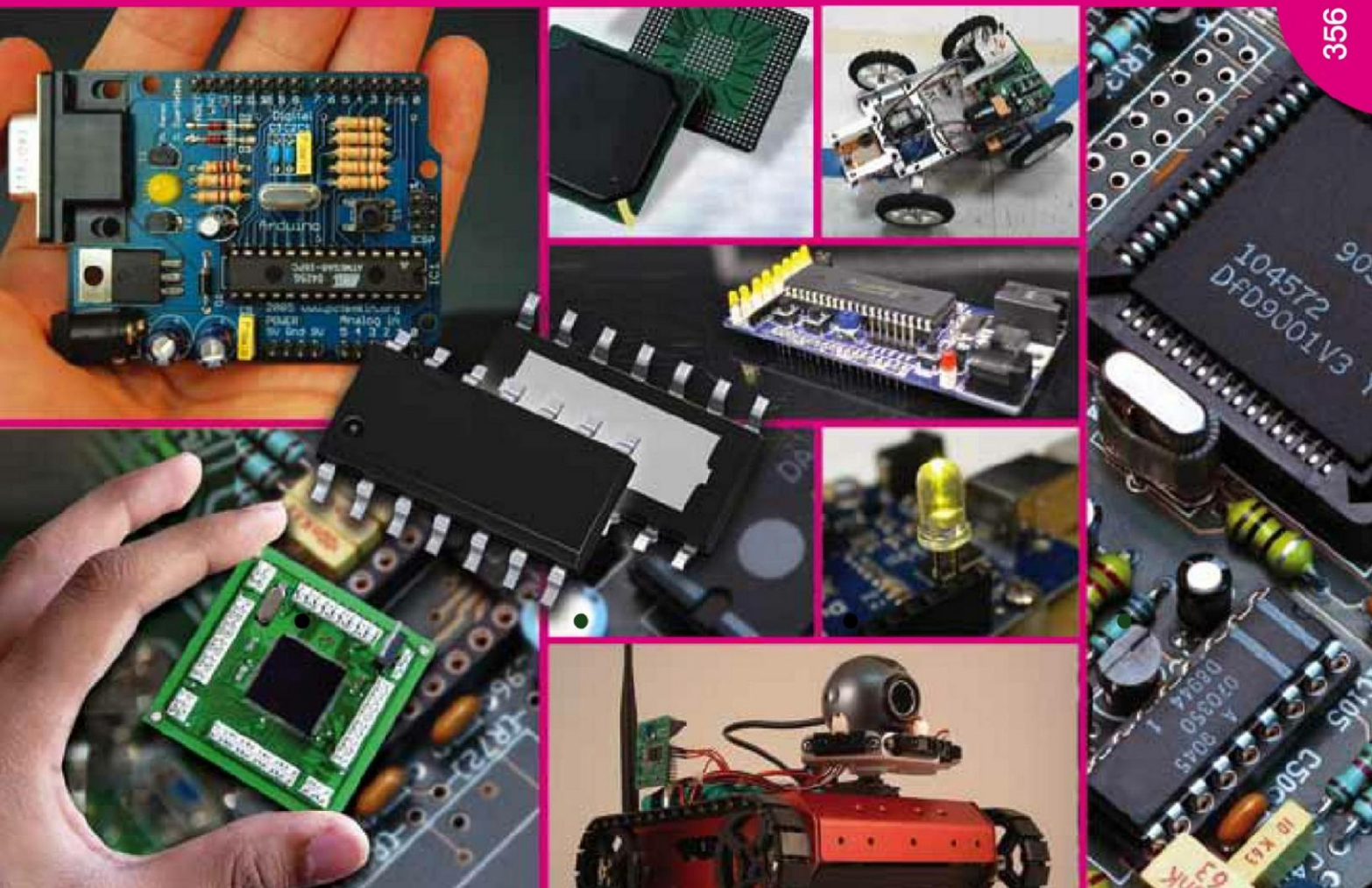
An EFY Group Publication

Price: $10

51 Exciting Microcontroller Based Projects

- Arduino-Based RF Controlled Robot
- RFID-Based Automatic Vehicle Parking System
- Speedometer-Cum-Odometer For A Motorbike
- GPS- And GSM-Based Vehicle Tracking System
- GSM-Based Borewell Water Level Monitor
- Solar Charger For Dusk-to-Dawn Use
- Water-Level Controller-Cum-Motor Protector
- Secure Digital Access System Using iButton
- MSP430G2231-Based Temperature Indicator And Controller
- Interfacing Nokia Colour LCD With AVR Microcontroller

356 Pages



efy ♥ electronics

efy group started in 1969 with the launch of electronics for you magazine. the love and support we received from indian electronics fraternity motivated us to launch many more products and services. some of them have grown into leaders in their respective categories...

electronics for you
india's #1 electronics magazine. leader in whole of south asia. read by half-a-million techies with a keen interest in electronics.

electronics bazaar
india's #2 electronics magazine and #1 b2b magazine. read by decision makers who either manufacture, buy or sell electronics goods or components.

efy yellow pages
india's only b2b yellow pages for electronics. it lists 2500 product categories, and the 7000-plus organisations in india, which supply them. excellent resource for buyers.

efy expo
launched in 2011, it is india's fastest growing electronics expo, and the only one that gets the engineering, manufacturing and trading communities under a single roof.

electronicsforu.com
india's #1 website for electronics engineers. 60% of its traffic is from india, the rest from all across the globe.

efy awards
india's leading awards for electronics brands and organisations–26 are decided by the readers of efy magazine, and 4 by a knowledgeable jury.

elcina-efy awards
a joint effort with india's leading electronics association (ELCINA), these awards are decided through self-nomination followed by a jury review.

facebook.com/designelectronics
world's #1 facebook community for electronics engineers! 100,000+ design engineers are friends of this page. 66% are from India, the rest from all across the globe.

FOR ENQUIRIES, CALL ATUL AT: +91-11-40596600 (EXT 228) OR 8800094211
EMAIL: efyenq@efyindia.com

MICROCONTROLLER-BASED PROJECTS

2nd Edition

First Published in this Edition, October 2010
Second Edition, May 2013

ISBN : 978-81-88152-25-4

Published by Ramesh Chopra for EFY Enterprises Pvt Ltd,
D-87/1, Okhla Industrial Area, Phase 1, New Delhi 110020
Typeset at EFY Enterprises Pvt Ltd

MICROCONTROLLER-BASED PROJECTS

2nd Edition

EFYGROUP
Technology Drives Us

EFY Enterprises Pvt Ltd
D-87/1 Okhla Industrial Area, Phase-1
New Delhi 110020

PREFACE TO THE SECOND EDITION

This is the second edition of the Microcontroller Based Projects book. Therefore, it has all the materials covered in the first edition, and also include 25 new projects with some new microcontrollers not covered in the first edition. This book, a collection of 51 microcontroller-based projects, which appeared in Electronics For You during 2001-2012 is brought out for the benefit of our new readers.

The book has been divided into five sections same as in the first edition viz : Security Systems, Domestic Applications, Industrial Applications, Measurement, Display Systems and Robotics. However, many new projects have been introduced in second edition.

The new features covered in this edition include:

1. Apart from the AT89C51, AT89C2051, AT89C52, AT89S8252, ATmega16, ATmega8535, PIC16F84, MC68HC705KJ1, MC908JL16C, and MC68HC705J1A microcontrollers covered in first edition, the new microcontrollers like MSP430G2231, PIC16F72, PIC16F877A, AT89S52, ATmega328, ATmega2560 and P89V51RD2 have been covered here
2. The projects based on Arduino, RFID(radio frequency identification), GPS(global positioning system) and GSM (global system for mobile communications) are some of the main attractions in this edition.
3. This edition also covers iButton, CCTV controller, solar tracker, touchscreen controller, data acquisition and logging and much more!
4. The source code download link is given at the end of each article

Although no textbook is perfect, each has unique features that makes it better than the other. This is one of such books very much suitable for practicing engineers, electronics students, teachers and hobbyists.

The main strength of this book is its audience-centric approach, which we hope will make for an enjoyable reading. It directly addresses the students, practicing engineers and hobbyists, giving the real-life examples and orienting the strategies to practical applications. The book will not only serve as practical reference book for the students but also help them to construct the projects required by the college/university.

Knowledge of Assembly, C or Basic programming language is not mandatory to use this book but readers with all or any of these languages experience will be helpful. The only prerequisite for the readers is that they should have the knowledge of basics of electronics including digital electronics in order to understand the working of the projects and the microcontrollers described here.

Some of the projects described in this book are available at our associates M/s Kits'n'Spares in the form of kits (including printed boards, components and source codes) at economical prices. More details can be found on www.kitsnspares.com website.

The joy of learning electronics comes through practice. A practical approach to learning is the easiest method to master the subject in electronics. By going through the descriptions of the projects, readers may be able to construct each projects without much difficulty. We hope that this book will serve as an ideal source for those building stand-alone projects using 8-bit microcontrollers.

Additional Resources

All the source codes of the projects in this book are available on www.efymag.com website. EFY has forum and websites for interacting with experts for problems related to electronics projects including the projects published in EFY. One of the popular sites is EFY's Electronics Design Community on Facebook. For microcontroller related topic you can visit http://electronicsforu.com/newelectronics/microcontrollers/

For various other topics related to electronics you can visit EFY forum on http://forum.electronicsforu.com/forum.php

About EFY Labs

EFY Group has modern lab setup for R&D and testing various electronics projects for publications. All the projects published in EFY were tested at EFY Labs. Apart from this online edition, all the print versions including Microcontroller-Based Projects (First edition), Simple Projects You Can Make At Home, Electronics Pojects Volume 1 through 25, Chip-Talk and Learn to Use Microprocessors books were compiled by EFY Labs.

About EFY Group

Electronics For You, South Asia's most popular electronics magazine is one of the products of EFY Group. The Group currently offers a bouquet of specialised publications which include—Linux For You, BenefIT, Facts For You and Electronics Bazaar—also enjoy a huge readership, and have managed to attract non-technical readers with their simple language and easy-on-the-eye design.

The Group also publishes directories and books, and organises several leading technology events. Its web-portals, which include electronicsforu.com, efytimes.com, bpotimes.com, linuxforu.com and eleb2b.com, have become leaders in their respective categories. The EFY Awards, Open Source India (LinuxAsia), and Edutech Expo are three annual events organised by the group.

Overview

The summary of each project covered in this book is given below.

1. Access Control System

It is an access control system that allows only authorized persons to access a restricted area. The system comprises a small electronic unit with a numeric keypad, which is fixed outside the entry door to control a solenoid-operated lock.

2. PIC16F84-Based Coded Device Switching System

This project is based on PIC16F84 microcontroller chip used for preventing unauthorised access to devices or solenoid-operated locks/electrical devices. Different passwords are used to access/operate different devices. So each user can access respective devices by entering the device number followed by the password. When anyone trying to access the device enters the incorrect password three times, the circuit sounds an alarm.

3. Secured Room Access System

This ATmega8535 microcontroller-Based digital is an access control system that allows only authorised persons to access a restricted area. When someone tries to enter the restricted area by entering invalid passwords continuously, the system locks itself and can be unlocked only by the master user. The system comprises a numeric keypad, a solenoid-operated lock and LCD display.

4. RFID-Based Security System

A radio-frequency identification (RFID) based access-control system allows only authorised persons to enter a particular area of an establishment. The authorised persons are provided with unique tags, using which they can access that area.

5. Secure Digital Access System using iButton

Access control forms a vital link in a security chain. Here we describe a secure digital access system using iButton that allows only authorised persons to access a restricted area.

6. A Remote Controlled 6-Camera CCTV Switcher

Here is a remote-controlled CCTV switcher circuit to monitor six cameras on a single or dual monitor/TV.

7. PIC Microcontroller-Based Electronic Lock

An electronic lock allows activation of an electric appliance only on entering the correct password. Here we present such an electronic locking system in which a PIC16F877A microcontroller plays the role of the processing unit.

8. Water-Level Controller-Cum-Motor Protector

It uses the AT89C51 microcontroller as the main controller. It can be used to switch on and off the motor depending upon the level of the water in the tank. The circuit also protects the pump against dry running and high voltage fluctuations of mains supply. The status of water levels and voltages are displayed on an LCD display.

9. Remotely Programmable RTC-Interfaced Microcontroller for Multiple Device Control

This project is based on AT89C52 Microcontroller and DS12887 Real-Time-Clock chips. It also includes an 82C55 Programmable Peripheral Interface chip to program the switching operation of 24 electrical appliances.

10. Remote-Controlled Digital Audio Processor

This project is based on AT89C51 microcontroller, TDA7439 audio processor and NEC format remote

control. It has four stereo input channels with a single stereo output with attenuation control, 80-step control for volume and 15-step control for bass, midrange and treble. The processors in this system control various functions of each channel and output them to the audio amplifier.

11. Solar Charger for Dusk-to-Dawn use

This AT89C2051 microcontroller-based solar charge controller has built-in digital voltmeter for battery status indication on the LCD with various battery protection controls. This controller is suitable for 10-40W solar panels.

12. Automatic Flush System

The project is used to automatically flush the fixture (in a toilet or urinal) when the user departs. It employs the AT89C2051 microcontroller and an infrared sensor to detect a user approaching the fixture and then waits until the user departs. It also flushes before the person departs if the person is present for more than the preset time.

13. MSP430G2231-Based Temperature Indicator and Controller

The ambient temperature must be within certain limits for instruments to work properly. Temperature measurement is important in industrial automation. This project is a temperature indicator cum controller using low-power MSP430G2231 microcontroller from Texas Instruments.

14. Sun Tracker With Position Display

The solar panels are able to receive peak sunlight only for a short time period of the day because the sun keeps on continuously moving from east to west during the entire day. This sun tracker circuit allows the solar panel to track the sun's position, ensuring maximum power generation.

15. Presence Sensing Lights Controller

Many a times we forget to switch off appliances like lights, fans and air-conditioner before leaving home. This leads to a considerable wastage of electricity apart from reducing the life of the appliances. This circuit allows to turn on the light only when someone is inside the room.

16. Touchscreen Control for Wheelchair

Wheelchairs are used by people for whom walking is difficult or impossible due to illness, injury or disability. Here we describe a microcontroller-based wheelchair, the speed and direction of which can be controlled from a touchscreen.

17. RF-Based Multiple Device Control Using Microcontroller

Here we describe how to control electrical and electronic gadgets from a remote location using radio frequency (RF) transmission. An RF interface is used instead of infrared (IR) to avoid the drawbacks of an IR interface.

18. GSM-Based Borewell Water Level Monitor

This GSM-based system will automatically give the farmer an alert on his mobile phone when the water level in the borewell drops below or rises to the threshold level for pumping. The user can also remotely switch on or switch off the pump motor by sending an SMS from his mobile phone.

19. Wireless Water-Level Indicator

Using this system, you can remotely monitor the water level of an overhead tank that is placed up to 30 metres away.

20. Make Your Own Digital Alarm Clock

Here we have described a microcontroller-based digital alarm clock. The time and alarm can be customised by the user, and are shown on the liquid crystal display (LCD) of the system.

21. Wireless Equipment Control Using AT89C51

Here is a microcontroller-based wireless equipment controller that can switch on or switch off up to four devices at a desired time interval set by the user in the transmitter.

22. Triggering Circuit for SCR Phase Control

Controllable triggering circuits are often needed to control the output voltage of SCR-/triac-based converters. In the present circuit, the firing angle of SCR is manually controlled through two pushbutton switches. The PIC16F84 microcontroller is programmed to detect the zero-crossing instants of the two halves of the mains input cycles as well as the signals received via pushbuttons switches.

23. Phase-Angle Control of SCR using AT 89C51

This article describes a microcontroller AT89C51-based phase-angle controller. The microcontroller is programmed to fire SCR over the full range of half cycles— from 0 to 180°—to get a good linear relationship between the phase angle and the delivered output power. The phase angle is displayed on an LCD panel. LED indicators are used for displaying the status of SCR.

24. Beverage Vending Machine Controller

This tea/coffee/soup vending machine controller uses Freescale's MC908JL16 microcontroller chip. The controller is programmable and user-friendly. You can set the quantity of the beverages through a button switch provided on the front panel of the controller as per your requirements. Thus, cups of any size can be filled at any time.

25. AT89C51-Based DC Motor Controller

It is an AT89C51 microcontroller-based project capable of controlling the direction of rotation of a 6V DC motor. Start, stop and change of direction of the motor controlled by push-button switches and indicated by LED as well as in the LCD display. Time settings are possible for forward and reverse running of the motor.

26. GPS- and GSM-Based Vehicle Tracking System

It is an ATmega16 microcontroller-based project for tracking a vehicle using global positioning system (GPS) and global system for mobile communication (GSM)

27. Battery Bank Protector With Multiple Features

This project can monitor the charge level, voltage run time and temperature of your battery bank. It is based on PIC16F877A microcontroller and a 16x2 LCD module.

28. Microcontroller-Based Intelligent Traffic Light System

It is an infra red based traffic light controller that detects the traffic movement. P89V51RD2 is the heart of the system. The firmware contains different traffic light intervals(red, yellow and green light delays in seconds) depending on the vehicle count in a particular traffic lane.

29. RFID-Based Automatic Vehicle Parking System

The AT89S52 microcontroller is the heart of this project. The RFID reader installed at the entry gate detects the RFID tags from the car owner. The firmware is designed such that RFID tag can be recharged. The entry gate barrier opens only when the tag with sufficient balance is detected by the reader.

30. Microcontroller-Based Scientific Calculator

A scientific calculator gives you quick access to certain mathematical functions. This project is designed to solve problems in science, engineering and mathematics.

31. Arduino-Based Vehicle Parking Counter

This vehicle counter counts the number of cars and the vacant space available in a parking lot and shows the values

on a three-digit dual-colour display. The number of cars is shown in red colour and the space available in green colour.

32. Eight-Channel Data Acquisition & Logging System

The AVR microcontroller-based system described here does the job of acquiring the analogue data such as temperature and sending it to a remote terminal for monitoring.

33. Programmable Industrial On-Off Timer With RF Remote

Some of the features of this project include: 'on' time and 'off' time can be programmed (from 1 to 60 seconds), repeat (or continuous) and single operation, fully remote-controlled within 100-metre range, user-friendly front-panel controls and display panel with LCD, emergency stop buttons (on control panel as well as on remote), provision of potential-free relay contacts for connecting any 230V AC at 10A or 28V DC at 10A device/application.

34. Digital Thermometer-Cum-Controller

This standalone digital thermometer controls the temperature of a device according to its requirement. It also displays the temperature on four 7-segment displays in the range of –55°C to +125°C. At the heart of the circuit is the microcontroller AT89S8252, which controls all its functions. IC DS1821 is used as temperature sensor.

35. AT89S52-Based Industrial Timer

It is based on the AT89S52 microcontroller that performs countdown operation starting from the digit 9999 (minutes/seconds) with four 7-segment displays showing the time left. The relay energises when the start switch is pressed and remains On till the countdown reaches 0000. Four push-to-on switches are used to start/stop, select either minutes or seconds, and set the initial value for countdown operation (using up and down keys).

36. Low-Cost LCD Frequency Meter

Frequency meters have always been expensive tools for the average hobbyists. Here is a compact and low-cost LCD based frequency meter built around AT89C2051 microcontroller and liquid-crystal displays (LCDs), that can measure up to 250 kHz. The LCD module used is a16 alphanumeric characters and two lines with backlight option.

37. Sophisticated But Economical School Timer

This versatile programmable school timer can be used to display real time clock as well as school bell timings on four 7-segment LED displays. It can store about 20 bell timings and activates the bell through a relay at every predetermined period. The project is based on MC68HC705J1A microcontroller.

38. RPM Counter Using Microcontroller AT 89C4051

Counting the revolutions per minute (RPM) of motors— determining the motor speed—is essential in the field of industrial automation. Here is a project based on microcontroller AT89C4051 that measures the RPM of a running motor and shows on an LCD.

39. Speedometer-Cum-Odometer for Motorbike

It is a digital speedometer-cum-odometer which can be installed on a motorbike. The circuit uses an AT89C2051 microcontroller and displays the readout on a 16x2 LCD display. The speed is displayed in km/hour and distance traveled in kilometers. It has self reset facility after completion of 99,999.9 km.

40. Traffic Light Count-Down Timer With Dual Colour Display

Here we describe a dual-colour traffic count-down timer that displays count-down time in either red or green colour matching with the signal light.

41. Rank Display System for Race and Quiz Competitions

In a game like 'fastest finger first' quiz, the winner is the one who takes the least time in successfully completing the task given. Sometimes there may be two or more players who appear to complete the task in equal time

and difficult for the judge to announce the winner. Here is a circuit based on AT89C51 microcontroller that can resolve the time-difference ambiguity and indicate correct ranking of all the participants on a 16x2 LCD module. It is designed for a maximum of eight participants playing at a time.

42. AT89C51-Driven Data Display

This project shows how you can use the RS-232 serial link to transfer data from a control unit to a handheld unit. Both the unit comprises of AT89C51 microcontroller to store the data. The data stored in the control unit, such as someone's birthday, is transferred to a handheld unit and displayed on the 16-character x 4-line type LCD screen.

43. Interfacing a Graphics LCD With The Microcontroller

Here is a project for interfacing a graphics LCD module with an AT89S8252 microcontroller. The graphics LCD used here is an SS24E12DLNW-E from UTC having a pixel size of 240×128 and is based on T6963 controller IC. It can be used for graphics or text display in any combination, like one area of 240×64 for graphics and another area of 240×64 for text. Also, the entire area of 240×128 can be used either for graphics or text.

44. Versatile Programmable Star Display

Most of the running lights display circuits available in the market are not programmable. Here's a versatile star display project that provides digital control of all the functions interactively and can be programmed for any desired display sequence. It is built around Atmel's AT89C2051 microcontroller to drive eleven incandescent light bulbs/tubes.

45. AT89C51-Based Moving-Message Display

The circuit presented here uses 16 common-anode, single-digit, alphanumeric displays to show 16 characters at a time using AT89C51 microcontroller.

46. LED Light Chaser for Five Lighting Effects

Light chaser circuits can be used to create lighting animation sequences. Here is one such application based on AT89C51 microcontroller that generates five different lighting effects using 120 LEDs connected to 24 input/output (I/O) lines of the microcontroller.

47. Interfacing Nokia Colour LCD with AVR Microcontroller

Here we describe how to use the colour LCD of a Nokia handset (model 6100, 7210, 6610, 7250 or 6220) with Philips PCF8833 chipset through ATmega2560 ATMEL AVR microcontroller.

48. Cellphone-Operated Land Rover

Here is a robot with a mobile phone attached to it. You can control the robot from another mobile phone by making a call to the mobile phone attached to the robot. The received signal is processed by the ATmega16 AVR microcontroller and control the motor drivers to drive the motors for forward or backward motion or a turn.

49. Automated Line-Following Robot

It is an AT89C51 microcontroller-based self-operating robot that detects and follows a line drawn on the floor. The path to be taken is indicated by a white line on a black surface. The control system senses the line and manoeuvre the robot to stay on course while constantly correcting the wrong moves using feedback mechanism.

50. Arduino-Based RF Controlled Robot

Here we present a simple Arduino-board based robot that can be driven remotely using an RF remote control.

51. PC-Based Wireless Control for Toy Car

It is a P89V51RD2 microcontroller based project to show how you can control a toy car through your PC's serial port using a pair of ASK transmitter and receiver RF modules.

Table of Contents

Security Systems

Domestic Applications

Industrial Applications

Measurement

Display Systems

Robotics

Security Systems

ACCESS CONTROL SYSTEM

■ VINAY CHADDHA

Security is a prime concern in our day-to-day life. Everyone wants to be as much secure as possible. An access-control system forms a vital link in a security chain. The microprocessor-based digital lock presented here is an access-control system that allows only authorised persons to access a restricted area.

System overview

The block diagram of the access-control system is shown in Fig. 1. The system comprises a small electronic unit with a numeric keypad, which is fixed outside the entry door to control a solenoid-operated lock. When an authorised person enters a predetermined number (password) via the keypad, the relay operates for a limited time to unlatch the solenoid-operated lock so the door can be pushed/pulled open. At the end of preset delay, the relay de-energises and the door gets locked again. If the entered password is correct the unit gives three small beeps, and if the entered password is wrong it gives a longer beep of one second.

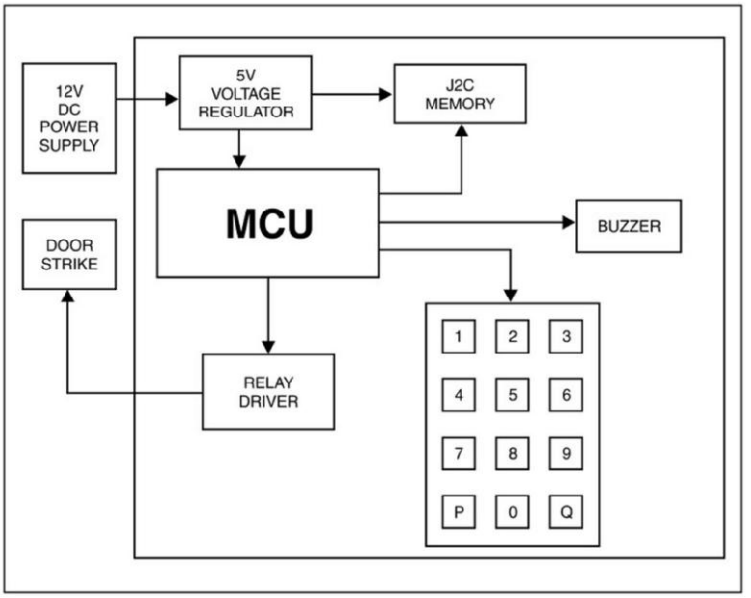


Fig. 1: Block diagram of the access-control system

PARTS LIST	
Semiconductors:	
IC1(U1)	- MC68HC705KJ1 microcontroller
IC2 (U2)	- ST24C02 I2C EEPROM
IC3 (MN1)	- MN1280 reset stabiliser
IC4 (Reg1)	- 7805 +5V regulator
T1, T2 (Q1, Q2)	- BC547 npn transistor
D1, D2	- 1N4007 rectifier diode
LED1	- Red LED
Resistors (all ¼-watt, ±5% carbon, unless stated otherwise):	
R1-R6	- 10-kilo-ohm
R7-R9	- 1-kilo-ohm
Capacitors:	
C1, C2	- 33pF ceramic disk
C3, C4, C6, C7	- 0.1µF ceramic disk
C5	- 10µF, 10V electrolytic
Miscellaneous:	
Xtal (Y1)	- 4MHz quartz crystal
PZ1 (BZ1)	- Ceramic piezo buzzer
Con1	- Power-supply connector
Con2	- 2-pin male/female Berg connectors - 7-pin male/female Berg connectors
SW1-SW12	- Tactile keyboard switch
RL1 (RLY1)	- 1C/O, 12V, 250-ohm miniature relay

The system uses a compact circuitry built around Motorola's MC68HC705KJ1 microcontroller and a non-volatile I²C EEPROM (ST24C02) capable of retaining the password data for over ten years.

The user can modify the password as well as relay-activation time duration for door entry. This version of software enables use of the unit even without the I²C EEPROM. (However, without EEPROM, the password and relay-activation time duration will be reset to default values on interruption of the power supply.)

Hardware details

Fig. 2 shows the access control circuit. Its main components are a microcontroller, I²C memory, power supply, keypad, relay, and buzzer.

Microcontroller. The 16-pin MC68HC705KJ1 microcontroller from Motorola has the following features:

- Eleven bidirectional input/output (I/O) pins
- 1240 bytes of OTPROM program memory

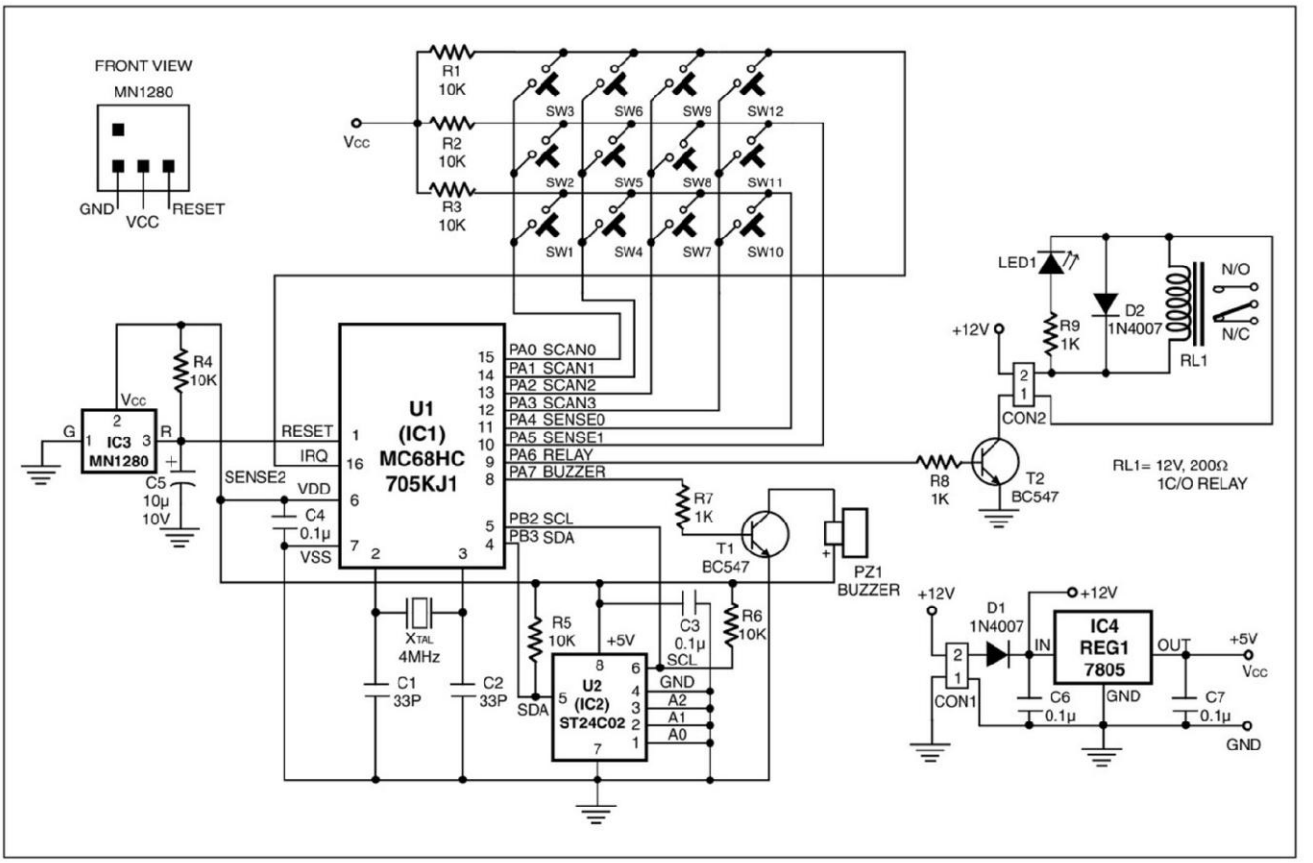


Fig. 2: Schematic diagram of the access-control system

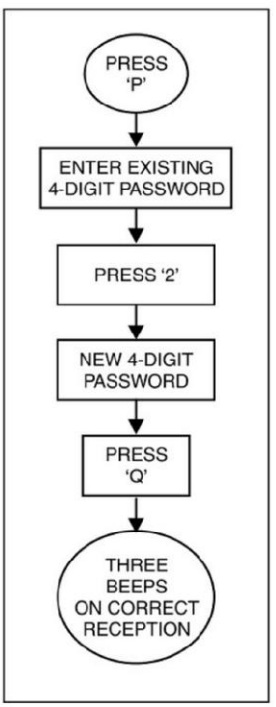


Fig. 3: Flow-chart for changing the password

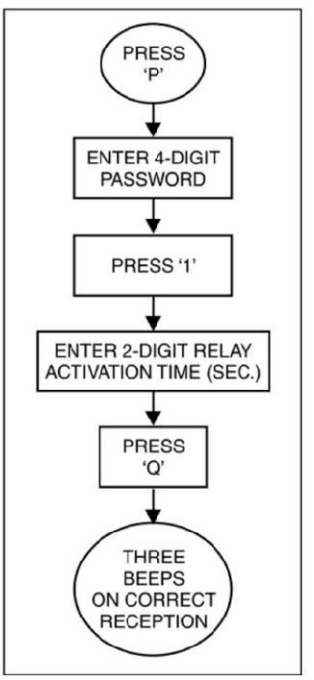


Fig. 4: Flow-chart for changing the relay-activation duration

• 64 bytes of user RAM
• 15-stage multiple-function timer

Out of eleven I/O pins, seven lines have been used for the keyboard, one for the buzzer, one for relay operation, and two (SCL and SDA, i.e. serial clock and serial data lines) for communication with I²C EEPROM.

I²C memory. A two-wire serial EEPROM (ST24C02) is used in the circuit to retain the password and the relay-activation time duration data. Data stored remains in the memory even after power failure, as the memory ensures reading of the latest saved settings by the microcontroller.

This I²C bus-compatible 2048-bit (2-kbit) EEPROM is organised as 256×8 bits. It can retain data for more than ten years. Using just two lines (SCL and SDA) of the memory, the microcontroller can read and write the bytes corresponding to the data required to be stored.

(***Note.*** For details of the microcontroller and programming of I²C EEPROM, you may refer to the article 'Caller ID Unit Using Microcontroller' published in April '99 issue of EFY and the article 'Remote-controlled Audio Processor Using Microcontroller' published in Sep. '99 issue of EFY also in Electronics Projects Vol. 20. The

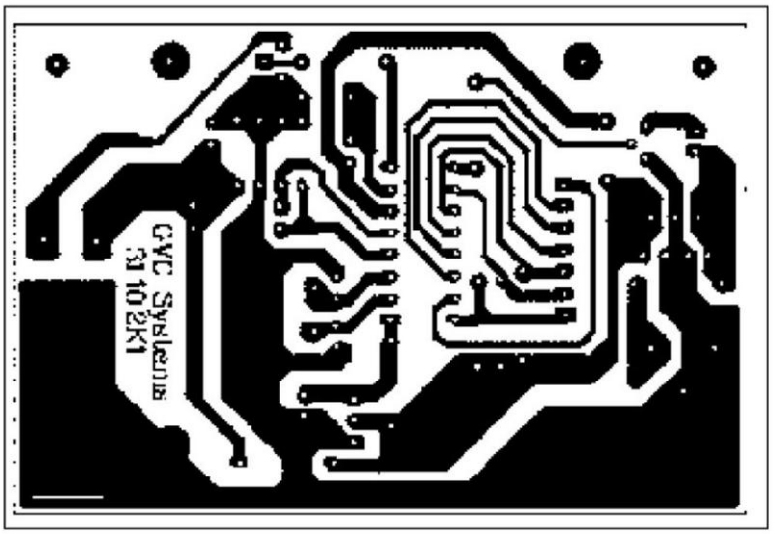


Fig. 5: Actual-size, single-side PCB for the access-control system without keypad (Main PCB)

information pertaining to I²C EEPROM is available on STMicroelectronics' Website.)

Power supply. The power supply unit provides a constant 5V supply to the entire unit. This is a conventional circuit using external 12V DC adaptor and fixed 3-pin voltage regulator 7805. Diode D1 is used in series with 12V input to avoid damage to the unit in case reverse voltage is applied by mistake.

Keypad. A 12-key numeric keypad for password entry is connected to the microcontroller. The keypad is also used for modifying the default password as well as relay-activation time period. To economise on the use of I/O pins, we use only seven pins for scanning and sensing twelve keys.

The keypad is arranged in a 3x4 matrix.

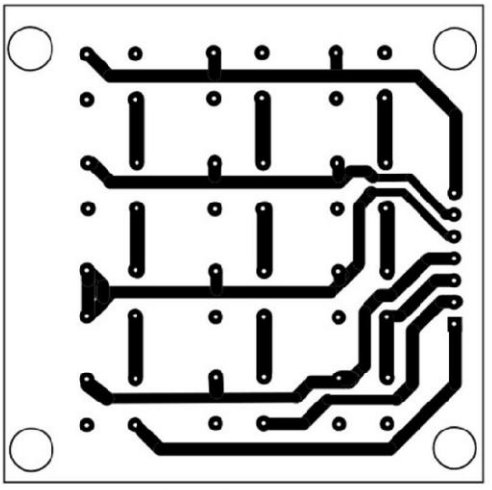


Fig. 6: Actual-size, single-side PCB for the keypad

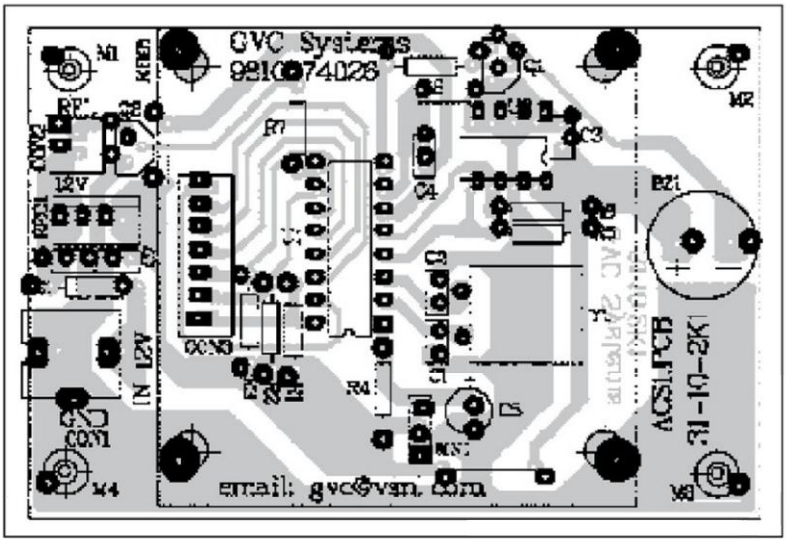


Fig. 7: Component layout for the PCB in Fig. 5

There are four scan lines/pins, which are set in output mode, and three sense keys, which are used as input lines to the microcontroller.

At 5ms interval, the microcontroller sets one of the four scan lines as low and other three scan lines as high, and then checks for the status of sense lines one by one. If any of the sense lines is found low, it means that a key at the intersection of a specific scan line and sense line has been pressed.

Similarly, after 5 ms, the next scan line is made low and remaining three scan lines are taken high, and again all three sense lines are checked for low level. This way the microcontroller can check whether any of the twelve keys is pressed.

Due to the high speed of the microcontroller, status of different keys is checked in less than 100 ms and a key-press is detected and identified. As the keys are pressed manually by the user, this delay of 100 ms is not noticeable. The net result is that we save on I/O pins of the microcontroller by sacrificing almost nothing.

Relay. A single-pole double-throw (SPDT) relay is connected to pin 9 of the microcontroller through a driver transistor. The relay requires 12 volts at a current of around 50 mA, which cannot be provided by the microcontroller. So the driver transistor is added. The relay is used to operate the external solenoid forming part of a locking

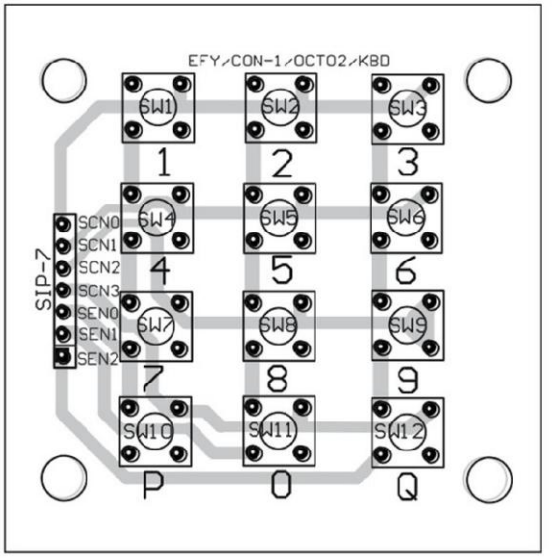


Fig. 8: Component layout for the PCB in Fig. 6

device or for operating any other electrical device. Normally, the relay remains off. As soon as pin 9 of the microcontroller goes high, the relay operates.

Buzzer. The buzzer is connected to pin 8 of the microcontroller. It beeps to indicate key and password entry. The buzzer gives a small beep whenever a key is pressed. In the case of a wrong password entry the buzzer gives a long beep, and in the case of a right password entry it gives three short beeps. The buzzer also gives short beeps as long as the relay remains energised.

Operation

The complete design is based on two parameters: the password and the relay-activation time duration. Both these parameters can be changed without making any change in the hardware. The user can change these parameters any number of times using the keypad. The flow-charts for changing the password and relay-

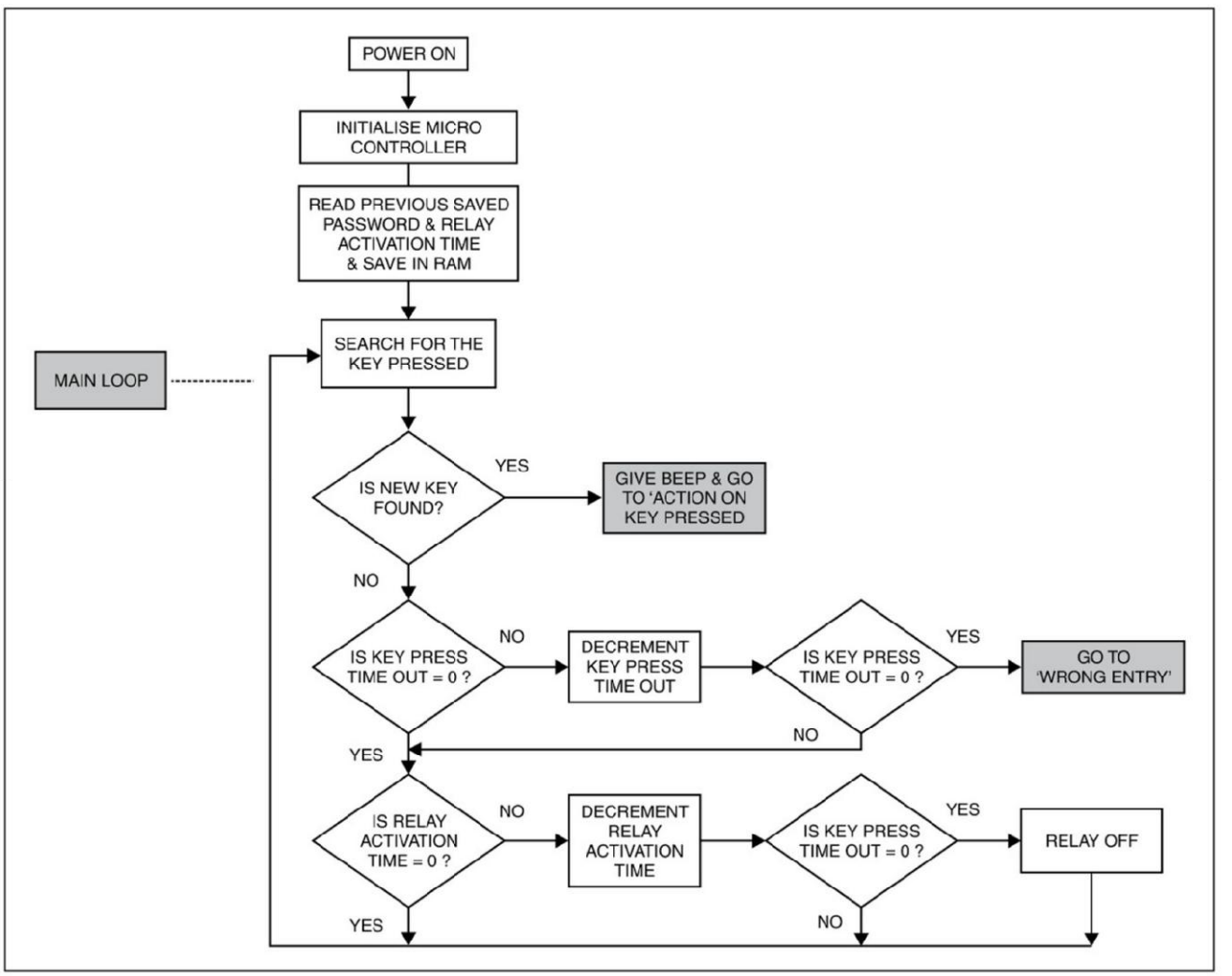


Fig. 9(a): Flow-chart for the access-control system, continued in Figs 9(b) and 9(c)

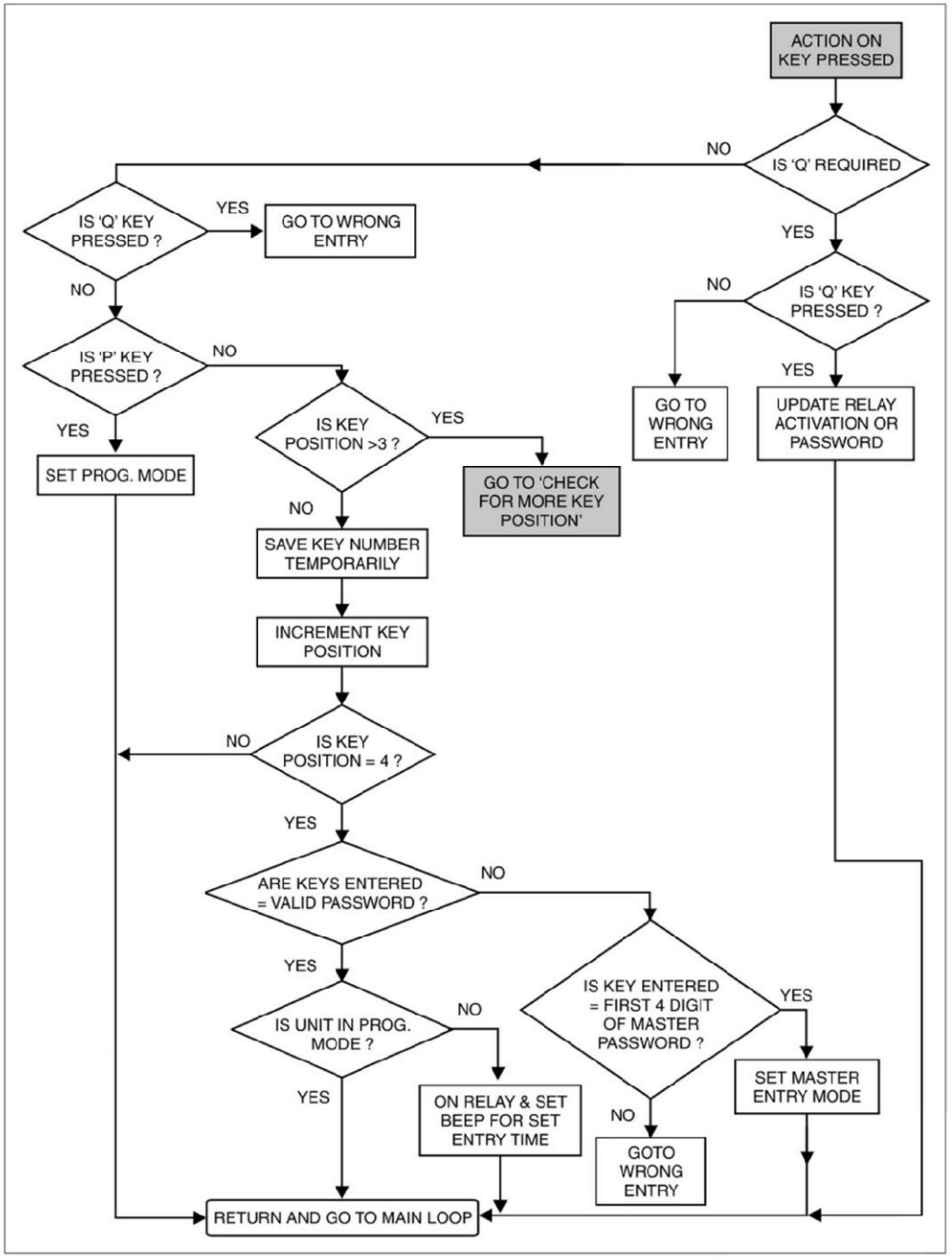


Fig. 9(b): Flow-chart for the access-control system, continued from Fig. 9(a)

activation time duration are shown in Figs 3 and 4, respectively.

Testing

Actual-size, single-side PCBs for the access control system (without keypad) and that of the keypad are shown in Figs 5 and 6, respectively, with their component layouts in Figs 7 and 8, respectively. During assembly ensure proper mating of Con 3 (female) on main PCB with SIP-7 (male) connector mounted on trackside of keypad PCB. After assembling the unit, check various points without inserting the programmed microcontroller and memory ICs as follows:

• Connect the external power source (a DC adaptor capable of delivering 200 mA at 12V DC), ensuring correct polarity.

• Check input and output voltages of regulator 7805. Ensure that the input voltage is 8-12V DC from an external source. The output at pin 3 of the 7805 should be 5 volts.

• Check 5 volts at pin 6 of the MCU (IC1) and pin 8 of the memory (IC2) with respect to ground pin 7 of IC1 and pin 4 of IC2.

• Check relay operation by shorting pin 9 of the MCU socket to 5 volts using a small wire. Normally, the relay would remain off. However, when pin 9 of the MCU socket is connected to 5V, the relay should energise.

• Check buzzer operation by shorting pin 8 of the MCU socket to 5 volts using a small piece of wire. Normally, the buzzer would be off. As soon as you short pin 8 of the MCU socket to +5V, the buzzer will produce a continuous beep sound.

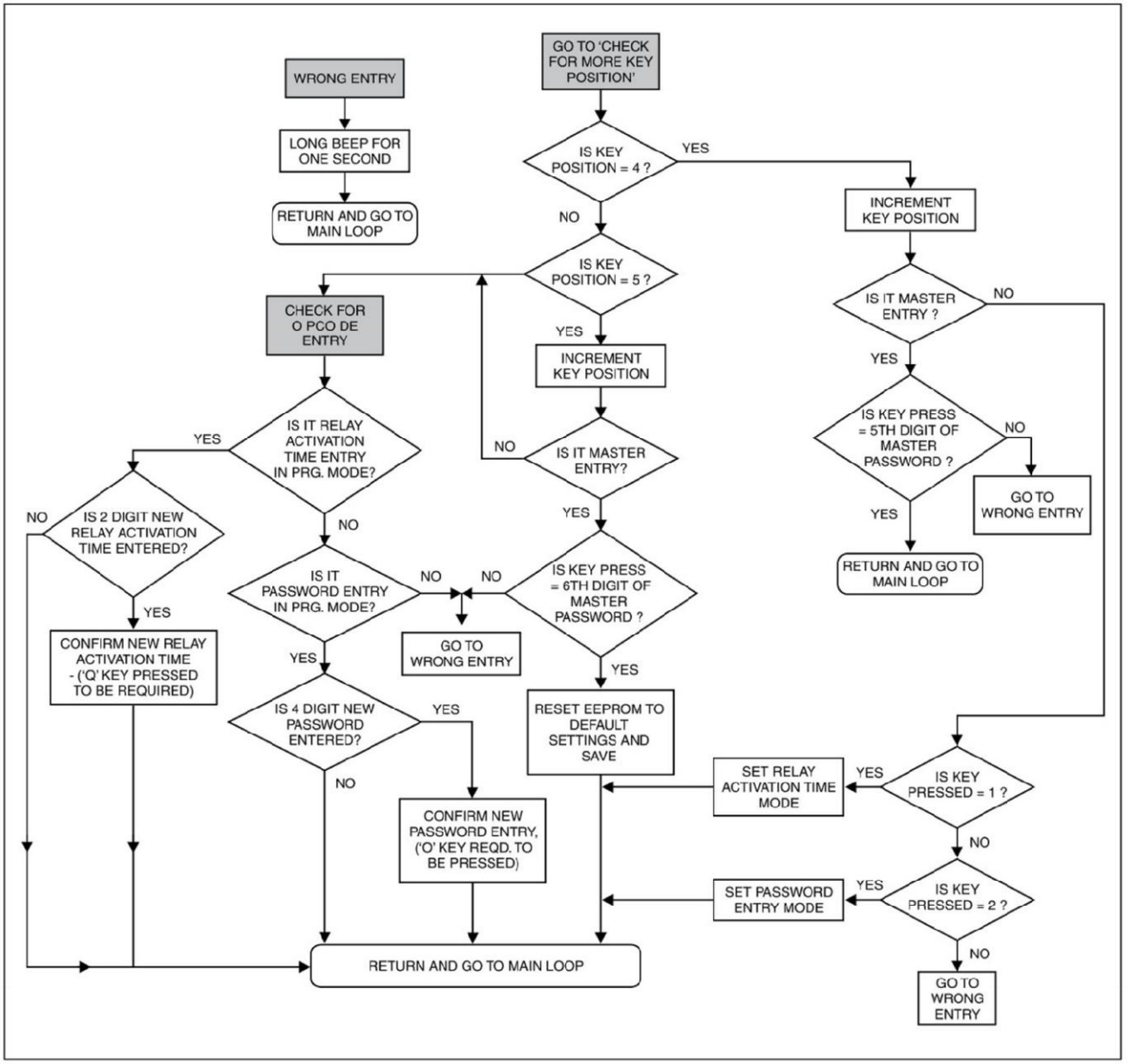


Fig. 9(c): Flow-chart for the access-control system, continued from Fig. 9(b)

• Physically check that only the capacitors of 27 to 33 pF are connected to crystal pins 2 and 3 of the MCU. For a higher-value capacitor, the crystal will not work.

Operation

Switch off the supply and insert only the microcontroller. Ensure correct direction and correct insertion of all the pins. Switch on the unit. On entering 1111 (default password) through the keypad, the relay will operate for around 10 seconds (default time duration). Each key-press gives a short beep. The buzzer will also beep for 10 seconds when the relay is 'on'. On entering some other code, say, 9999, the relay should not operate and the buzzer should give a long beep.

Change the password and the relay time. Check the operation with new password and relay activation period. Since there is no memory, the new password and relay time entered will be lost as soon as you switch off the unit.

The next time you switch on the unit, the password is again set to 1111 and the relay time to 10 seconds as default parameters.

Now insert the memory IC and change the password and the relay-activation time duration. On changing the same, the new password and changed relay-activation time are saved in the memory, which will be recalled at the next power-on. (***Note.*** In case you have forgotten the changed password, you cannot operate the unit unless you install a new/blank memory.)

Caution. Take care while connecting and using the live 220V wires.

The software

For software development the author has taken the help of Understanding Small Microcontrollers, MC68HC705KJ1 Technical Data book, and In-Circuit Simulator User's Manual. The development tools used include WinIDE software for KJ1 (including editor, assembler, simulator and programmer), in-circuit simulator (referred to as JICS board), and IBM PC with Windows OS and CD drive.

DOS-based programs can also be used for software development. So if you are comfortable with DOS or have an old computer with limited hard disk capacity, you will still face no difficulty.

Program development steps. You can write the software by using the following steps:

1. Read and understand the microcontroller's operation and instructions as well as the operation of WinIDE software. (The help option of the software will clear most of your doubts.) You should also have a clear knowledge of the logic sequence of the end-product operation. For this, you can make a flow-chart. (Flow-chart for this access control system is shown in Figs 9(a)-(c). The corresponding software source code is given at the end of this article.)

2. Convert the flow-charts to source program in Assembly language making use of the instruction set of the microcontroller and assembler directives. You can use any text editor for writing the same or use the text editor of the Integrated Development Environment (IDE), which also includes assembler, simulator, and programming software. The Assembly-level program is to be saved in a file with .asm extension.

3. Assemble the source code, i.e. convert the source code (file with extension .ASM) into object code (machine language) using assembler/compiler tab of environmental setting in WinIDE. The object code will be in S19 format, i.e. the object code file will have extension .S19. You can also choose options within the dialogue box to generate listing file with extension .LST and .MAP file for source-level debugging. Thus if your source program was titled 'main.asm', you will get main.s19, main.lst, and main.map files after successful assembly.

4. Simulate your program using the WinIDE software, JICS board, and the target board (the PCB with keyboard, memory, buzzer, etc). JICS board is connected to the computer through serial port (9-pin/25-pin) of the computer. The target board is connected to JICS board through a 16-pin DIP header cable.

During simulation you may find that the program is not behaving properly. Assuming that your hardware is okay, the most probable reason is an error in writing the software. So look for faults in your logic/code and rectify them. You should be able to simulate complete functions without using the actual microcontroller chip.

5. Now, program the microcontroller with the developed and tested software. After programming the microcontroller, insert it into the circuit and check all functions again.

Possible modifications

The circuit can be modified to have more than one password, advanced functions like real-time clock, computer connectivity via serial/parallel port to log data, and interfacing to a bar code reader instead of keypad for opening the lock. These additions may entail using a different microcontroller with more memory and I/O pins, but using essentially the same hardware configuration while writing a fresh program.

Note. The MN1280 is attached to reset pin 9 of the microcontroller. If the MN1280 is not available, you can use only the RC circuit.

Download Source Code: http://efymag.com/Access_Control_System-Oct02.zip

MAIN.ASM

```
;;*****************************************
;;PROJECT     :- ACCESS CONTROL (GENERAL)
;;VERSION      :- 01
;;STARTING DATE   :- 09-10-2k  day - monday
;;IC                                        :-
KJ1
;;HARDWARE :- 12 KEYS\1LED\1HOOTER\1MEMORY
;;HARDWARE REC.   :- 06-10-2k
;;FEATURES    :- ENTER PASSWORD TO OPEN DOOR
;*****************************************
                org             0c0h
$setnot         testing
$include        "stdj1.asm"
$include        "ports.asm"
$include        "variable.asm"

key_word        equ             14h
key_word1       equ             28h
second_last_kw  equ             5h
last_key_word   equ             7h

et_buff         db               2

org             300h

$include        "iic.asm"
$include        "macro.asm"
$include        "readkbd2.asm"

start:          rsp

;***************************** INITIALISE PORT ****
*****************************

   init_port        ddra             ;; initialise
port a
   init_port       porta

   init_port        ddrb             ;; initialise
port b
   init_port       portb

;************************** CLEAR MEMORY\INITIALISE
TIMER *********************************
clear_mem       ;; clear Ram

init_timer      ;; initialise timer

chk_mem         ;; check EEPROM

;; if bad_mem flag = 1 then goto read_defval
;; if bad_mem flag = 0 then read values from eeprom

brset   bad_mem,status,read_defval

;; program comes here when bad_mem flag = 00
;; at power on e_add & mem_ptr = 00
;;************************** READ VALUES FROM EEPROM
*************************
;; read 2 byte password/entry time from EEPROM
read_mem_val    clr     mem_ptr
                clr     e_add
read_nxt_val:    jsr       get_eeprom_info ;; read
from eeprom
lda     e_dat           ;; save read value in e_dat
ldx     mem_ptr         ;; set index reg as pointer
sta     password,x      ;; save read value in
cmp  #0ffh            ;; if value read from EEPROM
is ff then
;; goto read def val
                beq     read_defval
inc     e_add           ;; increment e_add
inc     mem_ptr         ;; increment ptr
lda     mem_ptr
cmp     #max_iic_bytes  ;; is all 3 bytes read
bne     read_nxt_val    ;; if no goto read_mem_val
bra     main_loop       ;; if yes goto main_loop

read_defval:    jsr     read_def_val

;;************************ MAIN LOOP *************
*************************
;; after every one tick over call sense_kbd
;; after every half second over call chk_set_beep
;; after every second check kbd_timeout\entry_time_
out
main_loop:       brclr    one_tick,tim_status,main_
loop
                bclr    one_tick,tim_status
                jsr     kbd_sense

chk_hs_over      brclr    half_sec,tim_status,chk_1_
sec
                bclr    half_sec,tim_status
                jsr     chk_set_beep

chk_1_sec       brclr   one_sec,tim_status,ret_act-
1sec
                bclr    one_sec,tim_status

;; program comes here after every second over
;  ************************ DECREMENT  KBD  TIMEOUT
*****************************
a1s_tstkbd      tst     kbd_timeout             ; if
timeout = 0 then
beq     tst_eto                   ; goto check for
entry time
dec      kbd_timeout              ; else decrement
kbd time
tst      kbd_timeout               ; again chk kbd
timeout
bne      tst_eto                    ; if # 0 goto
tst_eto
jsr     wrong_entry             ; give wrong entry
signal

;************************  DECREMENT  ENTRY  TIME
*****************************
;; check for entry time = 00
tst_eto:        tst     entry_time_out          ; if
timeout = 00 then
beq     ret_act1sec             ; ret_act1sec
dec      entry_time_out           ; else decrement
timeout
tst      entry_time_out          ; again chk entry
time
bne      ret_act1sec              ; if # zero goto
ret_act1sec
bclr    led_arm,led_port        ; else ON led arm

ret_act1sec

; ********************** CHECK FOR KEY ***********
*****************************
```

```
; if new key found flag set then goto act kbd else
goto main_loop
chkbd            brclr    new_key_found,status,ret_
chkbd  ; if new key found then set
bclr     new_key_found,status            ; flag
jsr      act_kbd                         ; call act-
kbd
ret_chkbd                jmp             main_loop
; else goto main loop

;***************************** ACTKBD *************
*****************************
;; set key press timeout to 10 seconds
act_kbd:          lda      #10t                    ;
set key press timeout = 10secs
sta      kbd_timeout

lda      kbd_pos                  ; read kbd pos

;*************************** KEY PROGRAM OK PRESSED
*****************************
act_kbd1:        cmp      #k_pgm_ok               ; is
pgm ok key pressed
bne      act_kbd2                 ; if no goto act_
kbd2
jsr      chk_po_status            ; if yes call chk_
po_status
bra      ret_actkbd               ; goto ret_actkbd

;; program here checks for po_password\po_entry_time
flag
;; if po_password\po_entry_time flag = 1 and if some
other key press
;; accept pgm_ok_key then goto wrong entry
;; else goto chk_pgm_key
act_kbd2              brclr      po_password,entry_
status,chk4poet
                jmp      wrong_entry

chk4poet:            brclr      po_entry_time,entry_
status,chk_pgm_key
                jmp      wrong_entry

;*************************** KEY PROGRAM PRESSED
*****************************
chk_pgm_key:     cmp      #k_program              ;
is pgm_ok key press
bne      act_kbd3                 ; if no goto act_
kbd3
bset     pgm_mode,status          ; if yes set flag
of pgm_mode
clr      buff_pointer            ; clear all pointers
clr      entry_status            ; clear entry status
clr      kbd_timeout
bra      ret_actkbd      ; give beep while returning

;************************** OTHER KEY PRESSED *****
*****************************
;; check for password code
;; first chk for buff pointer is buffer pointer > 3
if yes then goto is_it_mode
;; else take first digit pressed in kbd_buff,second
digit in kbd_buff+1
;; third digit in kbd_buff+2 & fourth digit in
kbd_buff+3

act_kbd3         ldx      buff_pointer    ;; is all 4
digit password enters
                 cpx      #3
bhi      is_it_mode      ;; if yes then goto is_it_
mode
lda      kbd_pos          ;; else store kbd_pos in
kbd_buff+ptr
sta      kbd_buff,x
inc      buff_pointer    ;; increment pointer
lda      buff_pointer     ;; is it 4th digit to be
entered
cmp      #4              ;; if no then return
bne      ret_actkbd

;; program comes here when all 4 keys entered
;; check for valid code
;; if not valid code then give long beep and clear
buff_pointer\kbd_timeout
;; and return
;; else clear sys_arm flag and give accp beep
jsr      pack_buff               ; call pack buffer

;; check for 4 key press
;; if it is equals to password then
;;    return
;; if it is not equals to password then goto wrong
entry
                 lda     kbd_buff
                 cmp     password
                 bne     chk4master_kw
                 lda     kbd_buff+1
                 cmp     password+1
                 bne     chk4master_kw

;; PROGRAM COMES HERE WHEN 4 DIGIT CORRECT PASSWORD
IS ENTERED
                 brset   pgm_mode,status,ret_actkbd
                  bset    led_arm,led_port        ;
off led arm
lda      entry_time              ; set entry_time_out
sta      entry_time_out          ;
jmp      entry_over              ; call entry_over

;; here program checks for master key word
;; if key sequence entered is equals to first 4 mater
key word then
;;    e_key_word flag is set
;; else
;;     long beep is heard as wrong entry

chk4master_kw:
lda      kbd_buff
cmp      #key_word                      ;; 14
bne      wrong_entry
lda      kbd_buff+1
cmp      #key_word1                     ;; 28
bne      wrong_entry
bset     es_key_word,entry_status
bra      ret_actkbd

;; program comes here when unit is in programming
mode and 4 digit password enters
;; if 4 digit entered # password then goto wrong
entry
;; else return
xxxx:            lda      kbd_buff                ;
compare kbd_buff with
cmp      password            ; password
bne      wrong_entry        ; if # goto wrong entry
lda      kbd_buff+1      ; if = compare kbd_buff+1
```

```
with
cmp     password+1              ; password+1
bne      wrong_entry              ; if # goto wrong
entry
ret_actkbd      jmp     quick_beep              ;
give small beep after every
                                                ;
key press
ret_actkbd1:    rts                             ;
return


is_it_mode:     cpx     #04                     ; is
buffer pointer = 4
bne     chk4parameters          ; if # goto chk4pa-
rameters
inc      buff_pointer             ; else increment
pointer

                brclr es_key_word,entry_status,iim1
;; program comes here when key word entry is checked
;; check is 5th key press = 8 then return
;; else
;;     goot wrong key and give long beep
                lda   kbd_pos
                 cmp   #second_last_kw          ;;
next digit is 5
                bne   wrong_entry
                jmp   ret_actkbd
iim1:
;; key 1 is for entry time
;; key 2 for password change
lda     kbd_pos                 ; read kbd_pos
cmp     #01                     ; is key 1 press
bne     chk2                    ; if # goto chk2
set_entry_time  bset    es_entry_time,entry_status ;
set flag of es_entry_time
bra     ret_actkbd              ; return

chk2:           cmp     #02                     ; is
key 2 press
bne     chk3                    ; if # goto chk3
set_new_password bset    es_password,entry_status ;
else set flag of es_password
bra     ret_actkbd              ; return

chk3:
;;************************ WRONG ENTRY ************
*****************************
wrong_entry     jsr     long_beep               ;
give long beep
                jmp     entry_over              ;
goto entry over


;; program comes here when buffer pointer is > 4
chk4parameters:
cpx     #05                     ; if buff_pointer >
5 then
bne     more_parameters         ; goto more_param-
eters
inc      buff_pointer             ; else increment
pointer

                    brclr       es_key_word,entry_
status,c4p1
                lda       kbd_pos
                  cmp       #last_key_word      ;
last digit for master key word is 7
                bne       wrong_entry
                jmp       master_reset_eeprom

c4p1:
;; program comes here when buff_pointer = 6
;; check is it es_entry_time = 1
;;      if yes then store key press in last_key _val
;;      set flag of po_entry_time
;;      return
;;      if no then goto chk4es_pw
                    brclr    es_entry_time,entry_
status,chk4es_pw
                lda     kbd_pos
                sta     et_buff
                jmp     ret_actkbd

;; program comes here when buff_pointer = 6 and
es_entry_time = 0
;; check es_password flag
;;      if flag set then
;;      save key press in kbd_buff
;;      else goto wrong entry
more_parameters:
                    brclr    es_entry_time,entry_
status,chk4es_pw
                bset    po_entry_time,entry_status
                lda     kbd_pos
                sta     et_buff+1
                tst     et_buff
                bne     ret_actkbd
                tst     et_buff+1
                bne     ret_actkbd
                jmp     wrong_entry

chk4es_pw:          brclr      es_password,entry_
status,wrong_entry
                 lda     buff_pointer           ;
subtract buff_pointer with 6
                sub     #6
tax                     ; set subtracted val as pointer
lda     kbd_pos      ; read kbd_pos
sta   kbd_buff,x           ; save in kbd_buff+ptr
inc     buff_pointer            ; increment pointer
lda     buff_pointer            ; if pointer = 10
                cmp     #10t                    ; if
no then return
bne     ret_actkbd
bset    po_password,entry_status ; else set po_pass-
word flag
bra     ret_actkbd              ; return

e n t r y _ t a b l e                           d b
5t,2,4,6,8,10t,12t,14t,16t,18t

;; program comes here when pgm_ok key press
;; chck is po_entry_time flag = 1
;; if yes then
;;      set last key press as pointer
;;        take corresponding entry time from entry
table
;;      and save in entry_time
;;      goto com_po_ret
chk_po_status:    brclr      po_entry_time,entry_
status,chk4popassword
                bclr    po_entry_time,entry_status
                jsr     pack_et_buff
                bra     com_po_ret

;; program comes here when po_entry_time = 0
```

```
;; program here checks for po_password
;; if po_password = 1 then
;;      call pack_buff
;;      store change password in password variable
;;      store in eeprom
;;      call entry_over
;;      give acc_beep
;;      return
chk4popassword
                        brclr    po_password,entry_
status,chk4more
                bclr    po_password,entry_status
upd_password    jsr      pack_buff                ;
call pack_buff
lda     kbd_buff                ; save kbd_buff in
sta     password                ; password
lda     kbd_buff+1            ; save kbd_buff+1 in
sta     password+1            ; password+1

com_po_ret      jsr      store_memory             ;
save changed parameter in eeprom
jsr     entry_over              ; call entry over
jsr      acc_beep                 ; give acceptance
beep
jmp     ret_actkbd1             ; return

chk4more        bra     wrong_entry              ;
else give long beep


;; SUBROUTINES :-
;;***************

;***************************** ACCEPTANCE BEEP ******
*************************
;; give beep thrice
acc_beep        jsr     short_beep
                jsr     short_delay
                jsr     short_delay
                jsr     short_beep
                jsr     short_delay
                jsr     short_delay
                jmp     short_beep

;***************************** ENTRY OVER ***********
*************************
;; clear pointer\timeout\entry_status\pgm_mode flag
entry_over:     bclr    pgm_mode,status
                clr     buff_pointer
                clr     kbd_timeout
                clr     entry_status
                rts

; ************************** SHORT DELAY **********
*************************
short_delay     lda     running_ticks
                add     #beep_time
                sta     delay_temp
sd_wait         lda     delay_temp
                cmp     running_ticks
                bne     sd_wait
                rts

;***************************** LONG ENTRY **********
*************************
;; give this beep when wrong entry
;; giva a long beep for around 1 sec
;; stay here till 1 second is over
long_beep       lda     #ticks_1_sec
                sta     buzzer_time_out
                bclr    buzzer,buzzer_port
lb_wait:        bsr     delay
                bsr     toggle_buzzer_pin
                tst     buzzer_time_out
                bne     lb_wait
                bset    buzzer,buzzer_port
                rts

;************************** SHORT BEEP
************************************
;; this routine is called from accp_beep and when
entry time # 0
;; and after every key press
;; beep for small time
;; set buzzer_time_out = beep_time
;; wait untill buzzer time out # 00
quick_beep:
short_beep      lda     #beep_time
                sta     buzzer_time_out
                bclr    buzzer,buzzer_port
sb_wait:        bsr     delay
                bsr     toggle_buzzer_pin
                tst     buzzer_time_out
                bne     sb_wait
                bset    buzzer,buzzer_port
                rts

;;************************ TOGGLE BUZZER PIN ******
*************************
;; if buzzer time out # 00 then toggle buzzer pin
toggle_buzzer_pin:
                brset   buzzer,buzzer_port,reset_
buzzer
                bset    buzzer,buzzer_port
                bra     ret_tbp
reset_buzzer:   bclr    buzzer,buzzer_port
ret_tbp:        rts

;;  ************************  DELAY  FOR  HALF  MSEC
*******************************
;; this delay is approximately = 499usec
;; 2+4+[(5+4+3)83]= 10998cycles
;; 998/.5 = 499usec = .5msec
delay:          lda     #83t
                sta     temp
wait_0:         dec     temp
                tst     temp
                bne     wait_0
                rts
;***************************** PACK BUFFER *********
*************************
pack_buff       lda     kbd_buff
                lsla
                lsla
                lsla
                lsla
                ora     kbd_buff+1
                sta     kbd_buff
                lda     kbd_buff+2
                lsla
                lsla
                lsla
                lsla
                ora     kbd_buff+3
                sta     kbd_buff+1
                rts

;***************************** STORE MEMORY *********
```

```
*************************
;; store 2byte password in eeprom
store_memory:   brset   bad_mem,status,ret_sm
                clr     e_add           ;; clear
e_add
                clr     mem_ptr         ;; clear
mem_ptr
nxt_data:
;; read data from RAM location
;; and store it in memory
ldx     mem_ptr         ;; set index register as ptr
lda     password,x      ;; read upper byte of pass-
word
sta     e_dat           ;; save in e_dat
jsr     set_eeprom_info ;; tx to eeprom
inc     e_add           ;; increment address
inc     mem_ptr         ;; increment pointer
lda     mem_ptr         ;; is all 3 bytes written
cmp     #max_iic_bytes  ;; if not goto nxt_data
bne     nxt_data        ;; else return
ret_sm:         rts
;;************************ TIMINT ***************
**************************
timint:         lda             #def_timer      ;;
set tscr = 14h
 sta            tscr
 bset           one_tick,tim_status ;; set flag for
0ne tick over
 inc            ticks           ;; increment ticks
 inc            running_ticks
;; if buzzer time out is not zero
;; then decrement buzzer timeout
;; interrupt comes here afetr every 8.2msec

 tst            buzzer_time_out
 beq            chk_half_sec
 dec            buzzer_time_out

chk_half_sec:   lda             ticks           ;;
compare ticks with
 cmp              #ticks_in_hsec  ;; ticks in half
sec
 bne            chk4secover     ;; if # goto chk-
4secover
  bset          half_sec,tim_status ;; set flag of
half sec over

chk4secover     lda             ticks           ;;
compare ticks with
  cmp             #ticks_1_sec    ;; ticks in one
second
   bne            ret_timint      ;; if # then re-
turn
   bset             half_sec,tim_status ;; set flag
of half sec
    bset            one_sec,tim_status  ;; set flag
of one sec

;    clr            running_ticks
     clr            ticks           ;; clear ticks
dummy:
ret_timint:     rti

;; start beep when entry or exit time is not zero
chk_set_beep    tst     entry_time_out
                beq     ret_csb
                jsr     short_beep
ret_csb         rts

;; master key word received
;; if key entered in following sequence then reset
EEPROm to default settings
;; Key word is 142587
;;  default  setting  is  that  password  entry  will
change to 1111
master_reset_eeprom:
                bsr     read_def_val
                 jsr     acc_beep                ;
give acceptance beep
                jsr     entry_over
                bra     store_memory

read_def_val    clrx
rdv_loop:       lda     def_table,x
                sta     password,x
                incx
                cpx     #max_iic_bytes
                bne     rdv_loop
                rts
;;  here  program  pack  entry  time  from  et_buff\
et_buff+1
;; first byte is in et_buff
;; second byte is in et_buff+1
;; output to entry_time var

;; for decimal selection multiply first number by 10t
and then add with next number

pack_et_buff:   lda     et_buff
                ldx     #10t
                mul
                add     et_buff+1
                sta     entry_time
                rts

;;************************** DEFAULT TABLE *******
**************************
def_table        db       11h      ; password     ;;
change defult password from 1234 to 1111
                db      11h     ; password+1
                db      10t     ; entry time

                org     7cdh
                jmp     start

                org     7f1h
                db      20h

                org     7f8h
                fdb     timint

                org     7fah
                fdb     dummy

                org     7fch
                fdb     dummy

                org     7feh
                fdb     start
```

IIC.ASM

```
;; IIC_TX
;; function : transfer 5 bytes from iic_buff to iic
bus
;; input    : iic_buff
;; output   : to iic
;; variables: rega, regx
;; constants: scl
;;            sda
;;            iicport
;;            iicont


;; input in a register
byte_iic:       bset            sda,iicont      ;
set sda as cutput port
ldx             #8              ; count of 8 bits
bit_iic:        rola                            ;
shift msb to carry
bcc              sda_low        ; if no carry(msb
low)
sda_high:       bset            sda,iicport     ;
carry set msb high
                bra             pulse_scl
sda_low:        bclr            sda,iicport
pulse_scl:      bsr              delay_small    ;
delay
                bset            scl,iicport     ;
set scl high
bsr             delay_small
bclr             scl,iicport    ; then scl is set
low
 ;              bsr             delay_small
decx                            ; is count over
bne             bit_iic         ; no next bit
bclr            sda,iicont      ; leave sda high by
making it input
bsr             delay_small
bsr             delay_small
bset            scl,iicport
bsr             delay_small
clc                              ; normal - clear
carry
brclr           sda,iicport,byte_over ;error if ackn
not rcvd
sec                             ; error - set carry
byte_over:      bclr             scl,iicport    ;
set scl low
bsr             delay_small
bsr             delay_small
bclr            sda,iicport     ;
rts                             ; leave with sda as
input

delay_small:    nop
    nop
    nop
    nop
    nop
    rts

set_eeprom_info
iic_tx:
;; generate start condition
;; first set sda then scl then make sda low while
scl is high
;; on return sda is low and scl is low
;; variables : iic_counter,iic_buff(six bytes)

restart_tx:

                bsr             gen_start
                lda             #0a0h
                bsr             byte_iic
                     bcs                    restart_tx
; restart if carry set
                lda             e_add
                bsr             byte_iic
                bcs             restart_tx
                lda             e_dat
                bsr             byte_iic
                bcs             restart_tx

;; generate stop condition
;; sda is set as output and low
;; fisrt sda is cleared  the scl is set high
;; then make sda high keeping scl high
;; on return scl is high and sda is also high

gen_stop:       bclr            sda,iicport
bset            sda,iicont       ; set sda as output
jsr             delay_small
bset            scl,iicport
bsr             delay_small
bset            sda,iicport             ; leave with
sda and
rts                                      ; scl high
and output

gen_start:       bset             sda,iicont      ;
sda as o/p
bset            sda,iicport             ; and high
bsr             delay_small
bset             scl,iicport             ; scl also
high
bsr             delay_small

                bclr            sda,iicport
                bsr             delay_small
                bclr            scl,iicport
                rts

get_eeprom_info
;; iic_rx
;; generate start byte
;; transfer address byte with bit 0 set to 1
;; if memory write e_add also
;; read one byte
;; and save in iic_status
;; generate stop byte
;; input : iicbuff (one byte- address of iic)
;; output : iic_status
;; variables : rega,regx
;; constants : scl,sda,iicport,iicont
iic_rx:
restart_rx:

                bsr             gen_start
                lda             #0a0h
dev_addr:           jsr                         byte_iic
; sda is input on return
                bcs             restart_rx
```

```
                lda             e_add
                     jsr                byte_iic
; second byte as mem add
                bcs             restart_rx
                bsr             gen_start
                lda             #0a1h
                     jsr                byte_iic
; sda is input on return
read_iicbyte:   ldx             #8
read_iicbit:      bset                  scl,iicport
; set scl high
;                   jsr               delay_small
; delay
;                   bclr              scl,iicport
; and again low
brset           sda,iicport,iic_1       ; read data
bit
iic_0           clc
                bra             read_iic
iic_1           sec
read_iic        rola
                    jsr                 delay_small
; delay
                    bclr                scl,iicport
; and again low
                decx
                bne             read_iicbit
                sta             e_dat
                bra             gen_stop
```

STDJ1.ASM

```
porta   equ     00h
portb   equ     01h
ddra    equ     04h
ddrb    equ     05h
pdra    equ     10h
pdrb    equ     11h
tscr    equ     08h
tcr     equ     09h
iscr    equ     0ah
copr    equ     7f0h
```

VARIABLE.ASM

```
last_key_val    db      00
entry_status    db      00
es_password     equ     1
es_entry_time   equ     2
po_entry_time   equ     3
po_password     equ     4
es_key_word     equ     5

temp            db      00
active_scan     db      00
kbd_temp        db      00
delay_temp      db      00
running_ticks   db      00
mem_ptr         db      00

kbd_timeout     db      00
buff_pointer    db      00
kbd_buff        db      00,00,00,00

status          db      00
new_key_found   equ     7
key_alarm       equ     6
bad_mem         equ     5
sys_arm         equ     4
pgm_mode        equ     3

password         db       00,00          ;; stored
in eeprom
entry_time       db       00             ;; stored
in eeprom

buzzer_time_out db      00
beep_time       equ     10t

entry_time_out  db      00
hooter_time     equ     2
hooter_alarm_tout db    00

e_add           db      00
e_dat           db      00
iic_buff        db      00


kbd_pos         db      00
last_key        db      00
same_key        db      00

def_timer       equ     14h

tim_status      db      00
one_tick        equ     7
half_sec        equ     6
one_sec         equ     5
one_min         equ     4

mins            db      00
ticks_1_sec     equ     122t
ticks_in_hsec   equ     61t
ticks           db      00
max_iic_bytes   equ     3

key_scan_cntr   db      00
```

READKBD2.ASM

```
scan_table:     db      0eh,0dh,0bh,07h
key_scan_port   equ     porta

;; sense2 line is at irq

kbd_sense
sense_line      lda     key_port                ;read
key port
                and     #30h
                bil     key_found
                ora     #40h
                cmp     #70h
```

```
                bne     key_found                ; no
some key pressed
                 bra     no_key_found             ;
yes no
key pressed

key_found       sta     kbd_temp
                lda     key_port                ;com-
pare key_port with kbd table
                 and     #0fh                    ;
remove unused line
                ora     kbd_temp
                clrx

try_nxt_code    cmp     kbd_table,x
beq     key_matched             ;if equal goto key
matched
incx                            ;else increment in-
dex register
cmpx    #max_keys                 ;compare it with
maximum keys
bne     try_nxt_code            ;if not equal goto
try nxt code

no_key_found    ldx     #0fh                    ;
k e y _ m a t c h e d                           t x a
;load accumulator with 'X'
cmp     kbd_pos                   ;compare it with
kbd pos
beq     ret_kbs                 ;if equal return
cmp     last_key                  ;compare it with
last key
bne     new_key                 ;if equal return
inc     same_key                ;else goto new key
& inc same
lda     same_key                ;for max debounce
load same key
cmp     #max_debounce           ;compare it with 4
bne     ret_kbs                 ;if not equal goto
ret kbs
upd_key                 lda             last_key
;load last key
sta     kbd_pos                 ;store it at kbd pos
cmp     #0fh                    ;is it key release
beq     ret_kbs                 ;yes-do not set flag
bset    new_key_found,status    ;set bit of new key
found in
bra     ret_kbs                   ;status and goto
ret kbs

new_key         sta     last_key
                clr     same_key
                bra     kbs_over

ret_kbs                 lda             kbd_pos
;load kbd pos
                cmp     #0fh                    ;
                bne     kbs_over                ;

change_sense    inc     key_scan_cntr
                lda     key_scan_cntr
                cmp     #04
                blo     cs1
                clr     key_scan_cntr

cs1:            lda     key_scan_port
                and     #0f0h
                 sta     key_scan_port          ;
reset all scan lines to zero on ports
                 ldx     key_scan_cntr          ;
output scan table to scan port one by one
                lda     scan_table,x
                ora     key_scan_port
                sta     key_scan_port
ret_sense_line
kbs_over        rts

max_keys        equ     12t

$if testing
max_debounce    equ     1
$elseif
max_debounce    equ     3t
$endif


;; code1                pin
;;scan0           bit     pa0     ; 16
;;scan1           bit     pa1     ; 15
;;scan2           bit     pa2     ; 14
;;scan3           bit     pa3     ; 13
;;sense0          bit     pa4     ; 12
;;sense1          bit     pa5     ; 11
;;sense2          bit     irq

;; code 0       13-irq  (pa3-pa5)
;; code 1       16-12   (pa0-pa4)
;; code 2       16-11   (pa0-pa5)

;; code 3       16-irq  (pa0-irq)
;; code 4       15-12   (pa1-pa4)
;; code 5       15-11   (pa1-pa5)

;; code 6       15-irq  (pa1-irq)
;; code 7       14-12   (pa2-pa4)
;; code 8       14-11   (pa2-pa5)

;; code 9       14-irq  (pa2-irq)
;; code 10      13-12   (pa3-pa4)       ;; key pro-
gram

;; code 12      13-11   (pa3-irq)       ;; key pro-
gram ok

kbd_table       db      057h            ;; code for
00
                db      06eh            ;; code for
01
                db      05eh            ;; code for
02
                db      03eh            ;; code for
03
                db      06dh            ;; code for
04
                db      05dh            ;; code for
05
                db      03dh            ;; code for
06
                db      06bh            ;; code for
07
                db      05bh            ;; code for
08
                db      03bh            ;; code for
09
                db      067h            ;; code for
pgm key
                db      037h            ;; code for
pgm ok key
```

PORTS.ASM

```
k_program       equ     10t
k_pgm_ok        equ     11t

scl             equ     2
sda             equ     3
iicport         equ     portb
iicont          equ     ddrb
                        ;; 7     6     5     4     3
2     1     0
def_ddra         equ      0cfh    ;; hoot   led   sen1
sen0 scan3 scan2 scan1 scan0
def_porta       equ     080h    ;; active low hooter
and led
;; at power on system armed led
def_ddrb        equ     0ch     ;; x      x     x     x
sda   scl   x      x
def_portb       equ     00

key_port        equ     porta

scan0           equ     0       ; 16
scan1           equ     1       ; 15
scan2           equ     2       ; 14
scan3           equ     3       ; 13
sense0          equ     4       ; 12
sense1          equ     5       ; 11
;;sense2        equ     irq      ; irq

led_port        equ     porta
led_arm         equ     6
toggle_led      equ     40h

buzzer_port     equ     porta
buzzer          equ     7
```

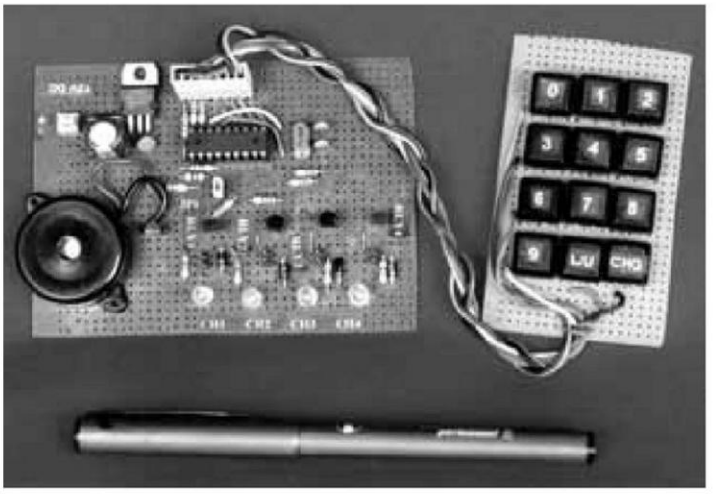
MACRO.ASM

```
$macro          chk_mem
bclr            bad_mem,status  ;; clear flag bad_mem
jsr             gen_start       ;; call gen_start
lda             #0a0h           ;; send device add
= 0a0h
jsr             byte_iic        ;; to memory
bcc              cm_over         ;; of carry clear
then return
bset               bad_mem,status   ;; if carry set
then set flag
cm_over                                         ;;
bad mem
$macroend

;; clear memory from 0c0h
$macro          clear_mem
ldx     #0c0h           ;clear memory
next_mm         clr     ,x
                incx
                bne     next_mm
$macroend


;; intialise timer
$macro          init_timer
                lda     #def_timer
                sta     tscr
                  cli                           ;enable
interrupt
$macroend
;; intialise porta , portb
$macro          init_port       port
                lda             #def_%1
                sta             %1
$macroend
```

PIC16F84-BASED CODED DEVICE SWITCHING SYSTEM

■ VIJAY KUMAR P.

Here's a microcontroller-based code lock that can be used for preventing unauthorised access to devices or for solenoid-operated locks/electrical devices. This code lock is built around Microchip's PIC16F84 microcontroller. Different passwords are used to access/operate different devices. So the code lock can be used as a multiuser code lock, where the users can access respective devices by entering the device number followed by the password. The password can be changed by the user and no external back-up supply is needed to retain the password. The password length for each device can be between 4 and 15 digits, as desired by the user.



Working model of PIC16F84-based coded device switching system

A buzzer has been added to provide suitable feedback with respect to the data entered via the keypad. The number of beeps indicates whether the data has been entered correctly or not. When anyone trying to access the device enters the incorrect password three times, the circuit sounds an alarm.

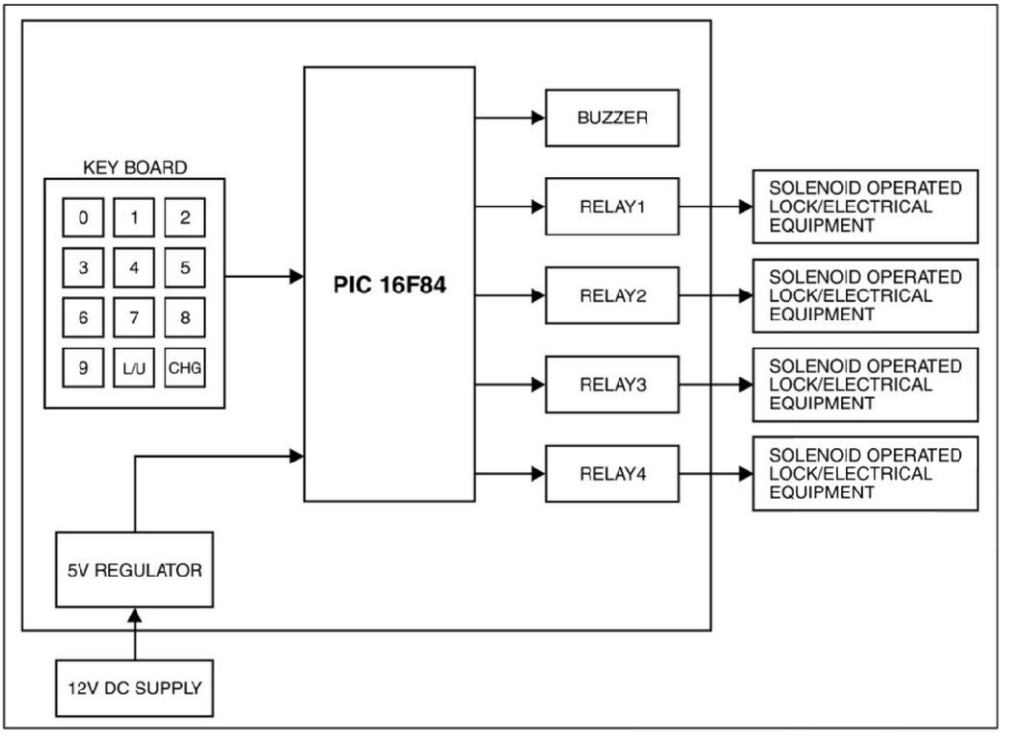


Fig. 1: Block diagram of PIC16F84-based coded device switching system

The alarm can be configured to work in two modes: auto-reset and latch-up. In the auto-reset alarm mode, all the keys pressed are ignored and the buzzer keeps beeping continuously for one minute, and thereafter the code lock resets automatically.

However, if you want additional security, you can enable the latch-up mode. In this mode the code lock never switches to the normal mode from the alarm mode and the only way to reset the code lock is to interrupt the power. When not in use, the code lock goes into sleep mode, and it

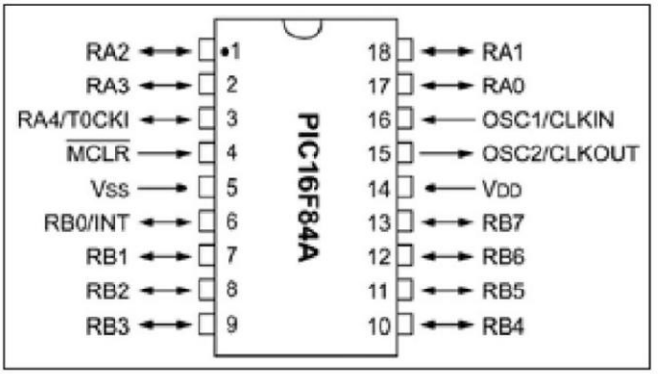


Fig. 2: Pin details of PIC18F84 microcontroller

PARTS LIST

Semiconductors:

IC1	- 7805 +5V regulator
IC2	- PIC16F84 microcontroller
T1-T5	- BC547 npn transistor
D1-D5	- 1N4007 rectifier diode
LED1-LED4	- Red LED

Resistors (all ¼-watt, ±5% carbon, unless stated otherwise):

R1	- 10-kilo-ohm
R2	- 4.7-kilo-ohm
R3-R5	- 220-ohm
R6-R10	- 2.2-kilo-ohm
R11-R14	- 1-kilo-ohm

Capacitors:

C1	- 470µF, 35V electrolytic
C2, C3	- 0.1µF ceramic disk
C4, C5	- 33pF ceramic disk

Miscellaneous:

RL1- RL4	- 12V, 285-ohm, 1c/o relay (OEN58 type 1C)
X_{TAL}	- 4MHz crystal
PZ1	- Piezobuzzer
S1-S12	- Push-to-on tactile switch

wakes up if any key is pressed. This feature reduces the power consumption by the microcontroller.

The main features of PIC16F84 microcontroller are:

1. Program and data memory are in separate blocks, with each having its own bus connecting to the CPU
2. Reduced instruction set controller (RISC) with only 35 instructions to learn
3. 1024 words (14-bit wide) of program memory
4. 68 bytes of data RAM
5. 64 bytes of data EEPROM
6. 8-bit wide data bus
7. 15 special-function registers (SFRs)
8. 13 input/output (I/O) pins with individual direction control
9. Code protection
10. Built-in power-on-reset, power-up timer, oscillator start-up timer
11. Power-saving sleep mode

Circuit description

Fig. 1 shows the block diagram of the microcontroller-based code lock. Pin diagram of PIC16F84 microcontroller is shown in Fig. 2. Basically, the circuit (shown in Fig. 3) comprises PIC16F84 microcontroller (IC2), 4x3 matrix keyboard, relays and buzzer.

The microcontroller. PIC16F84 is an 8-bit CMOS microcontoller. Its internal circuitry reduces the need for external components, thus reducing the cost and power consumption and enhancing the system reliability. The microcontroller has two ports, namely, Port A and Port B. Out of the available 13 bidirectional I/O pins of Ports A and B, seven pins are used for keyboard interfacing, four pins are used to drive the relays corresponding to the four devices and one pin is used to read the jumper status for selecting the alarm mode. One can reset the microcontroller only by interrupting the power.

The password is stored in the internal 64-byte EEPROM memory of the microcontroller at addresses 0x00 through 0x3F. The memory can be programmed and read by both the device programmer and the CPU when the device is not code protected. It is non-volatile and can retain data for more than 40 years.

Four special-function registers are used to read and write the EEPROM. These registers are named as EECON1, EECON2, EEDATA and EEADR, respectively. Register EEDATA holds 8-bit data for read/write and register EEADR holds the address of the EEPROM location being accessed. Register EECON1 contains the control bits, while register EECON2 is used to initiate the read/write operation.

Oscillator. The internal oscillator circuitry of the microcontroller generates the device clock. The microcontroller can be configured to work in one of the four oscillator modes:

1. External resistor-capacitor
2. Low-power crystal (oscillation frequency up to 200 kHz)
3. Crystal/resonator (oscillation frequency up to 4 MHz)
4. High-speed crystal/resonator (oscillation frequency up to 10 MHz)

In this circuit, the oscillator is configured to operate in crystal mode with a 4MHz crystal along with two 33pF capacitors.

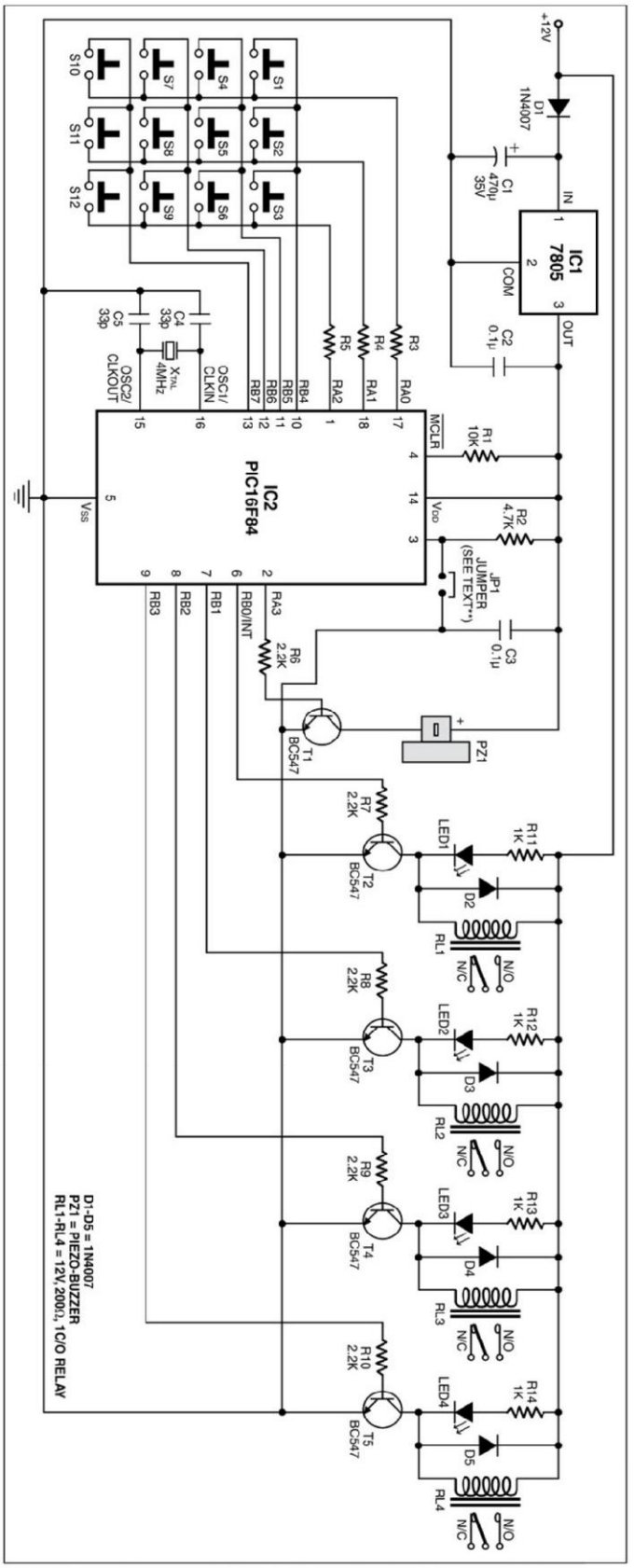


Fig. 3: Circuit diagram of PIC16F84-based coded device switching system

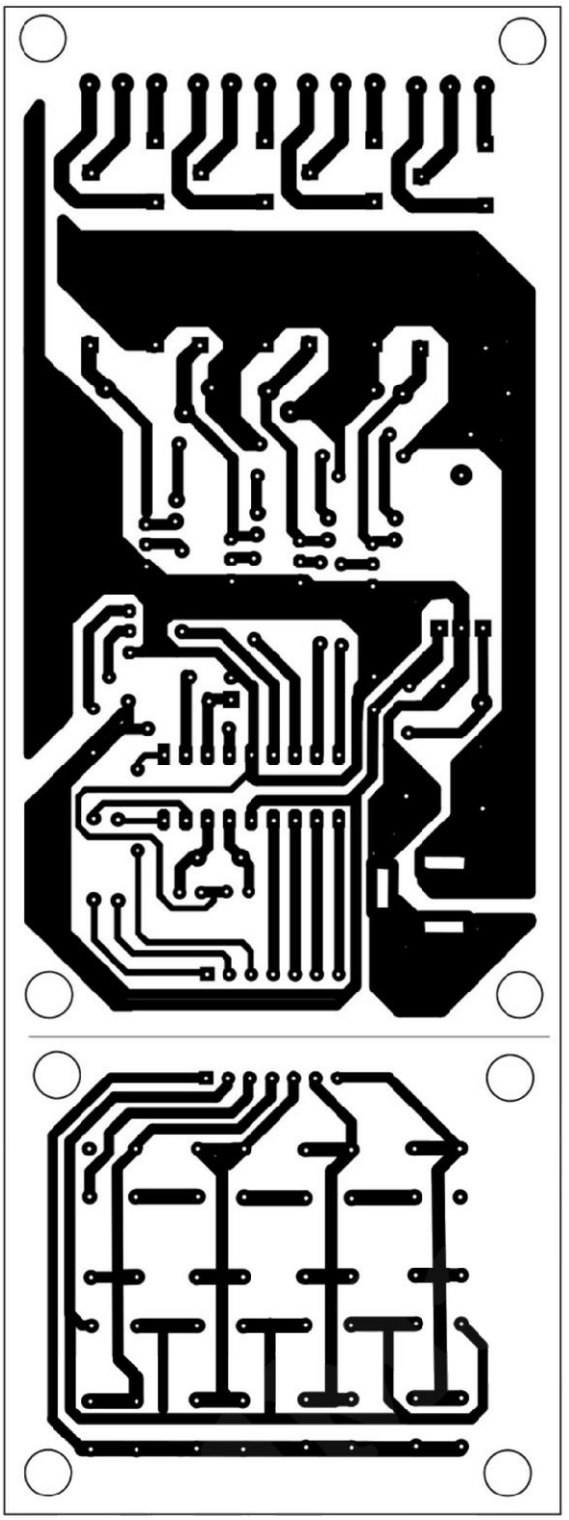


Fig. 4: Actual-size, single-side PCB layout for PIC16F84-based coded device switching system

Reset circuit. The built-in power-on reset circuitry of the microcontroller eliminates the need for the external power-on reset circuit. In the

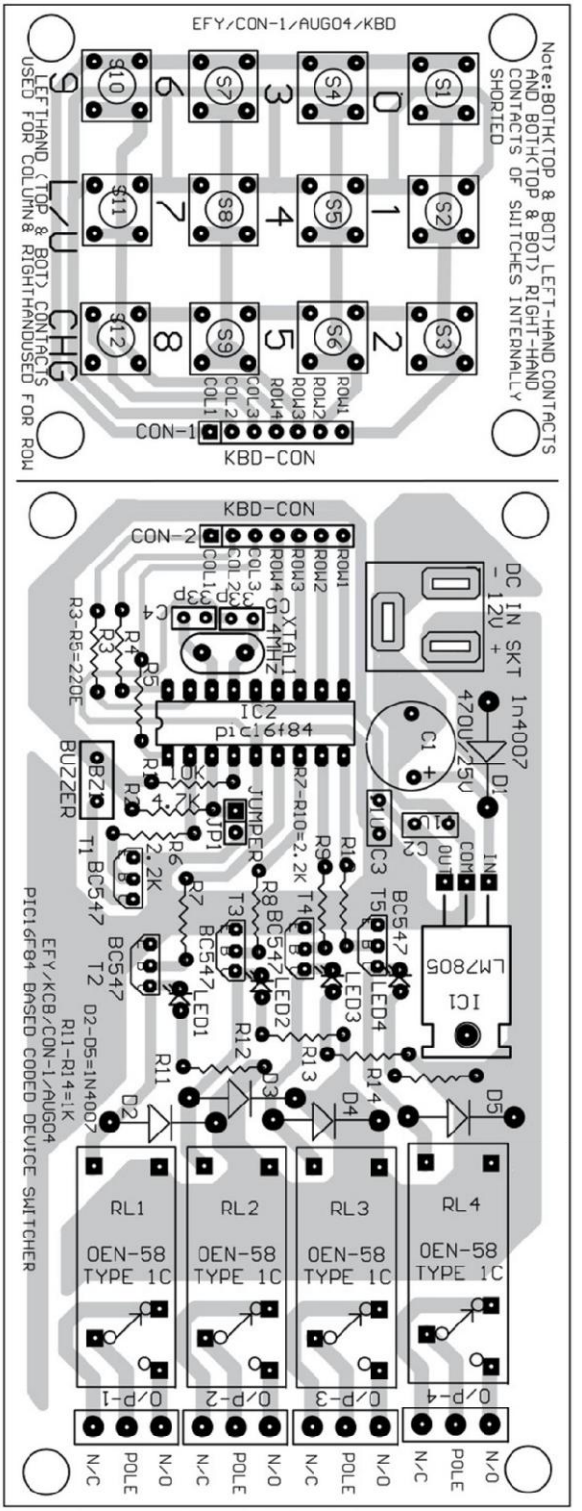


Fig. 5: Component layout for the PCB

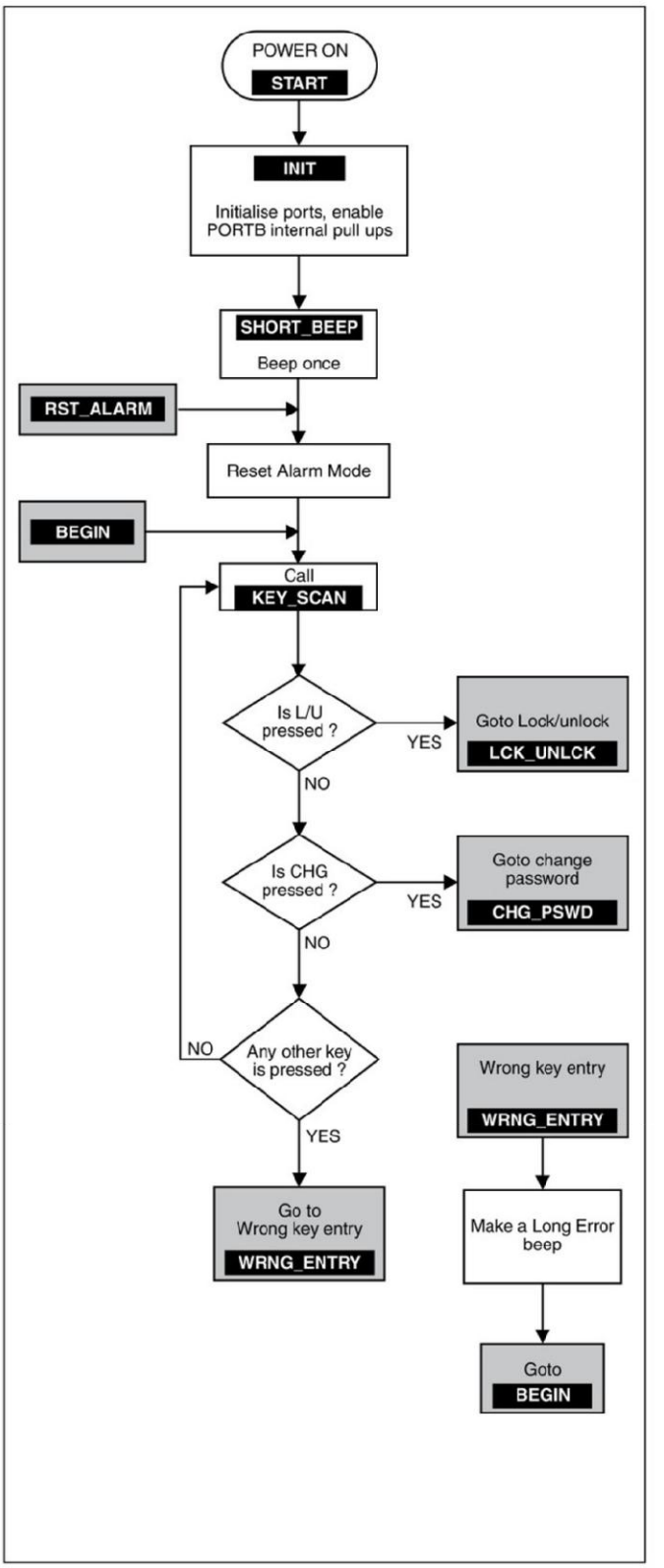


Fig. 6: Flow-chart of the main program

circuit, MCLR pin is tied to V_{DD} through resistor R1 (10 kilo-ohms) to enable power-on reset. The internal power-up timer (PWRT) provides a nominal 72ms delay from power-on reset. This delay allows V_{DD} to rise to an acceptable level when the microcontroller is powered

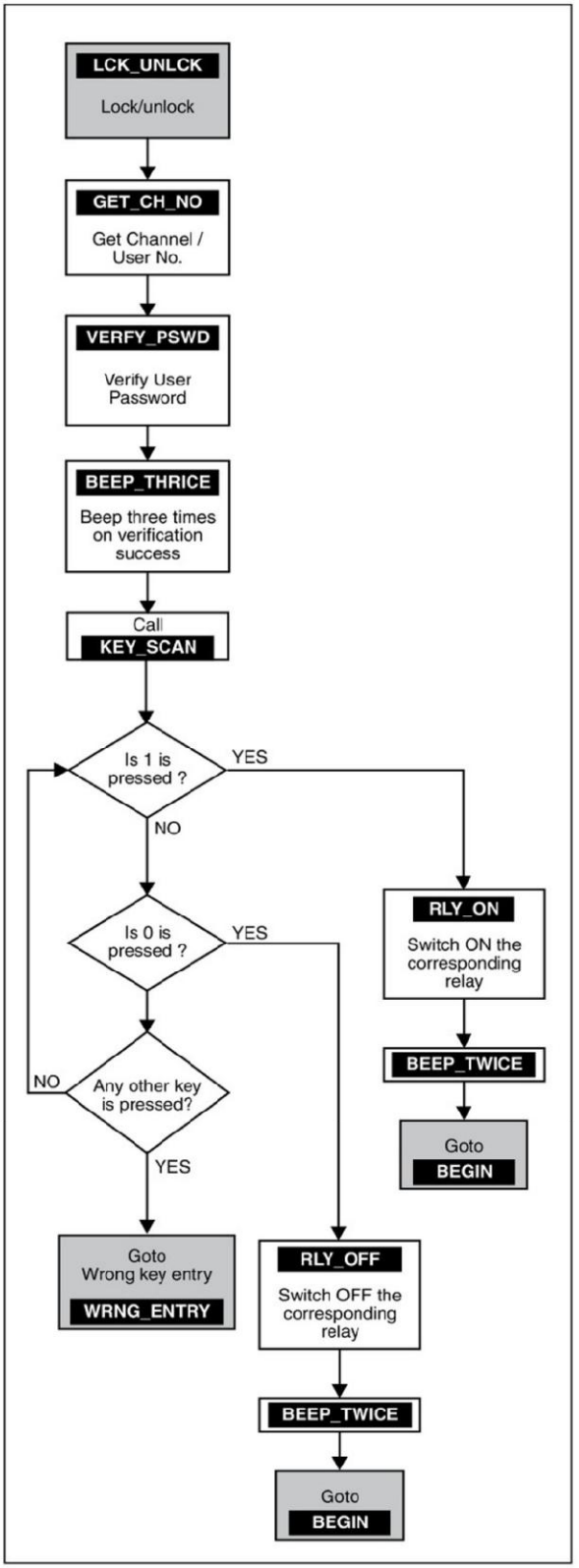


Fig. 6(a): Flow-chart for locking/unlocking the code lock

on. The oscillator start-up timer (OST) provides 1024-oscillator cycle delay after the power-up timer delay is over. This ensures that the crystal oscillator has started and is stable.

Power supply. The 12V DC supply for the circuit is obtained from a 12V adaptor with 500mA rating. Any other source such as a 12V lead-acid battery can also be used. This 12V DC is used for operation of the relays used in the circuit. The regulated +5V supply for the microcontroller is derived using regulator IC 7805 (IC1). Diode D1 protects the circuit from reverse supply connections. Capacitor C1 filters out the ripples present in the incoming DC voltage.

Keyboard. The 12-key matrix keyboard comprises 12 tactile pushbutton switches arranged in four rows and three columns as shown in Fig. 3. Data is entered via this keyboard.

Ports A and B of the microcontroller are bidirectional I/O ports. Three lines of Port A (RA0 through RA2) are used as the output-scan lines and four lines of Port B (RB4 through RB7) are used as the input-sense lines. Port B of IC2 has weak internal pull-ups, which can be enabled through the software. This eliminates the need for connecting external pull-up resistors to pins 10 through 13. Resistors R2 through R4 protect Port A's output drivers from shorting together when two keys of the same row are inadvertantly pressed simultaneously.

In the scanning routine, initially all the scan lines are made low and it is checked whether all the keys are in released state. If all the keys are in released state, the processor is put into sleep (power-down) mode. The interrupt-on-change feature of Port-B pins RB4 through RB7 is used to wake up the processor from sleep.

When any key is pressed, one of the sense lines becomes low. This change in the pin status causes an interrupt to wake up the microcontroller (IC2) from sleep.

Now each scan line is made low while keeping the remaining scan lines in high state. After making a scan line low, the status of the sense lines is read. If any of the sense lines is found low, it means that a key at the intersection of the current scan line and the low sense line has been pressed. If no key is found to be pressed, the next scan line is made low and again scan lines are checked for low state. This way all the twelve keys are checked for any pressed key by the microcontroller.

Since mechanical tactile switch keys are used, pressing of a single key may be considered by the microcontroller as pressing of many keys due to the bouncing of the keys. To avoid this, the processor is made to wait up to a debounce delay of 20 ms during the pressing or releasing of a key. Within this debounce delay, all the bounces get settled out, thus debouncing the key.

In sleep (power-down) mode, the device oscillator is turned off and the microcontroller is placed in its lowest-current consumption state. Also note that the mi-

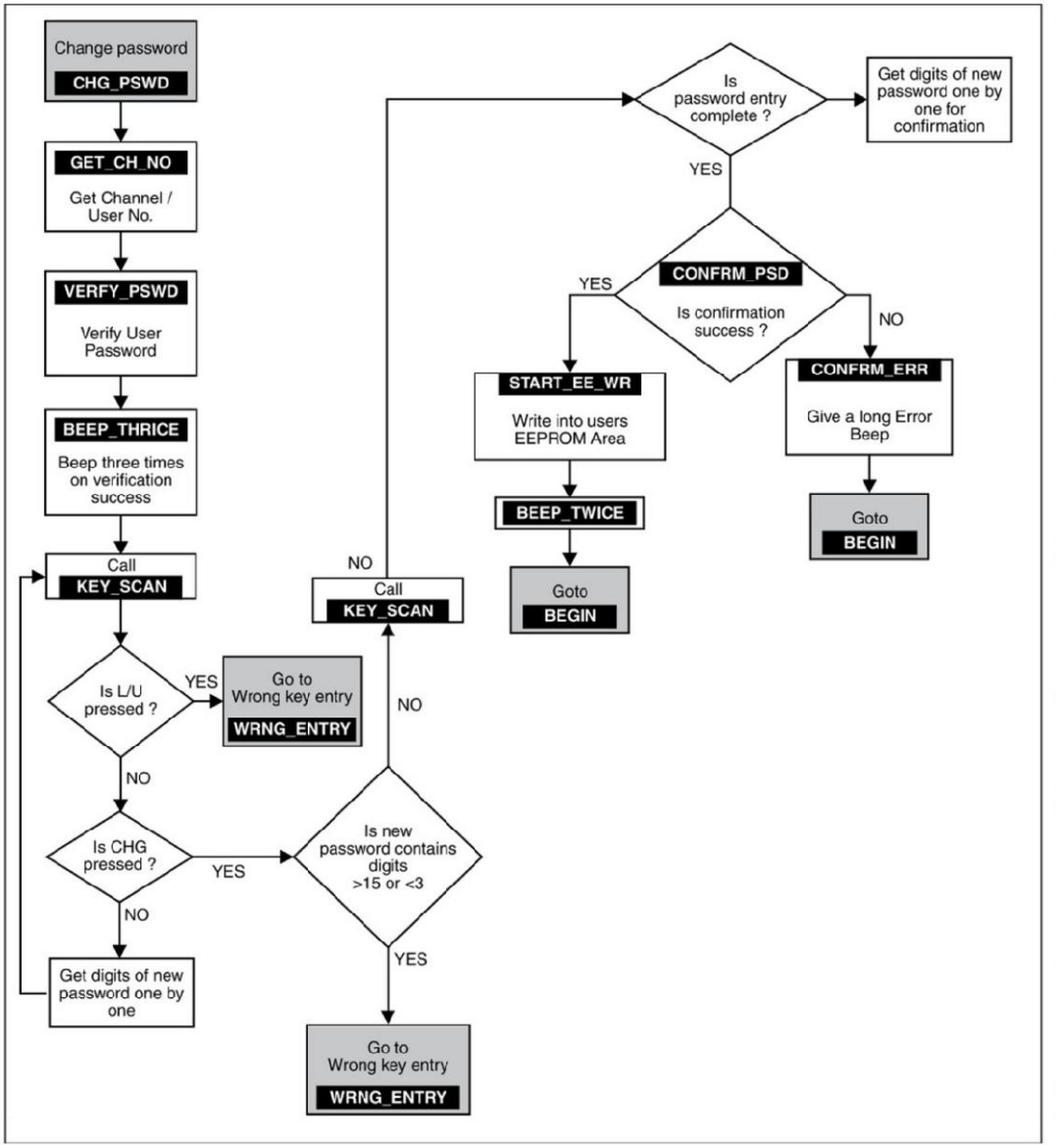


Fig. 6(b): Flow-chart for changing the password of the code lock

crocontroller's I/O pin status remains unaltered during sleep mode.

Relays. To turn on/off the equipment or to lock/unlock the solenoid-operated locks, four relays (RL1 through RL4) are provided—one for each channel. Since the current-driving capacity of the port pins of PIC16F84 (IC2) is not enough to drive the relays directly, transistors T2 through T5 are used to boost the current to drive relays RL1 through RL4, respectively.

The bases of transistors T2 through T5 are connected to Port-B pins 6 through 9 (RB0 through RB3) through base-current-limiting resistors R7 through R10, respectively. The equipment or solenoid-operated locks can be connected to the normally open (N/O) contacts of these relays. Diodes D2 through D5 are used as freewheel clamp diodes. The series combination of a red LED (LED1 through LED4) and a current-limiting resistor (R11 through R14) is connected across each relay coil.

Buzzer. Pin 2 (RA3) of IC2 is connected via resistor R6 and transistor T1 to piezobuzzer PZ1. The buzzer gives a short beep when any key is pressed. In the case of a wrong data entry, the buzzer gives a long beep to indicate the error. On successful password verification, it gives three short beeps, and after successful password change, it gives two short beeps. When a wrong password is entered consecutively for three times, the buzzer sounds an alarm.

Construction and testing

An actual-size, single-side, PCB layout for PIC16F84-based coded device switching system is shown in Fig. 4 and its component layout in Fig. 5.

The main circuit and the matrix keyboard can be assembled on separate PCBs. First check the assembled

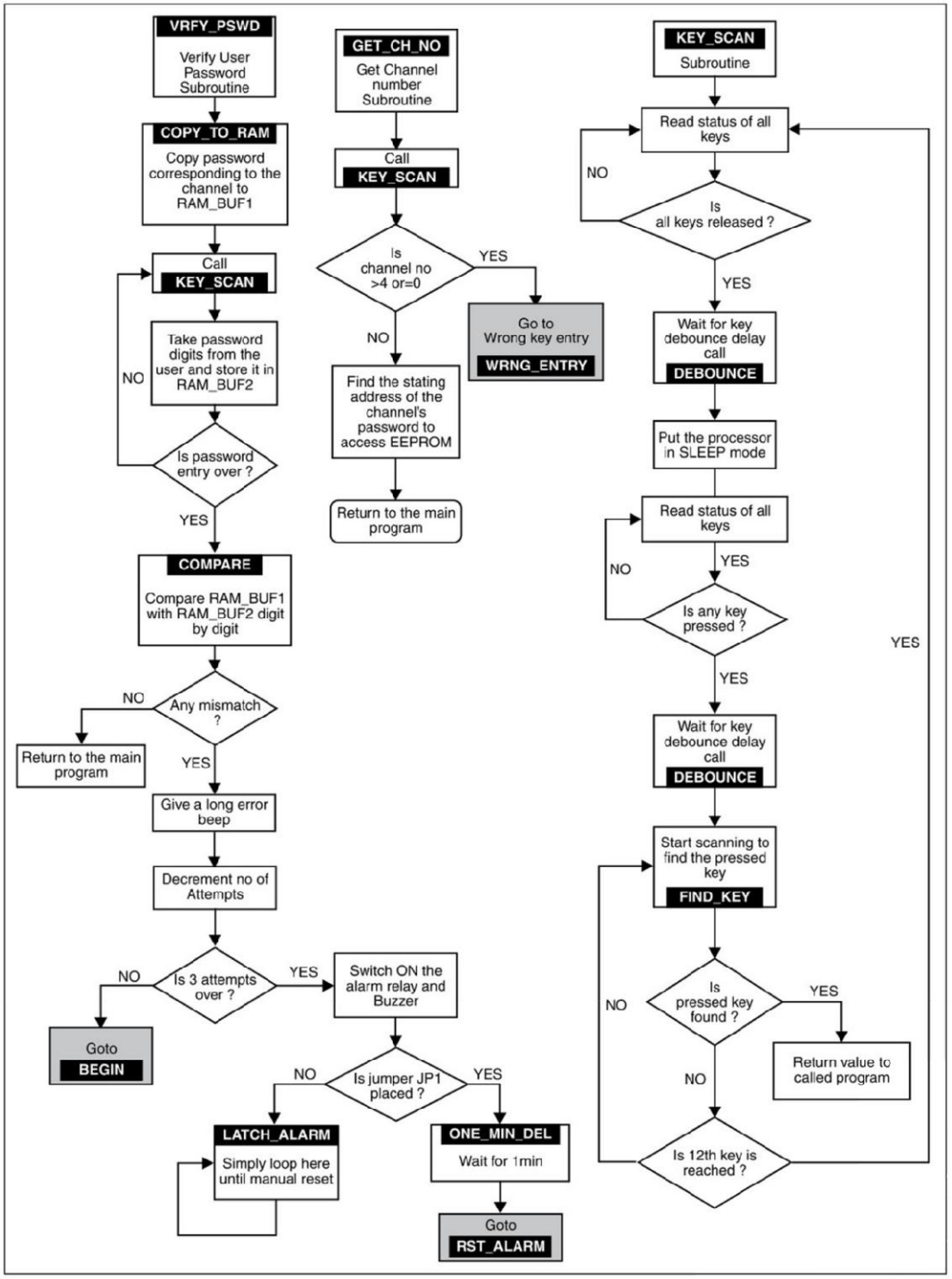


Fig. 6(c): Flow-chart for password verification, device (channel) selection and key scanning

PCBs for proper connections as per the circuit diagram. Then connect the main PCB to the matrix keyboard PCB using 7-pin SIP connectors and wires, ensuring one-to-one connection between the two PCBs. Connect the external 12V DC supply with the correct polarity, without inserting the PIC microcontroller into the socket, and follow these steps:

1. Check whether +5V is available at output pin 3 of regulator IC1 (7805).

2. Now check the availability of +5V at pins 4 and 14 of IC2 before placing IC2 into the socket.

3. To check the buzzer operation, connect pin 2 of IC2 socket to +5V available at pin 3 of IC1. Now the buzzer should beep continuously.

4. Check the operation of the four relays by connecting pins 6 through 9 of IC2 socket one by one to +5V.

5. Before placing jumper JP1, check the voltage at pin 3 of IC2 using a multimeter. The meter should read +5V or logic 1. Now on placing jumper JP1, the meter should read 0V or logic 0 at pin 3.

Now remove the supply and insert the programmed PIC16F84 microcontroller into the socket and switch on the supply. After power-on, the buzzer beeps once to indicate that the microcontroller is ready to take the user

data. Now you can lock/unlock or change the password as described below. Initially the four channels can be accessed using the default password '1234.'

Operating procedure

For unlocking/switching on the equipment:

1. Press the lock/unlock button (L/U) on the keypad.
2. Now enter the device number by pressing the button corresponding to the device number. The valid device numbers are 1 to 4. For example, if you want to access device No. 1 (RL1), press button '1.'
3. Now enter your password digits one by one. Note that the default password is '1234.'
4. The buzzer gives three short beeps to indicate successful verification of the password. If the entered password is incorrect, the buzzer gives a long beep to indicate error. To try again, repeat the procedure from step 1.
5. If the entered password is correct, you can unlock or switch on device No. 1 by pressing button '1.' When you press the key, the relay corresponding to this device gets energised and it remains in this state until you lock/switch it off again.

For locking/switching off the equipment:

Follow the aforesaid steps 1 through 4 and press button '0.' Now the relay corresponding to the device you want to turn off de-energises and it remains in this state until you unlock/switch it on again.

For changing the password:

1. Press the password change button (CHG) on the keypad.
2. Now press the device number.
3. Enter your current password.
4. On successful verification of the password, the buzzer gives three short beeps. If the entered password is wrong, the buzzer will give a long beep. Now if you want to try again, repeat the procedure from step 1.
5. Enter your new password. The length of the password should be between 4 and 15 digits.
6. End the password entry by pressing again CHG button.
7. Again enter your new password for confirmation. On successful confirmation, your new password gets replaced by the old password and the buzzer beeps twice to indicate successful password change. In case the password entered for confirmation is wrong, the buzzer gives a long beep to indicate error and the old password remains unaltered.

So whether you're locking, unlocking or changing the device, wrong password entry makes the buzzer to give a long error beep and the users are required to start afresh from step 1. In case you forget the password of the device, it can't be controlled until you reprogram the microcontroller.

Mode of operation. When anyone fails to enter the correct password in three attempts, the code lock circuit switches to alarm mode and the buzzer starts beeping continuously. All the keys pressed (for further attempts) are ignored by the code lock during alarm mode.

Placing the jumper between pin 3 (RA4) of IC2 and Ground enables the auto-reset alarm mode. Whereas removing the jumper enables the latch-up mode (see Fig. 3). If the auto-rest alarm mode is enabled, the code lock automatically resets after about one minute. If the latch-up alarm mode is enabled, the code lock never resets from the alarm mode until the user manually resets it by interrupting the power. Note that in the alarm mode the status of device-controlling relays remains unaltered.

Software

The software is written in Microchip's Assembly language. Fig. 6 shows the flow-chart for the program. In the flow-chart, important labels and subroutine names used in the program are also mentioned within the corresponding process boxes to enable easy understanding of the program. For instructions, you may refer to the PIC16F84 datasheet. The code is compiled and hex file is generated using MPLAB IDE. You can generate the hex file by using the MPASM.exe assembler also. The hex file generated can be burnt into the microcontroller using any PIC programmer that supports PIC16F84. We've used here PICburner to program the PIC. It was published in EFY's Sept. 2002 issue.

Download Source Code: http://www.efymag.com/admin/issuepdf/PIC16F84%20Device%20Switching.zip

CODLOCK.LST

```
MPASM 03.20 Released     CODLOCK.ASM   7-1-2004  16:25:54       PAGE  1

LOC  OBJECT CODE     LINE  SOURCE TEXT
  VALUE

                     00001 ;*****************************
                     00002 ;
                     00003 ;TITLE: "MICROCONTROLLER BASED 4
                                 CHANNEL CODE LOCK"
                     00004 ;PROCESSOR PIC16F84
                     00005 ;Oscillator:XT 4MHz crystal Oscillator
                     00006 ; Default password:1234 for ch1 - ch4
                     00007 ;
                     00008 ; Author:VIJAYA KUMAR.P
                     00009 ; EMAIL:vijay_kum_p@yahoo.co.in
                     00010 ;
                     00011 ;************************************
                     00012
                     00013 ;------------------------------------
                     00014
                     00015    #INCLUDE "p16f84.inc" ;Header file
                               inclusion directive.
                     00001     LIST
                     00002 ; P16F84.INC  Standard Header File,
                            Version 2.00 Microchip Technology,
                            Inc.
                     00136       LIST
                     00016
                     00017
                     00018 ; NOTE: This header file consists of
                            definations of all special function
                     00019 ; registers (SFRs) and their associated
                            bits.
                     00020
                     00021 ;------------------------------------
                     00022
                     00023  ;*********Configuration  bit  set-
tings*******
                     00024
                     00025  LIST P=PIC16F84  ;processor type
                                              PIC16F84A
                     00026
2007 0001            00027   __CONFIG  _XT_OSC  &_PWRTE_ON    &
                            _CP_ON & _WDT_OFF
                     00028
                     00029 ; SETTING : XT oscillator mode,power up
                            timer ON, code protect on,watch dog
                     00030 ; timer OFF
                     00031
                     00032 ;------------------------------------
                     00033 ; Defining Default  password. First time
                            after programming 16f84 you need
                     00034 ; to use default password 1234 for all
                            4 channels.
                     00035 ;------------------------------------
                     00036
2100                 00037   ORG  0X2100 ;Starting adderss of
                            ch1's password
2100 0001 0002 0003  00038   DE 1,2,3,4 ;default password for
                            ch 1
     0004
210F                 00039   ORG 0X210F
210F 0004            00040   DE D'04' ;Default password length =
                            4 digits
                     00041
2110                 00042   ORG  0X2110 ;Starting adderss of
                            ch2's password
2110 0001 0002 0003  00043   DE 1,2,3,4 ;Default password for
                            ch 2
     0004
211F                 00044   ORG 0X211F
211F 0004            00045   DE D'04' ;Default password length=4
                            digits
                     00046
2120                 00047   ORG 0X2120 ;Starting adderss of ch3's
                            password
2120 0001 0002 0003  00048   DE 1,2,3,4 ;Default password for
                            ch 3
     0004
212F                 00049   ORG 0X212F
212F 0004            00050   DE D'04' ;Default password length=4
                            digits
                     00051
2130                 00052   ORG 0X2130 ;Starting adderss of ch4's
                            password
2130 0001 0002 0003  00053   DE 1,2,3,4 ;Default password for
                            ch 4
     0004
213F                 00054   ORG 0X213F
213F 0004            00055   DE D'04' ;Default password length=4
                            digits
                     00056
                     00057
                     00058 ;************************************
                     00059   ;VARIABLE AND CONSTANT DATA
                             DECLARATIONS
                     00060
                     00061
                     00062 ; variables
                     00063
0000000C             00064   DEL_COUNT1 EQU  0X0C ;Counters used
                             to obtain software delay.
0000000D             00065   DEL_COUNT2 EQU  0X0D
0000000E             00066   DEL_COUNT3 EQU  0X0E
0000000F             00067   KEY_IN   EQU  0X0F
                             ;Holds the value of pressed key.
00000010             00068   KEY_NO   EQU  0X10
                             ;Holds key no.
00000011             00069   SCAN_CODE EQU  0X11  ;Holds scan
                             code.
00000012             00070   KB_TEMP  EQU  0X12  ;Temporary
                             variable to hold key value
00000013             00071   RAM_BUF1_PNT EQU  0X13  ;Pointer reg
                             to RAM_BUF1
00000014             00072   RAM_BUF2_PNT  EQU  0X14  ;Pointer reg
                             to RAM_BUF2
00000015             00073   DIGIT_COUNT  EQU  0X15 ;Holds no of
                             digits
00000016             00074   PSD_DIGIT EQU 0X16  ;Holds password
                             digit
00000017             00075   NO_OF_ATTEMPTS EQU 0X17 ;Holds no
                             of attempts
00000018             00076   CH_NO  EQU 0X18 ;Holds channel/user
                             no
00000019             00077   EEADDR_TEMP EQU 0X19 ;Temporary
                             store to hold  EEPROM addr
00000020             00078   NO_OF_BEEPS EQU 0X20 ;Holds the
                             number of beeps
00000021             00079   BUZ_DEL_CNT EQU 0X21 ;Counters used
                             to obtain 1min delay
00000022             00080   TEN_SEC_CNT EQU 0X22
00000023             00081   ONE_MIN_CNT EQU 0X23
00000024             00082   NO_OF_DIGITS EQU 0X24 ;No of digits
                             in a password
                     00083
                     00084 ; constant data declarations
                     00085
00000030             00086   RAM_BUF1  EQU 0X30 ;Starting address
                             of RAM_BUF1
00000040             00087   RAM_BUF2  EQU 0X40  ;Starting address
                             of RAM_BUF2
                     00088
                     00089
                     00090 ;************************************
                     00091 ; program starts from here as soon as
                             you switch on the code lock circuit.
                     00092
0000                 00093   ORG 0X0000  ;Reset vector
0000                 2823 00094    GOTO START
                     00095
                     00096 ;************************************
                     00097 ; Interrupt service routine ISR for
                             timer0 starts from here.
                     00098 ; This  ISR  is  encountered  for every
50ms.
```

```
                        00099 ; NOTE:This ISR is used only to obtain
                              1 minute delay.
                        00100
0004                    00101    ORG 0X0004  ;Interrupt vector
0004            138B    00102    BCF INTCON,GIE ;Dissable all inter-
upts
0005 1D0B               00103   BTFSS INTCON,T0IF ;Is T0IF ==1?
0006 0009               00104   RETFIE   ;If No return form ISR
0007 110B               00105   BCF INTCON,T0IF ;If YES clear it
0008 0BA2               00106   DECFSZ TEN_SEC_CNT,F ;Decrement
                              TEN_SEC_CNT and test if 0
0009 280D               00107   GOTO LOAD_TMR0 ;If !0 goto
                              LOAD_TMR0,if 0,
000A 0BA3               00108   DECFSZ ONE_MIN_CNT,F ;Decrement
                              ONE_MIN_CNT and test if 0
000B 2810               00109   GOTO  LOAD_TEN_SEC ;If !0 goto
                              LOAD_TENS_SEC
000C 2825               00110   GOTO RST_ALARM ;If 0 goto RST_ALARM
                        00111
000D 303F               00112   LOAD_TMR0 MOVLW 0X3F  ;Count for
                              50ms
000E 0081               00113   MOVWF TMR0
000F 0009               00114    RETFIE
                        00115
0010 30C8               00116   LOAD_TEN_SEC MOVLW 0XC8  ;Count for
                              10sec
0011 00A2               00117    MOVWF TEN_SEC_CNT
0012 0009               00118    RETFIE
                        00119
                        00120 ;*************************************
                        00121 ; INITIALISATION SUBROUTINE
                        00122
                        00123 ; This part of the program intialises
                              the required ports and SFRs.
                        00124
0013 0183               00125   INIT  CLRF STATUS  ;Switch to bank0
0014 0185               00126   CLRF    PORTA  ;Clear PORTA
0015 0186               00127   CLRF  PORTB  ;Clear PORTB
0016 1683               00128   BSF  STATUS,RP0 ;Switch to bankl
0017 30F0               00129   MOVLW  B'11110000'  ;Sets pins of
                              portb as iiiioooo
0018 0086               00130   MOVWF  TRISB  ;Where i=input &
                              o=output
0019 3010               00131   MOVLW  B'00010000' ;Sets pins of
                              porta as oooioooo
001A 0085               00132   MOVWF  TRISA
001B 3007               00133   MOVLW 0X07  ;Enable weak internal
                              pull ups,
001C 0081               00134   MOVWF  OPTION_REG ;asigns prescalar
                              to TMR0 with
                        00135 ; 1:256 ratio.
001D 1283               00136   BCF  STATUS,RP0  ;Switch to bank 0
001E 158B               00137   BSF  INTCON,RBIE ;Enable portb int on
                              change
001F 138B               00138   BCF  INTCON,GIE ;Dissable all the
                              interrupts
0020 3003               00139   MOVLW  0X03  ;Max no of atempts = 3
0021 0097               00140   MOVWF  NO_OF_ATTEMPTS
0022 0008               00141   RETURN   ;Return from sub routine
                        00142
                        00143 ;*************************************
                        00144 ; The main program starts from here
                        00145
0023 2013               00146   START    CALL  INIT  ;Call
                              initalization subroutine
0024 216C               00147   CALL  SHORT_BEEP ;Now the buzzer
                              beeps once
                        00148
0025 1185               00149   RST_ALARM BCF PORTA,3  ;Switch off
                              buzzer
                        00150
                        00151 ; here the program waits until L/U or
                              CHG key is pressed.
                        00152
0026 2033               00153   BEGIN  CALL KEY_SCAN ;Call kb
                              scanning routine
0027 0092               00154   MOVWF KB_TEMP  ;W -->KB_TEMP
0028 3A0A               00155   XORLW 0X0A   ;W XOR H'0A' -->W
0029 1903               00156   BTFSC STATUS,Z ;Is L/U key is
                              pressed ?
002A 2875               00157   GOTO  LCK_UNLCK ;If yes goto
                              LCK_UNLCK
002B 0812               00158   MOVF  KB_TEMP,W ;KB_TEMP -->W
002C 3A0B               00159   XORLW 0X0B   ;W XOR 0B -->W
002D 1903               00160   BTFSC  STATUS,Z ;Else Is CHG key is
                              pressed ?
002E 28FB               00161   GOTO  CHG_PSWD ;If yes goto
                              CHG_PSWD
002F 2831               00162   GOTO    WRNG_ENTRY
                              ;Give a long error beep on wrng key
0030 2826               00163   GOTO  BEGIN  ;Else simply LOOP_HERE
                        00164
                        00165 ;*************************************
                        00166 ; the program control comes here when
                              any wrong data entry is made.
                        00167
0031 2172               00168   WRNG_ENTRY CALL  LONG_BEEP
0032 2826               00169   GOTO BEGIN
                        00170
                        00171 ;*************************************
                        00172 ; KEYBOARD SCANING ROUTINE
                        00173 ;
                        00174 ; This subroutine when called returns
                              the value of key pressed in
                        00175 ; w register and makes the buzzer to
                              beep once for every key press.
                        00176 ; This routine uses the wake up on key
                              press feature and reduces power
                        00177 ; consumption by the PIC while not in
                              use.
                        00178 ;*************************************
                        00179
0033                    00180   KEY_SCAN
0033  3010               00181   KEY_RELEASE  MOVLW B'00010000'
                              ;Clearing PORTA pins but
0034  0585               00182   ANDWF PORTA,F  ;Retaining the RA4
                              status
0035  0806               00183   MOVF  PORTB,W  ;Read PORTB into W
                              reg
0036  39F0               00184   ANDLW  B'11110000' ;Mask the lower
                              nibble
0037  3AF0               0185    XORLW  B'11110000' ;W Xor 11110000 --
                              >W
0038  1D03               00186   BTFSS STATUS,Z ;Is all keys are
                              released ?
0039  2833               00187   GOTO  KEY_RELEASE ;If not goto
                              KEY_RELEASE
003A  206C               00188   CALL  DEBOUNCE ;If yes debounce the
                              key
003B  0806               00189   MOVF PORTB,W  ;Clear previous
                              mismatch condition
003C  100B               00190   BCF INTCON,RBIF ;Clear RBIF
003D  0063               00191   SLEEP   ;Put the processor in Sleep
                              mode
                        00192
                        00193
003E  3010               00194   ANY_KEY  MOVLW B'00010000' ;Clearing
                              PORTA pins but
003F  0585               00195   ANDWF PORTA,F  ;Retaining the RA4
                              status
0040  0806               00196   MOVF PORTB,W  ;PORTB -->W reg
0041  3AFF               00197   XORLW 0XFF  ;W XOR 0XFF -->W reg
0042  1903               00198   BTFSC STATUS,Z ;Is any key
                              pressed ?
0043  283E               00199   GOTO ANY_KEY  ;If no goto ANY_KEY
0044  206C               00200   CALL DEBOUNCE ;If yes debounce the
                              key
0045  3000               00201   MOVLW  0X00
0046  0090               00202   MOVWF  KEY_NO ;Initialise  KEY_NO
                              to 0
                        00203
0047 3010               00204   FIND_KEY MOVLW B'00010000'
0048 0585               00205   ANDWF PORTA,F  ;Retaining the RA4
                              status
0049 205E               00206   CALL  SCAN_TABLE ;Get the scan code
004A 0091               00207   MOVWF  SCAN_CODE  ;Move
                              SCAN_CODE to W reg
004B 3907               00208   ANDLW  B'00000111' ;Mask 5 MSB's
004C 0485               00209   IORWF  PORTA,F  ;w --> porta while
                        00210 ; Retaining the RA4 status
```

```
004D 0806               00211   MOVF  PORTB,W  ;Read PORTB to W reg
004E 39F0               00212   ANDLW  B'11110000'  ;Mask the lower
                                nibble of PORTB
004F 008F               00213   MOVWF  KEY_IN  ;Move the key value to
                                key_in
0050 0811               00214   MOVF  SCAN_CODE,W ;SCAN_CODE -->
                                W reg
0051 39F0               00215   ANDLW  B'11110000' ;Mask lower nibble
                                of scan code
0052 060F               00216   XORWF  KEY_IN,W ;compare read key
                                with scan code
0053 1903               00217   BTFSC  STATUS,Z ;Test for Z flag
0054 285B               00218   GOTO  RET  ;If Z=1 goto RET else
                                continue
0055 0A90               00219   INCF KEY_NO,F ;Increment key no
0056 0810               00220   MOVF  KEY_NO,W ;KEY_NO -->W REG
0057 3C0C               00221   SUBLW  0X0C  ; W - 12 -->W
0058 1D03               00222   BTFSS  STATUS,Z ;Test whether key
                                no=12th key
0059 2847               00223   GOTO  FIND_KEY ;If no goto FIND_KEY
005A 2833               00224   GOTO  KEY_SCAN ;If yes goto start new
                                scan
005B 216C               00225   RET  CALL    SHORT_BEEP ;Now the
                                buzzer will beep once
005C 0810               00226   MOVF  KEY_NO,W ;Pressed Key no-->w
005D 0008               00227   RETURN   ;Return from key scan
                        00228
                        00229
                        00230 ;*************************************
                        00231 ; LOOK UP TABLE FOR KEY CODE
                        00232 ; This look up table is used by the
                                keyboard
                                scan subroutine and look up
                        00233 ; table returns the scancode in w
                                register
                                when called by placing key number
                        00234 ; in KEY_NO
                        00235 ;*************************************
                        00236
005E 0810               00237   SCAN_TABLE MOVF  KEY_NO,W ;KEY_NO
                                -->W reg
005F 0782               00238   ADDWF  PCL,F  ;PCL+W -->PCL reg
                        00239
0060 34E6               00240   RETLW  B'11100110'  ;Scan code for
                                key0
0061 34E5               00241   RETLW  B'11100101' ;Scan code for
                                key1
0062 34E3               00242   RETLW  B'11100011' ;Scan code for
                                key2
0063 34D6               00243   RETLW  B'11010110' ;Scan code for
                                key3
0064 34D5               00244   RETLW  B'11010101' ;Scan code for
                                key4
0065 34D3               00245   RETLW  B'11010011' ;Scan code for
                                key5
0066 34B6               00246   RETLW  B'10110110' ;Scan code for
                                key6
0067 34B5               00247   RETLW  B'10110101' ;Scan code for
                                key7
0068 34B3               00248   RETLW  B'10110011' ;Scan code for
                                key8
0069 3476               00249   RETLW  B'01110110' ;Scan code for
                                key9
006A 3475               00250   RETLW B'01110101' ;Scan code for
                                L/U key
006B 3473               00251   RETLW B'01110011' ;Scan code for
                                CHG key
                        00252
                        00253 ;*************************************
                        00254 ; DELAY FOR DEBOUNCING THE KEY
                        00255 ; This delay routine produces a key
                                board debounce delay of 20ms
                        00256 ;*************************************
                        00257
006C 301C               00258   DEBOUNCE  MOVLW  0X1C
006D 008D               00259   MOVWF  DEL_COUNT2
006E 30F0               00260   KB_DLOOP1  MOVLW  0XF0
006F 008C               00261   MOVWF  DEL_COUNT1
0070 0B8C               00262   KB_DLOOP  DECFSZ DEL_COUNT1,F
0071 2870               00263   GOTO  KB_DLOOP
0072 0B8D               00264   DECFSZ  DEL_COUNT2,F
0073 286E               00265   GOTO  KB_DLOOP1
0074 0008               00266   RETURN
                        00267
                        00268 ;*************************************
                        00269 ; ROUTINE FOR LOCKING /UNLOCKING
                        00270 ; When you press L/U key the program
                                control comes here.
                        00271 ;*************************************
                        00272
0075 20A1               00273   LCK_UNLCK CALL  GET_CH_NO ;Get
                                channel/user no
0076 20B4               00274   CALL  VRFY_PASWD ;Call verify
                                password
                                subroutine
0077 0398               00275   DECF CH_NO,F  ;Decrement CH_NO
0078 0818               00276   MOVF CH_NO,W  ;CH_NO -->W reg
0079 0798               00277   ADDWF CH_NO,F  ;CH_NO x 2 -->CH_NO
007A 3003               00278   MOVLW 0X03  ;Reset no_of_attempts
                                to 3
007B 0097               00279   MOVWF NO_OF_ATTEMPTS
007C 217D               00280   CALL BEEP_THRICE ;Now the buzzer will
                                beep 3 times
                        00281
007D 2033               00282   SWITCH_RELAY CALL KEY_SCAN ;Call
                                Key scan subroutine
007E 0092               00283   MOVWF KB_TEMP  ;Store the key val in
                                KB_TEMP
007F 3A01               00284   XORLW 0X01
0080 1903               00285   BTFSC STATUS,Z ;Is key 1 is
                                pressed ?
0081 2887               00286   GOTO RLY_ON  ;If yes goto RLY_ON
0082 0812               00287   MOVF KB_TEMP,W
0083 3A00               00288   XORLW 0X00
0084 1903               00289   BTFSC STATUS,Z ;Is key 0 is
                                pressed ?
0085 288A               00290   GOTO RLY_OFF  ;If yes goto RLY_OFF
0086 2831               00291   GOTO WRNG_ENTRY ;If no goto
                                WRNG_ENTRY
0087 208D               00292   RLY_ON  CALL RLY_ON_TBL ;Call
                                RLY_ON table
0088 2178               00293   CALL BEEP_TWICE ;Now the buzzer will
                                beep twice
0089 2826               00294   GOTO BEGIN  ;Goto BEGIN
                        00295
008A 2097               00296   RLY_OFF  CALL RLY_OFF_TBL ;Call
                                RLY_OFF table
008B 2178               00297   CALL BEEP_TWICE ;Now the buzzer will
                                beep twice
008C 2826               00298   GOTO BEGIN  ;Goto BEGIN
                        00299
                        00300 ;*************************************
                        00301 ; RELAY_ON_TABLE
                        00302 ;*************************************
                        00303
008D 0818               00304   RLY_ON_TBL MOVF CH_NO,W
008E 0782               00305   ADDWF PCL,F
008F 1406               00306   BSF PORTB,0  ;Switches ON ch1's relay
0090 0008               00307   RETURN
0091 1486               00308   BSF PORTB,1  ;Switches ON ch2's relay
0092 0008               00309   RETURN
0093 1506               00310   BSF PORTB,2  ;Switches ON ch3's relay
0094 0008               00311   RETURN
0095 1586               00312   BSF PORTB,3  ;Switches ON ch4's relay
0096 0008               00313   RETURN
                        00314
                        00315 ;*************************************
                        00316 ; RELAY_OFF_TABLE
                        00317 ;*************************************
                        00318
0097 0818               00319   RLY_OFF_TBL MOVF CH_NO,W
0098 0782               00320   ADDWF PCL,F
0099 1006               00321   BCF PORTB,0  ;Switches OFF ch1's
                                relay
009A 0008               00322   RETURN
009B 1086               00323   BCF PORTB,1  ;Switches OFF ch2's
                                relay
009C 0008               00324   RETURN
009D 1106               00325   BCF PORTB,2  ;Switches OFF ch3's
                                relay
```

```
009E 0008                00326   RETURN
009F 1186                00327   BCF PORTB,3  ;Switches OFF ch4's
                                 relay
00A0 0008                00328   RETURN
                         00329
                         00330 ;*************************************
                         00331 ; This sub routine is used to take
                                 channel/user number and it also finds
                                 the staring
                         00332 ; address of ch's/user's password
                                 stored in EEPROM using Lookup table
                                 and places it
                         00333 ; in EEADDR_TEMP. This address will be
                                 used by COPY_TO_RAM subroutine.
                         00334 ;
                         00335 ;*************************************
                         00336
00A1 2033                00337   GET_CH_NO CALL  KEY_SCAN ;Ch/user
                                 no -->W
00A2 0098                00338   MOVWF CH_NO  ;[W] --> CH_NO
00A3 3A00                00339   XORLW  0X00
00A4 1903                00340   BTFSC  STATUS,Z ;Is entered key is
                                 0 ?
00A5 2831                00341   GOTO WRNG_ENTRY ;If yes WRNG_ENTRY
00A6 0818                00342   MOVF  CH_NO,W  ;If no CH_NO -->W
00A7 3C04                00343   SUBLW  0X04  ;Is entered key > 4 ?
00A8 1C03                00344   BTFSS  STATUS,C
00A9 2831                00345   GOTO  WRNG_ENTRY ;If YES goto
                                 WRNG_ENTRY
00AA 20AD                00346   CALL  EEADDR_LOOKUP ;If no CALL
                                 EEADDR look up table
00AB 0099                00347   MOVWF EEADDR_TEMP ;[W]
                                 -->EEADDR_TEMP
00AC 0008                00348   RETURN
                         00349
                         00350 ;*************************************
                         00351 ; LOOK UP TABLE FOR EEADDRESS
                         00352 ; This Lookup table returns the staring
                                 address of the ch's/user's password
                                 in
                         00353 ; EEPROM data memory when the channel/
                                 user number is passed into it.
                         00354 ;*************************************
                         00355
00AD 0818                00356   EEADDR_LOOKUP MOVF  CH_NO,W
00AE 0782                00357   ADDWF  PCL,F
00AF 0008                00358   RETURN
00B0 3400                00359   RETLW 0X00  ;Starting address of
                                 ch1's Password
00B1 3410                00360   RETLW  0X10  ;Starting address of
                                 ch2's Password
00B2 3420                00361   RETLW 0X20  ;Starting address of
                                 ch3's Password
00B3 3430                00362   RETLW 0X30  ;Starting address of
                                 ch4's Password
                         00363
                         00364 ;*************************************
                         00365 ;
                         00366 ; SUBROUTINE TO VERIFY PASSWARD
                         00367 ;
                         00368 ; This subroutine copies the passward
                                 saved in EEPROM into RAM_BUF1 then
                                 reads the
                         00369 ; password digits entered by the user
                                 and stores into RAM_BUF2 then
                                 compares
                         00370 ; RAM_BUF1 with RAM_BUF2 digit by
                                 digit.
                         00371 ; Returns to the called program if the
                                 match occures for all the digits. On
                                 mismatch it
                         00372 ; gives an long error beep and
                                 decrements the NO_OF_ATTEMPTS by one.
                                 If
                         00373 ; NO_OF_ATTEMPTS == 0 switches the
                                 code lock into alarm mode. and
                                 further
                         00374 ; key presses will be ignored.The
                                 codelock comes to the normal working
                                 after 1 minute.
                         00375 ; NOTE:the NO_OF_ATTEMPTS will not be
                         00376 ; decremented if the jumper is placed
                         00377 ; between RA4 and Gnd and hence will
                                 not switch into the alarm mode.
                         00378 ;*************************************
                         00379
00B4 20EB                00380   VRFY_PASWD CALL  COPY_TO_RAM ;Call
                                 COPY_TO_RAM sub routine
00B5 3030                00381   MOVLW  RAM_BUF1
00B6 3E0F                00382   ADDLW 0X0F  ;Initialize FSR to
00B7 0084                00383   MOVWF  FSR  ;the end of RAM_BUF1
00B8 0800                00384   MOVF  INDF,W  ;[INDF] -->W
00B9 00A4                00385   MOVWF  NO_OF_DIGITS ;[W]
                                 -->NO_OF_DIGITS
00BA 0095                00386   MOVWF  DIGIT_COUNT ;[W]
                                 -->DIGIT_COUNT
00BB 3040                00387   MOVLW  RAM_BUF2 ;Initialise FSR to
00BC 0084                00388   MOVWF  FSR  ;the starting of RAM_BUF2
                         00389
00BD 2033                00390   SCAN_NXT_BYTE CALL KEY_SCAN ;Call
                                 scan key routine
00BE 0080                00391   MOVWF INDF;[W]-->INDF
00BF 3C09                00392   SUBLW 0X09
00C0 1C03                00393   BTFSS STATUS,C ;Is L/U or CHG key
                                 pressed ?
00C1 2831                00394   GOTO  WRNG_ENTRY ;If yes goto
                                 WRNG_ENTRY
00C2 0A84                00395   INCF  FSR,F  ;Increment FSR by 1
00C3 0B95                00396   DECFSZ  DIGIT_COUNT,F ;Decrement
                                 DIGIT_COUNT by one,is it 0?
00C4 28BD                00397   GOTO  SCAN_NXT_BYTE ;If no go back to
                                 SCAN_NXT_BYTE
                         00398
                         00399
00C5 3030                00400   COMPARE  MOVLW  RAM_BUF1
                                 ;RAM_BUF1 pointer initialisation
00C6 0093                00401   MOVWF  RAM_BUF1_PNT
00C7 3040                00402   MOVLW  RAM_BUF2
00C8 0094                00403   MOVWF  RAM_BUF2_PNT ;RAM_BUF2
                                 pointer initialisation
00C9 0824                00404   MOVF  NO_OF_DIGITS,W
00CA 0095                00405   MOVWF  DIGIT_COUNT
                                 ;[NO_OF_DIGITS] --> DIGIT_COUNT
                         00406
00CB 0813                00407   COMP_CONT MOVF  RAM_BUF1_PNT,W
                                 ;[RAM_BUF1_PNT] -->W
00CC 0084                00408   MOVWF  FSR  ;[W]-->FSR
00CD 0800                00409   MOVF  INDF,W  ;password digit --> w
                                 reg 1 by 1
00CE 0096                00410   MOVWF  PSD_DIGIT ;[W] -->PSD_DIGIT
00CF 0814                00411   MOVF  RAM_BUF2_PNT,W
                                 ;[RAM_BUF2_PNT] -->W
00D0 0084                00412   MOVWF  FSR;[W]-->FSR
00D1 0816                00413   MOVF  PSD_DIGIT,W ;[PSD_DIGIT] -->W
00D2 0600                00414   XORWF  INDF,W  ;[W] xor [RAM_BUF2]
                                 -->W
00D3 1D03                00415   BTFSS  STATUS,Z ;Is Z==1 ?
00D4 28DA                00416   GOTO  WARN  ;If no goto WARN
00D5 0A93                00417   INCF  RAM_BUF1_PNT,F ;If yes incre-
ment
                                 RAM_BUF1_PNT by 1
00D6 0A94                00418   INCF  RAM_BUF2_PNT,F ;Increment
                                 RAM_BUF2_PNT by 1
00D7 0B95                00419   DECFSZ DIGIT_COUNT,F ;Decrement
                                 DIGIT_COUNT by 1, is it 0 ?
00D8 28CB                00420   GOTO  COMP_CONT  ;If no goto compare
                                 nxt digit
00D9 0008                00421   RETURN;If yes Return back
                         00422
                         00423
00DA 2172                00424   WARN  CALL  LONG_BEEP ;Make a long
                                 beep
00DB 0B97                00425   DECFSZ NO_OF_ATTEMPTS,F;Decrement
                                 NO_OF_ATTEMPTS,is it 0 ?
00DC 2826                00426   GOTO  BEGIN  ;If no goto BEGIN
00DD 1585                00427   ALARM  BSF  PORTA,3  ;Switch ON the
                                 buzzer
00DE 1A05                00428   BTFSC PORTA,4  ;Is the jumper placed?
00DF 28EA                00429   GOTO LATCH_ALARM ;If not goto
                                 latch_alarm
```

```
                        00430 ; If yes auto reset after 1 min
                        00431 ;************************************
                        00432 ; program now inactivates the codelock
                              for 1 minute
                        00433 ; 1min = 1uS(instuction cycle) x
                              256(prescalar count) x(195)tmr0
                              counts x200 x6
                        00434 ; count to be loaded in TMR0 = (256
                              -195) +2 =H'3F'
                        00435 ; 2 is added because after moving a
                              value to TMR0 reg the actual
                        00436 ; incremetation of TMR0 delays by 2
                              TMR0 clock cycles.
                        00437 ;------------------------------------
                        00438
00E0 110B               00439   ONE_MIN_DEL BCF INTCON,T0IF ;Clear
                              TMR0 interrupt flag
00E1 168B               00440   BSF INTCON,T0IE ;Enable TMR0
                              interrupt feature
00E2 3006               00441   MOVLW 0X06  ;Count for one minute
00E3 00A3               00442   MOVWF ONE_MIN_CNT
00E4 30C8               00443   MOVLW 0XC8  ;Count required to obtain
                              10s delay
00E5 00A2               00444   MOVWF TEN_SEC_CNT
00E6 303F               00445   MOVLW  0X3F  ;Count required to
                              obtain 50ms delay
00E7 0081               00446   MOVWF TMR0
00E8 178B               00447   BSF INTCON,GIE
                        00448
00E9 28E9               00449   INFI_LOOP  GOTO  INFI_LOOP ;Simply
                              loop here until 1 min
                        00450 ;------------------------------------
                        00451 ; The program control comes here only
                              if the jumper is not placed.(see
                              ckt dia)
                        00452
00EA 28EA               00453   LATCH_ALARM GOTO  LATCH_ALARM
                              ;Simply lopp here until manual reset
                        00454 ; by power interruption.
                        00455
                        00456
                        00457 ;************************************
                        00458 ;  ROUTINE TO COPY EEPROM CONTENT
                              TO RAM
                        00459 ;************************************
                        00460
00EB                    00461   COPY_TO_RAM
00EB 3030               00462   MOVLWRAM_BUF1 ;Initialize FSR to
                              the
00EC 0084               00463   MOVWF  FSR  ;Staring address of
                              RAM_BUF1
00ED 3010               00464   MOVLW D'16'
00EE 0095               00465   MOVWF  DIGIT_COUNT ;NO_OF_DIGITS
                              = 16 digits
00EF 0819               00466   MOVF EEADDR_TEMP,W
                              ;[EEADDR_TEMP] --> W
00F0 0089               00467   MOVWF  EEADR;[W] -->EEADR
                        00468
00F1 1683               00469   COPY_NXT_BYTE BSF  STATUS,RP0
                              ;Select bank1
00F2 1408               00470   BSF  EECON1,RD ;Enable Read mode
00F3 1283               00471   BCF  STATUS,RP0 ;Select bank0
00F4 0808               00472   MOVF  EEDATA,W ;[EEDATA]-->w
00F5 0080               00473   MOVWF  INDF  ;[W]-->INDF
00F6 0A84               00474   INCF  FSR,F  ;Increment FSR by 1
00F7 0A89               00475   INCF  EEADR,F  ;Increment EEADR by 1
00F8 0B95               00476   DECFSZ  DIGIT_COUNT,F ;Decrement
                              DIGIT_COUNT by 1,is it 0 ?
00F9 28F1               00477   GOTO  COPY_NXT_BYTE  ;If no goto
                              COPY_NXT_BYTE
00FA 0008               00478   RETURN;If yes return
                        00479
                        00480
                        00481 ;************************************
                        00482 ; ROUTINE TO CHG PASSWARD
                        00483 ;
                        00484 ; The program control comes here when
                              you press CHG key.First this
                              subroutine asks
                        00485 ; for channel no then old password if
                              the entered information is correct,
                              it takes the
                        00486 ; new password.then again takes the new
                              password for confirmation. on
                              confirmation
                        00487 ; on confirmation success old pasward
                              will be replaced by the new password.
                              On
                        00488 ; confirmation error the old password
                              will not be altered.
                        00489 ;************************************
                        00490
00FB 20A1               00491   CHG_PSWD CALL  GET_CH_NO ;Get the
                              user/channel no
00FC 20B4               00492   CALL  VRFY_PASWD ;Veryfy the old
                              password
00FD 217D               00493   CALL  BEEP_THRICE ;Beep thrice on
                              verificatin success
00FE 3003               00494   MOVLW 0X03  ;Reset NO_OF_ATTEMPTS
                              to 3
00FF 0097               00495   MOVWF NO_OF_ATTEMPTS
0100 3040               00496   MOVLW  RAM_BUF2 ;Initialise FSR to
                              the
0101 0084               00497   MOVWF  FSR  ;Starting address of
                              RAM_BUF2
0102 01A4               00498   CLRF NO_OF_DIGITS ;NO_OF_DIGITS=0
                        00499
0103 2033               00500   GET_NXT_BYTE CALL  KEY_SCAN ;Call
                              key scan routine
0104 0080               00501   MOVWF  INDF;[W] -->INDF
0105 0092               00502   MOVWF KB_TEMP  ;[W] --> KB_TEMP
0106 3A0A               00503   XORLW 0X0A
0107 1903               00504   BTFSC STATUS,Z ;Is L/U key pressed ?
0108 2831               00505   GOTO WRNG_ENTRY ;If yes goto
                              WRNG_ENTRY
0109 0812               00506   MOVF KB_TEMP,W ;If no KB_TEMP-->W
010A 3A0B               00507   XORLW 0X0B
010B 1903               00508   BTFSC STATUS,Z ;Is CHG key pressed ?
010C 2910               00509   GOTO PROCEDE  ;If yes goto PROCEDE
010D 0AA4               00510   INCF  NO_OF_DIGITS,F ;If no increment
                              NO_OF_DIGITS by 1
010E 0A84               00511   INCF FSR,F  ;Increment FSR by 1
010F 2903               00512   GOTO  GET_NXT_BYTE ;Goto
                              GET_NXT_BYTE
                        00513
0110 0824               00514   PROCEDE  MOVF NO_OF_DIGITS,W ;[NO
                              OF DIGITS] -->W
0111 0095               00515   MOVWF  DIGIT_COUNT ;[W]
                              -->DIGIT_CCUNT
0112 3C03               00516   SUBLW 0X03  ;Is new password
0113 1803               00517   BTFSC STATUS,C ;contains < 4 digits
?
0114 2831               00518   GOTO WRNG_ENTRY ;If yes goto
                              WRNG_ENTRY
0115 0824               00519   MOVF NO_OF_DIGITS,W ;If no W
                              --> NO_OF_DIGITS
0116 3C0F               00520   SUBLW D'15'  ;Is new password
0117 1C03               00521   BTFSS  STATUS,C  ;contains  -->  >15
digits?
0118 2831               00522   GOTO WRNG_ENTRY ;If yes goto
                              WRNG_ENTRY
0119 2178               00523   CALL BEEP_TWICE ;If no beep twice
011A 3030               00524   MOVLW RAM_BUF1 ;Initialise FSR to the
011B 0084               00525   MOVWF FSR  ;starting address of
                              RAM_BUF1
                        00526
                        00527
011C 2033               00528   GET_NXT_BYTE2 CALL  KEY_SCAN ;Call
                              scan key routine
011D 0080               00529   MOVWF  INDF;[W] -->INDF
011E 3C09               00530   SUBLW 0X09  ;[W] - 0x09 -->W
011F 1C03               00531   BTFSS STATUS,C ;Is L/U key is
                              pressed ?
0120 2831               00532   GOTO WRNG_ENTRY ;If yes goto
                              WRNG_ENTRY
0121 0A84               00533   INCF FSR,F  ;If no increment FSR
                              by 1
0122 0B95               00534   DECFSZ  DIGIT_COUNT,F ;Decrement
                              DIGIT_COUNT by 1,is it 0 ?
0123 291C               00535   GOTO  GET_NXT_BYTE2 ;If yes goto
```

```
                        GET_NXT_BYTE2
                    00536
0124 3030           00537   MOVLW  RAM_BUF1 ;RAM_BUF1_PNT
                            initialisation
0125 0093           00538   MOVWF  RAM_BUF1_PNT
0126 3040           00539   MOVLW  RAM_BUF2 ;RAM_BUF2_PNT
                            initialisation
0127 0094           00540   MOVWF  RAM_BUF2_PNT
0128 0824           00541   MOVF  NO_OF_DIGITS,W ;[No of digits]
                            -->W
0129 0095           00542   MOVWF  DIGIT_COUNT  ;[W]
                            -->DIGIT_COUNT
                    00543
012A 0813           00544   CONFRM_PSD MOVF  RAM_BUF1_PNT,W
012B 0084           00545   MOVWF  FSR  ;[RAM_BUF1_PNT] -->FSR
012C 0800           00546   MOVF  INDF,W  ;[RAM_BUF1]-->W
012D 0096           00547   MOVWF  PSD_DIGIT ;[W]-->PSD_DIGIT
012E 0814           00548   MOVF  RAM_BUF2_PNT,W
                            ;[RAM_BUF2_PNT] -->W
012F 0084           00549   MOVWF  FSR;[W]-->FSR
0130 0816           00550   MOVF  PSD_DIGIT,W ;[PSD_DIGIT] -->W
0131 0200           00551   SUBWF  INDF,W  ;[W]-[RAM_BUF2]-->W
0132 1D03           00552   BTFSS  STATUS,Z ;Is
                            [RAM_BUF1]==[RAM_BUF2] ?
0133 295D           00553   GOTO  CONFRM_ERR ;If no goto
                            CONFRM_ERR
0134 0A93           00554   INCF  RAM_BUF1_PNT,F ;If yes
                            increment RAM_BUF1_PNT by 1
0135 0A94           00555   INCF  RAM_BUF2_PNT,F ;Increment
                            RAM_BUF2_PNT by 1
0136 0B95           00556   DECFSZ DIGIT_COUNT,F ;Decrement
                            DIGIT_COUNT by 1,is it 0?
0137 292A           00557   GOTO  CONFRM_PSD  ;If no goto
                            CONFRM_PSD
0138 3040           00558   MOVLW RAM_BUF2 ;If yes point to the
0139 3E0F           00559   ADDLW 0X0F  ;end of RAM_BUF2
013A 0084           00560   MOVWF FSR
013B 0824           00561   MOVF NO_OF_DIGITS,W  ;Store the no of
                            digits
013C 0080           00562   MOVWF  INDF  ;in the password at the
                            end of
013D 3040           00563   MOVLW RAM_BUF2 ;RAM_BUF2
013E 0084           00564   MOVWF FSR
                    00565
013F 3010           00566   START_EE_WR MOVLW  D'16'  ;No of
                            bytes to write = 16
0140 0095           00567   MOVWF  DIGIT_COUNT
0141 0819           00568   MOVF  EEADDR_TEMP,W ;Set initial
                            EEPROM address
0142 0089           00569   MOVWF  EEADR
0143 1283           00570   BCF STATUS,RP0 ;Select bank0
0144 138B           00571   BCF  INTCON,GIE ;Dissable all
                            interrupts
                    00572
0145 0800           00573   WR_EEPROM MOVF  INDF,W  ;[INDF]
                            --> W
0146 0088           00574   MOVWF  EEDATA  ;W -->EEDATA
0147 1683           00575   BSF  STATUS,RP0 ;Select bank1
0148 1508           00576   BSF  EECON1,WREN ;Enable write mode
0149 3055           00577   MOVLW  0X55
014A 0089           00578   MOVWF  EECON2  ;H'55' must be written
                            to eecon2
014B 30AA           00579   MOVLW  0XAA  ;to start write sequence
014C 0089           00580   MOVWF  EECON2  ;followed by H'AA'
014D 1488           00581   BSF  EECON1,WR ;Set WR bit to start
                            writing
                    00582
014E 1E08           00583   POLL_EEIF BTFSS  EECON1,EEIF ;Is
                            write complete ?
014F 294E           00584   GOTO  POLL_EEIF ;If no goto POLL_EEIF
0150 1208           00585   BCF  EECON1,EEIF ;If yes clear EEIF
                            bit
0151 1988           00586   BTFSC  EECON1,WRERR ;Is WRERR is set?
0152 2945           00587   GOTO  WR_EEPROM ;If set write again
0153 0A84           00588   INCF  FSR,F  ;Increment FSR by 1
0154 1283           00589   BCF  STATUS,RP0 ;Select bank0
0155 0A89           00590   INCF  EEADR,F  ;Increment EEADR by 1
0156 0B95           00591   DECFSZ DIGIT_COUNT,F ;Decrement
                            DIGIT_COUNT by1 ,is it 0 ?
0157 2945           00592   GOTO  WR_EEPROM ;If NO go to write
                            next digit.
0158 1683           00593   BSF  STATUS,RP0 ;If yes select bank0
0159 1108           00594   BCF  EECON1,WREN ;Dissable Write mode
015A 1283           00595   BCF  STATUS,RP0 ;Select bank0
015B 217D           00596   CALL  BEEP_THRICE ;Beep thrice
015C 2826           00597   GOTO  BEGIN  ;Goto  BEGIN
                    00598
015D 2172           00599   CONFRM_ERR CALL  LONG_BEEP ;Give a
                            long beep on confirm Error
015E 2826           00600   GOTO  BEGIN  ;Goto  BEGIN
                    00601
                    00602
                    00603 ;*************************************
                    00604 ; DELAY SUBROUTINE FOR BUZZER ON
                            AND OFF TIME
                    00605 ;*************************************
015F 0821           00606   BUZ_DELAY MOVF BUZ_DEL_CNT,W
0160 008C           00607   MOVWF  DEL_COUNT1
0161 3040           00608   BUZ_LOOP1  MOVLW 0X40
0162 008D           00609   MOVWF DEL_COUNT2
0163 30FE           00610   BUZ_LOOP2 MOVLW  0XFE
0164 008E           00611   MOVWF  DEL_COUNT3
0165 0B8E           00612   BUZ_LOOP3 DECFSZ DEL_COUNT3,F
0166 2965           00613   GOTO BUZ_LOOP3
0167 0B8D           00614   DECFSZ  DEL_COUNT2,F
0168 2963           00615   GOTO BUZ_LOOP2
0169 0B8C           00616   DECFSZ DEL_COUNT1,F
016A 2961           00617   GOTO BUZ_LOOP1
016B 0008           00618   RETURN
                    00619 ;*************************************
                    00620 ; SUBROUTINES TO SOUND BUZZER
                    00621 ;*************************************
                    00622
016C 3001           00623   SHORT_BEEP MOVLW  0X01  ;Subroutine
                            to produce a short beep
016D 00A1           00624   MOVWF  BUZ_DEL_CNT
016E 1585           00625   BSF  PORTA,3
016F 215F           00626   CALL  BUZ_DELAY
0170 1185           00627   BCF PORTA,3
0171 0008           00628   RETURN
                    00629
0172 300A           00630   LONG_BEEP MOVLW  0X0A  ;Subroutine
                            to produce a long beep
0173 00A1           00631   MOVWF  BUZ_DEL_CNT
0174 1585           00632   BSF  PORTA,3
0175 215F           00633   CALL  BUZ_DELAY
0176 1185           00634   BCF PORTA,3
0177 0008           00635   RETURN
                    00636
0178 3005           00637   BEEP_TWICE MOVLW 0X05
0179 00A1           00638   MOVWF BUZ_DEL_CNT
017A 215F           00639   CALL BUZ_DELAY
017B 3002           00640   MOVLW 0X02  ;Subroutine to produce
                            2 short beeps
017C 2982           00641   GOTO BEEP_NOW
                    00642
017D 3005           00643   BEEP_THRICE MOVLW 0X05
017E 00A1           00644   MOVWF BUZ_DEL_CNT
017F 215F           00645   CALL BUZ_DELAY
0180 3003           00646   MOVLW 0X03  ;Subroutine to produce 3
                            short beeps
0181 2982           00647   GOTO BEEP_NOW
                    00648
0182 00A0           00649   BEEP_NOW MOVWF NO_OF_BEEPS
0183 3004           00650   BEEP_AGAIN MOVLW 0X04
0184 00A1           00651   MOVWF BUZ_DEL_CNT
0185 215F           00652   CALL BUZ_DELAY
0186 216C           00653   CALL SHORT_BEEP
0187 0BA0           00654   DECFSZ NO_OF_BEEPS,F
0188 2983           00655   GOTO  BEEP_AGAIN
0189 0008           00656   RETURN
                    00657
                    00658   END ;The progam ends here
```

SECURED ROOM ACCESS SYSTEM

■ **ASHAY MATHUR**

Security is a prime concern in our day-to-day life. And access control system forms a vital link in a security chain. The microcontroller-based digital lock presented here is an access control system that allows only authorised persons to access a restricted area. When someone tries to enter the restricted area by entering invalid passwords continuously, the system locks itself and can be unlocked only by the master user.

The system comprises a small electronic unit with a numeric keypad, which is fixed outside the entry door to control a solenoid-operated lock. When an authorised person enters a predetermined number (password) via the keypad, the relay energises for a limited time to unlock the solenoid-operated lock, so door can be pushed/pulled open. At the end of the preset delay, the relay de-energises and the door gets locked again. A prompt message is displayed on the LCD module.

Circuit description

The system uses a compact circuitry built around AVR microcontroller ATmega8535. The ATmega8535 is a low-power CMOS 8-bit microcontroller based on the AVR-enhanced RISC architecture. It provides the following features: 8 kB of in-system programmable Flash memory with read-while-write capabilities, 512-byte EEPROM,

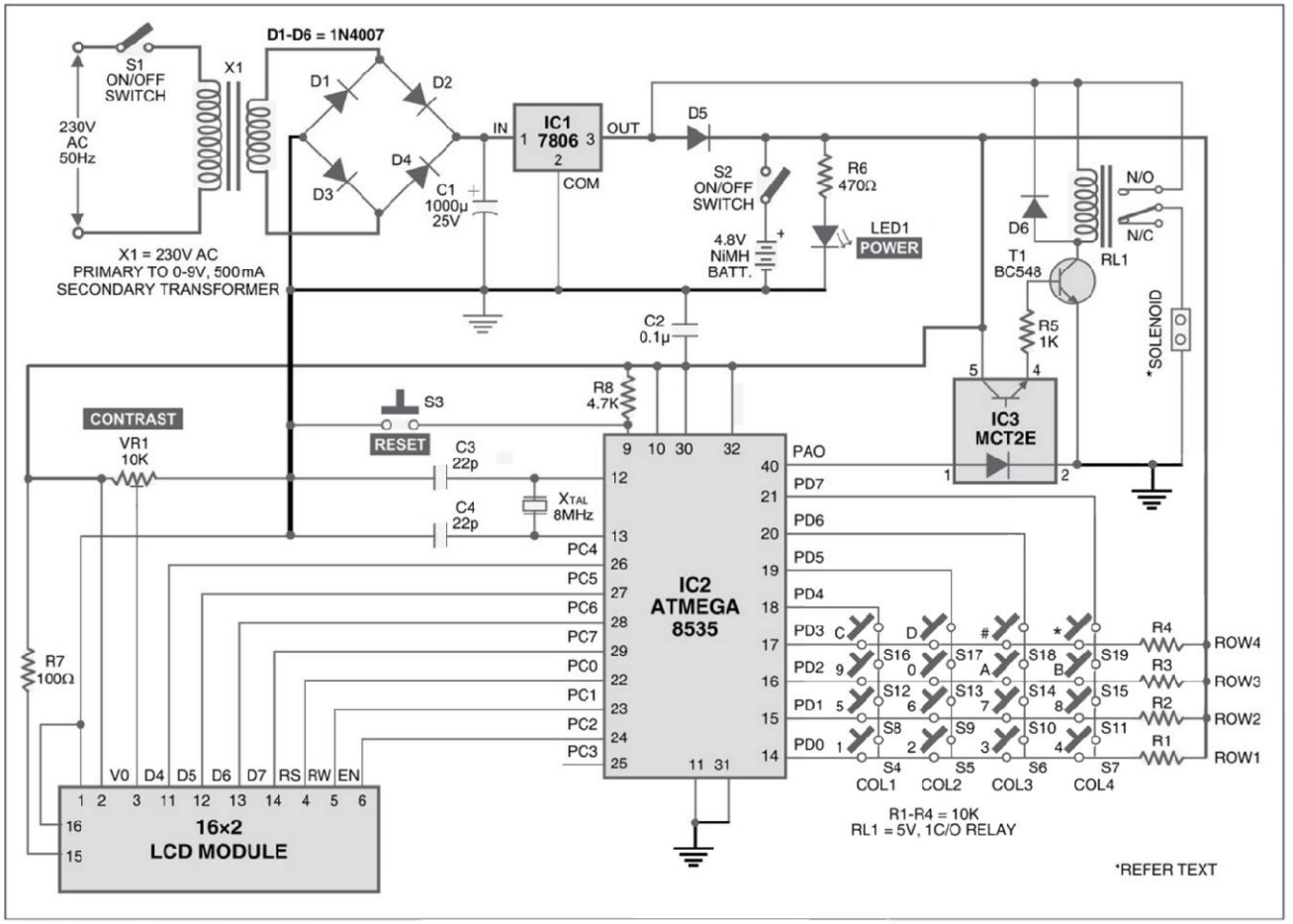


Fig. 1: Secured Room Access System

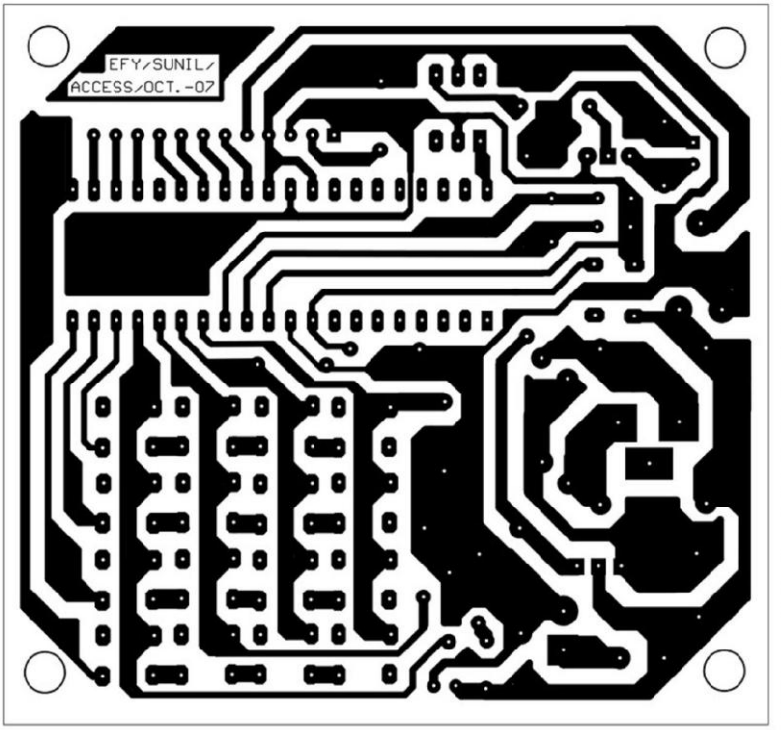


Fig. 2: A single-side, actual-size PCB layout for secured room access system

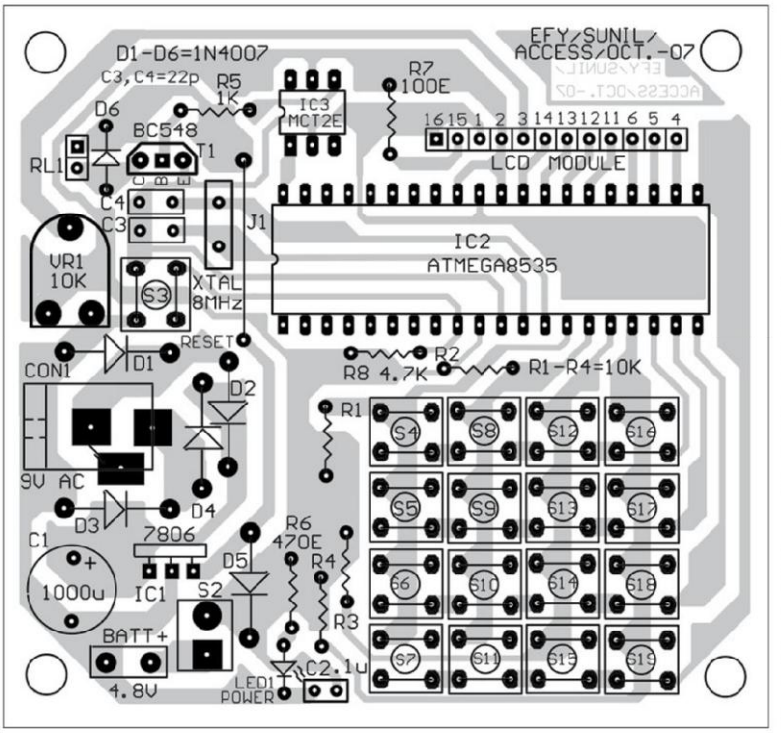


Fig. 3: Component layout for the PCB

512-byte SRAM, 32 general purpose I/O lines, 32 general-purpose working registers, three flexible timer/counters with compare modes, and internal and external interrupts. The built-in power-on-reset circuitry of the microcontroller eliminates the need for external power-on-reset circuit.

Switch S3 is used to reset the system, which is accessible only to the master user. Port D (PD0 through PD7) is interfaced with the numeric keypad. Port C is interfaced with a 16-x2-line LCD. Four pins (PC4 through PC7) of Port C are used as data lines for the LCD module and three lines (PC0 through PC2) are used for controlling the LCD. Pin 40 (PA0) of port A is connected to the relay driver circuit through optocoupler MCT2E (IC3) and transistor T1.

When port pin PA0 goes high, the internal transistor of IC3 drives transistor T1 into saturation and relay RL1 energises. As the solenoid valve is connected through normally-closed (N/C) contact of the relay, the solenoid coil de-energises and the gate is locked. An 8MHz crystal is used with two 22pF capacitors for providing clock. Preset VR1 is used to adjust the contrast of the LCD.

The 230V, 50Hz AC mains is stepped down by transformer X1 to deliver a secondary output of 9V, 500 mA. The transformer output is rectified by a full-wave bridge rectifier comprising diodes D1 through D4, filtered by capacitor C1 and regulated by IC 7806 (IC1). Use adequate heat-sink for 7806 as the solenoid draws a high current. LED1 glows when power is 'on' and resistor R6 acts as the current limiter.

A 16-key numeric keypad for password entry is connected to the microcontroller. The keypad is also used for password change and application of master password when required. To economise the use of I/O pins, we have used here only eight pins for scanning and sensing 16 keys.

The keypad is arranged in a 4x4 matrix. There are four scan lines/pins, which are set in output mode, and four sense keys, which are used as input lines to the microcontroller.

At a small time interval, the microcontroller sets one of the four scan lines as low and the other three scan lines as high. Then it checks for the status of sense lines one by one at the intersection of a specific scan line and

sense line to find out if any key has been pressed.

Similarly, after a small time interval, the next scan line is made low and remaining three scan lines are taken high, and again all three sense lines are checked for low level. This way the microcontroller checks which of the 16 keys is pressed.

Due to the high speed of the microcontroller, the status of different keys is checked in less than 100 ms and a key press is detected and identified. As the keys are pressed manually by the user, this delay of 100 ms is not noticeable. The net result is that you save on I/O pins of the microcontroller by sacrificing almost nothing.

When a person wants to enter the room, he enters the 6-digit password, say '123456.' If the password matches successfully, the gate is unlocked for 15 seconds.

If you want to change the user password (123456) and enter the master password '291279,' the system will ask you to change the user password. On successfully entering the password, pin A0 of port A becomes high for 15 seconds, because of which transistor T1 starts conducting through the emitter of the optocoupler and the relay energises. The connection between the solenoid lock and the power supply is broken and the door is unlocked for 15 seconds.

An actual-size, single-side PCB for secured room access system (Fig. 1) is shown in Fig. 2 and its component layout in Fig. 3.

PARTS LIST

Semiconductor:

IC1	- 7806 6V regulator
IC2	- ATmega8535 AVR microcontroller
IC3	- MCT2E optocoupler
T1	- BC548 npn transistor
D1-D6	- 1N4007 rectifier diode
LED1	- 5mm light-emitting diode

Resistors (all ¼-watt, ±5% carbon):

R1-R4	- 10-kilo-ohm
R5	- 1-kilo-ohm
R6	- 470-ohm
R7	- 100-ohm
VR1	- 10-kilo-ohm preset

Capacitors:

C1	- 1000µF, 25V electrolytic
C2	- 0.1µF ceramic disk
C3, C4	- 22pF ceramic disk

Miscellaneous:

X1	- 230V AC primary to 9V, 500mA secondary transformer
S1, S2	- On/off switch
S3-S19	- push-to-on tactile switch
X_{TAL}	- 8 MHz crystal
RL1	- 6V, 1C/O relay
Batt.	- 4.8 volt rechargeable battery
	- LCD module 16 X 2 line
	- 6 volt operated solenoid lock

Software

The software for the AVR microcontroller is written in 'C' language and compiled using Code Vision AVR 'C' compiler. Since this compiler does not have library functions for the keypad, place 'kbd.h' file in the INC folder of the installation folder and 'kbd.lib' in the LIB folder of 'cvavr' folder. This file is included in the program and the same can be used.

Download source code: http://www.efymag.com/admin/issuepdf/Access%20Control.zip

SOURCE PROGRAM

```
#asm
 .equ __lcd_port=0x15
#endasm
eeprom long int pass_store= 123456, master_password=
291279;
#include <mega8535.h>
#include <lcd.h>
#include <stdio.h>
#include <delay.h>
#include <kbd.h>
#define relay PORTA.0

int i,j,k,fail=0;
long int id_value, pass_value, pass[6];
bit match=0;
void password()
{i=0;
do{
 delay_ms(50);
 while(!kbd_read()){}
 j=kbd_read();
 if(j<11)
 {if(j==10) j=0; pass[i]=j;
 lcd_gotoxy(5+i,1);
 lcd_putsf("*"); i++;}
} while(i<6);
delay_ms(100);
pass_value=pass[5]+  10*pass[4]+  100*pass[3]+
1000*pass[2]+ 10000*pass[1]+ 100000*pass[0];
}
void pass_change()
{
lcd_clear(); lcd_gotoxy(0,0);
lcd_putsf("Enter New ");
lcd_gotoxy(0,1);
lcd_putsf("Password:");
delay_ms(50); k=0;
do{
delay_ms(50);
while(!kbd_read()){}
j=kbd_read();
if(j<11)
{if(j==10) j=0;
pass[k]=j;
k++;
lcd_gotoxy(9+k,1);
```

```
lcd_putsf("*");}
} while(k<6);
pass_value=pass[5]+  10*pass[4]+  100*pass[3]+
1000*pass[2]+ 10000*pass[1]+ 100000*pass[0];
pass_store=pass_value;
id_value=pass_store;
delay_ms(100);
if(pass_value==id_value){lcd_clear();
lcd_putsf("Password changed");}
else {lcd_clear();
lcd_putsf("Verify failed");}
delay_ms(300);}
void unlock()
{
relay=~relay;lcd_clear();
lcd_gotoxy(1,0);
lcd_putsf("Door Unlocked");
lcd_gotoxy(2,1);
lcd_putsf("Please enter");
delay_ms(300);
for(k=0;k<20;k++)
{lcd_clear();
delay_ms(50);
lcd_gotoxy(1,0);
lcd_putsf("Door Unlocked");
lcd_gotoxy(2,1);
lcd_putsf("Please enter");
delay_ms(50);}
relay=~relay; return;
}
void main()
{
PORTA=0x00;
DDRA=0xFF;
kbd_init();
lcd_init(16);
lcd_clear();
lcd_gotoxy(4,0);
lcd_putsf("Welcome!");
delay_ms(100);
label1: lcd_clear();
lcd_gotoxy(1,0);
lcd_putsf("Enter Password:");
delay_ms(50);
password();
if(pass_value==pass_store){
match=1;
}
else if(pass_value==master_password) {
match=1;
pass_change();
match=0;
fail=0;
goto label1;
}
else {match=0;
}
if(match==1)
{
match=0;
fail=0;
delay_ms(100);
unlock();
goto label1;
}
else
{
lcd_clear();
 lcd_putsf("Invalid Password");
 match=0;
 fail=fail+1;
 delay_ms(200);
 if(fail==3)
 {
 lcd_clear();
 lcd_gotoxy(1,0);
 lcd_putsf("Contact System ");
 lcd_gotoxy(1,1);
 lcd_putsf("Administrator");
 }
 else
 {
 goto label1;
 }
 }
}
```

RFID-BASED SECURITY SYSTEM

■ BIKRAMJEET WARAICH

A radio-frequency identification (RFID) based access-control system allows only authorised persons to enter a particular area of an establishment. The authorised persons are provided with unique tags, using which they can access that area.

The system is based on microcontroller AT89C52 and comprises an RFID module, an LCD module for displaying the status and a relay for opening the door. Fig. 1 shows a user trying to open the door by placing an RFID tag near the RFID reader.

Fig. 1: a user is trying to open the door by placing an RFID tag near the RFID reader

Radio-frequency identification

You might be familiar with RFID systems as seen in access control, contactless payment systems, product tracking and inventory control, etc. Basically, an RFID system consists of three components: an antenna or coil, a transceiver (with decoder) and a transponder (RF tag) electronically programmed with unique information.

Fig. 2 shows a typical RFID system. In every RFID system, the transponder tags contain unique identifying information. This information can be as little as a single binary bit or a large array of bits representing such things as an identity code, personal medical information or literally any type of information that can be stored in digital binary format.

The RFID transceiver communicates with a passive tag. Passive tags have no power source of their own and instead derive power from the incident electromagnetic field. Commonly, at the heart of each tag is a microchip. When the tag enters the generated RF field, it is able to draw enough power from the field to access its internal memory and transmit its stored information. When the transponder tag draws power in this way, the resultant interaction of the RF fields causes the voltage at the transceiver antenna to drop in value. This effect is utilised by the tag to communicate its information to the reader. The tag is able to control the amount of power drawn from the field and by doing so it can modulate the voltage sensed at the transceiver according to the bit pattern it wishes to transmit.

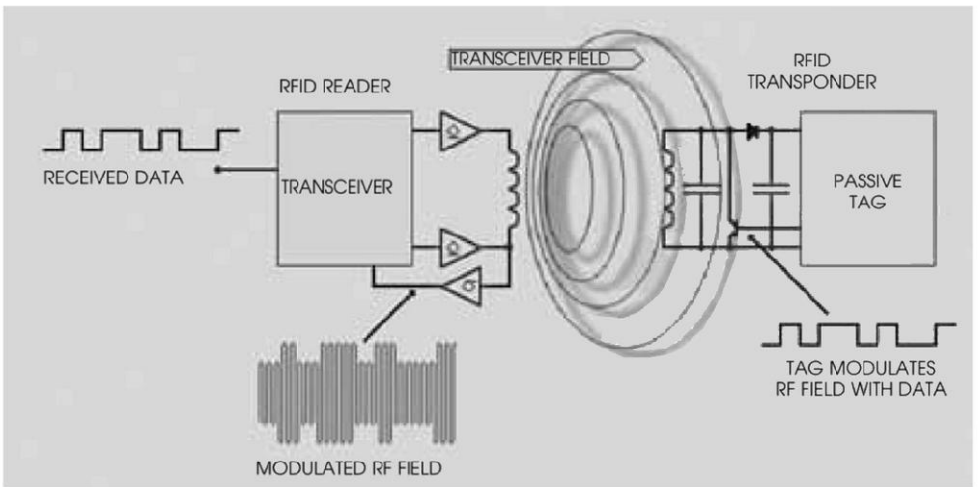


Fig. 2: A typical RFID system

Antenna. Fig. 3 shows the internal diagram of a

typical RFID antenna. An RFID antenna consists of a coil with one or more windings and a matching network. It radiates the electromagnetic waves generated by the reader to activate the tag and read/write data from it.

Antennae are the conduits between the tag and the transceiver which control the system's data acquisition and communication. These are available in a variety of shapes and sizes. Often, the antenna is packaged with the transceiver and decoder to become a reader, which can be configured either as a handheld or a fixed-mount device. The reader emits radio waves in ranges of anywhere from 2.54 cm (one inch) to 30 metres or more, depending upon its power output and the radio frequency used. When an RFID tag passes through the electromagnetic zone, it detects the reader's activation signal. The reader decodes the data encoded in the tag's integrated circuit (silicon chip) and the data is passed to the host computer for processing.

Tags (transponders). Fig. 4 shows the internal structure of a typical RFID tag. An RFID tag comprises a microchip containing identifying information and an antenna that transmits this data wirelessly to the reader. At its most basic, the chip will contain a serialised identifier, or licence plate number, that uniquely identifies that item, similar to the way many bar codes are used today.

There are three types of tags: active, passive and semi-passive.

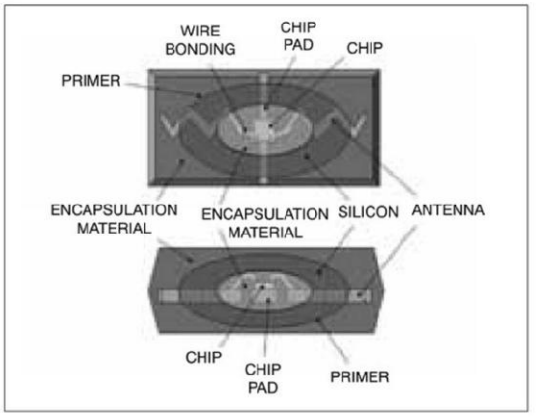


Fig. 3: Internal diagram of a typical RFID antenna

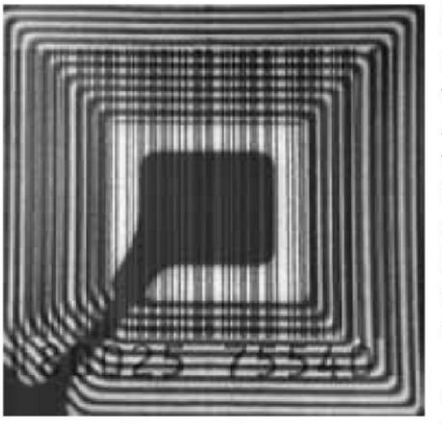


Fig. 4: internal structure of a typical RFID tag

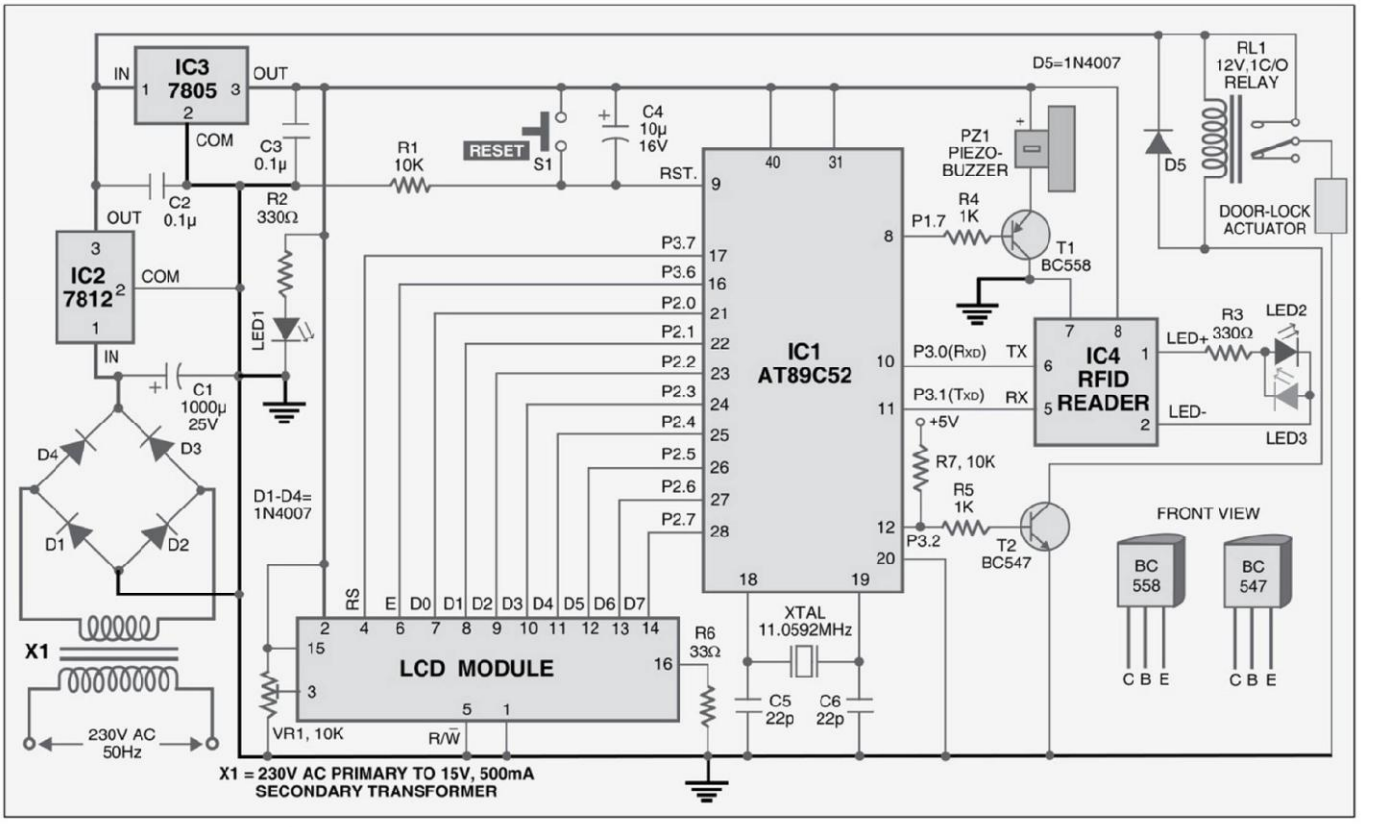


Fig. 5: Circuit of the RFID-based security system

Passive tags have no internal power source. These draw their power from the electromagnetic field generated by the RFID reader and then the microchip can send back information on the same wave. The reading range is limited when using passive tags.

Active transponders have their own transmitters and power source, usually in the form of a small battery. These remain in a low-power 'idle' state until they detect the presence of the RF field being sent by the reader. When the tag leaves the area of the reader, it again powers down to its idle state to conserve its battery. As a result, active tags can be detected at a greater range than passive tags.

Semi-passive tags have their own power source that powers only the microchip. These have no transmitter. They rely on altering the RF field from the transceiver to transmit their data.

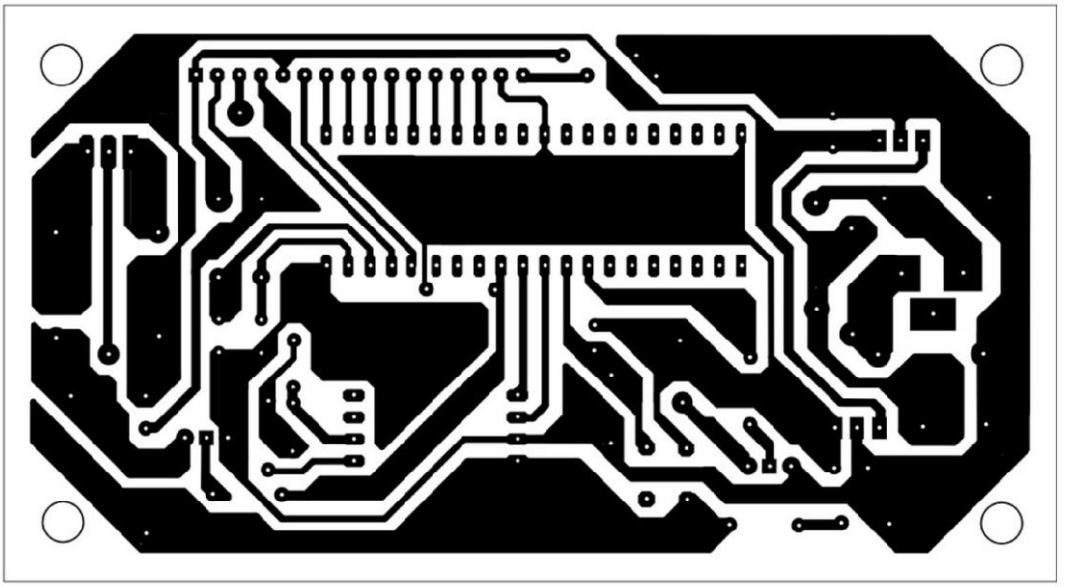


Fig. 6: An actual-size, single-side PCB for the RFID-based security system

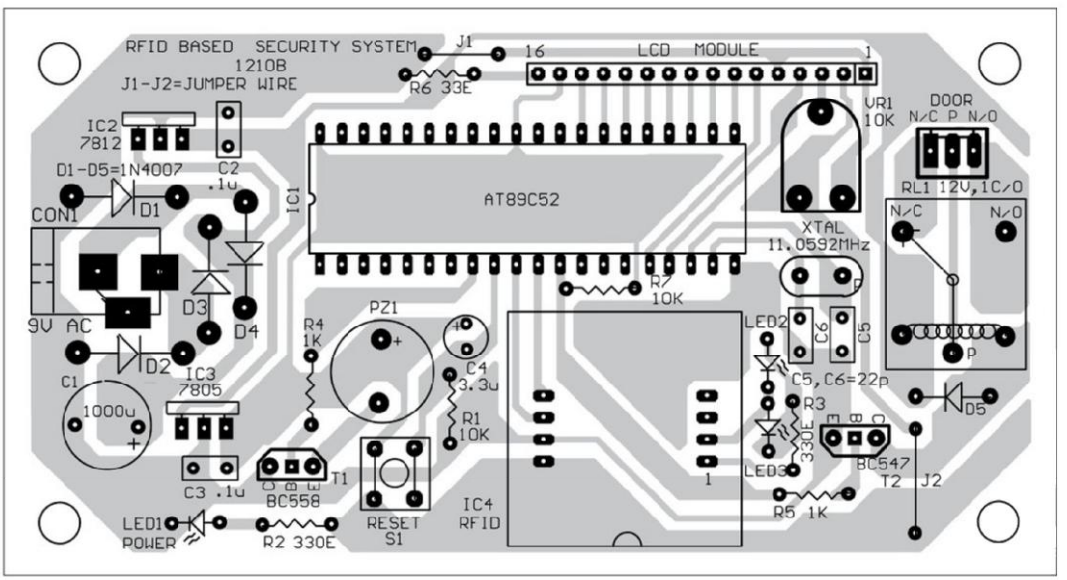


Fig. 7: Component layout for the PCB

There are three ways for data encoding into tags:

1. Read-only tags contain data, which is pre-written onto them by the tag manufacturer or distributor.
2. Write-once tags enable a user to write data to the tag one time in production or distribution processes.
3. Full read-write tags allow new data to be written to the tag as needed and later other data can be rewritten over the original data.

RF transceiver. The RF transceiver is the source of the RF energy used to activate and power the passive RFID tags. It may be enclosed in the same cabinet as the reader or it may be a separate piece of equipment. When provided as a separate piece of equipment, the transceiver is commonly referred to as an RF module. The RF transceiver controls and modulates the radio frequencies that the antenna transmits and receives. The transceiver filters and amplifies the back-scatter signal from a passive RFID tag.

Circuit description

Fig. 5 shows the circuit of the RFID-based security system. The compact circuitry is built around Atmel AT89C52 microcontroller. The AT89C52 is a low-power, high-performance CMOS 8-bit microcomputer with 8 kB of Flash programmable and erasable read-only memory (PEROM). It has 256 bytes of RAM, 32 input/output (I/O) lines, three 16-bit timers/counters, a six-vector two-level interrupt architecture, a full-duplex serial port, an on-chip oscillator and clock circuitry. The system clock also plays a significant role in operation of the microcontroller.

An 11.0592MHz quartz crystal connected to pins 18 and 19 provides basic clock to the microcontroller. Power-on reset is provided by the combination of electrolytic capacitor C4 and resistor R1. Switch S1 is used for manual reset. Port pins P2.0 through P2.7 of the microcontroller are connected to data port pins D0 through D7 of the LCD, respectively. Port pins P3.7 and P3.6 of the microcontroller are connected to register-select (RS) and enable (E) pins of the LCD, respectively. Read/write (R/$\overline{W}$) pin of the LCD is grounded to enable for write operation.

All the data is sent to the LCD in ASCII format for display. Only the commands are sent in hex form. Register-select (RS) signal is used to distinguish between data (RS=1) and command (RS=0). Preset VR1 is used to control the contrast of the LCD. Resistor R6 limits the current through the backlight of the LCD. Port pins P3.0 (R_{XD}) and P3.1 (T_{XD}) of the microcontroller are used to interface with the RFID reader.

When an authorised person having the tag enters the RF field generated by the RFID reader, RF signal is generated by the RFID reader to transmit energy to the tag and retrieve data from the tag. Then the RFID reader communicates through R_{XD} and T_{XD} pins of the microcontroller for further processing. Thus on identifying the authorised person, port pin P3.2 goes high, transistor T2 drives into saturation, and relay RL1 energises to open the door for the person. Simultaneously, the LCD shows "access granted" message and port pin P1.7 drives piezobuzzer PZ1 via transistor T1 for aural indication.

If the person is unauthorised, the LCD shows "access denied" and the door doesn't open. LED2 and LED3 show presence of the tag in the RFID reader's electromagnetic field.

To derive the power supply, the 230V, 50Hz AC mains is stepped down by transformer X1 to deliver a secondary output of 15V, 500 mA. The transformer output is rectified by a full-wave rectifier comprising diodes D1 through D4, filtered by capacitor C1 and regulated by ICs 7812 (IC2) and 7805 (IC3). Capacitor C2 bypasses the ripples present in the regulated supply. LED1 acts as the power indicator and R2 limits the current through LED1.

An actual-size, single-side PCB for RFID-based security system is shown in Fig. 6 and its component layout in Fig. 7. Assemble the circuit on a PCB as it minimises time and assembly errors. Carefully assemble the components and double-check for any overlooked error.

PARTS LIST

Semiconductors:

IC1	- AT89C52 microcontroller
IC2	- 7812, 12V regulator
IC3	- 7805, 5V regulator
IC4	- RFID reader module
T1	- BC558 pnp transistor
T2	- BC547 npn transistor
D1-D5	- 1N4007 rectifier diode
LED1-LED3	- 5mm LED
LCD	- 16-character, 2-line

Resistors (all ¼-watt, ±5% carbon unless stated otherwise):

R1, R7	- 10-kilo-ohm
R2, R3	- 330-ohm
R4, R5	- 1-kilo-ohm
R6	- 33-ohm

Capacitors:

C1	- 1000µF, 25V electrolytic
C2, C3	- 0.1µF ceramic disk
C4	- 10µF, 16V electrolytic
C4, C5	- 22pF ceramic disk

Miscellaneous:

X1	- 230V AC primary to 15V, 500mA secondary transformer
S1	- Push-to-on switch
X_{TAL}	- 11.0592MHz crystal
RL1	- 12V, 1C/O relay
PZ1	- Piezobuzzer
	- RFID tag
	- 12V DC door lock actuator

Software

The software for this project is given at the end of this article. It is written in 'C' language and compiled using Keil µVision4 compiler. The finally obtained '.hex' code is burnt into the microcontroller using a suitable programmer. The program is easy to understand.

Download source code: http://www.efymag.com/admin/issuepdf/RFID-Based-Security-System.zip

Note: The complete kit of this project is available with Kits'n'Spares.

RFID.C

```
#include<reg51.h>
sbit RS=P3^7;
sbit EN=P3^6;
sbit R=P3^2;       // for relay
sbit bz=P1^7;      // for piezo buzzer
void Rxmsg(void);
void lcdinit(void);
void lcdData(unsigned char l);
void lcdcmd(unsigned char k);
void DelayMs(unsigned int count);
void sucessRx(void);
void unknown(void);
void display(unsigned char s, t);
void welcome(void);
void main()
{
unsigned char i=0;
unsigned int j=0;
unsigned char c[15];
TMOD=0x20;                    // Configure the serial
port to 9600 baud rate
TH1=0xFD;
SCON=0X50;
TR1=1;
R=0;
lcdinit();
welcome();
bz=1;
while(1)
{
back:
for(i=0;i<15;i++)                          //command to
recv data
{
c[i]=0xFF;
}

while(RI==0);
for(i=0;i<15;i++)                          //command to
recv data
{
j=0;
while(RI==0)
{
if(j>=1000)
goto timeout;
j++;
}
c[i]=SBUF;
RI=0;
}
timeout:
for(i=0;i<15;i++)
{
if(c[i]=='1'  &&  c[i+1]=='E'  &&  c[i+2]=='0'  &&
c[i+3]=='0'  &&  c[i+4]=='7'  &&  c[i+5]=='C'  &&
c[i+6]=='A'  &&  c[i+7]=='0'  &&  c[i+8]=='3'  &&
c[i+9]=='C')
// RFID code
{
sucessRx();
DelayMs(1000);
R=1;
bz=0;
DelayMs(1000);
R=0;
bz=1;
DelayMs(1000);
lcdinit();
DelayMs(100);
welcome();
goto back;
}
}
unknown();
DelayMs(2000);
bz=0;
DelayMs(2000);
bz=1;
DelayMs(1000);
lcdinit();
DelayMs(100);
welcome();
}
}
void sucessRx()
{
unsigned int i=0;
unsigned char c[]="ACCESS GRANTED  ";
lcdcmd(0x01);
DelayMs(10);
lcdcmd(0x80);
DelayMs(10);
while(c[i]!='\0')
{
lcdData(c[i]);
i++;
}
}
void unknown(void)
{
unsigned int i=0;
unsigned char c[]="ACCESS DENIED";
lcdcmd(0x01);
DelayMs(10);
lcdcmd(0x80);
DelayMs(10);
while(c[i]!='\0')
{
lcdData(c[i]);
i++;
}
}
//-------------------------------
// Lcd initialization subroutine
//-------------------------------
void lcdinit(void)
{
lcdcmd(0x38);
DelayMs(250);
lcdcmd(0x0E);
DelayMs(250);
lcdcmd(0x01);
DelayMs(250);
lcdcmd(0x06);
DelayMs(250);
lcdcmd(0x80);
DelayMs(250);
}
//-------------------------------
// Lcd data display
//-------------------------------
void lcdData(unsigned char l)
{
P2=l;
RS=1;
EN=1;
DelayMs(1);
EN=0;
return;
}
//-------------------------------
// Lcd command
//-------------------------------
void lcdcmd(unsigned char k)
{
```

```
P2=k;
RS=0;
EN=1;
DelayMs(1);
EN=0;
return;
}
//---------------------------------
// Delay mS function
//---------------------------------
void DelayMs(unsigned int count)
{  // mSec Delay 11.0592 Mhz
    unsigned int i;                    // Keil v7.5a
    while(count) {
       i = 115;                        //  115  exact
value
                 while(i>0)
                 i--;
       count--;
    }
}
void welcome(void)
{
unsigned int i=0;
unsigned char c[]="WELCOME TO RFID";
unsigned char d[]="SECURITY SYSTEM";
lcdcmd(0x01);
DelayMs(10);
lcdcmd(0x80);
DelayMs(10);
while(c[i]!='\0')
{
lcdData(c[i]);
i++;
}
lcdcmd(0xc0);
i=0;
while(d[i]!='\0')
{
lcdData(d[i]);
i++;
}
}
```

SECURE DIGITAL ACCESS SYSTEM USING IBUTTON

■ CHIRUTKAR HARSHADKUMAR GOVINDRAO AND DR H.N. PANDYA

Access control forms a vital link in a security chain. Here we describe a secure digital access system using iButton that allows only authorised persons to access a restricted area.

The iButton is used here as a key to the access control system. Its unique identification (ID) number is used for authorisation. On detection of an authorised iButton, the system allows access. Thereafter, an automated lock key locks the system again. The system is permanently halted after five repeated false attempts. A service control unit built around an AVR microcontroller is interfaced to the iButton with 1-wire protocol for authentication of user validation of data.

iButton DS1990A

Here we have used the iButton DS1990A from Dallas Semiconductor (MAXIM). Its block diagram is shown in Fig. 1.

An iButton is a chip housed in a stainless-steel enclosure (refer Fig. 2). The electrical interface is reduced to the absolute minimum, i.e., a single data line plus a ground reference. The energy needed for operation is taken from the data line. The DS1990A serial number iButton is a rugged data carrier that acts as an electronic registration number for automatic identification. It contains a unique ROM code that is 64-bit long as shown in Fig. 3. The first eight bits are a 1-wire family code. The next 48 bits are a unique serial number. The last eight bits are a cyclic redundancy check (CRC) of the first 56 bits.

Data is transferred serially via the 1-wire protocol, which requires only a single data lead and a ground return. The iButton DS1990A provides the additional 1-wire protocol capability that allows the search ROM command to be interpreted by the DS1990A.

Circuit description

Fig. 4 shows the circuit of the secure digital access system using iButton. The circuit is built around an ATmega16 microcontroller.

The ATmega16 is an 8-bit microcontroller based on the AVR enhanced RISC architecture that executes powerful instructions in a single clock cycle. It has 16kB in-system programmable

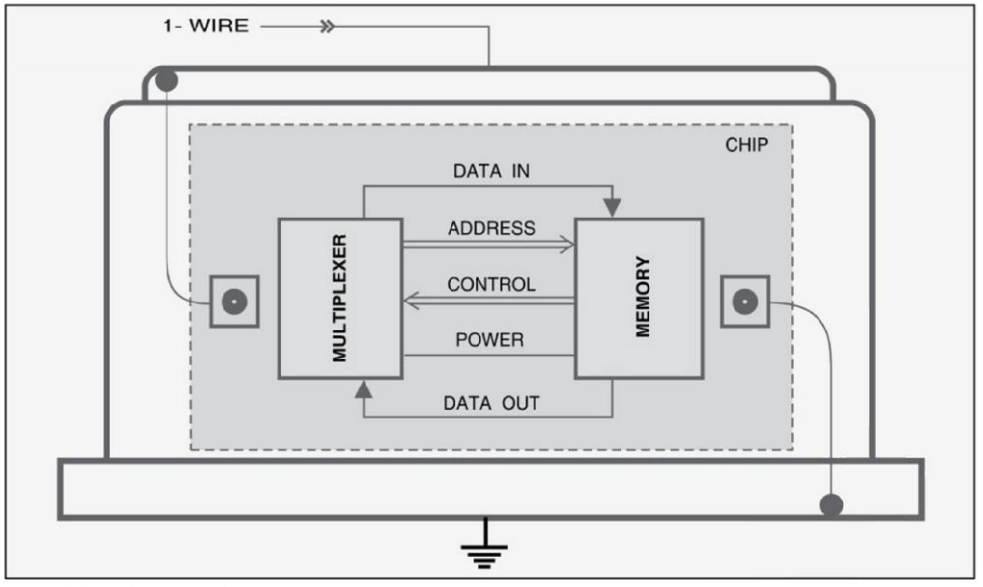


Fig. 1: Block diagram of iButton

Fig. 2: A typical iButton chip

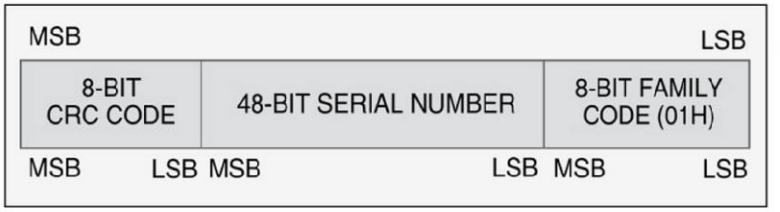


Fig. 3: 64-bit lasered ROM

flash program memory with read-while-write capabilities, 512 bytes of EEPROM, 1kB SRAM, 32 general-purpose input/output (I/O) lines, 32 general-purpose working registers, three flexible timers/counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented two-wire serial interface, a programmable watchdog timer with internal oscillator, an SPI serial port and six software-selectable power-saving modes.

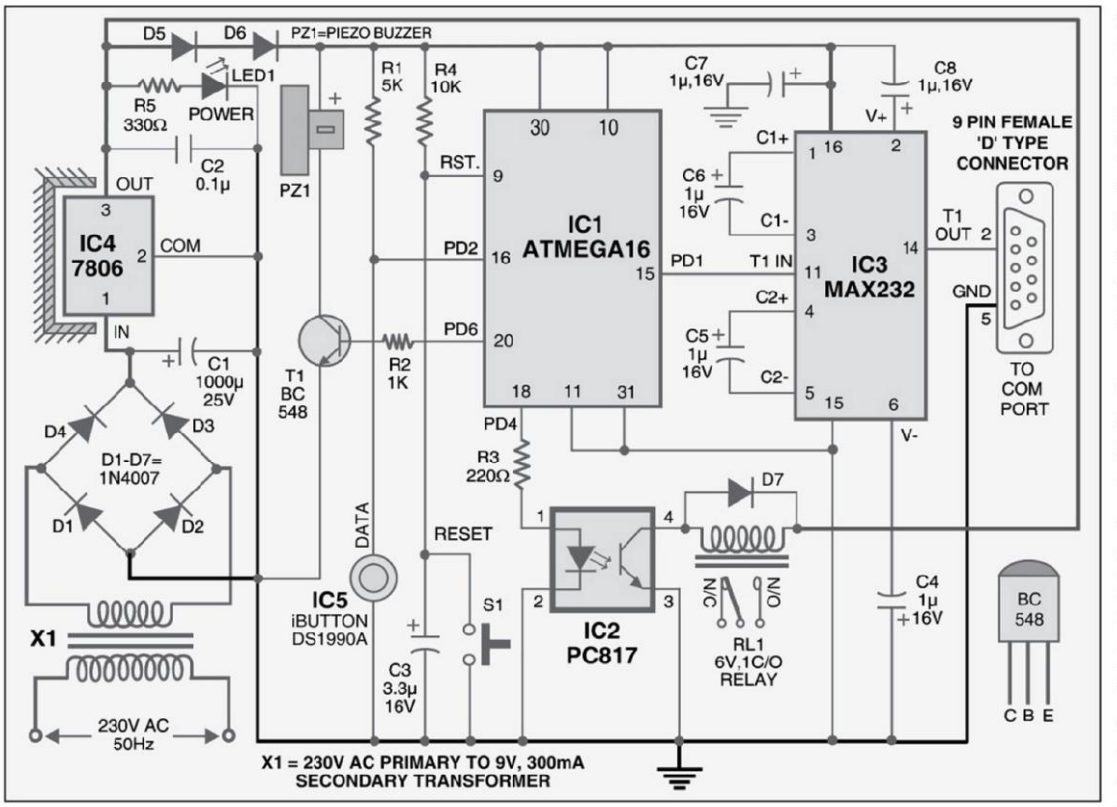


Fig. 4: Circuit of the secure digital access system using iButton

Piezobuzzer PZ1 is used as an audible indicator for true, fake, random touches and system halt. It is controlled from port pin PD6 of the microcontroller with the help of transistor T1. The iButton socket is connected to port pin PD2. Pull-up resistor R1 is used as required from the 1-wire protocol. Port pin PD4 controls the relay operation through optocoupler PC817 (IC2). The door locking mechanism, say, for a slide door, is connected to the contacts of relay RL1, which closes after some time automatically.

Whenever someone attempts to gain access by touching the iButton (IC5) using his own iButton, the firmware inside the AVR reads its unique ID and matches with the ID in the firmware. If the ID matches, port pin PD4 goes high, the internal LED of the optocoupler (IC2) glows and relay RL1 energises for the predefined time. Simultaneously, the buzzer sounds to indicate grant of access.

Thereafter, relay RL1 de-energises. The buzzer gets the PWM signal to produce sound from port pin PD6 of the microcontroller. The 1-wire communication is done through interrupt port pin PD2 (INT0), so the AVR does not have to poll the pin but respond when any change in signal is detected on the interrupt pin. Switch S1 is used for manual reset.

Port pin PD1 of the microcontroller is used to interface with the hyper terminal of the PC through RS-232 interface MAX232 IC (IC3) for iButton verification and checking. The microcontroller provides a transmit channel for serial data transfer. Transmit data pin (TXD) is specified at port pin PD1. The microcontroller is connected to T1 IN (pin 11) of MAX232. T1 OUT (pin 14) of IC3 is connected to pin 2 of the COM port

connector. The signals provided on these pins are TTL-level and must be boosted and inverted through a MAX232 converter to comply with the RS-232 standard.

The MAX232 has two internal charge pumps that convert +5V into ±10V (unloaded) for RS-232 driver operation. The first converter uses capacitor C6 to double the +5V input to +10V on capacitor C8 at pin 2. The second converter uses capacitor C5 to invert +10V to -10V on capacitor C4 at pin 6.

The power supply for this circuit is derived from 230V, 50Hz AC mains. Transformer X1 steps down 230V, 50Hz AC mains to deliver a secondary output of 9V, 300 mA. The transformer output is rectified by a full-wave rectifier comprising diodes D1 through D4, filtered by capacitor C1 and regulated by IC 7806 (IC4). Capacitor C2 bypasses the ripples present in the regulated supply. LED1 acts as the power indicator and R5 limits the current through LED1.

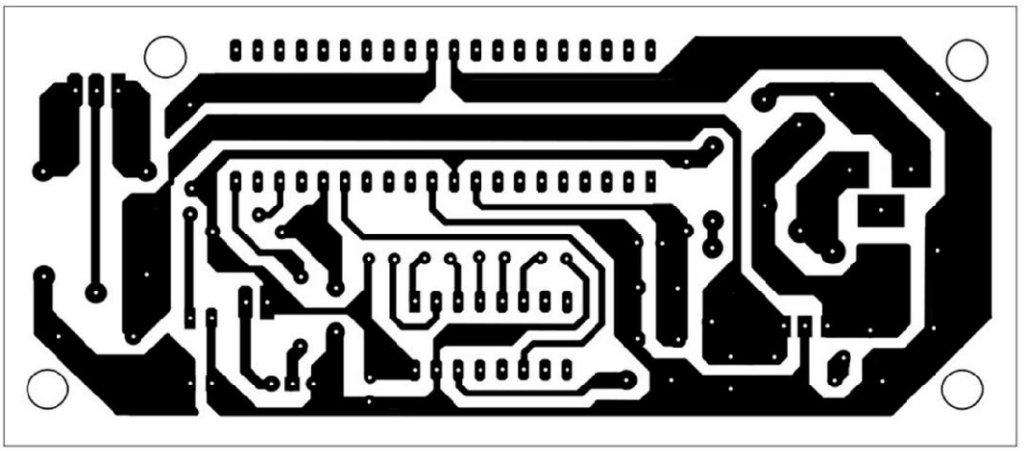


Fig. 5: An actual-size, single-side PCB for the secure digital access system using iButton

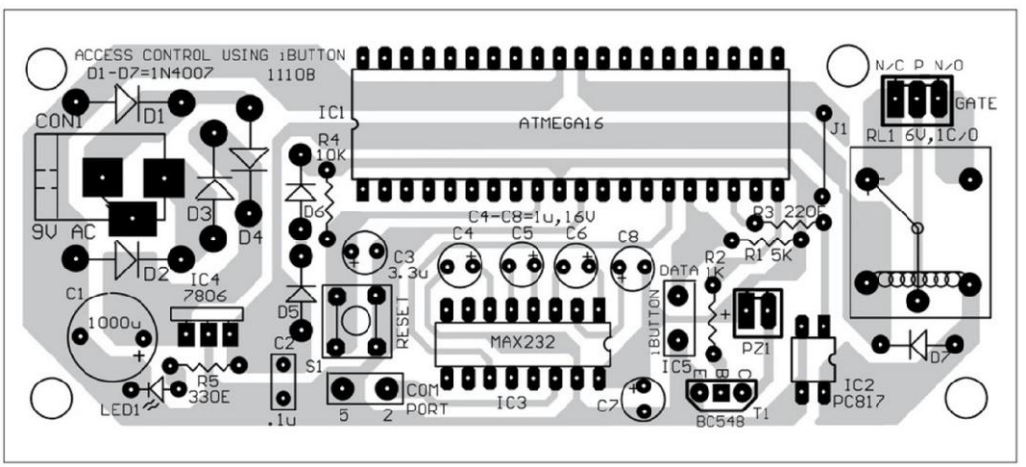


Fig. 6: Component layout for the PCB

Construction and testing

An actual-size, single-side PCB for the secure digital access system is shown in Fig. 5 and its component layout in Fig. 6. Assemble the circuit on a PCB as it minimises time and assembly errors. Carefully assemble the components and double-check for any overlooked error. Connect the assembled circuit to the COM port of the computer. The iButton access code and message is transferred is to the PC through the COM port using the Hyper Terminal program. For interfacing of the Hyper Terminal, refer EFY Oct. 2010 issue.

Software

The software for this project is given at the end of this article. It is written in 'Basic' language and compiled using BASCOM-AVR compiler. The source program is well commented and easy to understand.

Burn the finally obtained Intel hex code file into the AVR's flash memory using a suitable programmer. The microcontroller uses an 8MHz internally generated clock. To activate, program fuse bytes as follows:

Fuse low byte = D4

Fuse high byte = 99

The source program is developed quickly with the help of the BASCOM-AVR library function. The 1wrest, 1wread and 1wwrite functions are used for checking the presence of 1-wire device, reading of the unique ID of iButton and writing to the 1-wire device, respectively. Sound function is used for generating the beep sound from the buzzer. The iButton is connected to hardware interrupt pin INT0 so that any change on the pin can be detected and further processing for 1-wire done.

The program execution starts by initialising the input/output ports, the interrupt pin and its level of detection. Global interrupts are enabled and the interrupt service routine is ready to be executed when any interrupt is received. As soon as the normal program execution is interrupted, an interrupt is issued and the control of execution enters the interrupt service routine, where iButton is given reset command first and then the ROM command to fetch the unique ID of the iButton. This fetched ID is compared with the unique ID programmed in the firmware. If there is a mismatch between the two IDs, a fake parameter counter is incremented and the relay connected to port PD4 de-energises. If the two IDs match, the relay connected to port PD4 energises and simultaneously the buzzer connected to port PD6 sounds. If the unique ID is wrong sequentially five times, the system enters the major warning state where the buzzer sounds continuously and it can only be stopped by resetting the system.

Download source code: http://www.efymag.com/admin/issuepdf/Secure%20Digital%20Access%20System%20Using%20iButton.zip

PARTS LIST

Semiconductors:

IC1	- ATMega16 microcontroller
IC2	- PC817 optocoupler
IC3	- MAX232 RS-232 driver
IC4	- 7806, 6V regulator
IC5	- DS1990A iButton
T1	- BC548 npn transistor
D1-D7	- 1N4007 rectifier diode
LED1	- 5mm LED

Resistors (all ¼-watt, ±5% carbon):

R1	- 5-kilo-ohm
R2	- 1-kilo-ohm
R3	- 220-ohm
R4	- 10-kilo-ohm
R5	- 330-ohm

Capacitors:

C1	- 1000µF, 25V electrolytic
C2	- 0.1µF ceramic disk
C3	- 3.3µF, 16V electrolytic
C4-C8	- 1µF, 16V electrolytic

Miscellaneous:

X1	- 230V AC primary to 9V, 300mA secondary transformer
RL1	- 6V, 1C/O relay
PZ1	- Piezobuzzer
S1	- Push-to-on tactile switch
	9-pin D-type female connector

1WIRERESET.BAS

```
$regfile = "m16def.dat"
$baud = 9600
$crystal = 8000000
$hwstack = 32
' default use 32 for the hardware stack
$swstack = 64
' default use 10 for the SW stack
$framesize = 20
' default use 40 for the frame space
Config Portd = Output
$lib "mcsbyte.lbx"
Config Com1 = 9600 , Synchrone = 0 , Parity = None ,
Stopbits = 1 , Databits = 8 , Clockpol = 0
Config 1wire = Portd.2
'use this pin
Config Int0 = Low Level
Dim Ar(8) As Byte , A As Word , I As Byte , E As
Byte , W As Word
Dim Code1(8)as Byte
Dim Flag As Bit
'flag for correctness
'Code(8 Byte) = {&H01 ,&HDD , &H83 , &H07 , &H13 ,
&H00 , &H00 , &H1E}
Code1(1) = &H01
Code1(2) = &HDD
Code1(3) = &H83
Code1(4) = &H07
Code1(5) = &H13
Code1(6) = &H00
Code1(7) = &H00
Code1(8) = &H1E
On Int0 Int0_int
Enable Interrupts
Enable Int0
'enable the interrupt
E = 0
'flag for number of attempts
W = 65000
Do
Set Portd.4
'initially the relay is off
Loop
Int0_int:
Disable Interrupts
Disable Int0
Flag = 0
Wait 1
1wreset
'reset the device
Print "iButton Access Code:";
```

```
If Err = 0 And E <= 6 Then
'check DS1990A is present
      1wwrite &H33
'present, read code
'read ROM command
      Ar(1) = 1wread(8)
      For I = 1 To 8
            Print Hex(ar(i));
'print output
        Next
        Print
         For I = 1 To 8
             If Ar(i) <> Code1(i) Then
                Set Portd.4
                Flag = 0
'access not allowed
                For A = 10 To 1000
                     Sound Portd.6 , 4 , A
                Next
                Print "Access Code Not Match:Access
Denied";
                Print
                E = E + 1
                Print E;
                Print ":";
                Exit For
             Else
               Flag = 1
             End If
         Next
      If Flag = 1 Then
         E = 0
         Print Err;
         Reset Portd.4
'access allowed
         Sound Portd.6 , 10 , 50000
'siren on port PD6
         Print "--Access Allowed";
         Print
         Wait 2
      End If

Else
    Set Portd.4
'access not allowed
    For A = 50 To 500
                     Sound Portd.6 , 1 , A
                Next
    Print "Rejected";
    Print
    If E = 5 Then
'5 attempts of correctness
               Do
           Sound Portd.6 , 4 , W
'SOUND  pin, duration, pulses
           W = W + 1
           If W > 1000 Then
              W = 10
           End If
           Loop
    End If
End If
'End If
Enable Int0
Enable Interrupts
Return
```

REMOTE-CONTROLLED 6-CAMERA CCTV SWITCHER

■ S. DAS GUPTA

Closed-circuit television (CCTV) uses video cameras to transmit signals to specific monitors. It differs from broadcast television in that the signal is not openly transmitted, though it may employ point-to-point wireless links. CCTV is often used for surveillance in areas that need security, such as banks, casinos, shops, departmental stores and airports or military installations.

Here is a remote-controlled CCTV switcher circuit to monitor six cameras on a single or dual monitor/TV. The salient features of this switcher are:

1. Cameras: Six cameras with one-way audio (audio incoming only)
2. Operation: Fully remote-controlled (infrared commander)
3. Front display: LCD to display the camera selected and all other functions/parameters
4. Camera selection: Manual (you can select any camera of your choice by pressing number keys or camera up/down keys on the remote) and auto (the switcher selects the cameras one by one and holds videos of each camera till the preprogrammed time)
5. Auto mode: You can change 'on' time of each camera (hold time) individually through the remote. Time can be programmed from 0 to 255 seconds.
6. Camera bypass: You can set any camera in bypass mode by entering 0-second 'on' time for cameras which you want to bypass or don't want to watch during auto mode.
7. Audio: One-way audio. In this model, you can hear the audio from the camera selected by the switcher.
8. Video: Two video-out RCA sockets with 75-ohm impedance 1Vp-p are available at the rear panel of the master unit. You can connect two monitors in these sockets.
9. Other features: Wide input voltage range, long-range infrared remote commander, microcontroller-based design sup-

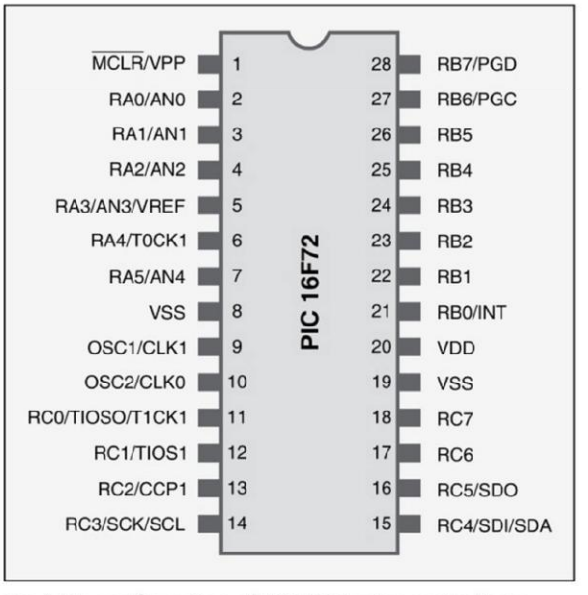


Fig. 1: Pin configuration of PIC16F72 microcontroller in PDIP package

PARTS LIST

Semiconductors:

IC1	- PIC16F72 microcontroller
IC2-IC4	- CD4066 quad bilateral CMOS switch
IC5	- UTC820 power amplifier
T1-T6	- BC547
IC6	- 78S12 voltage regulator
IC7	- 7805 voltage regulator
X_{TAL1}	- 4MHz crystal
D1-D4	- 1N4007 diode
LED1-LED6	- LED, 5mm red
LED7	- LED, 5mm green
IRx	- TSOP1738 IR receiver

Resistors (all ¼-watt, ±5% carbon unless stated otherwise):

R1	- 10-kilo-ohm
R2, R13	- 2.2-kilo-ohm
R3-R8	- 470-ohm
R9, R10	- 75-ohm
R15	- 47-ohm, 1-watt
R11, R16	- 47-ohm
R12	- 1-ohm
R14	- 1-kilo-ohm
R17-R22	- 1-kilo-ohm
VR1	- 10-kilo-ohm
VR2	- 10-kilo-ohm, log, potmeter

Capacitors:

C1	- 1000µF, 35V electrolytic
C2	- 220µF, 40V electrolytic
C3, C10, C13, C16	- 100µF, 25V electrolytic
C4, C5	- 33pF ceramic
C6, C8, C14, C18	- 0.1µF mylar
C7, C9	- 47µF, 16V electrolytic
C11	- 680pF ceramic
C12	- 22µF, 25V electrolytic
C15	- 220µF, 25V electrolytic
C17	- 10µF, 25V electrolytic

Miscellaneous:

X1	- 0-15V, 2A secondary transformer
S1	- DPDT rocker switch
SK1-SK6	- 5-pin DIN socket
SK7, SK8	- RCA socket
LCD	- 16×2 LCD with backlight
Remote	- No. 6710V00079B (type LG TV)
LS	- 8-ohm, 6.4cm speaker
CON1	- 7-pin bergstrip connector (male/female)
CON2-CON7	- 2-pin SIP connector (male/female)
CON8	- 12-pin bergstrip connector (male/female)

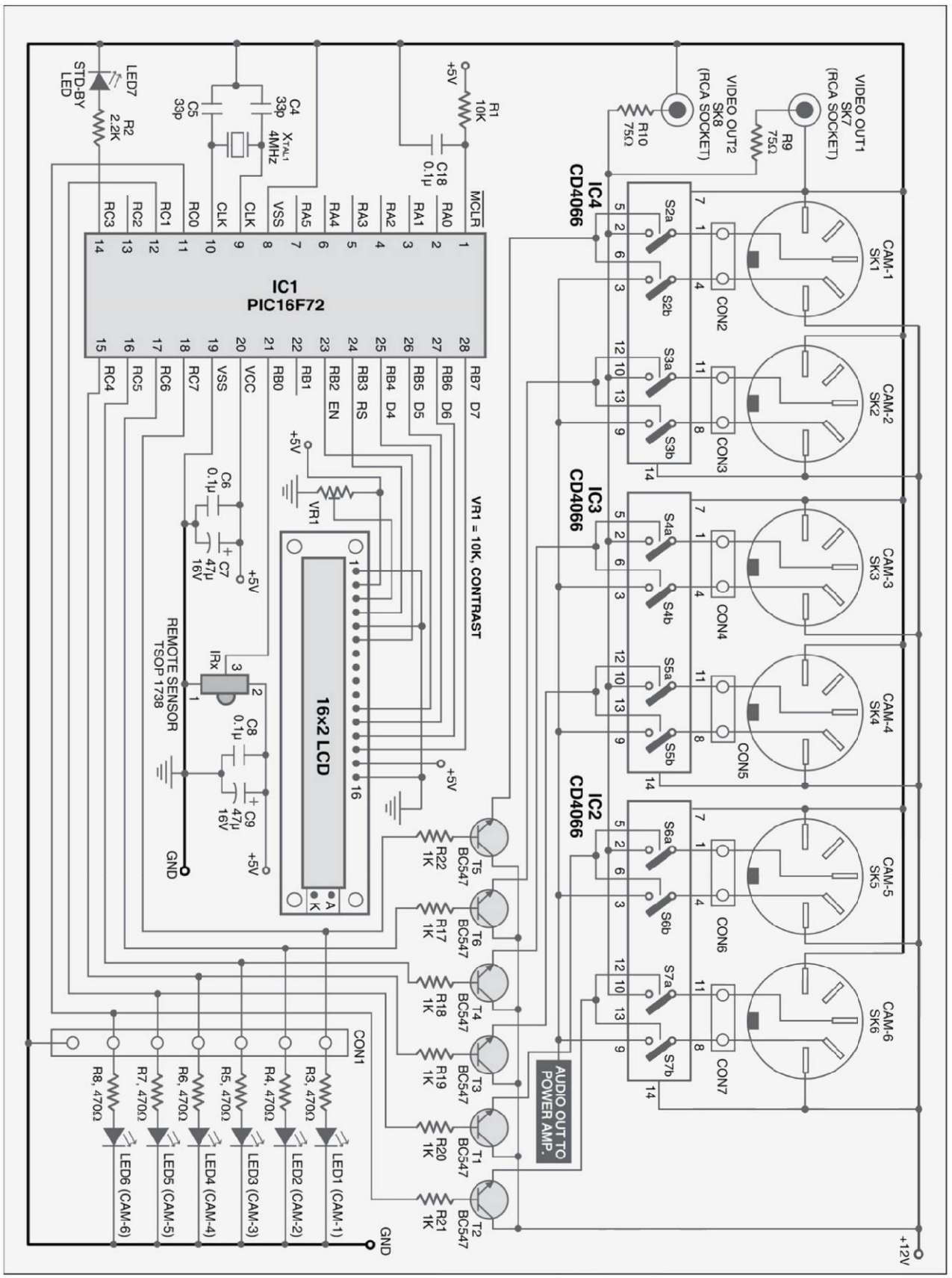


Fig. 2: CCTV switcher circuit

ported with CMOS digital integrated circuits, liquid-crystal display to monitor the status and bilateral digital MOS switches for switching the analogue video/audio signals from cameras

At the heart of the circuit is Microchip's PIC16F72 microcontroller, which controls all the functions described in the design. Its pin configuration is shown in Fig. 1.

TABLE I

Pin Details of CCTV Camera DIN Socket

Socket	Camera	Signal	Switch (IC)	Enable pins to microcontroller
SK1	Camera-1	Video	S2a (IC4)	Port (RC7)-pin 18
SK1	Camera-1	Audio	S2b (IC4)	Port (RC7)-pin 18
SK2	Camera-2	Video	S3a (IC4)	Port (RC6)-pin 17
SK2	Camera-2	Audio	S3b (IC4)	Port (RC6)-pin 17
SK3	Camera-3	Video	S4a (IC3)	Port (RC5)-pin 16
SK3	Camera-3	Audio	S4b (IC3)	Port (RC5)-pin 16
SK4	Camera-4	Video	S5a (IC3)	Port (RC4)-pin 15
SK4	Camera-4	Audio	S5b (IC3)	Port (RC4)-pin 15
SK5	Camera-5	Video	S6a (IC2)	Port (RC1)-pin 12
SK5	Camera-5	Audio	S6b (IC2)	Port (RC1)-pin 12
SK6	Camera-6	Video	S7a (IC2)	Port (RC0)-pin 11
SK6	Camera-6	Audio	S7b (IC2)	Port (RC0)-pin 11

The circuit

Fig. 2 shows the main control circuit of the camera switcher. In this circuit, PIC16F72 microcontroller controls all the functions such as interfacing/initialising the LCD to display all the desired characters on the screen, decoding the RC-5 command on pressing the required keys of the remote control and providing high logic to enable corresponding bilateral CMOS analogue switches to connect the video signal from cameras to the video-output socket and audio signals to the power amplifier of the corresponding camera selected from the remote. Here we have used the remote control of an LG TV (remote code No. 6710V00079B). It is readily available in the market.

TABLE II

Truth Table for Port C

PORT	RC7	RC6	RC5	RC4	RC1	RC0	Camera-ON
Port C	1	0	0	0	0	0	Camera-1
Port C	0	1	0	0	0	0	Camera-2
Port C	0	0	1	0	0	0	Camera-3
Port C	0	0	0	1	0	0	Camera-4
Port C	0	0	0	0	1	0	Camera-5
Port C	0	0	0	0	0	1	Camera-6

A remote sensor is connected to port RB0 (pin 21), called interrupt port, of the controller and the software initialises interrupt routines along with timer-0 interrupt to decode the RC-5 command. The RC-5 protocol was described in March 2007 issue of EFY, so it is not discussed here.

Any transmission from the remote consists of two start bits, one toggle bit, 5-bit address and 6-bit command. Here we have not used the 5-bit address. A biphase-modulated bit can be thought of as two separate bits that are always the inverse of each other. A logical zero is represented by a '10' pattern on the IR input, while a logical one is represented by a '01' pattern. That is basically used to decode the received message.

Toggle bit is a particular property of the RC-5 protocol. This bit changes polarity every time you press a key and will remain unchanged as long as you hold the key. That enables the receiver to detect released keys, which helps to eliminate key bounces. We have also used this bit to detect so that if any key is kept pressed for a long time, the routine detects the command once, keeps the value in a general-purpose register called 'passdummy' (refer to '.asm' file), and does not process and store any further command till the remote key is released. The stored RC-5 command value in register 'passdummy' is used to perform the necessary function given in the source code.

The LCD shows the camera number selected from the remote and mode of operation (auto/manual).

For camera-hold time set for auto mode and other display text information, please see the LCD screenshot in Fig. 3. Use of the LCD makes the project more user-friendly.

All the cameras are to be connected to SK1 through SK6 (5-pin DIN connector) as shown in the circuit diagram. The extreme right and left pins are for 12V positive and negative supply (to be fed from the regulated

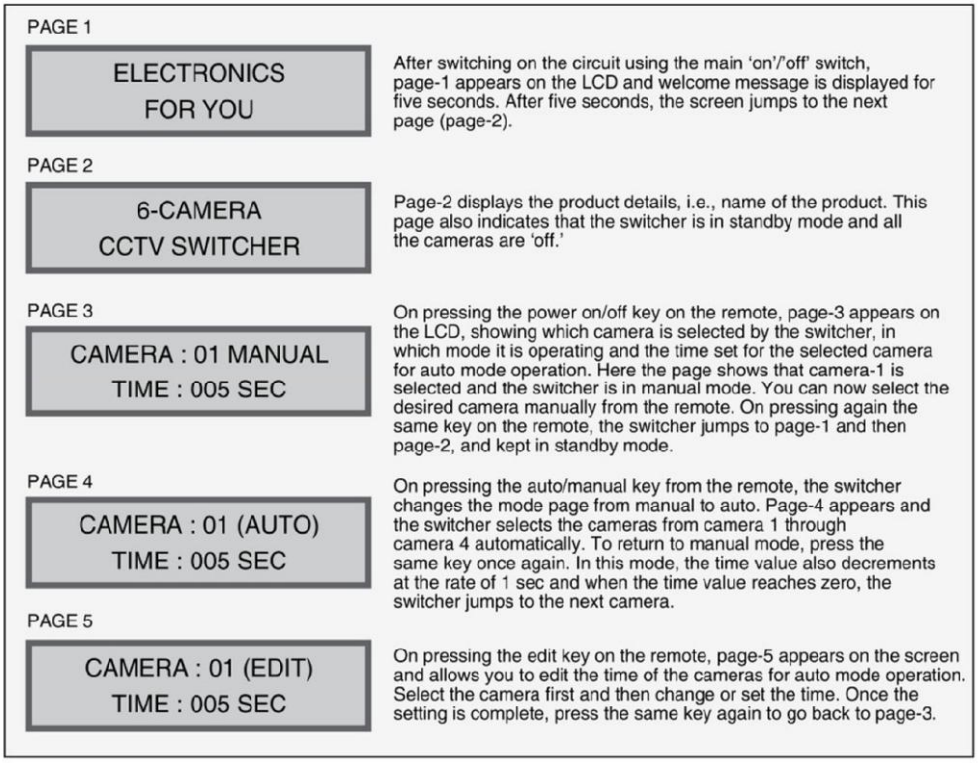


Fig. 3: Screenshots of LCD

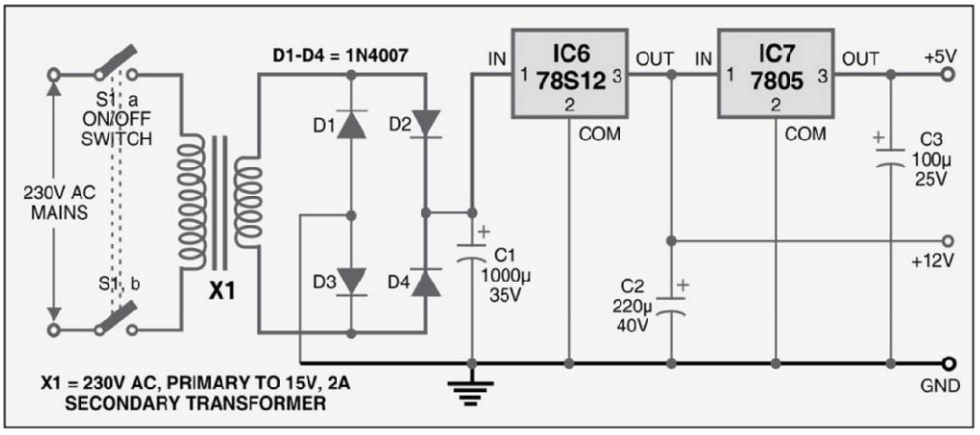


Fig. 4: Circuit for regulated power supply

power supply of the main switcher) for the built-in camera module. The 12V power supply is permanently connected to all the cameras. Audio and video signals from the cameras are selected using switch pairs S2a-S2b through S7a-S7b (built inside IC2 through IC4) and fed to the audio power amplifier and video output sockets, respectively.

Each IC has four CMOS bilateral switches for transmission or multiplexing of analogue or digital signals. It is pin-to-pin compatible with IC CD4016, but has much lower 'on' resistance, which is relatively constant over the input signal range.

Video and audio signals from SK1 (camera 1) are connected to S2a and S2b, respectively, and both the enable pins (to switch on both the switches) are shorted out and connected to port RC7 (pin 18) of the microcontroller. Similarly, audio and video signals from SK2 through SK6 are connected to IC2 through IC4 and then to Port C of IC1 as shown in Table I.

Enable pins of switch pairs S2a-S2b, S3a-S3b, S4a-S4b, S5a-S5b, S6a-S6b and S7a-S7b of IC2 through IC4 are shorted to each other and fed to the corresponding port of Port C as shown in Table I. The microcontroller provides high logic at Port-C pins to switch on a particular camera. The truth table for Port C is shown in Table II.

Immediately after switching on the mains, the standby LED (LED7) connected to Port-C pin RC3 glows and the LCD shows the welcome page (refer to the LCD screenshot).

On pressing the power key on the remote, the standby LED turns off and the LCD shows which camera is 'on,' in which mode it is now and default hold time for the corresponding camera in 'Auto' mode.

As described above, pins of Port C (RC0, RC1, RC4-RC7) are connected to IC2 through IC4 for selecting the camera to be switched on. LED1 through LED6 are used to check the logic status of Port C. Pressing keys '1' through '6' on the remote makes the corresponding LEDs (LED1 through LED6) glow, indicating 'on' condition of the cameras (CAM-1 through CAM-6).

Here a 16-character, 2-line LCD is used in 4-bit mode. Data lines D4 through D7 of the LCD are connected to Port-B pins RB4 through RB7. Enable (EN) and reset (RS) pins are connected to RB2 and RB3 pins, respectively, and the read/write (R/W) pin is connected to ground as we have used the LCD in writing mode only. Preset VR1 (10-kilo-ohm) is used for setting the LCD character contrast.

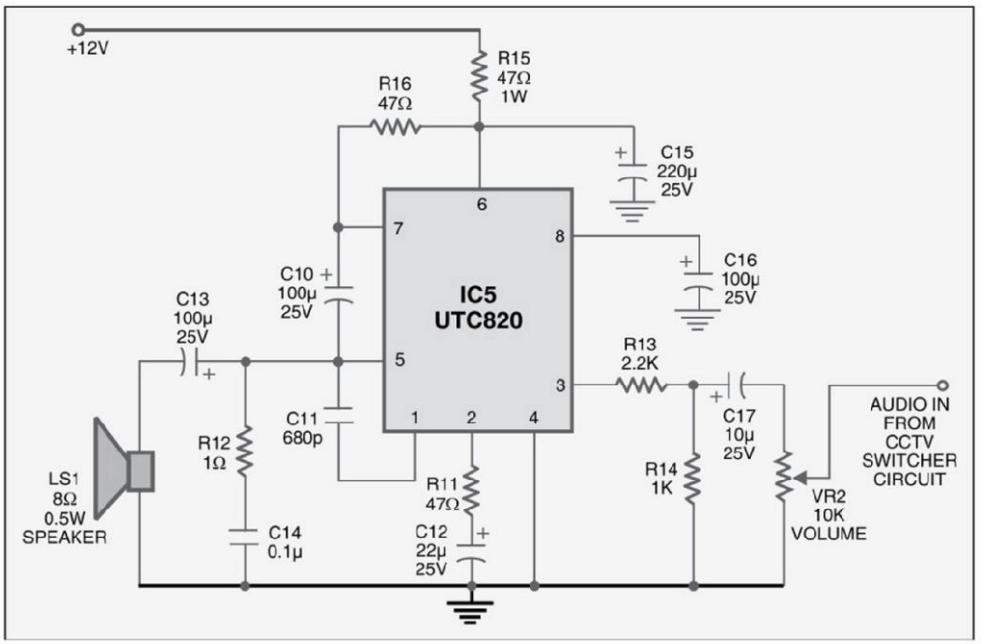


Fig. 5: Circuit diagram of power amplifier

Power supply

A simple circuit for regulated supply (shown in Fig. 4) is used here. The +12V regulated output is used to power all the six cameras, IC2 through IC4 and IC5 and fed to another 7805 voltage regulator (IC7) to get the regulated +5V output for the microcontroller (IC1) and the LCD.

The current rating of the transformer should be around 2A, depending on the current consumption of the cameras. Use a good-quality heat-sink for both the regulators.

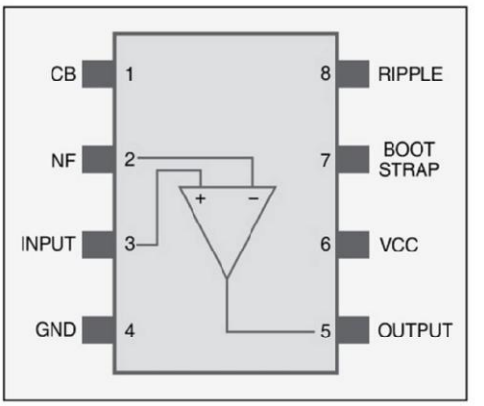


Fig. 6: Pin diagram of UTC820 audio power amplifier

Power amplifier

A simple power amplifier (see Fig. 5) is used to amplify the sound signal from the cameras and hear it on a small speaker.

The audio power amplifier is built around IC UTC820 (IC5). UTC820 is a monolithic integrated circuit audio amplifier delivering an output of 1.2 watts at 9V on an 8-ohm speaker load with 10 per cent total harmonic distortion (THD) and good ripple rejection. It is designed for audio-frequency class-B amplifier with a wide operating supply voltage range of 3V to 14V and minimum external components.

The pin configuration of IC UTC820 is shown in Fig. 6. Components at pin 2 of the IC5 decide closed-loop voltage gain (dB) of the amplifier. Here a 47-ohm resistor (R11) and a 22µF capacitor (C12) are used and, with these values, about 45dB closed-loop voltage gain is achieved. You can reduce the component values for more gain but it's not required here as maximum gain of 75 dB is possible. Capacitor C10 (100µF) connected between pins 5 and 7 of IC5 is used for bootstrap and improving the low-frequency signal. The amplifier output at pin 5 is connected to the speaker via C13. The audio signal from the cameras is fed to input pin 3 of the IC via volume control VR2 (10-kilo-ohm), capacitor C17 and resistor R13. Here the output power is limited to about 500 mW at 8-ohm load approximately, so use of a good-quality 6.4cm speaker is recommended.

Functions of remote control keys

The functions of various remote keys are shown in Fig. 7.

Power on/off key. Press it once to switch on the switcher and press again to turn off or keep the switcher in

standby mode and vice versa. (AC mains should be 'on' in both the cases.)

Camera selection key. Select any of the six cameras manually by pressing any of the numerical keys '1' through '6' once. For example, to select camera-3 press '3' on the remote, and to select camera-5 press '5' on the remote.

Auto/manual. By default, the microcontroller sets the mode of selection to manual. (In that case, you have to select the cameras by pressing the number keys or the camera up/down keys.) To change to automatic selection (auto) mode, press 'Mute' key and the switcher selects the cameras one by one automatically. Pressing this key again will return the mode of selection to manual, and vice versa.

Camera up key (PR∆ key). By pressing the camera-up key, the switcher selects the next camera. For example, if the LCD is showing camera-2, pressing this key once makes the switcher jump to camera-3. Each successive pressing of this key will make the switcher jump to the next camera.

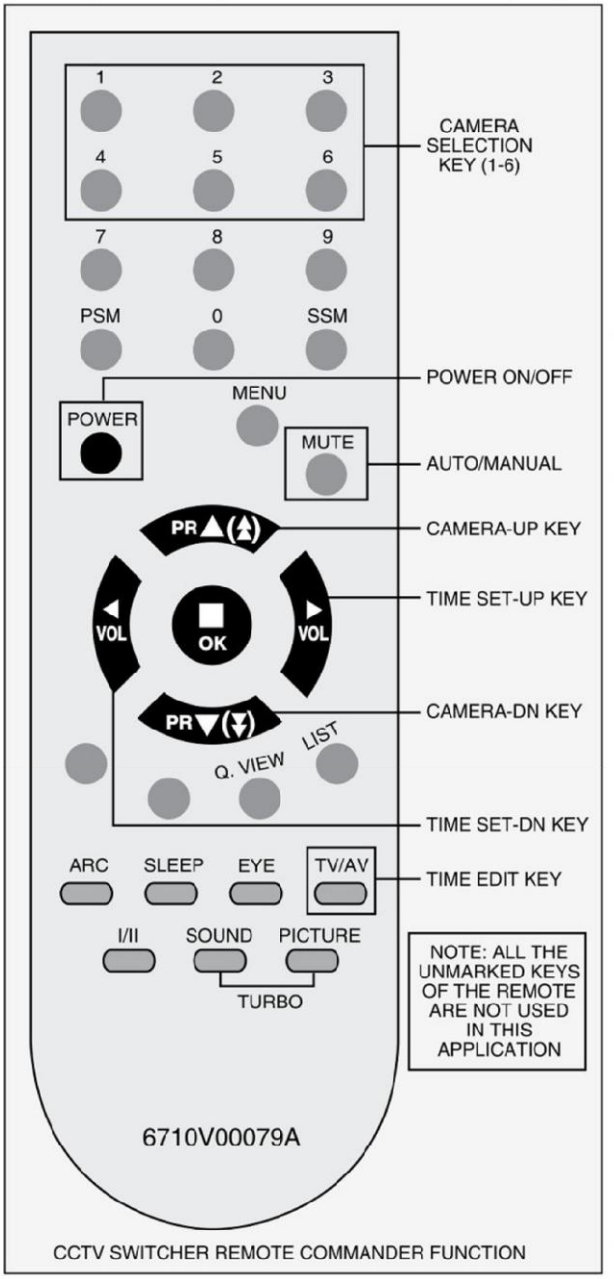


Fig. 7: Remote control of LG TV

Camera down key (PR∇ key). By pressing the camera-down key, the switcher selects the preceding camera. For example, if the LCD is showing camera-4, pressing this key once makes the switcher jump back to camera-3. On each successive pressing of this key, the switcher jumps back to the previous camera.

Time set up/down key (Vol+/Vol– key). In auto mode, the switcher selects the cameras one by one and the picture of the selected camera is held up for a certain time. By default, the microcontroller sets this hold time as 5 seconds. You can change this time for each camera individually from 0 to 255 seconds.

To change this time, press the edit key (TV/AV) once. The switcher goes to the edit mode. Select the camera (for which you want to change the time) by pressing the camera up/down key. On each successive pressing of the time set up key, the time increments up to 255 seconds maximum.

On each successive pressing of the time set-down key, the time decrements up to zero. If you set 0-second time for any camera, the switcher bypasses this camera during selection in auto mode. For example, if you set the time as 0-second for camera-3, in auto mode, the switcher selects camera-1, camera-2, camera-4, camera-5 and camera-6. Here the switcher bypasses camera-3. This function is useful if you want to skip certain cameras.

Edit key (TV/AV). To change the camera hold-up time for 'auto mode,' enter 'edit' mode first by pressing this key once and then set the desired time by pressing the time up/down key as described above. To exit edit mode after setting the time, press the same key on the remote once again. The controller goes back to manual mode.

Testing and troubleshooting

After assembling the circuit without microcontroller, apply the power using 'on'/'off' switch S1. Check the DC voltage at pin 3 of regulator IC7. It should be +5V. Check the same voltage at pins 1 and 20 of the base of IC1.

Now, switch off the power, insert the microcontroller in its base and recheck all the connections from the PCB to

the LCD and IR sensor TSOP1738. On applying the power, page-1 and page-2 text should display on the LCD screen. Using the remote, switch on the switcher and change the functional status on the LCD by referring to CCTV switcher remote commander (shown in Fig. 7) and LCD screenshots (shown in Fig. 3).

Adjust VR1 for maximum contrast on the LCD. If you don't get any output on the LCD, troubleshoot as follows:

1. No output on the LCD. Switch off the power and check continuity between the LCD and various pins of IC1 and regulator IC as shown in the schematic diagram. If the connections are Ok, switch on the power, vary VR1 and check the voltage at pin 3 of the LCD (0V to 5V).

2. No backlight on the LCD. Check the voltage at pin 15 (+5V) and pin 16 (0V) of the LCD.

3. Page 1/page 2 is shown on the screen but the remote commander doesn't work (nothing happens on pressing functional keys of the remote). Check the supply at pin 2 of the IR sensor. It should be +5V. If +5V is available at pin 2, check the voltage at pin 3 of the IR sensor. If there is no voltage at pin 3, replace the IR sensor or check the PCB, etc for shorting.

Once the LCD shows page-1 and page-2 text properly and the remote commander is working, switch off the power and then switch it on again using S1. Now the LCD should show page-1 and page-2 as shown in Fig. 3. At the same time, pin 14 of the MCU should go high (+5V) and the power standby LED should glow. Now press 'Power' key on the remote. Pin 14 of the MCU should go low and the standby LED should turn off, and vice versa, on pressing the same key again. If this doesn't happen, replace the microcontroller.

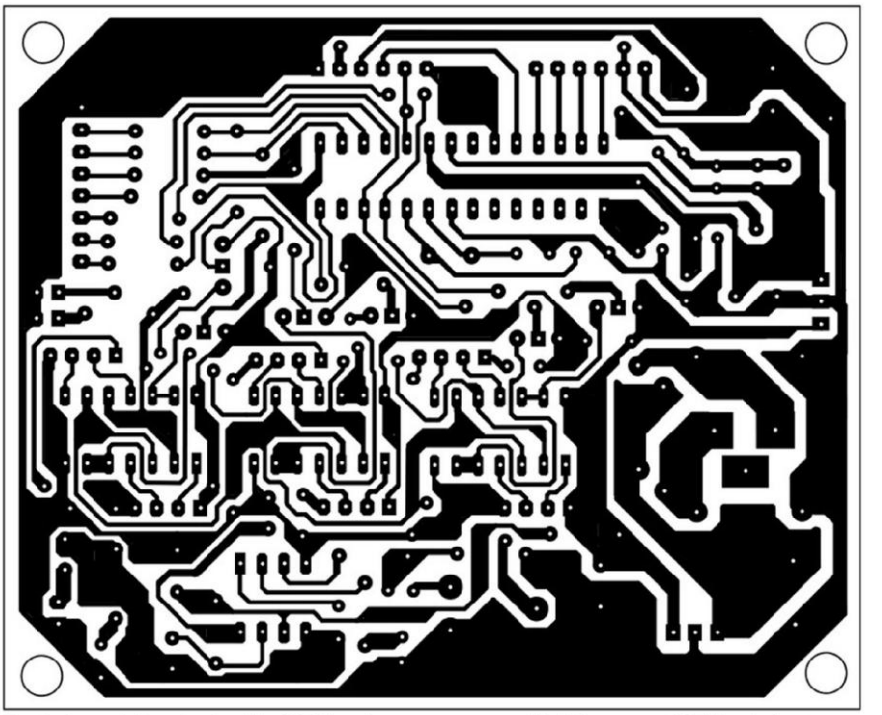


Fig. 8: Actual-size, single-side PCB for the remote-controlled 6-camera CCTV switcher

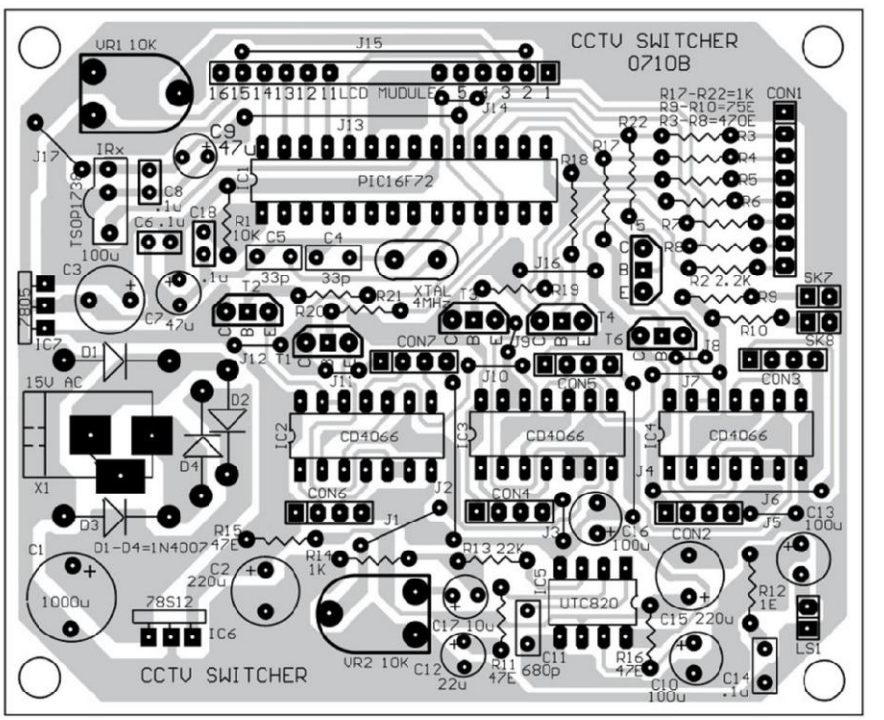


Fig. 9: Component layout for the PCB

Connect LED1 through LED6 to connector CON1. Press 'Power' key on the remote to switch on the switcher. Page-3 text should appear on the LCD, showing camera-1 is 'on,' the switcher is in manual mode and the default time of camera hold for auto mode. At the same time, Port-C pin RC7 should go high and LED1 should glow. Select other cameras either by pressing keys 1 through 6 or camera up/down keys of the remote

and check whether the corresponding LEDs (LED1 through LED6) are glowing (refer to Port-C truth table). Simultaneously, page-3 should show the corresponding camera number accordingly. If the LEDs don't glow according to the truth table, replace the microcontroller.

When camera-1 is selected, the LCD should show camera-1 and LED1 should glow. If the monitor does not show the video of camera-1, or you hear no audio from the power amplifier, check 5V logic at pins 5 and 6 of IC4. If there is no voltage at these pins, check the availability of 0.6V at the base of transistor T5, +5V at its emitter and +12V at its collector. Follow the same procedure for all other cameras and their corresponding switching circuits.

Software

The source program is written in Assembly language and compiled using MPASM tool suite. The explanation of the Assembly source code is available on the link given below.

Construction

A single-side, solder-side PCB layout for the microcontroller-based remote-controlled six-camera CCTV switcher is shown in Fig. 8 and its component layout in Fig. 9.

Connect CCTV cameras in the sockets provided (SK1 through SK6). Also connect the LCD and remaining components (as shown in Fig. 2) and +5V and +12V from the power supply circuit. Now a welcome message will appear. Press 'Power' key on the remote to start the switcher.

Download source code: http://www.efymag.com/admin/issuepdf/Microcontroller%20Based%20CCTV%20Switcher.zip

PIC MICROCONTROLLER-BASED ELECTRONIC LOCK

■ ANSHUMAN BEZBORAH

An electronic lock allows activation of an electric appliance only on entering the correct password. Here we present such an electronic locking system in which a PIC16F877A microcontroller plays the role of the processing unit. The MCU is interfaced with a 4×4 matrix keypad and a 16×2 LCD to form the user interface. Using this circuit, you can make any electrical appliance password-protected. It can also be used as an

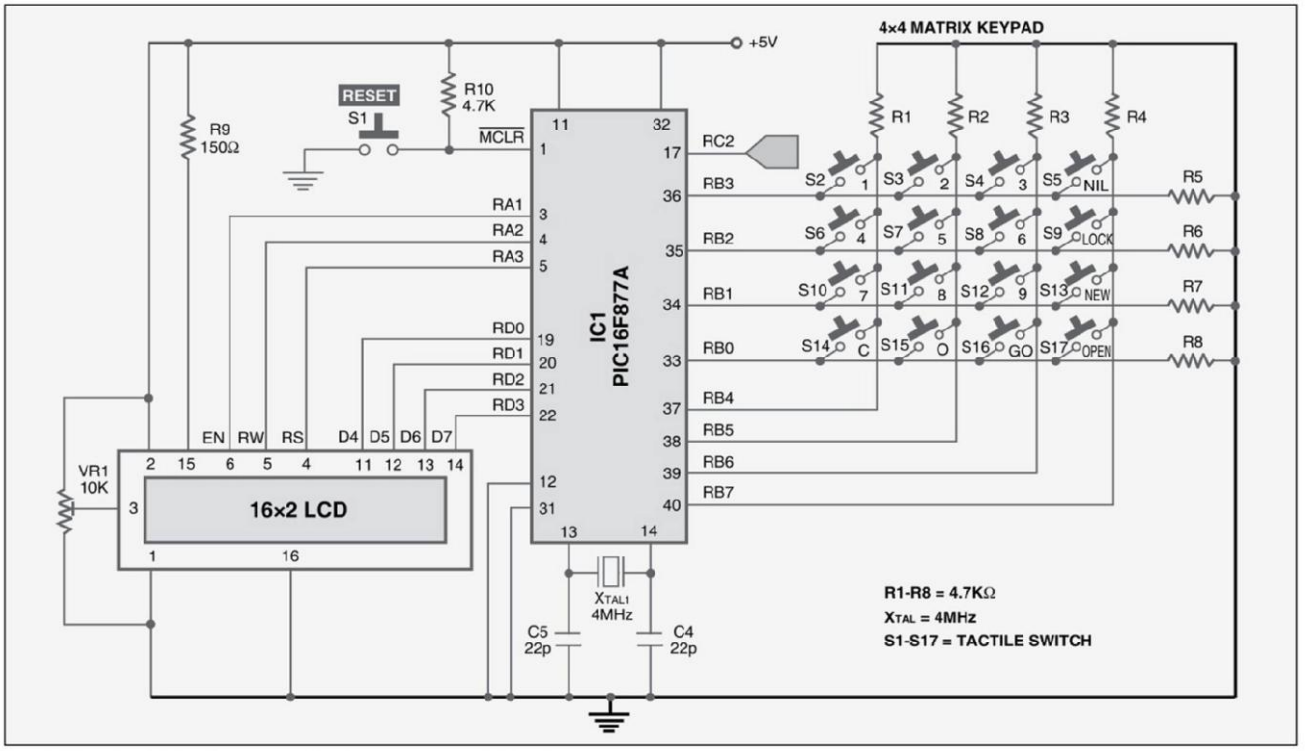


Fig. 1: Circuit of PIC microcontroller-based electronic lock

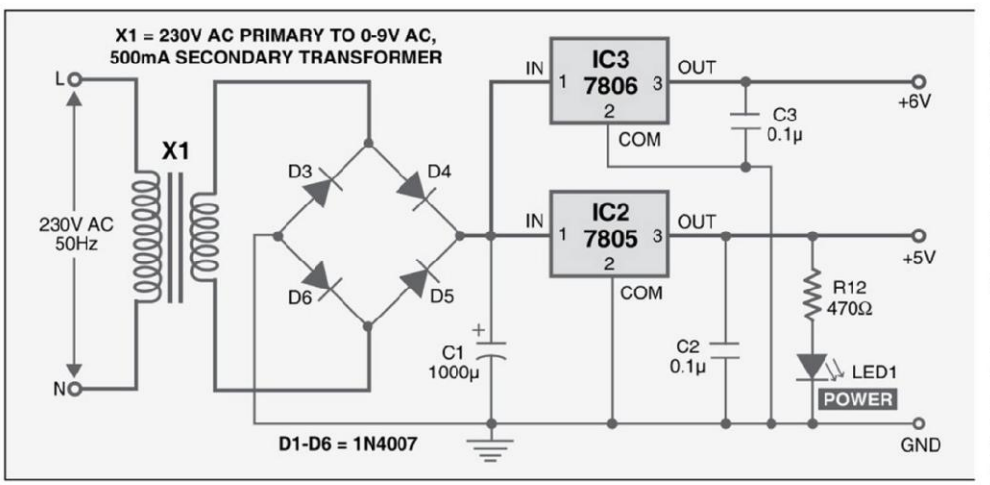


Fig. 2: Power supply circuit

electronic door lock by interfacing the output of the circuit with an electrically actuated door lock. The system turns on the appliance on entering a four-digit password set by the user.

Circuit description

Fig. 1 shows the circuit of the PIC microcontroller-based electronic lock. It can be divided into five

PARTS LIST

Semiconductors:

IC1	- PIC16F877A microcontroller
IC2	- 7805 voltage regulator
IC3	- 7806 voltage regulator
LED1	- 5mm Light-emitting diode
D1	- 1N4148 diode
D2-D6	- 1N4007 diode
T1	- SL100 transistor

Resistors (all ¼-watt, ±5% carbon unless stated otherwise):

R1-R8, R10	- 4.7-kilo-ohm
R9	- 150-ohm
R11	- 10-kilo-ohm
R12	- 470-ohm
VR1	- 10-kilo-ohm preset

Capacitors:

C1	- 1000µF, 25V electrolytic
C2, C3	- 0.1µF ceramic
C4, C5	- 22pF ceramic

Miscellaneous:

X_{TAL}	- 4MHz crystal oscillator
X1	- 230V AC primary to 0-9V, 500mA secondary transformer
	- HD44780-based 16×2 LCD
RL1	- 6V, 1C/O relay
S1-S17	- Push-to-on tactile switch

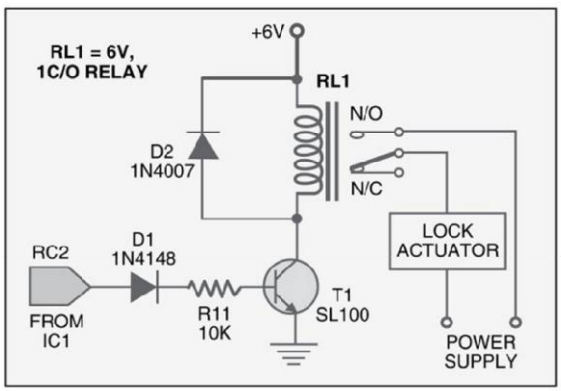


Fig. 3: Relay driver circuit

sections: input (4×4 matrix keypad), processing unit (PIC16F877A MCU), appliance controller (relay driver), display (16×2 LCD), and power supply.

PIC16F877A MCU. The PIC16F877A is an 8-bit microcontroller based on reduced instruction set computer (RISC) architecture. It has 8k×14-bit flash program memory, 368 bytes of RAM and many other internal peripherals like analogue-to-digital converter, USART, timers, synchronous serial port, compare captures and pulse-width modulation modules, EEPROM and analogue comparators.

The job of the MCU in this project is to receive signals from the input device (keypad) and take corresponding actions. Whenever any key is pressed on the keypad, the software program in the MCU identifies the pressed key and accordingly turns on or turns off the appliance. Simultaneously, it also displays a message on the LCD screen.

4×4 matrix keypad. A 4×4 matrix keypad is used to give commands and the password to the MCU. It consists of 16 keys (S2-S17) arranged in the form of a square matrix of four rows and four columns. Each key in the matrix is labeled according to the operation assigned to it. The connections from the pin-outs of the keypad to the MCU pins are shown in Fig. 1. Rows 1 through 4 are connected to pins RB3, RB2, RB1 and RB0 of Port B of the MCU, respectively. Columns 1 through 4 are connected to pins RB4 through RB7 of Port B, respectively.

16×2 LCD. A Hitachi HD44780 16×2 LCD is used to display various messages. It also displays an asterisk mark (*) for each digit of the password entered. Control lines EN, RW and RS of the LCD module are connected to pins RA1, RA2 and RA3 of Port A of the MCU, respectively. Commands and the data to be displayed are sent to the LCD module in nibble mode from Port D of the MCU. The higher four data bits of the LCD (D4 through D7) are connected to the lower nibble of Port D (RD0 through RD3) of the MCU.

Relay driver. RC2 pin of Port C of the MCU is interfaced with the relay driver circuit (shown in Fig. 3) to switch on or switch off the AC load (appliance). A relay driver circuit is nothing but a simple electronic circuit that drives an electromechanical relay. In this project, a 6V, single-changeover relay is used for switching the appliance 'on' or 'off.' Transistor SL100 plays the role of the relay driver.

Whenever the user enters the correct password, RC2 pin goes high (RC2=1). Consequently, transistor SL100 is triggered to energise the relay and the appliance turns 'on.' When RC2 is low (RC2=0), the appliance turns 'off.' Free-wheeling diode 1N4007 protects the relay driver circuit from the reverse voltage developed in the relay coil.

You can also use optocoupler MCT2E to isolate the relay driver circuit from the microcontroller circuit. Whenever the user enters the correct password, RC2 pin goes high (RC2=1) and the internal LED of the MCT2E IC glows, which, in turn, triggers the internal transistor of MCT2E.

Power supply. Fig. 2 shows the power supply circuit. The 230V AC mains supply is stepped down to 9V AC using step-down transformer X1. The output from the secondary of the transformer is rectified by a bridge rectifier comprising diodes D3 through D6 and filtered by capacitor C1. The filtered output is regulated by ICs 7805 and 7806 connected in parallel to obtain the required 5V and 6V, respectively.

Software

The software code is written in 'C' language and compiled using Hitech C compiler in MPLAB IDE. MPLAB IDE is a very powerful software development tool for Microchip's MCUs. It can be downloaded from www.microchip.com free of cost. It consists of tools like text editor, assembler, cross compliers and simulator. Hitech C compiler is meant for Microchip's PIC10/12/16 series of MCUs. Its Lite edition comes for free with newer versions of MPLAB IDE like MPLAB v8.2 or v8.3, and it can also be downloaded for free from www.htsoft.com.

The tasks performed by the software are:

1. Identify the key
2. Take the action allotted to the identified key

The key identification is done by identifying the row and the column to which the key belongs. Fig. 1 shows how the keypad is connected to Port B of the MCU. The lower nibble of Port B is declared as output pins (scan lines) and the upper nibble is declared as input pins (return lines). The number 0Fh is written to Port B so that the lower four bits become high and the upper four bits become low.

Functions of Various Keys of the Keypad and Their Labeling

S.No.	Row	Column	Label	Operation/digit entry
1	1	1	1	Digit '1'
2	1	2	2	Digit '2'
3	1	3	3	Digit '3'
4	1	4	Nil	No operation allotted
5	2	1	4	Digit '4'
6	2	2	5	Digit '5'
7	2	3	6	Digit '6'
8	2	4	Lock	Lock or turn off the appliance
9	3	1	7	Digit '7'
10	3	2	8	Digit '8'
11	3	3	9	Digit '9'
12	3	4	New	Change the password
13	4	1	C	Clear or backspace
14	4	2	0	Digit '0'
15	4	3	Go	Should be pressed after entering the password
16	4	4	Open	Open the lock (asks for password when pressed)

Whenever a key is pressed, the upper nibble pin (return line) of Port B, to which the column containing the key is connected, goes high. Thus the column is identified. Column identification is done using a switch-case block in the main program. On identifying the column, the rowfind(int) function is called, which does the job of row identification. To identify the row, scan lines are made low one by one in sequence and status of the return line corresponding to the key is checked. If it becomes low, the key belongs to that scan line or row. The row and column numbers are stored in two global variables 'row' and 'col.' A key debouncing delay of 20 ms is provided in the program by calling the DelayMs(20) function.

After identifying the key, the action( ) function is called in the main program, to perform the action corresponding to the identified key.

The detailed procedure for developing the project using MPLAB IDE, compiling the same using Hitech C compiler and burning the executable hex file to the microcontroller was explained in 'Construction' section of EFY's May 2010 issue.

The above description is available in a file named 'lock.c.' Functions lcd_init( ), lcd_goto(int), lcd_clear( ) and lcd_putch(char) are defined in a file named 'lcd.c' and the DelayMs(int) function is defined in the delay.c file. Add all the three 'C' files—lock.c, lcd.c and delay.c—as source files to the MPLAB IDE project. Save the project file as 'Elock.mcp.' Set configuration bits properly before building the project. Select the oscillator as XT and disable all other features like watchdog timer, power-up timer and brownout detection. After successfully building the project, the Elock.hex file is generated. Burn it into the chip using a suitable programmer, e.g., MPLAB ICD2.

Testing

Once the program is burnt into the chip and the hardware setup is ready, the user can test the system. When the power supply is switched on, message "Welcome" is displayed on the LCD screen. The default password set in the program is 1234.

To turn on the appliance, press 'Open' key. The system will ask for the password. Enter the password as 1234

and press 'Go.' The appliance should turn on (RC2=1) and the message "Password Accepted" should be displayed for two seconds followed by the message "Lock Open."

To turn off the appliance, press 'Lock.' The appliance should immediately turn off (RC2=0) and the message "Lock Closed" should be displayed on the LCD screen.

To set a new password, press 'New' key. The system should ask for the current and new passwords. Press 'Go' each time after you enter the four-digit password. The message "Password Saved" should appear for two seconds, followed by the message "Welcome." Now you can turn on the device by pressing 'Open' and then entering the new password that you have set. Key 'C' acts like 'Backspace' key in a PC's keyboard.

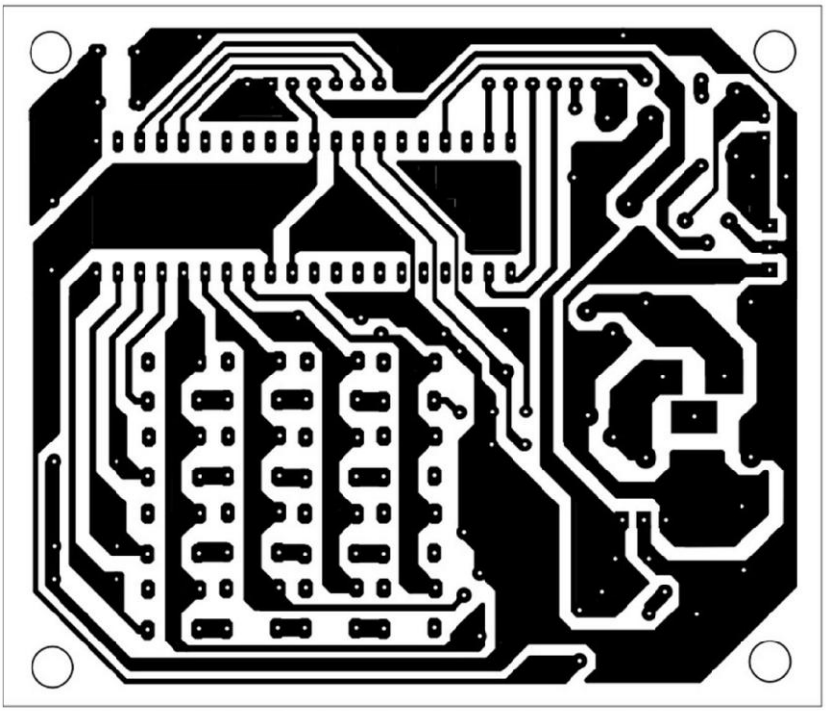


Fig. 4: Actual-size, single-side PCB for the PIC microcontroller-based electronic lock

Construction

A single-side, solder-side PCB layout for the PIC microcontroller-based electronic lock is shown in Fig. 4 and its component layout in Fig. 5.

Connect the 4×4 matrix keypad and 16×2 LCD to PIC16F877A microcontroller as shown in Fig. 1. Complete the remaining connections also as shown in Fig. 1. Connect the appliance to be controlled to RC2 (pin 17) of Port C through the relay-driver circuit as shown in Fig. 3. Connect 6V power supply from the power supply circuit to the relay driver circuit. The 5V supply required by the microcontroller is obtained from the 7805 regulator output.

Download source code: http://www.efymag.com/admin/issuepdf/Electronic%20Code%20Lock%20code%20new.rar

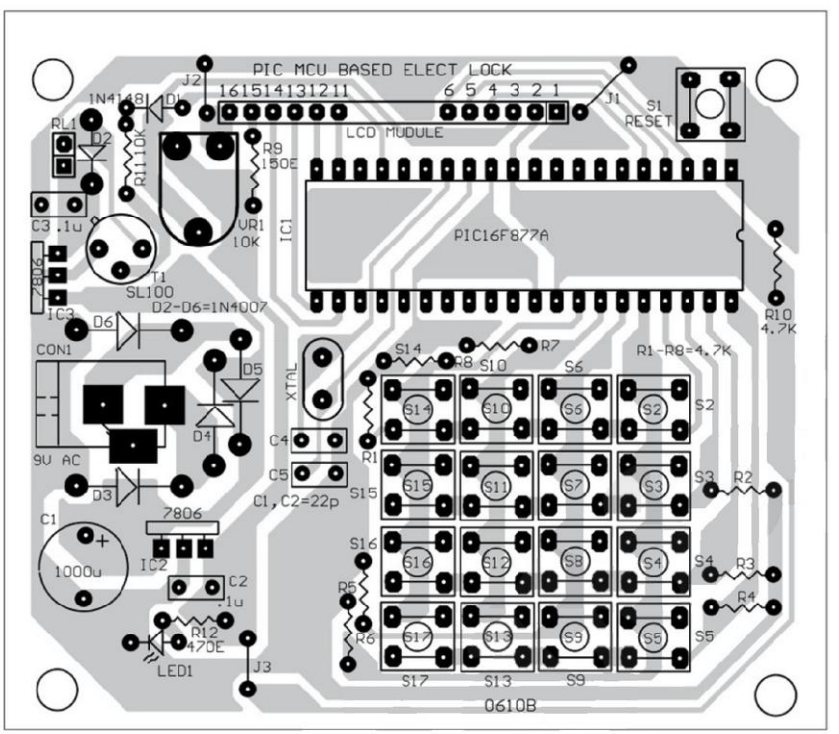


Fig. 5: Component layout for the PCB

Domestic Applications

WATER-LEVEL CONTROLLER-CUM-MOTOR PROTECTOR

■ GURSHARANJEET SINGH KALRA

Many a time we forget to switch off the motor pushing water into the overhead tank (OHT) in our households. As a result, water keeps overflowing until we notice the overflow and switch the pump off. As the OHT is usually kept on the topmost floor, it is cumbersome to go up frequently to check the water level in the OHT.

Here's a microcontroller-based water-level controller-cum-motor protector to solve this problem. It controls 'on' and 'off' conditions of the motor depending upon the level of water in the tank. The status is displayed on an LCD module. The circuit also protects the motor from high voltages, low voltages, fluctuations of mains power and dry running.

Circuit description

Fig. 1 shows the circuit of the microcontroller-based water-level controller-cum-motor protector. It comprises operational amplifier LM324, microcontroller AT89C51, optocoupler PC817, regulator 7805, LCD module and a few discreet components.

The AT89C51 (IC2) is an 8-bit microcontroller with four ports (32 I/O lines), two 16-bit timers/counters, on-chip oscillator and clock circuitry. Eight pins of port-1 and three pins of port-3 are interfaced with data and control lines of the LCD module. Pins P3.0, P3.1 and P3.6 are connected to RS (pin 4), R/W (pin 5) and E (pin 6) of the LCD, respectively. Pin EA (pin 31) is strapped to Vcc for internal program executions. Switch S2 is used for backlight of the LCD module.

Power-on-reset is achieved by connecting capacitor C8 and resistor R14 to pin 9 of the microcontroller. Switch S1 is used for manual reset.

The microcontroller is operated with a 12MHz crystal. Port pins P2.0 through P2.2 are used to sense the water level, while pins P2.3 and P2.4 are used to sense the under-voltage and over-voltage, respectively. Pin P3.4 is used to control relay RL1 with the help of optocoupler IC3 and transistor T5 in the case of under-voltage, over-voltage and different water-level conditions. Relay RL1 operates off a 12V supply. Using switch S3, you can manually switch on the motor.

The LM324 (IC1) is a quad operational amplifier (op-amp). Two of its op-amps are used as comparators to detect under- and over-voltage. In normal condition, output pin 7 of IC1 is low, making pin P2.3 of IC2 high. When the voltage at pin 6 of N1 goes below the set reference voltage at pin 5 (say, 170 volts), output pin 7 of N1 goes high. This high output makes pin P2.3 of IC2 low, which is sensed by the microcontroller and the LCD module shows 'low voltage.'

In normal condition, pin 1 of N2 is high. When the voltage at pin 2 of N2 goes above the set voltage at pin 3, output pin 1 of N2 goes low. This low signal is sensed by the microcontroller and the LCD module shows 'high voltage.'

Presets VR1 and VR2 are used for calibrating the circuit for under-

PARTS LIST

Semiconductors:

IC1	- LM324 quad op-amp
IC2	- AT89C51 microcontroller
IC3	- PC817 optocoupler
IC4	- 7805, 5V regulator
T1-T4	- BC548 npn transistor
T5	- SL100 npn transistor
D1-D14	- 1N4007 rectifier diode

Resistors (all ¼-watt, ±5% carbon):

R1, R2, R7, R11, R12	- 1-kilo-ohm
R3, R9	- 560-kilo-ohm
R4, R5, R8	- 2.7-kilo-ohm
R6	- 330-ohm
R10	- 470-ohm
R13	- 100-ohm
R14	- 10-kilo-ohm
R15-R17	- 100-kilo-ohm
R18-R20	- 2.2-kilo-ohm
R21, R22	- 33-ohm
RNW1	- 10-kilo-ohm resistor network
VR1, VR2	- 470-ohm preset
VR3	- 10-kilo-ohm preset

Capacitors:

C1-C3	- 1000µF, 35V electrolytic
C4	- 220µF, 16V electrolytic
C5, C6	- 33pF ceramic disk
C7	- 100µF, 35V electrolytic
C8	- 10µF, 16V electrolytic

Miscellaneous:

X1	- 230 AC primary to 12V, 500mA secondary transformer
RL1	- 12V, 1C/O relay
X_{TAL}	- 12MHz crystal
S1	- Push-to-on switch
S2, S3	- On/off switch
	- LCD module (1×16)

and over-voltage, respectively.

The AC mains is stepped down by transformer X1 to deliver a secondary output of 12V at 500 mA. The transformer output is rectified by a full-wave bridge rectifier comprising diodes D5 through D8, filtered by capacitor C2, and used for the under- and over-voltage detection circuitry.

The transformer output is also rectified by a full-wave bridge rectifier comprising diodes D1 through D4, filtered by capacitor C1 and regulated by IC4 to deliver regulated 5V for the circuit.

When water in the tank rises to come in contact with the sensor, the base of transistor BC548 goes high. This high signal drives transistor BC548 into saturation and its collector goes low. The low signal is sensed by port pins of microcontroller IC2 to detect empty tank, dry sump and full tank, respectively.

An actual-

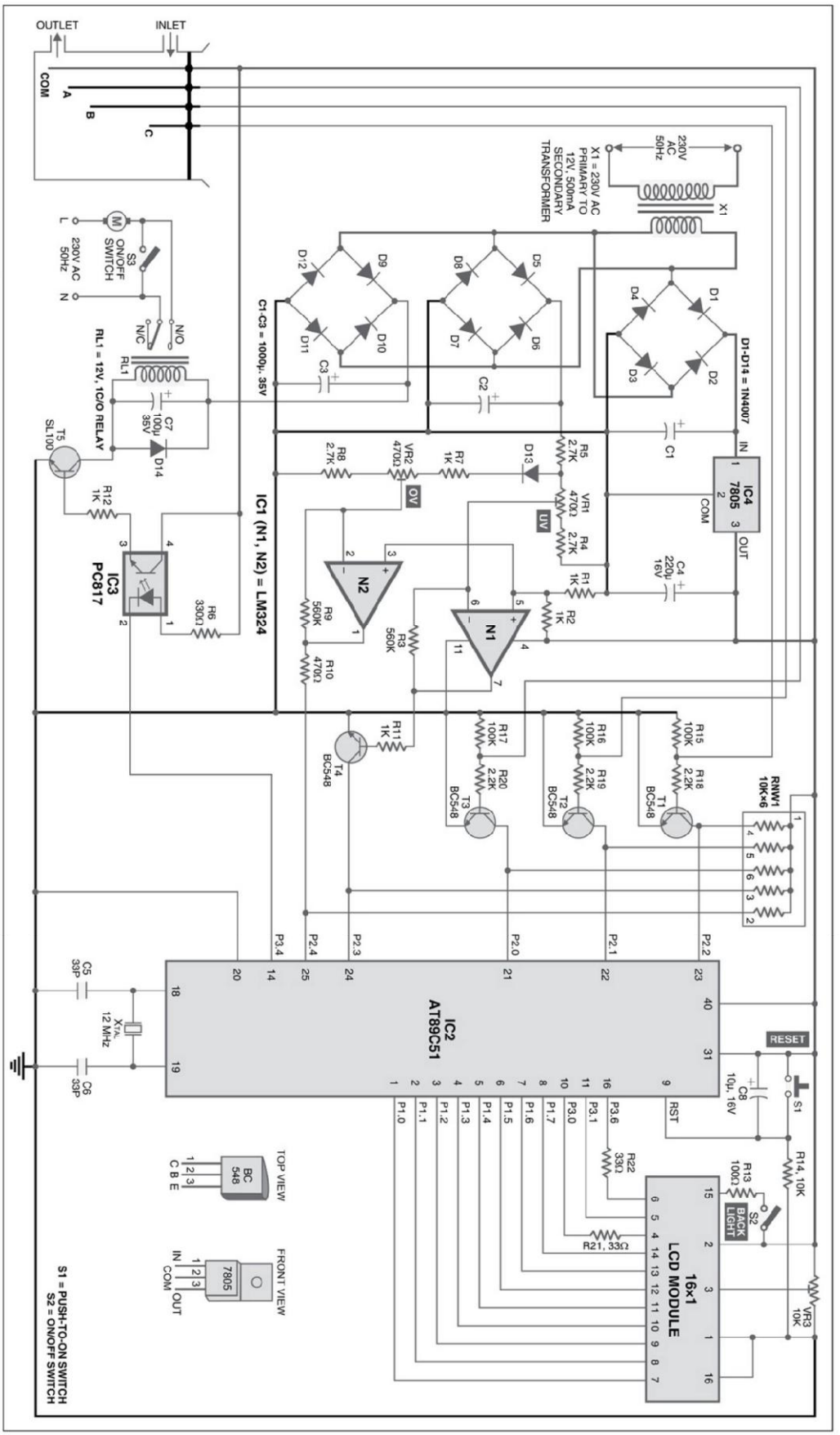


Fig. 1: Circuit of water-level controller-cum-motor protector

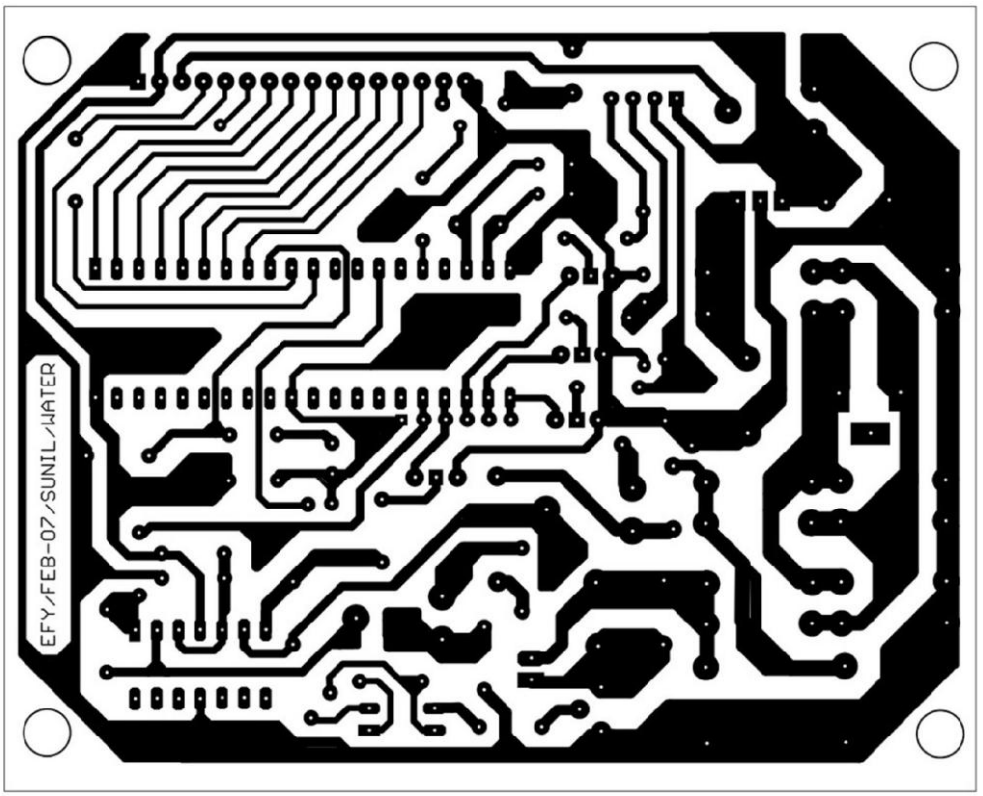


Fig. 2: Actual-size, single-side PCB layout of water-level controller-cum-motor protector

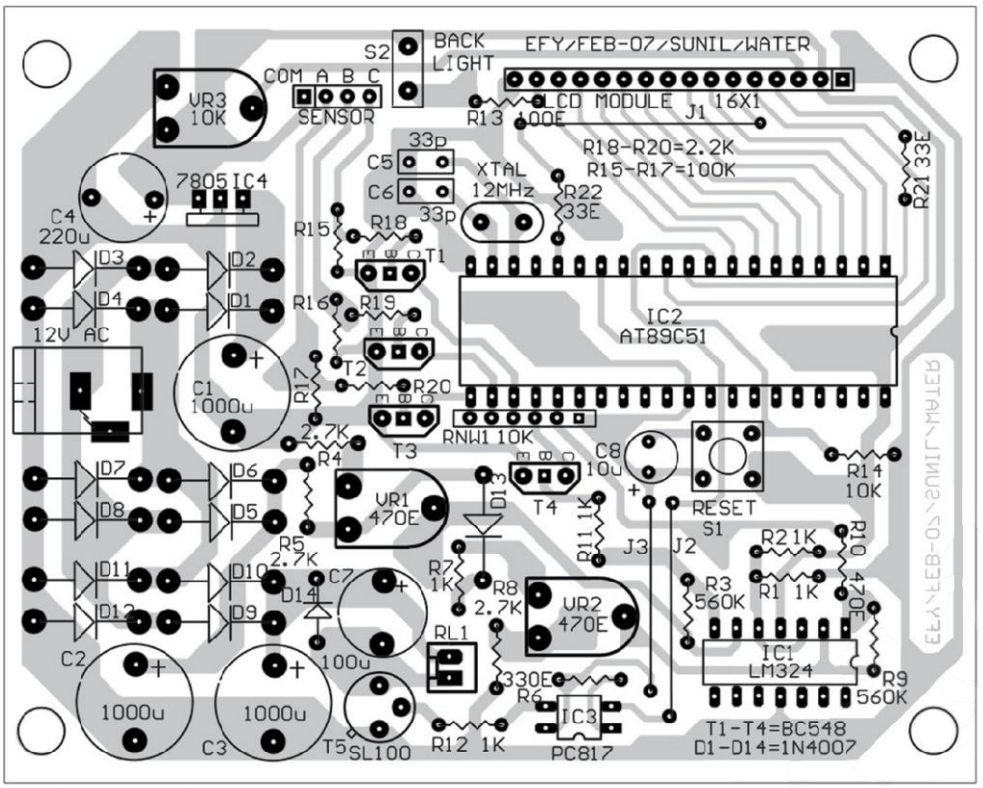


Fig. 3: Component layout for the PCB

size, single-side PCB for the water-level controller-cum-motor protector (Fig. 1) is shown in Fig. 2 and its component layout in Fig. 3.

Operation

When water in the tank is below sensor A, the motor will switch on to fill water in the tank. The LCD module will show 'motor on.' The controller is programmed for a 10-minute time interval to check the dry-run condition of the motor. If water reaches sensor B within 10 minutes, the microcontroller comes out of the dry-run condition and allows the motor to keep pushing water in the tank.

The motor will remain 'on' until water reaches sensor C. Then it will stop automatically and the microcontroller will go into the standby mode. The LCD module will show 'tank full' followed by 'standby mode' after a few seconds. The 'standby mode' message is displayed until water in the tank goes below sensor A.

In case water does not reach sensor B within 10 minutes, the microcontroller will go into the dry-running mode and stop the motor for 5 minutes, allowing it to cool down. The LCD module will show 'dry-sump1.'

After five minutes, the microcontroller will again switch on the motor for 10 minutes and check the

status at sensor B. If water is still below sensor B, it will go into the dry-running mode and the LCD module will show 'dry-sump2.'

The same procedure will repeat, and if the dry-run condition still persists, the display will show 'dry-sump3' and the microcontroller will not start the motor automatically. Now you have to check the line for water and manually reset the microcontroller to start operation.

In the whole procedure, the microcontroller checks for high and low voltages. For example, when the voltage is high, it will scan for about two seconds to check whether it is a fluctuation. If the voltage remains high after two seconds, the microcontroller will halt running of the motor. Now it will wait for the voltage to settle down. After the voltage becomes normal, it will still check for 90 seconds whether the voltage is normal or not. After normal condition, it will go in the standby mode and start the aforementioned procedure.

Practical applications

This controller is useful for single-phase operated motor-pumps and the pumps that suck water from the ground water tank. A small push-to-off manual switch in series with sensor A can also make it useful for pumps that suck water from Jal Board's supply. Because of the particular timing of this water supply, the controller must be switched on within the timing of the water supply and switched off when water is not being supplied.

When the controller is 'on' during the supply timings, it will wait for the tank to get empty before starting the motor. However, you can also start the motor using the pushbutton. The motor will turn on ignoring the status of the water level and will go through the aforementioned procedure.

Sensor positions in the tank

Four non-corrosive metallic sensors are installed in the tank as shown in Fig. 1. Sensor COM is connected to Vcc supply from the circuit. Sensor A detects the empty tank to start the motor. Sensor B detects dry-running condition of the motor and sensor C detects the full tank to stop the motor. Make sure that sensor B is around 2 cm above sensor A to check the dry-running condition properly.

Calibration

Care must be taken when calibrating for under- and over-voltages. Always calibrate when the relay is in 'on' position. If you calibrate in the standby mode, it will trip at a voltage nearly 10 volts lower than the set voltage due to the loading effect.

Software

The source code is written in Assembly language and assembled using 8051 cross-assembler. The generated Intel hex code is burnt into microcontroller AT89C51 using a suitable programmer. The software is well-commented and easy to understand. All the messages are displayed on the LCD module.

Download source code: http://www.efymag.com/admin/issuepdf/Waternew.zip

WATER.LST

```
PAGE 1

                        1
                        2
$MOD51
0000 02002F             3     START:          LJMP
MAIN_PGR    ;GO TO MAIN PROGRAMME
002F                    4     ORG 002FH
002F 7438               5     MAIN_PGR:       MOV A,
# 38H             ;INITIALIZE LCD
0031 1200A0             6                     LCALL
WR_LCD
0034 740F               7                     MOV A,
#0FH
0036 1200A0             8                     LCALL
WR_LCD
0039 7406               9                     MOV A,
#06H
003B 1200A0            10                     LCALL
WR_LCD
```

```
003E 7401            11                    MOV A,
#01H
0040 1200A0          12                    LCALL
WR_LCD
0043 7480            13                    MOV A,
#80H
0045 1200A0          14                    LCALL
WR_LCD
0048 7453            15                    MOV A,
#53H            ;WRITE DATA TO LCD
004A 1200C0          16                    LCALL
LCD_RAM   ;i.e.," STANDBY-MODE"
004D 7454            17                    MOV A,
#54H
004F 1200C0          18                    LCALL
LCD_RAM
0052 7441            19                    MOV A,
#41H
0054 1200C0          20                    LCALL
LCD_RAM
0057 744E            21                    MOV A,
#4EH
0059 1200C0          22                    LCALL
LCD_RAM
005C 7444            23                    MOV A,
#44H
005E 1200C0          24                    LCALL
LCD_RAM
0061 7442            25                    MOV A,
#42H
0063 1200C0          26                    LCALL
LCD_RAM
0066 7459            27                    MOV A,
#59H
0068 1200C0          28                    LCALL
LCD_RAM
006B 74B0            29                    MOV A,
#0B0H
006D 1200C0          30                    LCALL
LCD_RAM
0070 74C0            31                    MOV A,
#0C0H           ;JUMP TO 9TH POSITION
0072 1200A0          32                    LCALL
WR_LCD           ;OR SECOND LINE
0075 744D            33                    MOV A,
#4DH            ;ENTER DATA AGAIN
0077 1200C0          34                    LCALL
LCD_RAM
007A 744F            35                    MOV A,
#4FH
007C 1200C0          36                    LCALL
LCD_RAM
007F 7444            37                    MOV A,
#44H
0081 1200C0          38                    LCALL
LCD_RAM
0084 7445            39                    MOV A,
#45H
0086 1200C0          40                    LCALL
LCD_RAM
0089 02010A          41                    LJMP
SCAN              ;GO TO 010AH
00A0                 42                    ORG
00A0H
00A0 1200F0          43          WR_LCD: LCALL
SETT             ;CHECK FOR READY STATUS OF LCD
00A3 F590            44                    MOV
P1,A              ;MOVE CONTENTS OF A TO PORT 1
00A5 C2B0            45                    CLR
P3.0                ;WRITE COMMANDS TO LCD
00A7 C2B1            46                    CLR P3.1
00A9 D2B6            47                    SETB
P3.6
00AB C2B6            48                    CLR P3.6
00AD 22              49                    RET
;RETURN TO PROGRAMME
00C0                 50                    ORG
00C0H
00C0 1200F0          51       LCD_RAM: LCALL SETT
;CHECK READY STATUS OF LCD
00C3 F590            52                    MOV P1,A
;MOVE CONTENS OF A TO PORT 1
00C5 D2B0            53                    SETB
P3.0               ;WRITE TO DISPLAY RAM OF LCD
00C7 C2B1            54                    CLR P3.1
00C9 D2B6            55                    SETB
P3.6
00CB C2B6            56                    CLR P3.6
00CD 22              57                    RET
;GO TO PROGRAMME
00F0                 58                    ORG
00F0H
WATERNEW
PAGE 2

00F0 C2B6            59    SETT:   CLR P3.6
00F2 7590FF          60                    MOV P1,
#0FFH   ;SET PORT1 FOR INPUT
00F5 00              61                    NOP
;DELAY
00F6 C2B0            62                    CLR P3.0
00F8 D2B1            63                    SETB
P3.1
00FA C2B6            64            L1:     CLR P3.6
00FC D2B6            65                    SETB
P3.6
00FE 2097F9          66                    JB
P1.7,L1             ;IF NOT READY JUMP TO 00FA H
0101 C2B6            67                    CLR P3.6
0103 22              68                    RET
;BACK TO PROGRAMME
010A                 69                    ORG
010AH
010A D276            70            SCAN: SETB 76H
;SET USER FLAGS
010C D277            71                    SETB 77H
010E D278            72                    SETB 78H
0110 75A0FF          73                    MOV
P2,#0FFH            ;SET PORT2 FOR INPUT
0113 30A420          74            L4:   JNB
P2.4,L2                      ;IF H/V THEN GOTO 0136
H
0116 30A372          75                    JNB
P2.3,L3             ;IF L/V THEN GOTO 018B H
0119 30A0F7          76                    JNB
P2.0,L4             ;SCAN FOR TANK TO BE EMPTY
011C 8012            77                    SJMP
L130               ;GOTO 0130 H
011E 120750          78            LOOP: LCALL
TMR_10MIN   ;CALL 10 MIN. TIMER
0121 307867          79                    JNB
78H,L3                     ;L/V THEN GOTO 018B H
0124 30770F          80                    JNB
77H,L2              ;H/V THEN GOTO 0136 H
0127 20765E          81                    JB
76H,L5               ;DRY SUMP THEN GOTO 0188 H
012A 20A2FD          82                    JB
P2.2,$               ;WAIT UNTILL TANK FULL
```

```
012D 020260          83           LOOP1:LJMP
TANK_FULL           ;GOTO TANK FULL DISPLAY
0130 C2B4            84           L130:   CLR
P3.4                       ;START MOTOR
0132 0202D0          85                   LJMP
DIS_M_ON           ;DISPLAY MOTOR ON
0135 00              86                   NOP
;BLANK SPACE
0136 1201D8          87           L2:     LCALL
DLY_2SEC  ;WAIT FOR 2 SECONDS
0139 20A4D7          88                   JB
P2.4,L4             ;STILL H/V THEN GOTO 0113 H
013C D2B4            89           LOOP2:SETB
P3.4         ;H/V THEN OFF MOTOR
013E 00              90                   NOP
013F 00              91                   NOP
0140 1202B3          92                   LCALL
INI_LCD           ;INITIALIZE LCD
0143 7448            93                   MOV A,
#48H             ;DISPLAY HIGH-VOLTAGE
0145 1200C0          94                   LCALL
LCD_RAM
0148 7449            95                   MOV A,
#49H
014A 1200C0          96                   LCALL
LCD_RAM
014D 7447            97                   MOV A,
#47H
014F 1200C0          98                   LCALL
LCD_RAM
0152 7448            99                   MOV A,
#48H
0154 1200C0         100                   LCALL
LCD_RAM
0157 74B0           101                   MOV A,
#0B0H
0159 1200C0         102                   LCALL
LCD_RAM
015C 7456           103                   MOV A,
#56H
015E 1200C0         104                   LCALL
LCD_RAM
0161 744F           105                   MOV A,
#4FH
0163 1200C0         106                   LCALL
LCD_RAM
0166 744C           107                   MOV A,
#04CH
0168 1200C0         108                   LCALL
LCD_RAM
016B 74C0           109                   MOV A,
#0C0H            ;MOVE TO 9TH CHARACTER
016D 1200A0         110                   LCALL
WR_LCD
0170 7454           111                   MOV A,
#54H
0172 1200C0         112                   LCALL
LCD_RAM
0175 7441           113                   MOV A,
#41H
0177 1200C0         114                   LCALL
LCD_RAM
017A 7447           115                   MOV A,
#47H
017C 1200C0         116                   LCALL
LCD_RAM
WATERNEW
PAGE 3
```

```
017F 7445           117                   MOV A,
#45H
0181 1200C0         118                   LCALL
LCD_RAM
0184 020490         119                   LJMP
CHK_HV             ;GOTO 0490 H
0187 00             120                   NOP
;BLANK SPACE
0188 020226         121           L5:     LJMP
M_STOP             ;GOTO 0226 H
018B 1201D8         122           L3:   LCALL
DLY_2SEC            ;WAIT FOR 2 SECONDS
018E 20A382         123                   JB
P2.3,L4             ;VOLTAGE OK THEN GOTO 0113 H
0191 D2B4           124           LOOP3:SETB P3.4
;STOP MOTOR IF ON
0193 1202B3         125                   LCALL
INI_LCD                 ;INITIALIZE LCD
0196 744C           126                   MOV A,
#04CH            ;DISPLAY LOW-VOLTAGE
0198 1200C0         127                   LCALL
LCD_RAM
019B 744F           128                   MOV A,
#04FH
019D 1200C0         129                   LCALL
LCD_RAM
01A0 7457           130                   MOV A,
#57H
01A2 1200C0         131                   LCALL
LCD_RAM
01A5 74B0           132                   MOV A,
#0B0H
01A7 1200C0         133                   LCALL
LCD_RAM
01AA 7456           134                   MOV A,
#56H
01AC 1200C0         135                   LCALL
LCD_RAM
01AF 744F           136                   MOV A,
#04FH
01B1 1200C0         137                   LCALL
LCD_RAM
01B4 744C           138                   MOV A,
#04CH
01B6 1200C0         139                   LCALL
LCD_RAM
01B9 7454           140                   MOV A,
#54H
01BB 1200C0         141                   LCALL
LCD_RAM
01BE 74C0           142                   MOV A,
#0C0H            ;GOTO 9TH CHARACTER
01C0 1200A0         143                   LCALL
WR_LCD
01C3 7441           144                   MOV A,
#41H             ;START DISPLAY AGAIN
01C5 1200C0         145                   LCALL
LCD_RAM
01C8 7447           146                   MOV A,
#47H
01CA 1200C0         147                   LCALL
LCD_RAM
01CD 7445           148                   MOV A,
#45H
01CF 1200C0         149                   LCALL
LCD_RAM
01D2 0204B0         150                   LJMP
CHK_LV             ;GOTO 04B0 H
01D5 00             151
```

```
NOP
01D6 00              152
NOP
01D7 00              153
NOP
01D8                 154                    ORG
01D8H
01D8 7B03            155          DLY_2SEC: MOV R3,
#03H
01DA 7CFF            156            L8:     MOV R4,
#0FFH
01DC 7DFF            157            L7:     MOV R5,
#0FFH
01DE 00              158            L6:   NOP
01DF 00              159                    NOP
01E0 00              160                    NOP
01E1 00              161                    NOP
01E2 00              162                    NOP
01E3 00              163                    NOP
01E4 DDF8            164                    DJNZ
R5,L6 ;01DEH
01E6 DCF4            165                    DJNZ
R4,L7 ;01DCH
01E8 DBF0            166                    DJNZ
R3,L8 ;01DAH
01EA 22              167                    RET
;BACK TO PROGRAMME
0226                 168                    ORG
0226H
0226 D2B4            169            M_STOP: SETB
P3.4              ;STOP MOTOR
0228 1202B3          170                    LCALL
INI_LCD           ;INITIALIZE LCD
022B 7444            171                    MOV A,
#44H              ;START FILLING DISPLAY RAM OF LCD
022D 1200C0          172                    LCALL
LCD_RAM
0230 7452            173                    MOV A,
#52H
0232 1200C0          174                    LCALL
LCD_RAM
WATERNEW
PAGE 4

0235 7459            175                    MOV A,
#59H
0237 1200C0          176                    LCALL
LCD_RAM
023A 74B0            177                    MOV A,
#0B0H
023C 1200C0          178                    LCALL
LCD_RAM
023F 7453            179                    MOV A,
#53H
0241 1200C0          180                    LCALL
LCD_RAM
0244 7455            181                    MOV A,
#055H
0246 1200C0          182                    LCALL
LCD_RAM
0249 744D            183                    MOV A,
#04DH
024B 1200C0          184                    LCALL
LCD_RAM
024E 7450            185                    MOV A,
#50H
0250 1200C0          186                    LCALL
LCD_RAM
0253 74C0            187                    MOV A,
#0C0H             ;MOVE TO 9TH CHARACTER OF LCD
0255 1200A0          188                    LCALL
WR_LCD
0258 7431            189                    MOV A,
#31H              ;START WRITING AGAIN
025A 1200C0          190                    LCALL
LCD_RAM
025D 020300          191                    LJMP
L300               ;GOTO 0300 H
0260 D2B4            192     TANK_FULL: SETB P3.4
;STOP MOTOR
0262 00              193                    NOP
;BLANK SPACES FOR FURTHER EXPANSION
0263 00              194                    NOP
0264 00              195                    NOP
0265 00              196                    NOP
0266 00              197                    NOP
0267 00              198                    NOP
0268 00              199                    NOP
0269 1202B3          200                    LCALL
INI_LCD   ;INITIALIZE LCD
026C 7454            201                    MOV A,
#54H              ;WRITE TO DISPLAY RAM OF LCD
026E 1200C0          202                    LCALL
LCD_RAM
0271 7441            203                    MOV A,
#41H
0273 1200C0          204                    LCALL
LCD_RAM
0276 744E            205                    MOV A,
#4EH
0278 1200C0          206                    LCALL
LCD_RAM
027B 744B            207                    MOV A,
#4BH
027D 1200C0          208                    LCALL
LCD_RAM
0280 74B0            209                    MOV A,
#0B0H
0282 1200C0          210                    LCALL
LCD_RAM
0285 7446            211                    MOV A,
#46H
0287 1200C0          212                    LCALL
LCD_RAM
028A 7455            213                    MOV A,
#55H
028C 1200C0          214                    LCALL
LCD_RAM
028F 744C            215                    MOV A,
#4CH
0291 1200C0          216                    LCALL
LCD_RAM
0294 74C0            217                    MOV A,
#0C0H             ;GOTO 9TH CHARACTER OF LCD
0296 1200A0          218                    LCALL
WR_LCD
0299 744C            219                    MOV A,
#4CH              ;START DISPLAYING AGAIN
029B 1200C0          220                    LCALL
LCD_RAM
029E 1201D8          221                    LCALL
DLY_2SEC           ;DISPLAY IT FOR 2 SECONDS
02A1 020000          222                    LJMP
START               ;GOTO STANDBY MODE
02B3                 223                    ORG
02B3H
02B3 7438            224            INI_LCD:MOV A,
# 38H
```

```
02B5 1200A0          225                    LCALL
WR_LCD
02B8 740F            226                    MOV A,
#0FH
02BA 1200A0          227                    LCALL
WR_LCD
02BD 7406            228                    MOV A,
#06H
02BF 1200A0          229                    LCALL
WR_LCD
02C2 7401            230                    MOV A,
#01H
02C4 1200A0          231                    LCALL
WR_LCD
02C7 7480            232                    MOV A,
#80H
WATERNEW
PAGE 5

02C9 1200A0          233                    LCALL
WR_LCD
02CC 22              234                    RET
;BACK TO PROGRAMME
02CD 00              235                    NOP
02CE 00              236                    NOP
02CF 00              237                    NOP
02D0 1202B3          238          DIS_M_ON: LCALL
INI_LCD ;INITIALIZE LCD
02D3 744D            239                    MOV A, #
4DH            ;WRITE INTO DISPLAY RAM OF LCD
02D5 1200C0          240                    LCALL
LCD_RAM
02D8 744F            241                    MOV A,
#04FH
02DA 1200C0          242                    LCALL
LCD_RAM
02DD 7454            243                    MOV A,
#54H
02DF 1200C0          244                    LCALL
LCD_RAM
02E2 744F            245                    MOV A,
#04FH
02E4 1200C0          246                    LCALL
LCD_RAM
02E7 7452            247                    MOV A,
#52H
02E9 1200C0          248                    LCALL
LCD_RAM
02EC 74B0            249                    MOV A,
#0B0H
02EE 1200C0          250                    LCALL
LCD_RAM
02F1 744F            251                    MOV A,
#04FH
02F3 1200C0          252                    LCALL
LCD_RAM
02F6 744E            253                    MOV A,
#04EH
02F8 1200C0          254                    LCALL
LCD_RAM
02FB 02011E          255                    LJMP
LOOP              ;BACK TO MAIN PROGRAMME AT 011E H
02FE 00              256                    NOP
02FF 00              257                    NOP
0300 120320          258            L300: LCALL
TMR_5MIN           ;CALL 5 MINUTES TIMER
0303 020550          259                    LJMP
MAIN              ;GOTO MAIN PROGRAMME AT 0550 H
0320                 260                    ORG
0320H
0320 7A03            261      TMR_5MIN: MOV R2,#03H
0322 7BFF            262            L12:    MOV R3,
#0FFH
0324 7CFF            263            L11:    MOV R4,
#0FFH
0326 7DFF            264            L10:    MOV R5,
#0FFH
0328 00              265            L9:  NOP
0329 00              266                    NOP
032A 00              267                    NOP
032B 00              268                    NOP
032C DDFA            269                    DJNZ
R5, L9 ;0328
032E DCF6            270                    DJNZ
R4, L10 ;0326
0330 DBF2            271                    DJNZ
R3, L11 ;0324
0332 DAEE            272                    DJNZ
R2, L12 ;0322
0334 22              273                    RET
;BACK TO MAIN PROGRAMME
0430                 274                    ORG
0430H
0430 30A209          275            L430: JNB
P2.2,L43C             ;IF TANK FULL GOTO 043C H
0433 30A409          276                    JNB
P2.4,L43F           ;H/V THEN GOTO 043F H
0436 30A30F          277            L436:   JNB
P2.3,L448           ;L/V THEN GOTO 0448 H
0439 80F5            278            L439:   SJMP
L430                ;GOTO 0430 H
043B 00              279                    NOP
043C 02012D          280            L43C:   LJMP
LOOP1               ;ACK TO MAIN PROGRAMME
043F 1201D8          281            L43F:   LCALL
DLY_2SEC           ;WAIT FOR 2 SECONDS
0442 20A4F1          282                    JB
P2.4, L436           ;IF NOT H/V THEN GOTO 0436 H
0445 02013C          283                    LJMP
LOOP2               ;H/V THEN GOTO 013C H
0448 1201D8          284            L448:   LCALL
DLY_2SEC           ;WAIT FOR 2 SECONDS
044B 20A3EB          285                    JB
P2.3, L439           ;IF NOT L/V THEN GOTO 0439 H
044E 020191          286                    LJMP
LOOP3      ;IF L/V THEN GOTO 0191 H
0490                 287                    ORG
0490H
0490 00              288        CHK_HV:     NOP
0491 00              289                    NOP
0492 120500          290         L492:      LCALL
DLY_2MIN           ;WAIT FOR 2 MINUTES
WATERNEW
PAGE 6

0495 30A4FA          291                    JNB
P2.4,L492           ;CHECK FOR H/V AGAIN
0498 020000          292                    LJMP
START               ;GOTO START AGAIN
04B0                 293                    ORG
04B0H
04B0 00              294        CHK_LV: NOP
04B1 00              295                    NOP
04B2 120500          296            L4B2:   LCALL
DLY_2MIN  ;WAIT FOR 2 MINUTES
04B5 30A3FA          297                    JNB
P2.3, L4B2           ;CHECK FOR L/V AGAIN
04B8 020000          298                    LJMP
```

```
START              ;GOTO START AGAIN
0500                 299                    ORG
0500H
0500 7AFF            300          DLY_2MIN: MOV R2,
#0FFH
0502 7BFF            301            L502:   MOV
R3,#0FFH
0504 7CFF            302            L504:   MOV
R4,#0FFH
0506 00              303            L506: NOP
0507 00              304                    NOP
0508 00              305                    NOP
0509 00              306                    NOP
050A DCFA            307                    DJNZ
R4, L506
050C DBF6            308                    DJNZ
R3, L504
050E DAF2            309                    DJNZ
R2, L502
0510 22              310                    RET
;BACK TO MAIN PROGRAMME
054D                 311                    ORG
054DH
054D 020642          312          M_START: LJMP
SUB_BR     ;SUB BRANCH DUE TO SPACE PROBLEM
0550 30A447          313            MAIN: JNB
P2.4, L59A          ;CHECK FOR HIGH VOLTAGE
0553 30A34D          314                  JNB
P2.3, L5A3          ;CHECK FOR LOW VOLTAGE
0556 C2B4            315                  CLR
P3.4                      ;IF VOLTAGE OK THEN
START MOTOR
0558 1202B3          316                  LCALL
INI_LCD           ;INITIALIZE LCD
055B 744D            317                  MOV A,
#04DH            ;START WRITING TO DISPLAY RAM OF
LCD
055D 1200C0          318                  LCALL
LCD_RAM
0560 744F            319                  MOV A,
#04FH
0562 1200C0          320                  LCALL
LCD_RAM
0565 7454            321                  MOV A,
#54H
0567 1200C0          322                  LCALL
LCD_RAM
056A 744F            323                  MOV A,
#4FH
056C 1200C0          324                  LCALL
LCD_RAM
056F 7452            325                  MOV A,
#52H
0571 1200C0          326                  LCALL
LCD_RAM
0574 74B0            327                  MOV A,
#0B0H
0576 1200C0          328                  LCALL
LCD_RAM
0579 744F            329                  MOV A,
#4FH
057B 1200C0          330                  LCALL
LCD_RAM
057E 744E            331                  MOV A,
#4EH
0580 1200C0          332                  LCALL
LCD_RAM
0583 120750          333                  LCALL
TMR_10MIN         ;ENTER INTO 10 MINUTES TIMER
0586 307820          334                  JNB
78H, C_LV; 05A9H           ;LOW VOLTAGE THEN GOTO
05A9 H
0589 307714          335                  JNB
77H, C_HV; 05A0H           ;HIGH VOLTAGE THEN GOTO
05A0 H
058C 20761D          336                  JB 76H,
DRY ;05ACH              ;IF TANK DRY THEN GOTO 05AC
H
058F 30A2BB          337            L58F: JNB
P2.2, M_START              ;TANK FULL THEN GOTO
054D H
0592 30A405          338                  JNB
P2.4, L59A          ;HIGH VOLTAGE THENGOTO 059A H
0595 30A30B          339            L595: JNB
P2.3, L5A3          ;LOW VOLTAGE THEN GOTO 05A3 H
0598 80F5            340            L598: SJMP
L58F                ;REPEAT FROM 058F H
059A 1201D8          341            L59A: LCALL
DLY_2SEC            ;WAIT FOR 2 SECONDS
059D 20A4F5          342                  JB
P2.4, L595          ;IF NOT H/V THEN GO BACK TO
0595 H
05A0 02013C          343            C_HV: LJMP
LOOP2               ;STILL H/V THEN GOTO 013C H
05A3 1201D8          344            L5A3: LCALL
DLY_2SEC            ;WAIT FOR 2 SECONDS
05A6 20A3EF          345                  JB
P2.3, L598          ;IF NOT L/V THEN GO BACK TO
0598 H
05A9 020191          346            C_LV: LJMP
LOOP3               ;STILL L/V THEN GOTO 0191 H
05AC D2B4            347            DRY:  SETB
P3.4                ;STOP MOTOR
05AE 1202B3          348                  LCALL
INI_LCD           ;INITIALIZE LCD
WATERNEW
PAGE 7

05B1 7444            349                  MOV A,
#44H             ;START WRITING TO DISPLAY RAM OF
LCD
05B3 1200C0          350                  LCALL
LCD_RAM
05B6 7452            351                  MOV A,
#52H
05B8 1200C0          352                  LCALL
LCD_RAM
05BB 7459            353                  MOV A,
#59H
05BD 1200C0          354                  LCALL
LCD_RAM
05C0 74B0            355                  MOV A,
#0B0H
05C2 1200C0          356                  LCALL
LCD_RAM
05C5 7453            357                  MOV A,
#53H
05C7 1200C0          358                  LCALL
LCD_RAM
05CA 7455            359                  MOV A,
#55H
05CC 1200C0          360                  LCALL
LCD_RAM
05CF 744D            361                  MOV A,
#4DH
05D1 1200C0          362                  LCALL
```

```
LCD_RAM
05D4 7450          363                    MOV A,
#50H
05D6 1200C0        364                    LCALL
LCD_RAM
05D9 74C0          365                    MOV A,
#0C0H          ;GOTO 9TH CHARACTER OF LCD
05DB 1200A0        366                    LCALL
WR_LCD
05DE 7432          367                    MOV A,
#32H           ;START WRITING AGAIN
05E0 1200C0        368                    LCALL
LCD_RAM
05E3 120320        369                    LCALL
TMR_5MIN        ;WAIT FOR 5 MINUTES
05E6 30A447        370                    JNB
P2.4,HV ; 0630H            ;H/V THEN GOTO 0630 H
05E9 30A34D        371                    JNB
P2.3,LV ; 0639H            ;L/V THEN GOTO 0639 H
05EC C2B4          372                    CLR P3.4
;IF VOLTAGE OK THEN START MOTOR
05EE 1202B3        373                    LCALL
INI_LCD         ;INITILIZE LCD
05F1 744D          374                    MOV A,
#4DH           ;START WRITING TO DISPLAY RAM OF
LCD
05F3 1200C0        375                    LCALL
LCD_RAM
05F6 744F          376                    MOV A,
#4FH
05F8 1200C0        377                    LCALL
LCD_RAM
05FB 7454          378                    MOV A,
#54H
05FD 1200C0        379                    LCALL
LCD_RAM
0600 744F          380                    MOV A,
#4FH
0602 1200C0        381                    LCALL
LCD_RAM
0605 7452          382                    MOV A,
#52H
0607 1200C0        383                    LCALL
LCD_RAM
060A 74B0          384                    MOV A,
#0B0H
060C 1200C0        385                    LCALL
LCD_RAM
060F 744F          386                    MOV A,
#4FH
0611 1200C0        387                    LCALL
LCD_RAM
0614 744E          388                    MOV A,
#4EH
0616 1200C0        389                    LCALL
LCD_RAM
0619 120750        390                    LCALL
TMR_10MIN ;GOTO MINUTES TIMER
061C 307820        391                    JNB 78H,
L63F          ;L/V THEN GOTO 063F H
061F 307714        392                    JNB 77H,
L636          ;H/V THEN GOTO 0636
0622 207622        393                    JB 76H,
L647           ;STILL DRY SUMP THEN GOTO 0647 H
0625 30A21A        394          L625:     JNB
P2.2, SUB_BR ;0642H        ;TANK FULL THEN GOTO
0642 H
0628 30A405        395                    JNB
P2.4, HV ;0630H            ;H/V THEN GOTO 0630 H
062B 30A30B        396          N_HV:     JNB
P2.3, LV ;0639H            ;L/V THEN GOTO 0639 H
062E 80F5          397          N_LV:     SJMP
L625           ;REPEAT FROM 0625 H
0630 1201D8        398          HV:       LCALL
DLY_2SEC       ;WAIT FOR 2 SECONDS
0633 20A4F5        399                    JB P2.4,
N_HV ;062BH          ;IF NOT H/V THEN GOTO 062B H
0636 02013C        400          L636:     LJMP
LOOP2          ;STILL H/V THEN GOTO 013C H
0639 1201D8        401          LV:       LCALL
DLY_2SEC       ;WAIT FOR 2 SECONDS
063C 20A3EF        402                    JB P2.3,
N_LV ;062EH          ;IF NOT L/V THEN GOTO 062E H
063F 020191        403          L63F:     LJMP
LOOP3          ;STILL L/V THEN GOTO 0191 H
0642 D2B4          404        SUB_BR: SETB P3.4
;STOP MOTOR
0644 020260        405                    LJMP
TANK_FULL      ;TANK FULL THEN GOTO 0260 H
0647 D2B4          406          L647:     SETB
P3.4           ;STOP MOTOR
WATERNEW
PAGE 8

0649 00            407                    NOP
064A 00            408                    NOP
064B 1202B3        409                    LCALL
INI_LCD         ;INITIALIZE LCD
064E 7444          410                    MOV A,
#44H           ;START WRITING TO DISPLAY RAM OF
LCD
0650 1200C0        411                    LCALL
LCD_RAM
0653 7452          412                    MOV A,
#52H
0655 1200C0        413                    LCALL
LCD_RAM
0658 7459          414                    MOV A,
#59H
065A 1200C0        415                    LCALL
LCD_RAM
065D 74B0          416                    MOV A,
#0B0H
065F 1200C0        417                    LCALL
LCD_RAM
0662 7453          418                    MOV A,
#53H
0664 1200C0        419                    LCALL
LCD_RAM
0667 7455          420                    MOV A,
#55H
0669 1200C0        421                    LCALL
LCD_RAM
066C 744D          422                    MOV A,
#4DH
066E 1200C0        423                    LCALL
LCD_RAM
0671 7450          424                    MOV A,
#50H
0673 1200C0        425                    LCALL
LCD_RAM
0676 74C0          426                    MOV A,
#0C0H          ;GOTO 9TH CHARACTER OF LCD
0678 1200A0        427                    LCALL
WR_LCD
067B 7433          428                    MOV A,
#33H           ;START WRITING AGAIN
067D 1200C0        429                    LCALL
```

```
LCD_RAM
0680 80FE            430                    SJMP $
;STAY HERE UNTILL MANUAL RESET
0750                 431                    ORG
0750H
0750 7A05            432     TMR_10MIN: MOV R2,
#05H
0752 7BFF            433            L752:   MOV R3,
#0FFH
0754 7CFF            434            L754:   MOV R4,
#0FFH
0756 7DFF            435            L756:   MOV R5,
#0FFH
0758 00              436            L758:   NOP
0759 00              437                    NOP
075A 00              438                    NOP
075B 00              439                    NOP
075C 00              440                    NOP
075D DDF9            441                    DJNZ
R5, L758
075F DCF5            442                    DJNZ
R4, L756
0761 30A40C          443                    JNB
P2.4,L770           ;H/V THEN GOTO 0670 H
0764 30A311          444            L764:   JNB
P2.3, L778          ;L/V THEN GOTO 0678 H
0767 30A116          445            L767:   JNB
P2.1, L780          ; NOT DRY SUMP THEN GOTO 0680 H
076A DBE8            446            L76A:   DJNZ
R3, L754
076C DAE4            447                    DJNZ
R2, L752
076E 22              448                    RET
;BACK TO MAIN PROGRAMME
076F 00              449                    NOP
0770 1201D8          450            L770:   LCALL
DLY_2SEC           ;WAIT FOR 2 SECONDS
0773 30A412          451                    JNB
P2.4, 0788H         ;STILL H/V THEN GOTO 0788 H
0776 80EC            452                    SJMP
L764                ; NOT H/V THEN GOTO 0764 H
0778 1201D8          453            L778:   LCALL
DLY_2SEC           ;WAIT FOR 2 SECONDS
077B 30A30D          454                    JNB
P2.3, 078BH         ;STILL L/V THEN GOTO 078B H
077E 80E7            455                    SJMP
L767               ;NOT L/V THEN GOTO 0767 H
0780 1201D8          456            L780:   LCALL
DLY_2SEC           ;WAIT FOR 2 SECONDS
0783 30A108          457                    JNB
P2.1, 078EH         ;STILL NOT DRY SUMP THEN GOTO
078E H
0786 80E2            458                    SJMP
L76A               ;OTHERWISE GOTO 076A
0788 C277            459                    CLR 77H
;CLEAR FLAG 77H FOR H/V
078A 22              460                    RET
078B C278            461                    CLR 78H
;CLEAR FLAG 78H FOR L/V
078D 22              462                    RET
078E C276            463                    CLR 76H
;CLEAR FLAG 76H FOR DRY SUMP CHECK
0790 22              464                    RET
WATERNEW
PAGE 9

                     465                    END
                     466
                     467
```

```
VERSION 1.2k ASSEMBLY COMPLETE, 0 ERRORS FOUND
WATERNEW
PAGE 10

CHK_HV . . . . . C ADDR  0490H
CHK_LV . . . . . C ADDR  04B0H
C_HV . . . . . C ADDR  05A0H
C_LV . . . . . C ADDR  05A9H
DIS_M_ON . . . . . C ADDR  02D0H
DLY_2MIN . . . . . C ADDR  0500H
DLY_2SEC . . . . . C ADDR  01D8H
DRY. . . . . C ADDR  05ACH
HV . . . . . C ADDR  0630H
INI_LCD. . . . . C ADDR  02B3H
L1 . . . . . C ADDR  00FAH
L10. . . . . C ADDR  0326H
L11. . . . . C ADDR  0324H
L12. . . . . C ADDR  0322H
L130 . . . . . C ADDR  0130H
L2 . . . . . C ADDR  0136H
L3 . . . . . C ADDR  018BH
L300 . . . . . C ADDR  0300H
L4 . . . . . C ADDR  0113H
L430 . . . . . C ADDR  0430H
L436 . . . . . C ADDR  0436H
L439 . . . . . C ADDR  0439H
L43C . . . . . C ADDR  043CH
L43F . . . . . C ADDR  043FH
L448 . . . . . C ADDR  0448H
L492 . . . . . C ADDR  0492H
L4B2 . . . . . C ADDR  04B2H
L5 . . . . . C ADDR  0188H
L502 . . . . . C ADDR  0502H
L504 . . . . . C ADDR  0504H
L506 . . . . . C ADDR  0506H
L58F . . . . . C ADDR  058FH
L595 . . . . . C ADDR  0595H
L598 . . . . . C ADDR  0598H
L59A . . . . . C ADDR  059AH
L5A3 . . . . . C ADDR  05A3H
L6 . . . . . C ADDR  01DEH
L625 . . . . . C ADDR  0625H
L636 . . . . . C ADDR  0636H
L63F . . . . . C ADDR  063FH
L647 . . . . . C ADDR  0647H
L7 . . . . . C ADDR  01DCH
L752 . . . . . C ADDR  0752H
L754 . . . . . C ADDR  0754H
L756 . . . . . C ADDR  0756H
L758 . . . . . C ADDR  0758H
L764 . . . . . C ADDR  0764H
L767 . . . . . C ADDR  0767H
L76A . . . . . C ADDR  076AH
L770 . . . . . C ADDR  0770H
L778 . . . . . C ADDR  0778H
L780 . . . . . C ADDR  0780H
L8 . . . . . C ADDR  01DAH
L9 . . . . . C ADDR  0328H
LCD_RAM. . . . . C ADDR  00C0H
LOOP . . . . . C ADDR  011EH
LOOP1. . . . . C ADDR  012DH
LOOP2. . . . . C ADDR  013CH
WATERNEW
PAGE 11

LOOP3. . . . . C ADDR  0191H
LV . . . . . C ADDR  0639H
MAIN . . . . . C ADDR  0550H
```

```
MAIN_PGR . . . . . C ADDR  002FH
M_START. . . . . C ADDR  054DH
M_STOP . . . . . C ADDR  0226H
N_HV . . . . . C ADDR  062BH
N_LV . . . . . C ADDR  062EH
P1 . . . . . D ADDR  0090H
PREDEFINED
P2 . . . . . D ADDR  00A0H
PREDEFINED
P3 . . . . . D ADDR  00B0H
PREDEFINED
SCAN . . . . . C ADDR  010AH
SETT . . . . . C ADDR  00F0H
START. . . . . C ADDR  0000H
SUB_BR . . . . . C ADDR  0642H
TANK_FULL. . . . . C ADDR  0260H
TMR_10MIN. . . . . C ADDR  0750H
TMR_5MIN . . . . . C ADDR  0320H
WR_LCD . . . . . C ADDR  00A0H
```

REMOTELY PROGRAMMABLE RTC-INTERFACED MICROCONTROLLER FOR MULTIPLE DEVICE CONTROL

■ **RAJA GOPAL AKELLA**

This project based on Atmel AT89C52 and Dallas real-time-clock (RTC) chip DS12887 can be used to control and remotely program the switching operation of 24 electrically operated devices. The devices can be switched on/off at precise times repeatedly every day, every month. The microcontroller can be programmed for device control using a normal Philips TV remote control.

RC5 coding

Since the circuit makes use of Philips TV remote for device-switching time parameters, you need to know the fundamentals of the coding format used in these IR remotes.

The Philips IR format makes use of RC5 code, which is also known as 'bi-phase coding.' In RC5-coded signals (Fig. 2), each bit has a uniform duration. A transition in the middle of the time interval assigned to each bit encodes its logical value ('0' or '1'). A high-to-low transition assigns the bit a logic value of '0,' and a low-to-high transition assigns the bit a logic value of '1.' We need additional transitions at the beginning of each bit if a stream of equal bits is sent. However, there is no need of additional transitions if the next bit has a different logic value.

Table II shows how all the commands of an RC5 remote control are encoded.

The first two bits are 'start' bits, which are used to adjust and synchronise the receiver. These bits are used to calculate and analyse the bit length of the other bits.

The third bit is a 'toggle' bit, which is toggled every time a button is pressed at the remote control. This bit is used to identify whether the button is really pressed or whether an obstacle came in between the IR path of the remote and the IR receiver.

The five bits (A4 through A0) immediately following the toggle bit are used to identify the device (see Table III). So, a maximum of 32 devices can be identified to and respond individually to the same type of coding without any disturbance, i.e., one among the 64 devices can be identified uniquely. Addresses of some of the remotes are shown in Table III.

The six bits (C5 through C0) immediately following the five address bits are the control/command bits. Therefore a maximum of 64 commands can be equipped in an RC5-type remote. Some of the command codes (decimal equivalents), as used in this project, are

PARTS LIST

Semiconductors:	
IC1	- AT89C52 microcontroller
IC2	- 74LS573 octal D-type latch
IC3	- DS12887 real-time clock
IC4	- 74LS138 decoder
IC5	- 7400 NAND gate
IC6	- 82C55 programmable peripheral interface
IC7- IC9	- ULN2803 high current octal Darlington array
IC10	- 7805 5V regulator
IRX1	- TSOP1738 IR receiver module
BR1	- 1A bridge rectifier
T1	- BC547 npn transistor
Resistors (all ¼-watt, ±5% carbon):	
R1, R6-R29	- 4.7-kilo-ohm
R2, R3	- 10-kilo-ohm
R4	- 100-ohm
R5	- 1-kilo-ohm
VR1	- 10-kilo-ohm preset
VR2	- 1-kilo-ohm preset
Capacitors:	
C1, C2	- 33pF ceramic disk
C3	- 10µF, 16V electrolytic
C4-C8, C11	- 100nF ceramic disk
C9	- 1µF, 16V electrolytic capacitor
C10	- 1000µF, 35V electrolytic capacitor
Miscellaneous:	
X1	- 230V AC primary to 15V, 500mA secondary transformer
RL1-RL24	- 12V, 200-ohm, 1C/O relay
S1	- Push-to-on switch
	- 16×2 LCD module
X_{TAL}	- 11.09 MHz crystal

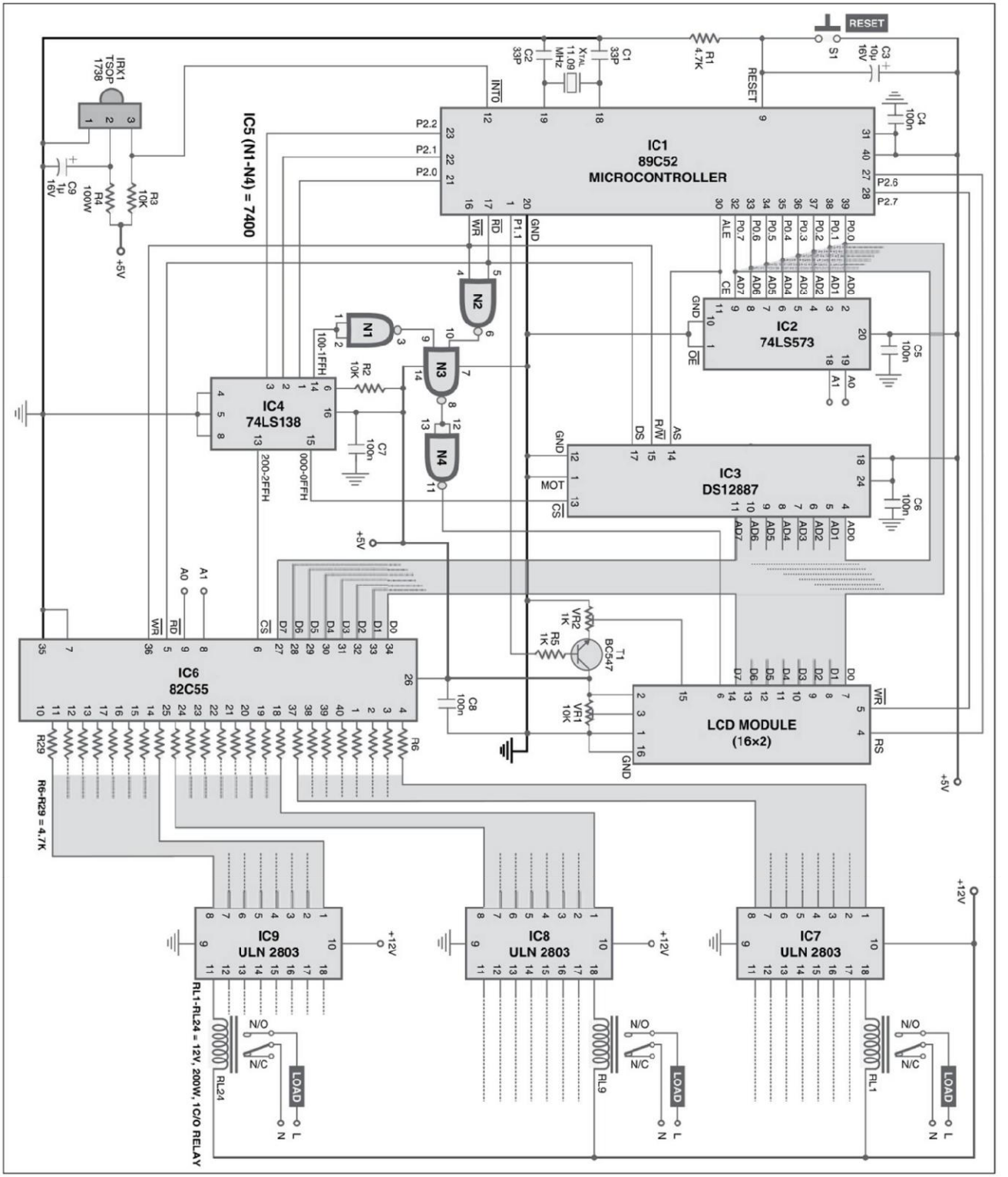


Fig. 1: Circuit for remotely programmable RTC-interfaced microcontroller for multiple device control

shown in Table IV.

When any of the command/control buttons on the remote is pressed, the coded signal is received by the IR receiver-demodulator TSOP1738. The output of the IR demodulator circuit is normally high, but when any

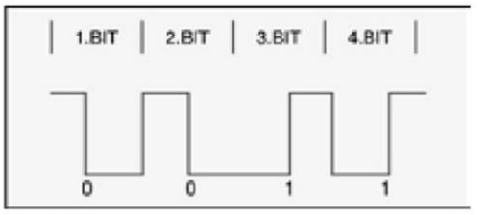


Fig. 2: RC5 coding scheme

of the buttons in the remote is pressed, a stream of low-going demodulated pulses will appear at its output. These pulses are fed to the external active-low interrupt input pin ($\overline{INT0}$) of

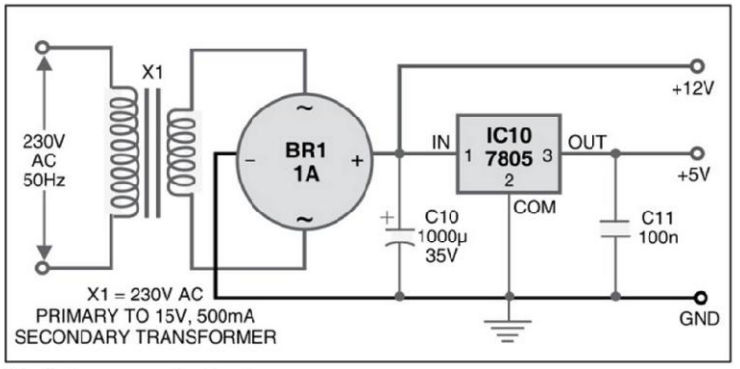


Fig. 3: Power supply circuit

89C52. On receipt of the first low-going pulse, the monitor program of 89C52 will get interrupted and jump to the location '0003H,' where the execution is redirected to 'receive' sub-routine of the program. The outputs from the sub-routine are:

1. Toggle bit, which toggles (either '0' or '1') each time the button in a remote is pressed.

2. Address byte, whose value is zero for a normal Philips-type TV remote control (see Table III).

3. Control byte, which has a unique value for each button in the remote control (see Table IV).

TABLE III
Remote Address Codes

Address	Device/Equipment
0	TV1
1	TV2
2	Videotext
3	Expansion for TV1 and TV2
4	Laser video player
5	VCR1
6	VCR2
7	Reserved
8	Sat1
27-31	Reserved

TABLE IV
Remote Command Codes

Button	Command	Function (as used)
0 – 9	0 – 9	Number keys
'--'	10	10+
'sfx'	36	20+
Mute	13	Delete task
AC	34	Clear prog memory
PWR	12	Change password
Timer	38	Change time
Search	30	Chk existing tasks
CH+	32	See next task
CH-	33	See before task
RCL	15	Turn on/off LCD back light
PP	14	Enter new task
Store	41	Enable/disable child lock

The hardware

Microcontroller AT89C52 is interfaced to DS12887 (RTC), a 16x2 LCD module and an 8255 programmable peripheral interface (PPI). The address-decoding circuitry comprises NAND gates 74LS00 and 3-to-8 line decoder 74LS138 as shown in Fig. 1. The interfacing circuitry for the external electrical appliances comprises Darlington array IC ULN2803. The addressing range of various peripheral devices is shown in Table I.

TABLE I
I/O Address Range of Peripheral Devices

Address range	Used address range	Device name
0000 00FF	0000 to 007F	DS12887 (RTC chip)
0100 01FF	0100	LCD Module (16×2)
0200 02FF	0200 to 0203	8255 (PPI)

Note. Please refer device datasheets for more details.

In 89C52 (IC1), port P0 is used for outputing multiplexed address (lower 8-bit) and data. The address is latched into 74LS573 (IC2) octal latch and RTC DS12887 (IC3) with the help of ALE (address latch-enable) output from

TABLE II
RC5 Coding Format

S	S	T	A4	A3	A2	A1	A0	C5	C4	C3	C2	C1	C0

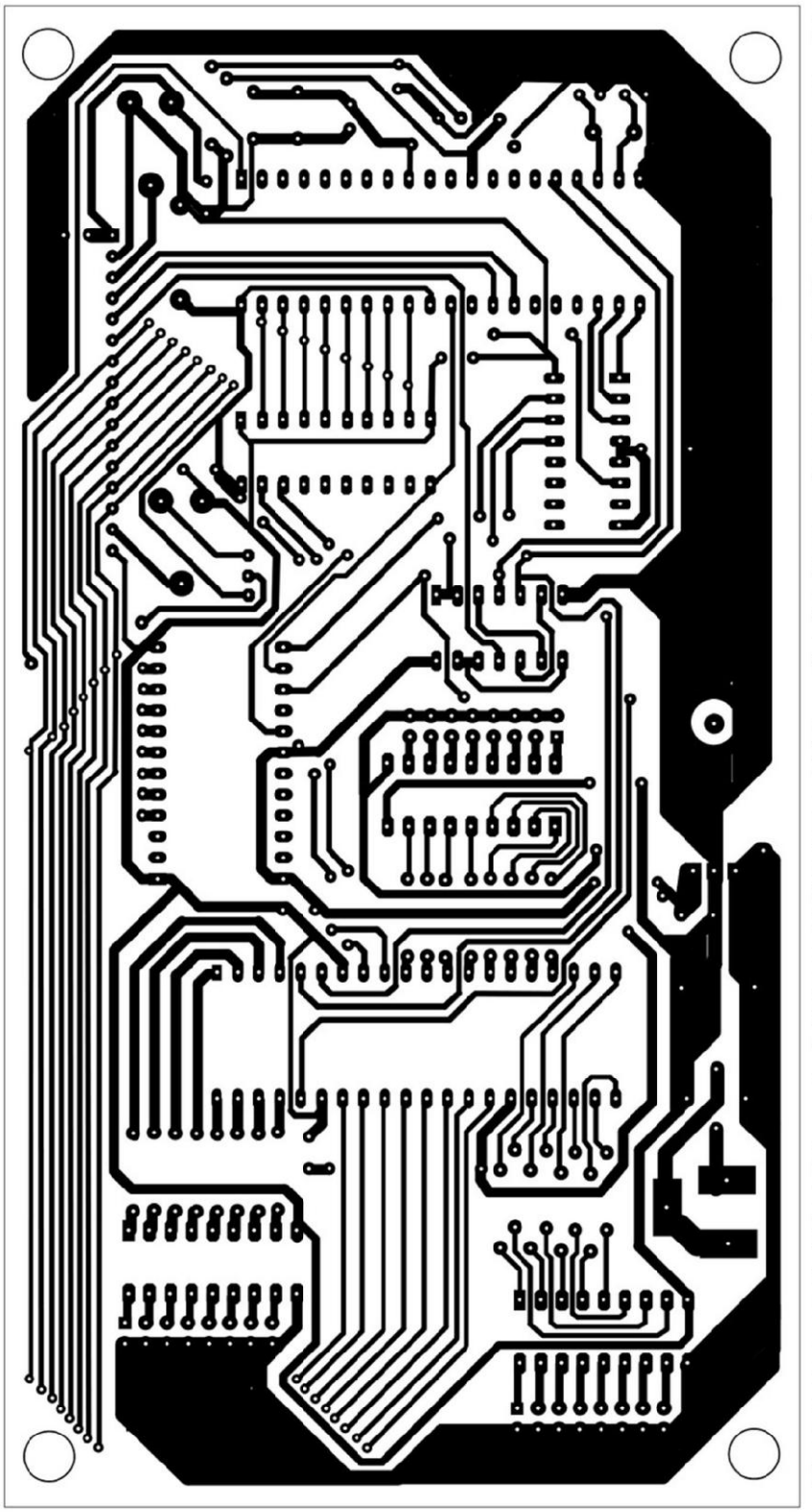


Fig. 4: Actual-size, single-side PCB for remotely programmable RTC-interfaced microcontroller for multiple device control

pin 30 of IC1.

Only two address lines from IC2 (A0 and A1) have been used for addressing the four registers of 8255 PPI (IC6) in conjunction with the chip-select signal at pin 6 (from IC4) and read/write signals from IC1. Higher-address bits from port P2 of IC1 (A8, A9 and A10 from output pins P2.0, P2.1 and P2.2) are used for generating the chip-select signals from 74LS138 (IC4) covering address ranges 000H-0FFH, 100H-1FF and 200-2FF for RTC, LCD module and PPI chip, respectively.

Quad NAND gate 7400 (IC5) in conjunction with read and write signals from IC1 and chip-select signal from pin 14 of IC4 is used for selecting the LCD module both during read and write cycles of IC1.

PPI chip 8255 is configured with port A, port B and port C as output ports for controlling up to 24 electrical appliances via relays RL1 through RL24. Relays are energised through high-current octal Darlington arrays inside ULN2803 (IC7 through IC9) in accordance with programmed data stored in the non-volatile RAM (NV RAM) of RTC chip DS12887. There is no need of connecting external free-wheeling diodes across relays as inbuilt diodes are provided in ULN2803 ICs.

All the 24 devices/electrical appliances are considered as 24 bits (three bytes at locations 200H, 201H and 202H) of the three ports (ports A, B and C) of 8255. The LCD is used for displaying real time (year, month, date, day and time in 24-hour mode) obtained from RTC DS12887 as also some other information during time setting, device programming, searching (device-switching programmed data), password entry, etc, as described later.

RTC DS12887 is clock-cum-calendar chip with 128 NV RAM (14 bytes used for its control registers and 114 bytes as general-purpose RAM). It has an inbuilt lithium battery and can retain stored data for over ten years in the absence of external power.

TABLE V

Memory Map of DS12887

Decimal	Memory Location	Data description	Pointer
0	0000	Seconds	Memory used by Clock
1	0001	Seconds Alarm	
2	0002	Minutes	
3	0003	Minutes Alarm	
4	0004	Hours	
5	0005	Hours Alarm	
6	0006	Day of the Week (Sun=1)	
7	0007	Date of the month	
8	0008	Month	
9	0009	Year	
10	000A	Register A	
11	000B	Register B	
12	000C	Register C	
13	000D	Register D	
14	000E	Data on Port A	24 bits treated as 24 devices
15	000F	Data on Port B	
16	0010	Data on Port C	
17	0011	Month	Prog 1 data
18	0012	Date	
19	0013	Hour	
20	0014	Minute	
21	0015	Device # (MSB indicates ON/OFF)	
22	0016	Month	Prog 2 data
23	0017	Date	
24	0018	Hour	
25	0019	Minute	
26	001A	Device # (MSB indicates ON/OFF)	
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
112	0070	Month	Prog 20 data
113	0071	Date	
114	0072	Hour	
115	0073	Minute	
116	0074	Device # (MSB indicates ON/OFF)	
117	0075	Month	Prog 21 data
118	0076	Date	
119	0077	Hour	
120	0078	Minute	
121	0079	Device # (MSB indicates ON/OFF)	
122	007A	Mem. Location not used	Nil
123	007B	*	Used to store the Password
124	007C	*	
125	007D	*	
126	007E	*	
127	007F	Pointer value	

Memory map of DS12887 is shown in Table V. Data stored in location 07FH (decimal 127) indicates the address of the last RAM location used. The relay-switching data that is output from ports A, B and C of the PPI is stored as consecutive bits at 00EH, 00FH and 010H locations of the RAM. The relay/device programming timing data is stored at five consecutive locations for each device. This data includes month, date, hour and minute in first four bytes, while the fifth byte contains 5-bit address of the device with MSB indicating 'on'/'off' status of the device. Bits 6 and 7 of this byte are 'don't care' bits. Address locations 123 through 126 are used for storing the 4-byte long password. Thus only 106 locations are available for storing the 5-byte long device data and as such the program for a maximum of only 21 devices out of 24 devices can be stored. The remaining three devices can be switched on/off through remote key operation as explained below.

Bit P1.1 output of IC1 is fed to transistor BC547 (T1) through R5. Transistor T1 acts like a switch for LCD backlight. So you can switch the backlight of LCD 'on'/'off' just by setting/resetting the P1.1 bit of 89C52.

Power supply (Fig. 3). While most of the circuit requires regulated 5V for its operation, the relays and Darlington drivers IC7 through IC9 (ULN2803) are operated with unregulated 12V DC supply. A step-down transformer rated at 15V AC secondary voltage at 500 mA is used to supply 12V unregulated and 5V regulated power to the circuit. The secondary output is rectified by 1A rated bridge rectifier BR1 and smoothed by 1000µF, 35V capacitor C10. The output from the capacitor is directly fed to all the relays and ULN2803 ICs. The same output is used for

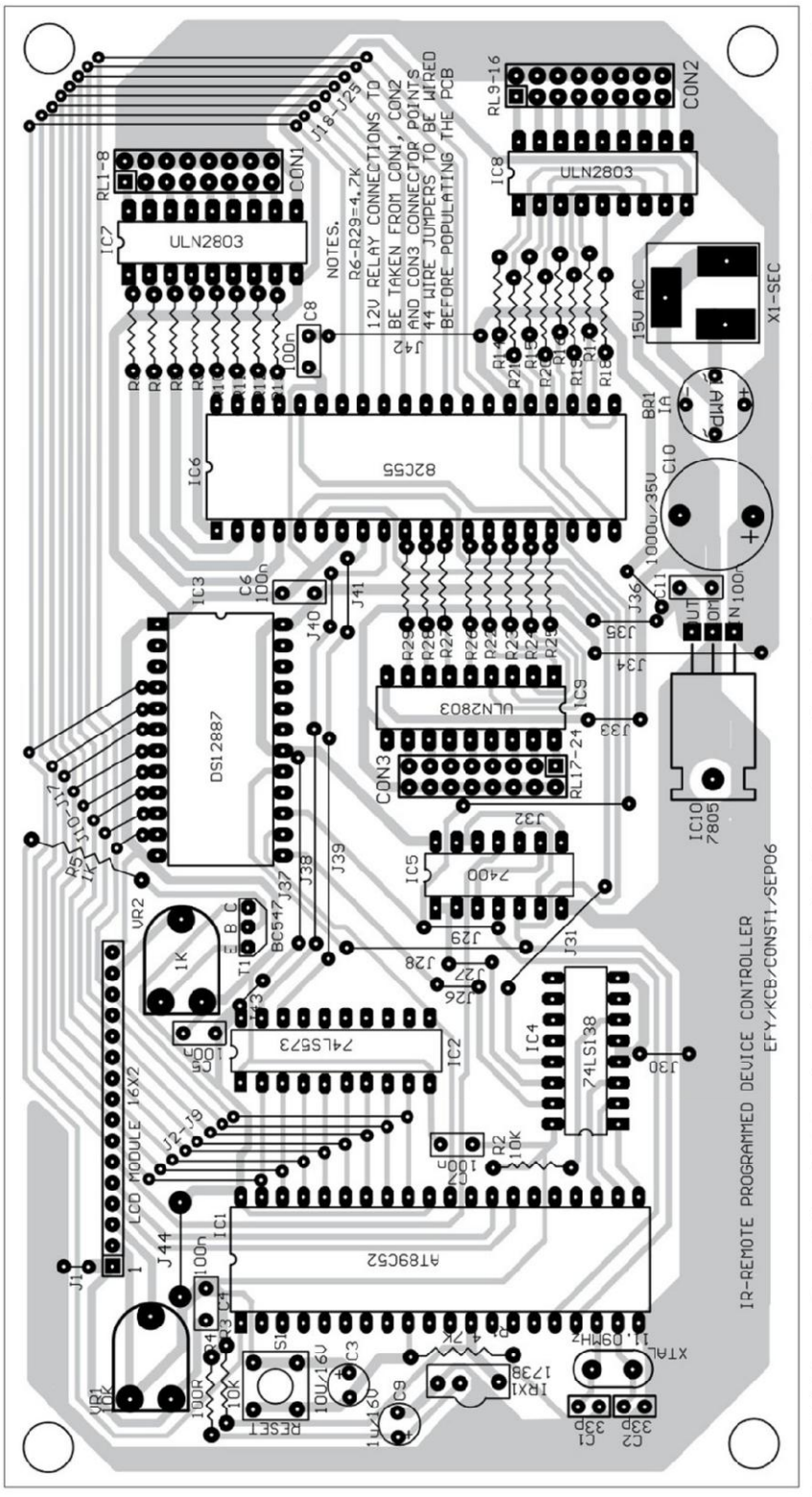


Fig. 5: Component layout for the PCB shown in Fig. 4

regulation by 7805 (IC10). The ripple in the regulator output is filtered by capacitor C11.

An actual-size, single-side PCB for the remotely-programmable RTC-interfaced microcontroller for multiple device control and power supply circuits (Figs 1 and 3) is shown in Fig. 4 and its component layout in Fig. 5. The connections for relays are to be extended from the 16-pin FRC connectors on the PCB. Each connector is meant for extending connections to eight relays.

The author's prototype is shown in Fig. 6.

Remote key operations

Refer Table IV for details of remote buttons/keys. The functions of these keys follow:

Keys '0' through '9', '--' and 'sfx'. Used to switch on/off the 24 devices manually. Each time you press any of these buttons, the status of the corresponding device will toggle, i.e., its output will be the complement of the previous state.

Button '--' is used as '10+' button. When it is pressed, the system waits for around three seconds before the next button (which should be between '0' and '9' to determine the device between '10' and '19') is pressed. Similarly, 'sfx' is used as '20+' key. The button following the 'sfx' button should be between '0' and '3' (as the project supports only 24 electrical appliances numbering '0' through '23').

RCL. Turns the LCD backlight 'on'/'off.'

PWR. Used to change the 4-digit password (initial value '0000'). When you press this button, the system will ask for the existing password. If the correct password is entered, it will ask for the new password. If a wrong password is entered, 'invalid' message will flash on the LCD.

Note that the password can be any 4-digit value, which need not be the numbers from '0000' to '9999.' Other buttons representing various codes are also accepted.

Fig. 6: Author's prototype

Timer. Used to change the real time. As the circuit operations depend on the real (set) time, changing the same is password-protected. A valid 4-digit password will let you change/set the time. When you press 'timer' button, the top row on the LCD defines the format 'Hr:Mn:ScWkDyMnYr.' You need to enter the valid data as follows:

Hr: 00 to 23 (24-hour mode)

Mn: 00 to 59 minutes

Sc: 00 to 59 seconds

Wk: 01 to 07 (01 is Sunday)

Dy: 01 to 31 dates

Mn: 01 to 12 (01 is January)

Yr: 00 to 99 (year)

Any value out of the range will not be accepted and message 'invalid value' will be displayed on the LCD.

Example

Program memory in DS12887

Task 1	Task 2	Task 3	...	...	Task 'n–1'	Task 'n'
When Task 2 is deleted, the whole memory is refreshed as:						
Task 1	Task 3	...	...	Task 'n–1' replaces task 'n–2'	Task 'n' replaces task 'n–1'	Empty

Store. Enables/disables the child lock function. You can lock the remote keypad by enabling the child lock. When you press this button, the system will prompt the message 'Lock?' or 'UnLock?' depending on the present status of the child lock. If '1' is pressed, the child lock feature is enabled/disabled. Any button other than '1' will be treated as zero.

PP. Takes you to programming of a task.

If the NV RAM is full in DS12887, the message 'prog memory full' will flash on the LCD.

If the memory is not full, a new device program is accepted by displaying a message in the first line of the LCD as 'Mn Dt Hr:Mn Dv S' and a blinking cursor will appear on the second line to take the data. 'Mn' indicates 'month' ('01' to '12'), 'Dt' indicates 'date' ('01' to '31'), 'Hr' indicates 'hours' ('00' to '23'), 'Mn' indicates 'minutes' ('00' to '59'), 'Dv' indicates 'device number' ('00' to '23') and 'S' stands for 'programmed status' ('1' for 'on' or '0' for 'off'). Enter the desired data in this format, which will get stored in the NV RAM of the RTC. If month (Mn) is entered as '00,' the same task will repeat every month on the same date and time. If date (Dt) is entered as '00,' the same task will repeat every day on the same time.

Search. Shows the existing device programs that are stored in the memory starting from location 011H onwards one by one. Each time, you need to press CH+/CH- button to move forward/backward. In this mode, you may delete the displayed device program data entry simply by pressing 'Mute' button. Then the program that is residing next to this task moves to the location of the deleted task and the whole memory is refreshed.

See the example shown above for clarity. The pointer value in memory location 007FH of DS12887 changes accordingly.

AC. Deletes the entire programmed data in one stroke. So use this key very cautiously.

RTC initialisation

When DS12887 is shipped from the factory, the oscillator is in disabled state. The following program will make DS12887 work and also reset the password to '0000' and make the program pointer to point to the location 0011H, i.e., it clears the existing tasks by making the program memory empty:

```
$MOD52
    ORG   00H
    JMP   MAIN
    ORG   20H
MAIN:   MOV     DPTR,   #000AH
        MOVX    A,      @DPTR
        ANL     A,      #0A0H
        MOVX    @DPTR,  A
        MOV     DPTR,   #007FH
        MOV     A,      #0011H
        MOVX    @DPTR,  A
        MOV     A,      #00H
        MOV     DPTR,   #007BH
        MOVX    @DPTR,  A
        INC     DPTR
        MOVX    @DPTR,  A
        INC     DPTR
        MOVX    @DPTR,  A
        INC     DPTR
        MOVX    @DPTR,  A
        JMP   $
    END
```

Before getting started, you need to make this program run for the first time after all the components and ICs are inserted into the circuit. That is, to make DS12887 work, burn the program shown above in microcontroller 89C52, put the programmed microcontroller in the circuit, switch on the circuit for five seconds and then turn it off. By doing so, the internal oscillator of DS12887 starts oscillating and IC DS12887 is now ready for use. Now, remove 89C52 from the circuit and load into it the main program to make the circuit work with all the features. (For more details on DS12887 RTC, refer to the datasheets.)

The monitor program in 89C52 gets the relevant time data (time, date, day, year, etc) from DS12887 RTC and displays it on the LCD. The data is also compared against the user-entered data (programmed timing data for multiple devices), which had been stored in the NV RAM of DS12887. When the timing data that was stored in the NV RAM equals the real-time data fetched from the DS12887, it sets/resets the MSB of the fifth byte of the stored program for that device.

Before burning the code for main program 'proj.asm' into AT89C52, erase the initialisation program 'rtcint.asm' that is programmed initially into it.

Download source code: http://www.efymag.com/admin/issuepdf/Remote%20Controlled.zip

REMOTE-CONTROLLED DIGITAL AUDIO PROCESSOR

■ KULAJIT SARMA

These days most audio systems come with remote controllers. However, no such facility is provided for normal audio amplifiers. Such audio controllers are not available even in kit form. This article presents an infrared (IR) remote-controlled digital audio processor. It is based on a microcontroller and can be used with any NEC-compatible full-function IR remote control.

This audio processor has enhanced features and can be easily customised to meet individual requirements as it is programmable. Its main features are:

1. Full remote control using any NEC-compatible IR remote control handset
2. Provision for four stereo input channels and one stereo output
3. Individual gain control for each input channel to handle different sources
4. Bass, midrange, treble, mute and attenuation control
5. 80-step control for volume and 15-step control for bass, midrange and treble
6. Settings displayed on two 7-segment light-emitting diode (LED) displays and eight individual LEDs

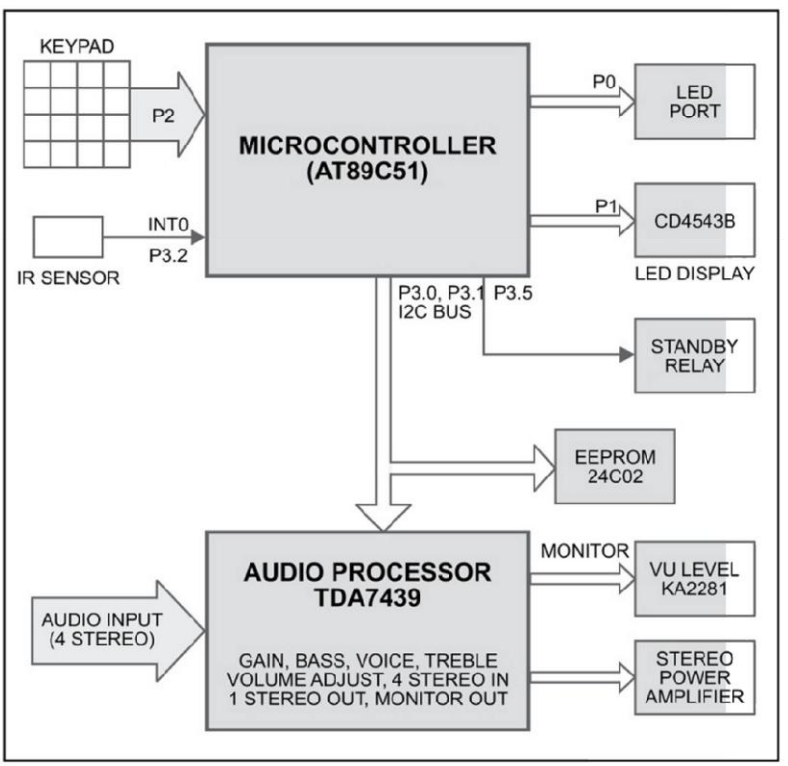


Fig. 1: Block diagram of the remote-controlled digital audio processor

PARTS LIST

Semiconductors:

IC1	-	AT89C51 microcontroller
IC2, IC3	-	CD4543 7-segment decoder/driver
IC4	-	TDA7439 audio processor
IC5	-	MC24C02 I²C EEPROM
IC6	-	KA2281 2-channel level meter driver
IC7	-	TSOP1238 IR receiver module
IC8	-	7809 9V regulator
IC9	-	7805 5V regulator
IC10	-	LM317 variable regulator
T1	-	BC558 pnp transistor
T2, T3, T5	-	BC547 npn transistor
T4	-	BD139 pnp transistor
BR1	-	W04M bridge rectifier
D1-D6	-	1N4004 rectifier diode
DIS1, DIS2	-	LTS543 7-segment display
DIS3	-	10-LED bargraph display
LED1-LED8	-	Red LED
LED9	-	Green LED

Resistors (all ¼-watt, ±5% carbon):

R1	-	8.2-kilo-ohm
R2-R24, R40-R49	-	1-kilo-ohm
R25, R28, R50, R53	-	10-kilo-ohm
R26, R29, R30, R34	-	2.7-kilo-ohm
R27	-	100-ohm
R31, R35	-	5.6-kilo-ohm
R32, R33	-	4.7-kilo-ohm
R36-R39	-	22-kilo-ohm
R51	-	220-kilo-ohm
R52	-	2.2-kilo-ohm

Capacitors:

C1, C2	-	33pF ceramic disk
C3, C10	-	10µF, 16V electrolytic
C4-C6, C39-C41	-	100nF ceramic disk
C7	-	4.7µF, 16V electrolytic
C8, C9	-	2.2µF, 16V electrolytic
C11, C20	-	5.6nF polyester
C12, C19	-	18nF polyester
C13, C18	-	22nF polyester
C14, C17	-	100nF polyester
C21-C28	-	0.47µF polyester
C29-C32	-	4.7µF, 25V electrolytic
C33, C34	-	10µF, 25V electrolytic
C35	-	1000µF, 25V electrolytic
C36	-	4700µF, 25V electrolytic
C37, C38	-	0.33µF ceramic disk
C42	-	470µF, 25V electrolytic

Miscellaneous:

X1	-	230V AC primary to 12V, 1A secondary transformer
RL1	-	9V, 160Ω, 2 C/O relay
X_{TAL}	-	12MHz crystal
S1-S7	-	Push-to-on switch
S8	-	On/Off switch
Remote	-	Creative's remote (NEC-compatible format)

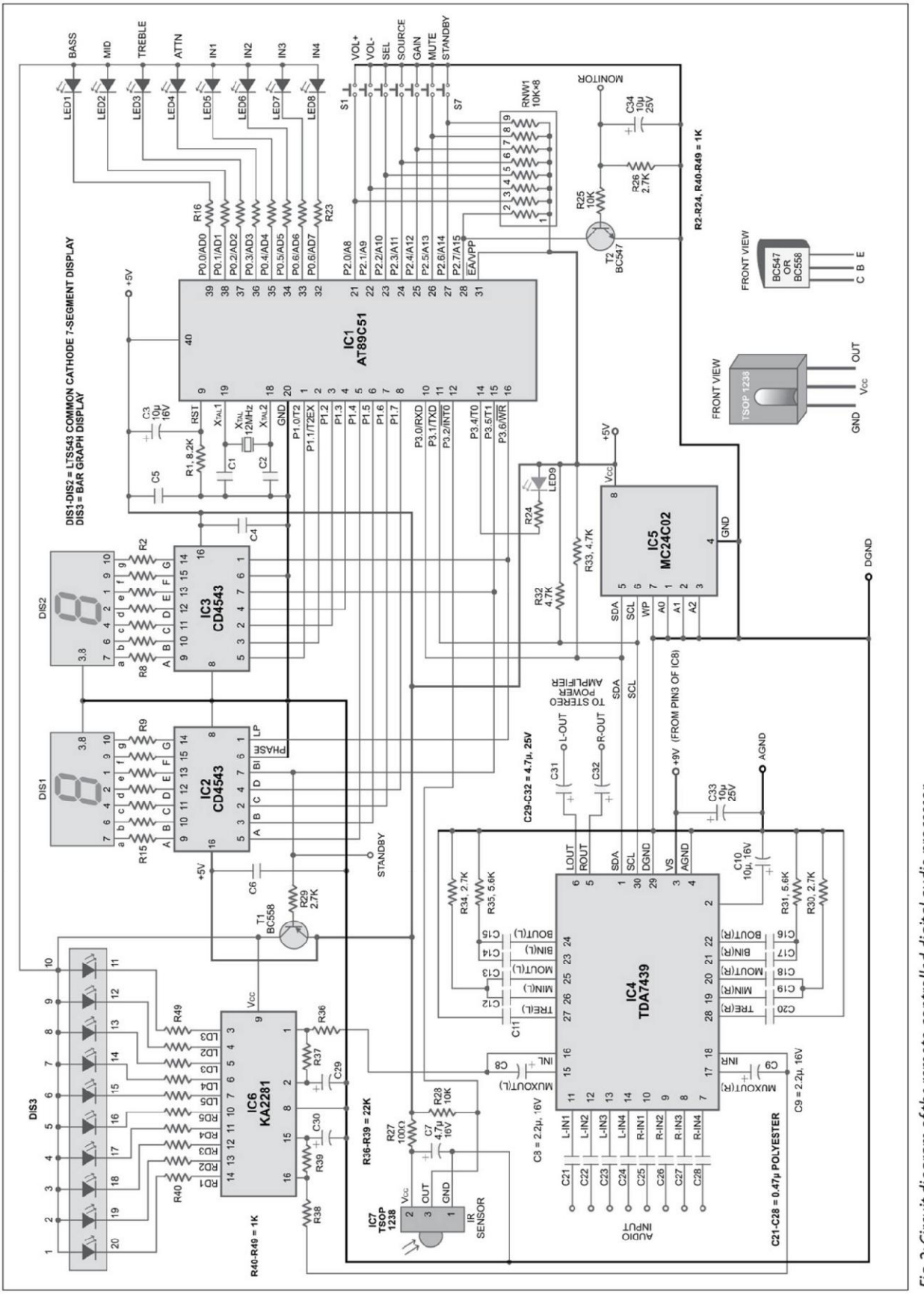


Fig. 2: Circuit diagram of the remote-controlled digital audio processor

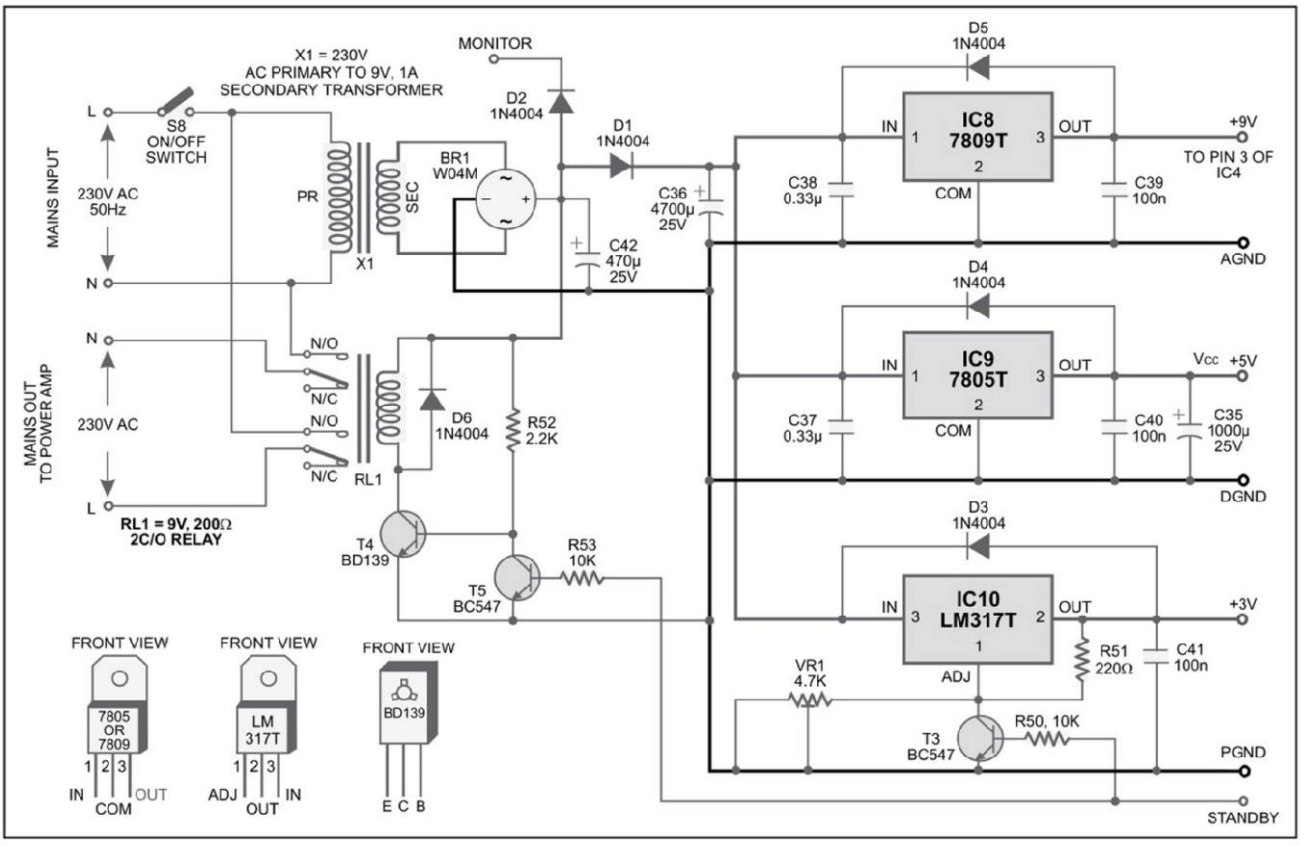


Fig. 3: Power supply

7. Stereo VU level indication on 10-LED bar display
8. Full-function keys on-board for audio amplifier control
9. All settings stored on the EEPROM
10. Standby mode for amplifier power control

Circuit description

Fig. 1 shows the block diagram of the remote-controlled digital audio processor. The system comprises Atmel's AT89C51 microcontroller (IC1), TDA7439 audio processor from SGS-Thomson (IC4) and I2C bus compatible MC24C02 EEPROM (IC5). The microcontroller chip is programmed to control all the digital processes of the system. The audio processor controls all the audio amplifier functions and is compatible with I2C bus. All the commands from the remote control are received through the IR sensor. The audio amplifier can also be controlled using the on-board keys.

Microcontroller. The function of the microcontroller is to receive commands (through port P3.2) from the remote handset, program audio controls as per the commands and update the EEPROM. A delay in updating the EEPROM is deliberately provided because normally the listener will change the value of a parameter continuously until he is satisfied.

The 40-pin AT89C51 microcontroller has four 8-bit input/output (I/O) ports.

Port 0 is used for indicating through LEDs the various functions selected via the remote/on-board keys.

Port 1 drives the 7-segment display using 7-segment latch/decoder/driver IC CD4543.

Port 2 is pulled up via resistor network RNW1 and used for manual key control.

Pins P3.0 and P3.1 of the microcontroller are used as serial data (SDA) and serial clock (SCL) lines for the I2C bus for communicating with the audio processor (TDA7439) and EEPROM (MC24C02). These two lines are connected to pull-up resistors, which are required for I2C bus devices. P3.2 receives the remote commands through the IR receiver module. Pin P3.4 is used for flashing LED9 whenever a remote command is received or

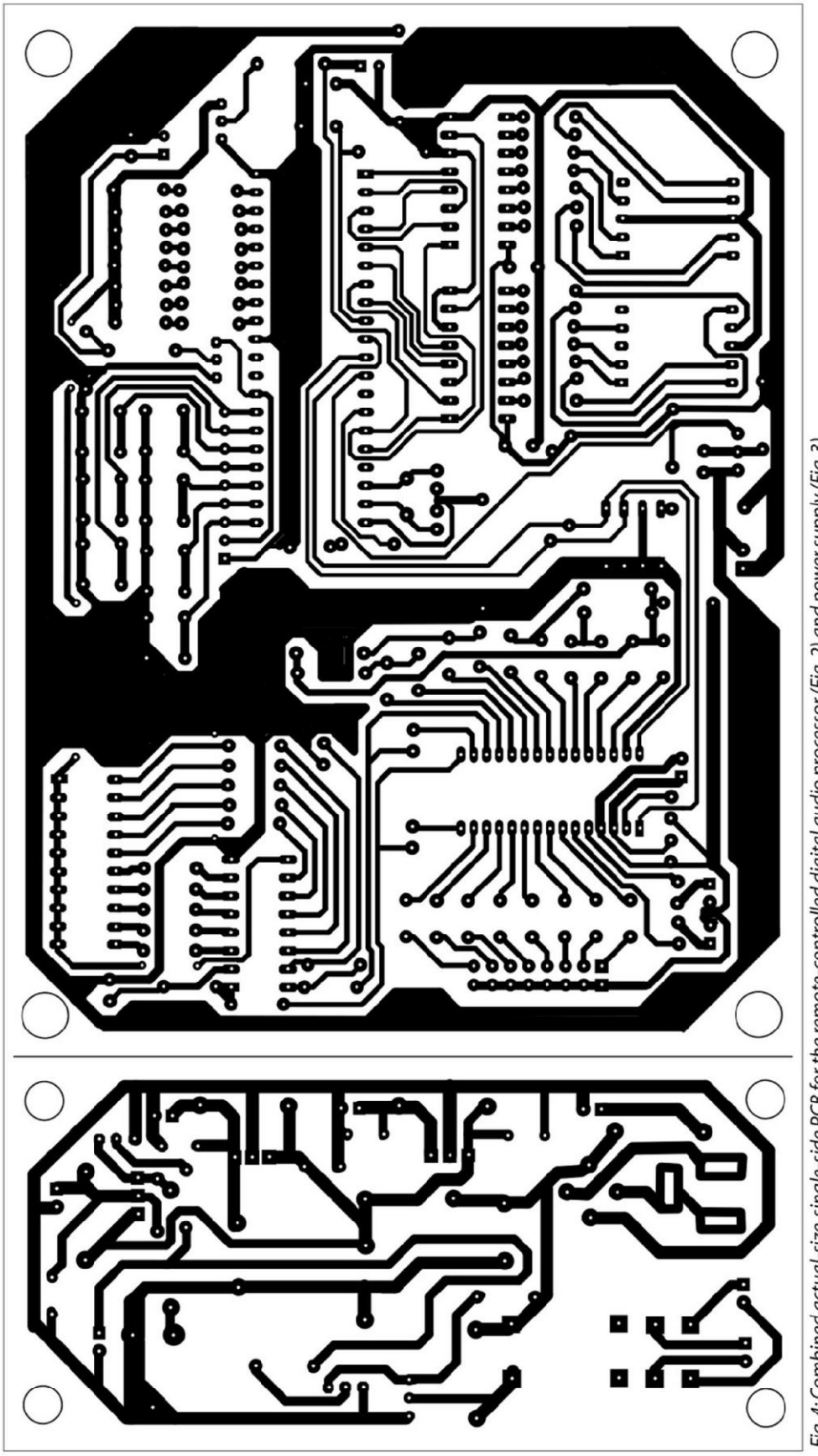


Fig. 4: Combined actual-size, single-side PCB for the remote-controlled digital audio processor (Fig. 2) and power supply (Fig. 3)

any key is pressed.

The microcontroller also checks the functioning of the memory (MC24C02) and the audio processor (TDA7439). If it is not communicating with these two ICs on the I2C bus, it flashes the volume level on the 7-segment displays.

Memory. IC MC24C02 is an I2C-bus compatible 2k-bit EEPROM organised as 256×8-bit that can retain data for more than ten years. Various parameters can be stored in it.

To obviate the loss of latest settings in the case of power failure, the microcontroller stores all the audio settings of the user in the EEPROM. The memory ensures that the microcontroller will read the last saved settings from the EEPROM when power resumes. Using SCL and SDA lines, the microcontroller can read and write data for all the parameters.

For more details on I2C bus and memory interface, please refer to the MC24C02 datasheet. Audio parameters can be set using the remote control handset or the on-board keys as per the details given under the 'remote control' section.

Audio processor. IC TDA7439 is a single-chip I2C-bus compat-

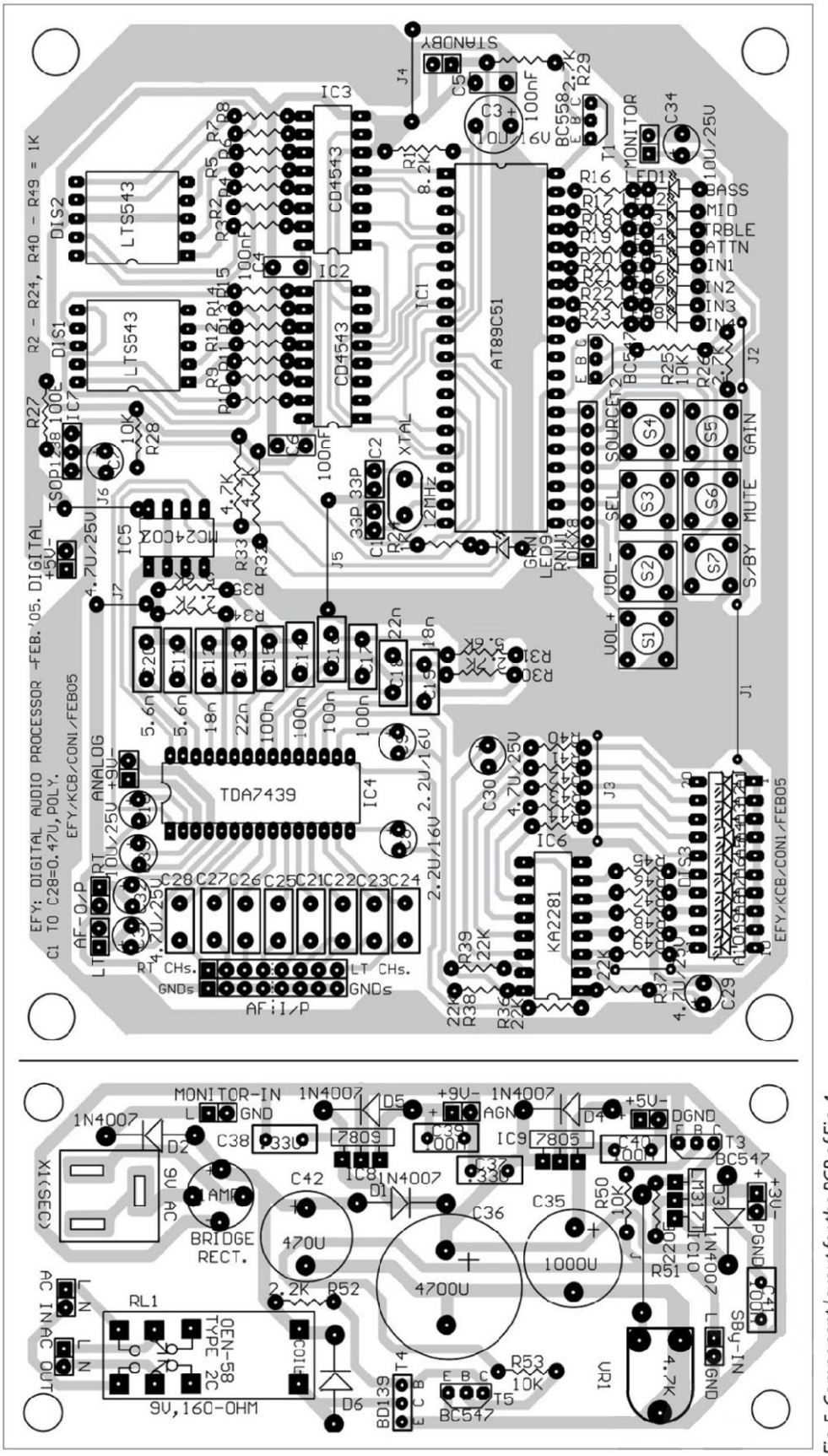


Fig. 5: Component layout for the PCB of Fig. 4

ible audio controller that is used to control all the functions of the audio amplifier. The output from any (up to four) stereo preamplifier is fed to the audio processor (TDA7439). The microcontroller can control volume, treble, bass, attenuation, gain and other functions of each channel separately. All these parameters are programmed by the microcontroller using SCL and SDA lines, which it shares with the memory IC and the audio processor.

Data transmission from the microcontroller to the audio processor (IC TDA7439) and the memory (MC24C02) and vice versa takes place through the two-wire I2C-bus interface consisting of SDA and SCL, which are connected to P3.0 (RXD) and P3.1 (TXD) of the microcontroller, respectively. Here, the microcontroller unit acts as the master and the audio processor and the memory act as slave devices. Any of these three devices can act as the transmitter or the receiver under the control of the master.

Some of the conditions to communicate through the I2C bus are:

1. Data validity: The data on the SDA

line must be stable during the high period of the clock. The high and low states of the data line can change only when the clock signal on the SCL line is low.

2. Start and Stop: A start condition is a high-to-low transition of the SDA line while SCL is high. The stop condition is a low-to-high transition of the SDA line while SCL is high.

3. Byte format: Every byte transferred on the SDA line must contain eight bits. The most significant bit (MSB) is transferred first.

4. Acknowledge: Each byte must be followed by an acknowledgement bit. The acknowledge clock pulse is generated by the master. The transmitter releases the SDA line (high) during the acknowledge clock pulse. The receiver must pull down the SDA line during the acknowledge clock pulse so that it remains low during the high period of this clock pulse.

To program any of the parameters, the following interface protocol is used for sending the data from the microcontroller to TDA7439. The interface protocol comprises:

1. A start condition (S)
2. A chip address byte containing the TDA7439 address (88H) followed by an acknowledgement bit (ACK)
3. A sub-address byte followed by an ACK. The first four bits (LSB) of this byte indicate the function selected (e.g., input select, bass, treble and volume). The fifth bit indicates incremental/non-incremental bus (1/0) and the sixth, seventh and eighth bits are 'don't care' bits.
4. A sequence of data followed by an ACK. The data pertains to the value for the selected function.
5. A stop condition (P)

In the case of non-incremental bus, the data bytes correspond only to the function selected. If the fifth bit is high, the sub-address is automatically incremented with each data byte. This mode is useful for initialising the device. For actual values of data bytes for each function, refer to the datasheet of TDA7439.

Similar protocol is followed for sending data to/from the microcontroller to MC24C02 EEPROM by using its chip address as 'A0H'.

Power supply. Fig. 3 shows the power supply circuit for the remote-controlled digital audio processor. The AC mains is stepped down by transformer X1 to deliver a secondary output of 9V AC at 1A. The transformer output is rectified by full-wave bridge rectifier BR1 and filtered by capacitor C42. Regulators IC8 and IC9 provide regulated 5V and 9V power supplies, respectively. IC10 acts as the variable power supply regulator. It is set to provide 3V regulated supply by adjusting preset VR1. Capacitors C39, C40 and C41 bypass any ripple in the regulated outputs. This supply is not used in the circuit. However, the readers can use the same for powering devices like a Walkman.

As capacitors above 10 μF are connected to the outputs of regulator ICs, diodes D3 through D5 provide protection to the regulator ICs, respectively, in case their inputs short to ground. Relay RL1 is normally energised to provide mains to the power amplifier. In standby mode, it is de-energised. Switch S2 is the 'on'/'off' switch.

Software

The software was assembled using Metalink's ASM51 assembler, which is freely available for download. The source code has been extensively commented for easier understanding. It can be divided into the following segments in the order of listing:

1. Variable and constant definitions
2. Delay routines
3. IR decoding routines
4. Keyboard routines
5. TDA7439 communication
6. MC24C02 communication
7. I2C bus routines
8. Display routines
9. IR and key command processing
10. Timer 1 interrupt handler
11. Main program

On reset, the microcontroller executes the main program as follows:

1. Initialise the microcontroller's registers and random-access memory (RAM) locations.

2. Read Standby and Mute status from the EEPROM and initialise TDA7439 accordingly.

3. Read various audio parameters from the EEPROM and initialise the audio processor.

4. Initialise the display and LED port.

5. Loop infinitely as follows, waiting for events:

• Enable the interrupts.

• Check the monitor input for AC power-off. If the power goes off, jump to the power-off sequence routine.

• Else, if a new key is pressed, call the DO_KEY routine to process the key. For this, check whether the NEW_KEY bit is set. This bit is cleared after the command is processed.

• Else, if a new IR command is received, call the DO_COM routine to process the remote command. For this, check whether the NEW_COM (new IR command available) bit is set. This bit is cleared after the command is processed.

• Jump to the beginning of the loop.

6. Power-off sequence. Save all the settings to the EEPROM, and turn off the display and standby relay.

Since the output of the IR sensor is connected to pin 12 (INT0) of the microcontroller, an external interrupt occurs whenever a code is received. The algorithm for decoding the IR stream is completely implemented in the 'external interrupt 0' handler routine. This routine sets NEW_COM (02H in bit memory) if a new command is available. The decoded command byte is stored in 'Command' (location 021H in the internal RAM). The main routine checks for NEW_COM bit continuously in a loop. Timer 0 is exclusively used by this routine to determine the pulse timings.

Decoding the IR stream involves the following steps:

1. Since every code is transmitted twice, reject the first by introducing a delay of 85 milliseconds (ms) and start timer 0. The second transmission is detected by checking for no-overflow timer 0. In all other cases, timer 0 will overflow.

2. For second transmission, check the timer 0 count to determine the length of the leader pulse (9 ms). If the pulse length is between 8.1 ms and 9.7 ms, it will be recognised as valid. Skip the following 4.5ms silence.

3. To detect the incoming bits, timer 0 is configured to use the strobe signal such that the counter runs between the interval periods of bits. The value of the counter is then used to determine whether the incoming bit is '0', '1' or 'Stop.' This is implemented in the RECEIVE_BIT routine.

4. If the first bit received is 'Stop,' repeat the last command by setting the NEW_COM bit.

5. Else, receive the rest seven bits. Compare the received byte with the custom code (C_Code). If these don't match, return error.

6. Receive the next byte and compare with the custom code. If these don't match, return error.

7. Receive the next byte and store in 'Command.'

8. Receive the next byte and check whether it is complement value of 'Command.' Else, return error.

9. Receive 'Stop' bit.

10. Set NEW_COM and return from interrupt.

Other parts of the source code are relatively straightforward and self-explanatory.

Remote control. The micro-controller can accept commands from any IR remote that uses NEC transmission format. These remote controllers are readily available in the market and use μPD6121, PT2221 or a compatible IC. Here, we've used Creative's remote handset.

All the functions of the system can be controlled fully using the remote or the on-board keys. By default, the display shows the volume setting and LEDs indicate the channel selected. LED9 glows momentarily whenever a command from the remote is received or any key is pressed.

Function adjustments are detailed below:

1. Volume: Use Vol+/Vol- key to increase/decrease the volume. The volume settings are shown on the two-digit, 7-segment display. Steps can be varied between '1' and '80.'

2. Mute and Standby: Using 'Mute' and 'Standby' buttons, you can toggle the mute and standby status, respectively. If 'Mute' is pressed, the display will show '00.' In 'Standby' mode, the relay de-energises to switch off the main amplifier. All the LEDs and displays, except LED9, turn off to indicate the standby status.

3. Input Select: To select the audio input source, press 'Channel' key until the desired channel is selected. The LED corresponding to the selected channel turns on and the input gain setting for that channel is displayed for

five seconds. Thereafter, the volume level is displayed on the 7-segment display.

4. Input Gain set: Press 'Gain' key. The LED corresponding to the channel will start blinking and the gain value is displayed. Use Vol+/Vol- key to increase/decrease the gain for that channel. Note that the gain can be varied from '1' to '15.' If you press 'Gain' key once more, and no key is pressed for five seconds, it will exit the gain setting mode and the volume level is displayed.

5. Audio: Press 'Audio Set' (Menu) key to adjust bass, middle, treble and attenuation one by one. Each time 'Audio Set' key is pressed, the LED corresponding to the selected function turns on and the function value is displayed. Once the required function is selected, use Vol+ and Vol- to adjust the setting. Bass, middle and treble can be varied from '07' to '7.' Values '0' through '7' indicate 'Boost' and '00' through '07' indicate 'Cut.' Attenuation can be varied from '0' to '40.'

Construction

The circuit can be easily assembled on any PCB with IC base. Before you install the microcontroller, memory and audio processor in their sockets and solder the IR receiver module, make sure that the supply voltage is correct. All parts, except the audio processor (TDA7439), require 5V DC supply. The audio processor is powered by 9V DC.

Download source code: http://www.efymag.com/admin/issuepdf/Audio%20Processor.zip

SOLAR CHARGER FOR DUSK-TO-DAWN USE

■ **ARUN KUMAR VADLA**

As the sources of conventional energy deplete day by day, resorting to alternative sources of energy like solar and wind energy has become need of the hour.

Solar-powered lighting systems are already available in rural as well as urban areas. These include solar lanterns, solar home lighting systems, solar streetlights, solar garden lights and solar power packs. All of them consist of four components: solar photovoltaic module, rechargeable battery, solar charge controller and load.

In the solar-powered lighting system, the solar charge controller plays an important role as the system's overall

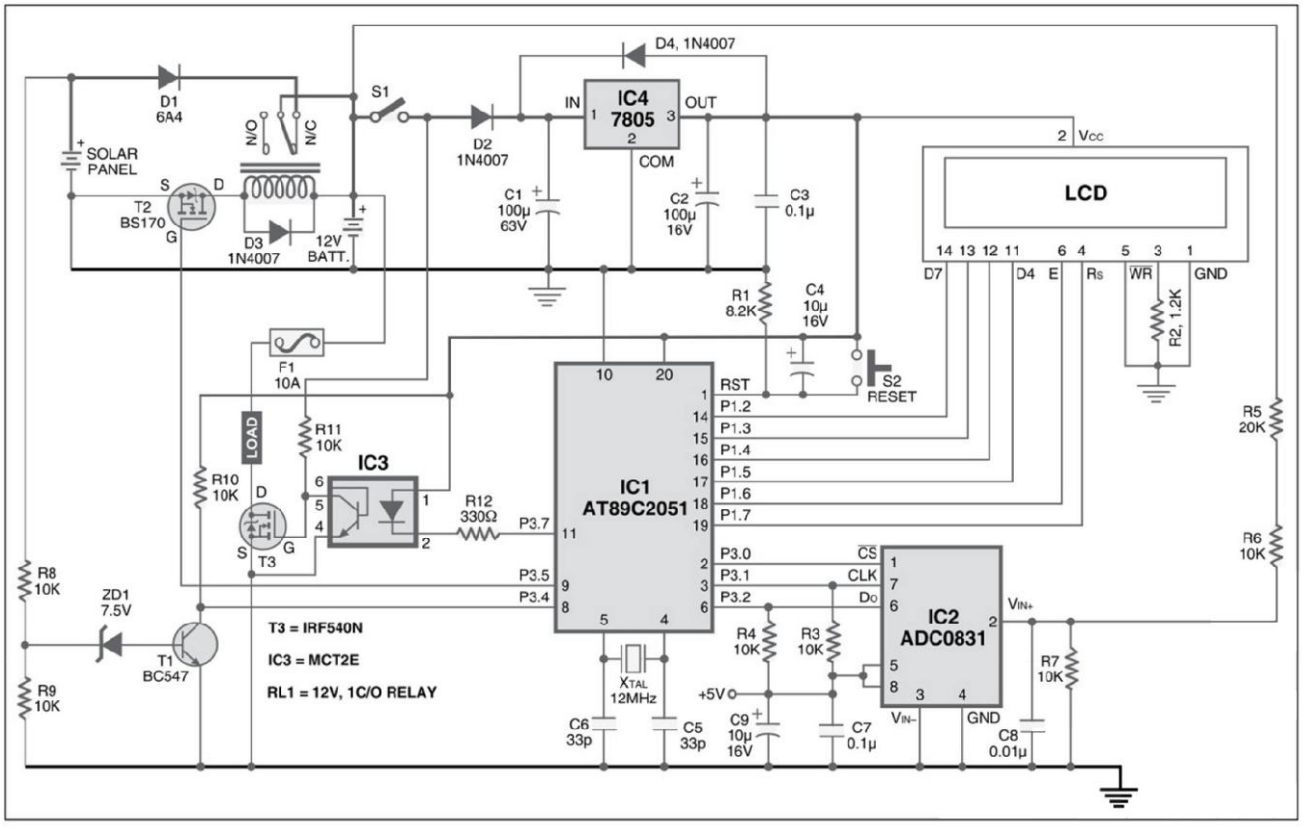


Fig. 1: Circuit of microcontroller-based solar charger

success depends mainly on it. It is considered as an indispensable link between the solar panel, battery and load.

The microcontroller-based solar charge controller described here has the following features:

1. Automatic dusk-to-dawn operation of the load
2. Built-in digital voltmeter (0V-20V range)
3. Parallel- or shunt-type regulation
4. Overcharge protection
5. System status display on LCD

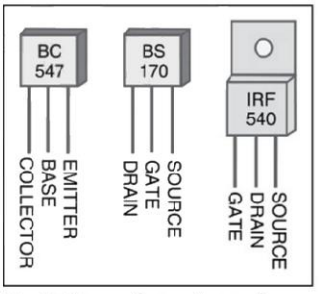


Fig. 2: Pin configurations of BC547, BS170 and IRF540

PARTS LIST

Semiconductors:

IC1	- AT89C2051 microcontroller
IC2	- ADC0831 analogue-to-digital converter
IC3	- MCT2E optocoupler
IC4	- 7805, 5V regulator
T1	- BC547 npn transistor
T2	- BS170 n-channel MOSFET
T3	- IRF540 n-channel MOSFET
D1	- 6A4 rectifier diode
D2-D4	- 1N4007 rectifier diode
ZD1	- 7.5V zener diode

Resistors (all ¼-watt, ±5% carbon):

R1	- 8.2-kilo-ohm
R2	- 1.2-kilo-ohm
R3, R4, R6-R11	- 10-kilo-ohm
R5	- 20-kilo-ohm
R12	- 330-ohm

Capacitors:

C1	- 100µF, 63V electrolytic
C2	- 100µF, 16V electrolytic
C3, C7	- 0.1µF ceramic disk
C4, C9	- 10µF, 16V electrolytic
C5, C6	- 33pF ceramic disk
C8	- 0.01µF ceramic disk

Miscellaneous:

S1	- On/off switch
S2	- Push-to-on switch
RL1	- 12V, 1C/O relay
X_{TAL}	- 12MHz crystal
LCD	- 16×2 line display
Solar panel	- 10-40W
	- 10A fuse

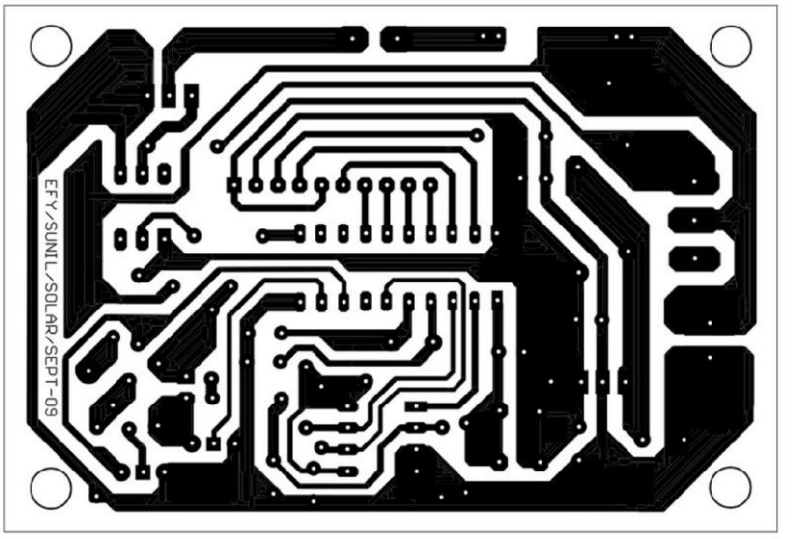


Fig. 3: A single-side, actual-size PCB layout for microcontroller-based solar charger

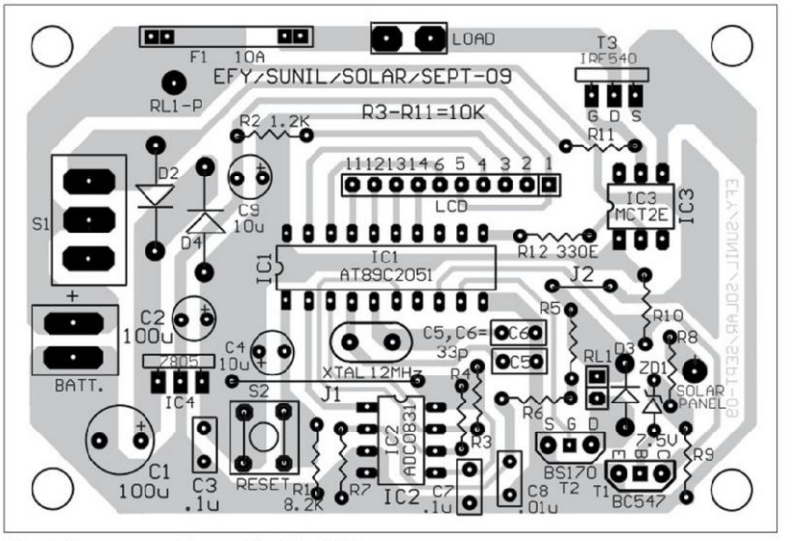


Fig. 4: Component layout for the PCB

6. Deep-discharge protection
7. Low battery lock
8. Charging current changes to 'pulsed' at full charge
9. Low current consumption
10. Highly efficient design based on microcontroller
11. Suitable for 10-40W solar panels for 10A load

The circuit of the solar charge controller is shown in Fig. 1. It comprises microcontroller AT89C2051, serial analogue-to-digital converter ADC0831, optocoupler MCT2E, regulator 7805, MOSFETs BS170 and IRF540N, transistor BC547, LCD and a few discrete components. Component description is given below.

Microcontroller. Microcontroller AT89C2051 is the heart of the circuit. It is a low-voltage, high-performance, 8-bit microcontroller that features 2 kB of Flash, 128 bytes of RAM, 15 input/output (I/O) lines, two 16-bit timers/counters, a five-vector two-level interrupt architecture, a full-duplex serial port, a precision analogue comparator, on-chip oscillator and clock circuitry. A 12MHz crystal is used for providing the basic clock frequency. All I/O pins are reset to '1' as soon as RST pin goes high. Holding RST pin high for two machine cycles, while the oscillator is running, resets the device. Power-on reset is derived from resistor R1 and capacitor C4. Switch S2 is used for manual reset.

Serial ADC. The microcontroller monitors the battery voltage with the help of an analogue-to-digital converter. The ADC0831 is an 8-bit successive approximation analogue-to-digital converter with a serial I/O and very low conversion time of typically 32 µs. The differential analogue voltage input allows increase of the

common-mode rejection and offsetting of the analogue zero input voltage. In addition, the voltage reference input can be adjusted to allow encoding of any smaller analogue voltage span to the full eight bits of resolution. It is available in an 8-pin PDIP package and can be interfaced to the microcontroller with only three wires.

LCD module. The system status and battery voltage are displayed on an LCD based on HD44780 controller. The backlight feature of the LCD makes it readable even in low light conditions. The LCD is used here in 4-bit mode to save the microcontroller's port pins. Usually the 8-bit mode of interfacing with a microcontroller requires eleven pins, but in 4-bit mode the LCD can be interfaced to the microcontroller using only seven pins.

Solar panel. The solar panel used here is meant to charge a 12V battery and the wattage can range from 10 to 40 watts. The peak unloaded voltage output of the solar panel will be around 19 volts. Higher-wattage panels can be used with some modifications to the controller unit.

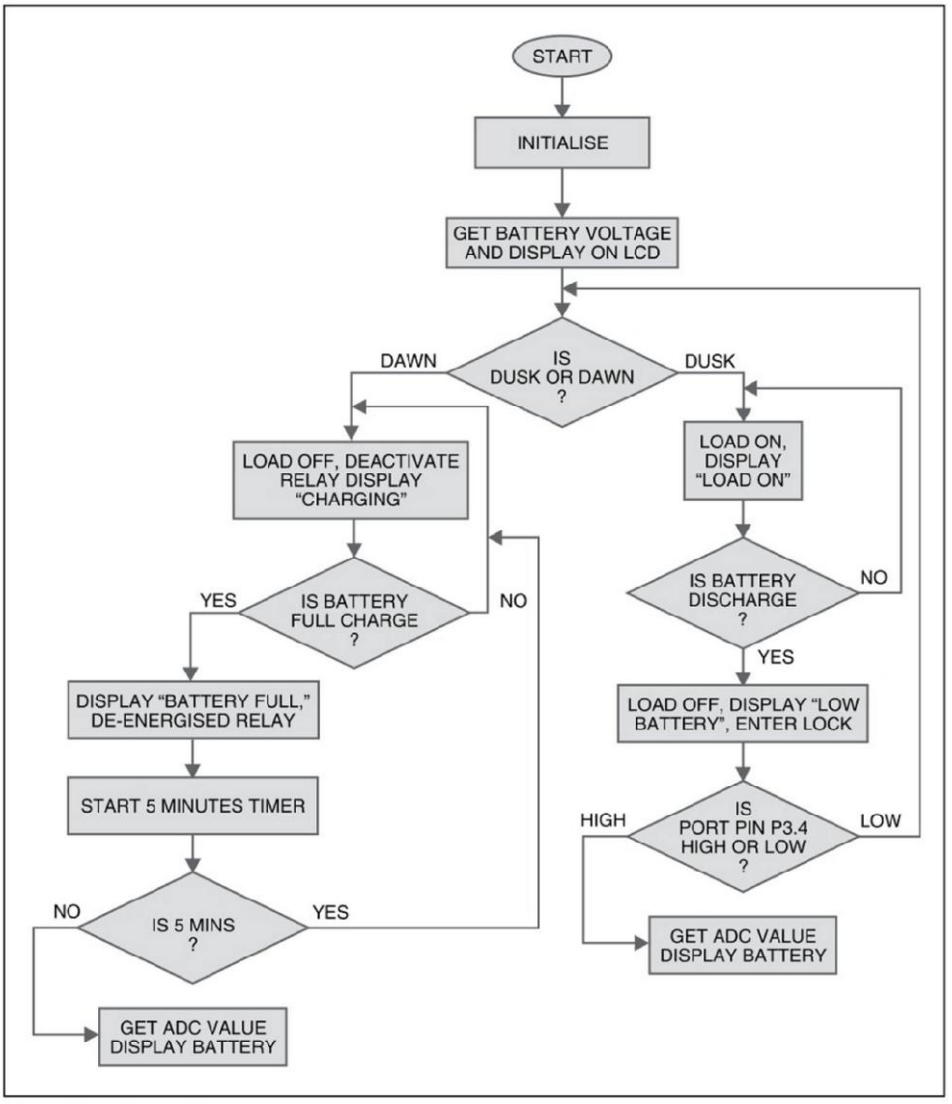


Fig. 5: Flow-chart of the source program

Rechargeable battery. The solar energy is converted into electrical energy and stored in a 12V lead-acid battery. The ampere-hour capacity ranges from 5 Ah to 100 Ah.

Dusk-to-dawn sensor. Normally, in a solar-photovoltaic-based installation—for example, solar home lighting system, solar lantern or solar streetlight—the load (the light) is switched on at dusk (evening) and switched off at dawn (morning). During daytime, the load is disconnected from the battery and the battery is recharged with current from the solar panel. The microcontroller needs to know the presence of the solar panel voltage to decide whether the load is to be connected to or disconnected from the battery, or whether the battery should be in charging mode or discharging mode. A simple sensor circuit is built using a potential divider formed around resistors R8 and R9, zener diode ZD1 and transistor T1 for the presence of panel voltage.

Charge control. Relay RL1 connects the solar panel to the battery through diode D1. Under normal conditions, it allows the charging current from the panel to flow into the battery. When the battery is at full charge (14.0V), the charging current becomes 'pulsed.' To keep the overall current consumption of the solar controller low, normally-closed (N/C) contacts of the relay are used and the relay is normally in de-energised state.

Load control. One terminal of the load is connected to the battery through fuse F1 and another terminal of

the load to an n-channel power MOSFET T3. MOFETs are voltage-driven devices that require virtually no drive current. The load current should be limited to 10A. One additional MOSFET is connected in parallel for more than 10A load current.

Circuit description

Basically, there are two methods of controlling the charging current: series regulation and parallel (shunt) regulation. A series regulator is inserted between the solar panel and the battery. The series type of regulation 'wastes' a lot of energy while charging the battery as the control circuitry is always active and series regulator requires the input voltage to be 3-4 volts higher than the output voltage. The current and voltage output of a solar panel is governed by the angle of incidence of light, which keeps varying.

Parallel regulation is preferred in solar field. In parallel regulation, the control circuitry allows the charging current (even in mA) to flow into the battery and stop charging once the battery is fully charged. At this stage, the charging current is wasted by converting into heat (current is passed through low-value, high-wattage resistor); this part of the regulation dissipates a lot of heat.

In this project, we have used parallel regulation technique but instead of wasting the charging current as heat, we have made it pulsed and applied to the battery to keep the battery topped-up.

After power-on, the microcontroller reads the battery voltage with the help of the ADC and displays the values on the LCD. It monitors the input signal from the dusk-to-dawn sensor and activates the load or charging relay RL1 accordingly. The digital voltmeter works up to 20V. As Vref of the ADC is connected to VCC (5V), the input voltage to the ADC cannot exceed +5V. A potential divider is used at pin 2 of the ADC (IC2) using resistors R5, R6 and R7 to scale down the voltage from 0V-20V to 0V-05V. The ADC output is multiplied four times and displayed on the LCD as battery voltage.

When the solar panel voltage is present, the dusk-to-dawn sensor provides a signal to the microcontroller, which then displays 'charging' message on the LCD. During charging, the battery voltage is continuously monitored. When the voltage reaches 14.0V, the microcontroller interrupts the charging current by energising the relay, which is connected to MOSFET BS170 (T2), and starts a 5-minute timer. During this stage, the LCD shows "battery full."

After five minutes, the relay reconnects the panel to the battery. This way, the charging current is pulsed at the intervals of five minutes and the cycle repeats until the panel voltage is present.

When the panel voltage falls below the zener diode (ZD1) voltage of the dusk-to-dawn sensor, the microcontroller senses this and activates the load by switching on MOSFET T3 via optocoupler IC3 and "load on" message is displayed.

In this mode, the microcontroller monitors for low battery. When the battery voltage drops below 10 volts, the microcontroller turns off the load by switching off MOSFET T3 and "battery low—load off" message is displayed.

Normally, when the load is switched off, the battery voltage tends to rise back and the load oscillates between 'on' and 'off' states. To avoid this, the microcontroller employs a hysteresis control by entering into a 'lock' mode during low-battery state and comes out of the lock mode when the dusk-to-dawn sensor receives the panel voltage (the next morning). During lock mode, the microcontroller keeps converting the ADC value and displays the battery voltage on the LCD.

Construction and testing

Pin configurations of transistor BC547, MOSFET BS170 and MOSFET IRF540 are shown in Fig. 2. An actual-size, single-side PCB for the microcontroller-based solar charger is shown in Fig. 3 and its component layout in Fig. 4. Wire the circuit on the PCB. Prior to inserting the programmed microcontroller into the PCB, check for soldering mistakes like shorts, and for proper connections using a multimeter. Mount power MOSFET IRF540N on a suitable heat-sink. Before switching on the controller unit, connect the leads of the battery, load and solar panel at appropriate places on the board.

Switch on the unit and the message "Solar Charge Controller–EFY" is displayed on the LCD for two seconds. The system status messages are displayed on line 1 of the LCD and the battery voltage is displayed on line 2. A small graphic representing the battery status is also displayed on line 2 of the LCD.

EFY note. 1. If the unit is switched on without the solar panel connected, the "Battery Low—Load Off"

message is displayed irrespective of the battery voltage. The display changes to "charging" as soon as the panel is connected.

2. There will be slight variation in the voltage displayed because of the tolerance levels of potential-divider resistors in the ADC section and Vref of the ADC being directly connected to VCC (the output of 7805 has an accuracy of 2-5 per cent) instead of dedicated temperature-compensated voltage reference.

Software

The source program for the project is written in Assembly language and assembled using Metalink's ASM51 assembler, which is freely available on the Internet for download. It is well commented for easy understanding and works as per the flow-chart shown in Fig. 5. The hex file 'solar.hex' is to be burnt into the microcontroller.

Download source code: http://www.efymag.com/admin/issuepdf/Microcontroller%20Based%20Solar%20Charger.zip

SOLAR.ASM

```
$MOD51
    ; LCD 4-BIT MODE CONNECTIONS
    RS   EQU   P1.7   ;LCD   REGISTER   SELECT

LINE
    EN EQU P1.6 ;LCD ENABLE LINE
    DB4 EQU P1.5 ;
    DB5 EQU P1.4 ;
    DB6 EQU P1.3 ;
    DB7 EQU P1.2 ;
    ; ADC0831 CONNECTIONS
    CS EQU P3.0
    CLK EQU P3.1
    DO EQU P3.2
    ;INPUT & OUTPUT

    DYI   EQU   P3.4   ;   SOLAR   PANEL   VOLTAGE

SENSOR
    CHG_RL   EQU   P3.5   ;   CHARGING   CONTROL

  RELAY
    LD_RL EQU P3.7 ; LOAD CONTROL RELAY
    DSEG

    ORG 0020H

    VAL1: DS 1
    VAL2: DS 1
    VAL3: DS 1
    ADC_VAL: DS 1
    BUF: DS 1
    CNT1: DS 1
    CNT2: DS 1
    IMG: DS 1
    FLAGS: DS 1
    OCF BIT FLAGS.0 ; OVER CHARGE FLAG
    LBF BIT FLAGS.1 ; LOW BATT FLAG
    CSEG

    ORG 0000H
    JMP MAIN

    ORG 000BH    ;Timer Interrupt0
    JMP COUNTDOWN
    MAIN:       MOV SP,#50H
                MOV P3,#0FFH
                MOV P1,#0FFH
                CLR CHG_RL
                CLR LD_RL
                LCALL PWR_DELAY
                LCALL INIT
                SETB CLK
                SETB DO
                SETB CS
                SETB DYI
                MOV VAL1,#00H
                MOV VAL2,#00H
                MOV VAL3,#00H
                MOV FLAGS,#00H
LOADCHAR: MOV BUF,#40H
    LCALL CMD
                MOV DPTR,#RCHAR
    REP:        CLR A
                MOVC A,@A+DPTR
                JZ SCREEN1
                MOV BUF,A
                LCALL DAT
                INC DPTR
                SJMP REP
SCREEN1: MOV BUF,#80H
    LCALL CMD
    MOV DPTR,#MSG1
    HERE:       CLR A
    MOVC A,@A+DPTR
    JZ NEXT
    MOV BUF,A
    LCALL DAT
    INC DPTR
    SJMP HERE
    NEXT:       MOV BUF,#0C0H
    LCALL CMD
    MOV DPTR,#MSG2
    HERE1:      CLR A
    MOVC A,@A+DPTR
    JZ OVER
    MOV BUF,A
    LCALL DAT
```

```
    INC DPTR
    SJMP HERE1
    OVER:       LCALL ONE_SEC_DELAY
                LCALL ONE_SEC_DELAY
                LCALL CLEAR
                MOV BUF,#0C0H
                LCALL CMD
                MOV DPTR,#MSG7
    HERE2:      CLR A
                MOVC A,@A+DPTR
                JZ CONVERT
                MOV BUF,A
                LCALL DAT
                INC DPTR
                SJMP HERE2
    CONVERT:LCALL DDELAY
    CLR CS ; INITIATE CONVERSION
    SETB CLK
    CLR CLK ; FIRST CLOCK
 SETB CLK
    CLR CLK ; SECOND CLOCK
     MOV A,#00H ; CLEAR A
    MOV R5,#08H ; 8 CLOCK PULSES
    AGAIN:      MOV C,DO
    RLC A
    SETB CLK
    CLR CLK
    DJNZ R5,AGAIN
    SETB CS
    MOV ADC_VAL,A
    MOV B,#79D
    MUL AB ; PRODUCT IN AB
    MOV R1,B ; HIGH BYTE IN B
    MOV R2,A ; LOW BYTE IN A
    LCALL HEX2BCD
                MOV VAL1,R7
                MOV VAL2,R6
                MOV VAL3,R5
                LCALL SENDVAL2LCD
    CHECK:      JNB LBF,CHECK2 ; SEE IF ANY FLAGS ARE
SET ,i,e LOW BATT FLAG OR BATT FULL FLAG
                JB DYI,CONVERT
    CHECK2:     JNB OCF,PROCEED
                JB DYI,NIGHT
                SJMP CONVERT
 PROCEED:       JB DYI,NIGHT
                CLR LD_RL ; OFF LOAD
                CLR LBF ; CLEAR LOW BATT FLAG
                MOV A,VAL2 ; SEE IF BATT.IS FULL
                XRL A,#04H
                JZ FULLCHG
                CLR  CHG_RL  ;  CONNECT  BATT.  TO

 PANEL
                MOV  DPTR,#MSG4  ;  DISPLAY  CHARGING

   MSG
                MOV IMG,#00H
                LCALL SENDSTAT2LCD
                LJMP CONVERT
 FULLCHG:       SETB   OCF   ;SET   OVERCHARGE
   FLAG
    SETB   CHG_RL   ;DISCONNECT   BATT.FROM

PANEL
    MOV   TH0,#03CH   ;START   5   MIN   TIMER

HERE
    MOV   TL0,#0B0H   ;DISCONNECT   BATT   FROM

PANEL
    MOV CNT1,#200D
    MOV CNT2,#30D
    SETB ET0
    SETB TR0
    SETB EA
    MOV   DPTR,#MSG5   ;   DISPLAY   BATT.FULL

  MSG
    MOV IMG,#01H
    LCALL SENDSTAT2LCD
    LJMP CONVERT
    NIGHT:      CLR  CHG_RL  ;  RECONNECT  BATT.

    TO PANEL
    CLR   TR0   ;   STOP   TIMER0   INCASE   ITS
                                                RUN-
NING
    CLR OCF ; CLEAR OVER CHARGE FLAG
    SETB LD_RL ; CONNECT LOAD TO BATT.
                MOV A,VAL1
                XRL A,#00H
                JZ LOWBAT
                MOV DPTR,#MSG3 ; DISPLAY LOAD ON MSG
                MOV IMG,#02H
                LCALL SENDSTAT2LCD
                LJMP CONVERT
    LOWBAT: SETB LBF
    CLR   LD_RL   ;   DISCONNECT   LOAD   FROM
                                                BATT.
    MOV   DPTR,#MSG6   ;   DISPLAY   BAT.OW   AND

LOAD OFF MSG
    MOV IMG,#03H
    LCALL SENDSTAT2LCD
    LJMP CONVERT
 SENDVAL2LCD: MOV BUF,#0C7H
    LCALL CMD
    MOV A,VAL1
    ORL A,#30H
    MOV BUF,A
    LCALL DAT
    MOV A,VAL2
    ORL A,#30H
    MOV BUF,A
    LCALL DAT
    MOV BUF,#'.'
    LCALL DAT
 MOV A,VAL3
    ORL A,#30H
    MOV BUF,A
    LCALL DAT
 RET
```

```
 SENDSTAT2LCD:  MOV BUF,#080H
    LCALL CMD
 HERE3:         CLR A
    MOVC A,@A+DPTR
    JZ PICT
    MOV BUF,A
    LCALL DAT
    INC DPTR
    SJMP HERE3
 PICT:          MOV BUF,#0CEH
    LCALL CMD
    MOV BUF,IMG
    LCALL DAT
 BACK:          RET
;*********************************
; TIMER0 ISR (5 MINUTES TIMER)
;*********************************
 COUNTDOWN:     CLR TR0
                MOV TH0,#03CH
                MOV TL0,#0B0H
                SETB TR0
                DJNZ CNT1,BACK2
                MOV CNT1,#200D
                DJNZ CNT2,BACK2
    CLR TR0 ; OFF 5 MIN TIMER
                CLR ET0
                CLR OCF ; CLEAR OVER CHARGE FLAG
                CLR  CHG_RL  ;  RE-CONNECT  BATT  TO

PANEL
    BACK2:      RETI
 Hex2BCD: MOV R3,#00D
    MOV R4,#00D
    MOV R5,#00D
    MOV R6,#00D
    MOV R7,#00D
    ACALL H2B
    RET
    H2B:        MOV B,#10D
    MOV A,R2
    DIV AB
    MOV R3,B ;
    MOV B,#10 ; R7,R6,R5,R4,R3
    DIV AB
    MOV R4,B
    MOV R5,A
    CJNE  R1,#00H,HIGH_BYTE  ;  CHECK  FOR

    HIGH BYTE
    SJMP ENDD
 HIGH_BYTE:     MOV A,#6
                ADD A,R3
                MOV B,#10
                DIV AB
                MOV R3,B
                ADD A,#5
                ADD A,R4
                MOV B,#10
                DIV AB
                MOV R4,B
                ADD A,#2
                ADD A,R5
                MOV B,#10
                DIV AB
                MOV R5,B
                CJNE R6,#00D,ADD_IT
                SJMP CONTINUE
 ADD_IT:        ADD A,R6
 CONTINUE:      MOV R6,A
    DJNZ R1,HIGH_BYTE
    MOV B, #10D
    MOV A,R6
    DIV AB
    MOV R6,B
    MOV R7,A
    ENDD: RET
ONE_SEC_DELAY:  MOV  R0,#10D  ;  One  second

delay routine
    RZ3: MOV R1,#100D
    RZ1: MOV R2,#250D
    RZ2: NOP
                NOP
                DJNZ R2,RZ2
                DJNZ R1,RZ1
                DJNZ R0,RZ3
    RET
PWR_DELAY: ; 15 mSec DELAY FOR LCD TO INTIALIZE AF-
TER POWER-ON
    MOV R4,#100D
    H2: MOV R3,#250D
    H1: DJNZ R3,H1
    DJNZ R4,H2
    RET
;******LCD SUBROUTINES********
    CMD: PUSH ACC ; SAVE ACCUMULATOR
    SETB EN
    CLR RS ; SELECT SEND COMMAND
    MOV A,BUF ; PUT DATA BYTE IN ACC
    MOV  C,  ACC.4  ;  LOAD  HIGH  NIBBLE  ON

DATA BUS
    MOV  DB4,C  ;  ONE  BIT  AT  A  TIME
                                          US-
ING...
    MOV C, ACC.5 ; BIT MOVE OPERATOINS
    MOV DB5,C
    MOV C, ACC.6
    MOV DB6,C
    MOV C, ACC.7
    MOV DB7,C
    CLR EN
    NOP
    SETB EN ; PULSE THE ENABLE LINE
    MOV  C,  ACC.0  ;  SIMILARLY,  LOAD  LOW

NIBBLE
    MOV DB4,C
    MOV C, ACC.1
    MOV DB5,C
    MOV C, ACC.2
    MOV DB6,C
    MOV C, ACC.3
    MOV DB7,C
```

```
     CLR EN
    NOP
    SETB EN ; PULSE THE ENABLE LINE
    LCALL MSDELAY
    POP ACC
    RET
; ******LCD SUBROUTINES*******
    DAT: PUSH ACC ; SAVE ACCUMULATOR
    SETB EN
    SETB RS ; SELECT SEND DATA
    MOV A, BUF ; PUT DATA BYTE IN ACC
    MOV  C,  ACC.4  ;  LOAD  HIGH  NIBBLE  ON

DATA BUS
    MOV  DB4,C  ;  ONE  BIT  AT  A  TIME
                                          US-
ING...
    MOV C, ACC.5 ; BIT MOVE OPERATOINS
    MOV DB5,C
    MOV C, ACC.6
    MOV DB6,C
    MOV C, ACC.7
    MOV DB7,C
    CLR EN
    NOP
    SETB EN ; PULSE THE ENABLE LINE
    MOV  C,  ACC.0  ;  SIMILARLY,  LOAD  LOW

NIBBLE
    MOV DB4,C
    MOV C, ACC.1
    MOV DB5,C
    MOV C, ACC.2
    MOV DB6,C
    MOV C, ACC.3
    MOV DB7,C
    CLR EN
    NOP
    SETB EN ; PULSE THE ENABLE LINENOP
    LCALL MSDELAY
    POP ACC
    RET
;******LCD SUBROUTINES******
    CLEAR: MOV BUF,#01H
    LCALL CMD
    RET
;******LCD SUBROUTINES*******
    DDELAY: MOV R5,#41D ; 4.1 mS DELAY
    QT2:        MOV R6,#50D
    QT1:        DJNZ R6,QT1
    DJNZ R5,QT2
    RET
;******LCD SUBROUTINES*******
 MSDELAY: MOV R5,#26D
    QT22:       MOV R6,#50D
    QT11:       DJNZ R6,QT11
    DJNZ R5,QT22
    RET
;******LCD SUBROUTINES*******
    INIT: MOV BUF,#30H ; FUNCTION
                SET - DATA BITS, LINES, FONTS
    LCALL CMD
    ACALL DDELAY ;INITIAL DELAY 4.1MSEC
    MOV BUF,#30H ; FUNCTION SET -
                                    DATA  BITS,
LINES, FONTS
    LCALL CMD
    ACALL DDELAY ;INITIAL DELAY 4.1MSEC
    MOV  BUF,#30H  ;  FUNCTION  SET  -  DATA

BITS, LINES, FONTS
    LCALL CMD
    ACALL DDELAY ;INITIAL DELAY 4.1MSEC
    MOV  BUF,#28H  ;2  LINES  5X7,  4-BIT

MODE
    LCALL CMD
 MOV BUF,#0CH ; DISPLAY ON
    LCALL CMD
    MOV  BUF,#01H  ;  CLEAR  DISPLAY,  HOME

CURSOR
    LCALL CMD
    MOV BUF,#06H ; SET ENTRY MODE
    LCALL  CMD  ;  INCREMENT  CURSOR  RIGHT,
                                          NO
SHIFT
 RET

            ORG 0320H
    MSG1:   DB ' SOLAR CHARGE ',00H
    MSG2:   DB ' CONTROLLER-EFY ',00H
    MSG3:   DB ' LOAD ON ',00H
    MSG4:   DB ' CHARGING ',00H
    MSG5:   DB ' BATTERY FULL ',00H
    MSG6:   DB 'BAT.LOW-LOAD OFF',00H
    MSG7:   DB 'Volts: ',00H
            ORG 0400H
    RCHAR:  DB  04D,31D,17D,31D,17D,31D,
                                    17D,31D  ;
CHARGING

DB      04D,31D,31D,31D,31D,31D,
                                    31D,31D ; FULL

DB      31D,31D,14D,04D,04D,14D,
                                    31D,31D ; LOAD

DB      04D,31D,17D,17D,17D,17D,
                                    17D,31D ; LOW
BATT

DB      31D,31D,31D,31D,31D,31D,
                                    31D,31D

DB      31D,31D,31D,31D,31D,31D,
                                    31D,31D

DB      31D,31D,31D,31D,31D,31D,
                                    31D,31D

DB      031D,31D,31D,31D,31D,31D,
            31D,31D,00H
            END
```

AUTOMATIC FLUSH SYSTEM

■ K.S. SANKAR

In manual flush systems, the user presses a button, which opens a flush valve allowing mains-pressure water to flow into the bowl, or sometimes the user presses directly a flush lever (a handle connected to a flushometer). The valve contains a pneumatic mechanism that closes it after a preset time.

Today, manual flush system has been replaced with a sensor-operated system that automatically flushes the fixture when the user departs.

The microcontroller-based automatic flush system presented here uses an infrared sensor to detect a user approaching the fixture, then it waits until the user departs.

PARTS LIST

Semiconductor:

IC1	- 7805, 5V regulator
IC2	- AT89C2051 microcontroller
IC3	- CD4050 hex non-inverting buffer
T1	- BC548 npn transistor
IRX1	- TSOP1738 IR receiver module
D1-D5	- 1N4007 rectifier diode
LED1-LED5	- 5mm LED
IR LED1, IR LED2	- IR LED

Resistors (all ¼-watt, ±5% carbon):

R1, R11-R14	- 330-ohm
R2	- 220-ohm
R3-R8	- 10-kilo-ohm
R9	- 4.7-kilo-ohm
R10	- 150-ohm
R15	- 1.2-kilo-ohm

Capacitors:

C1	- 1000µF, 25V electrolytic
C2, C3	- 10µF, 16V electrolytic
C4, C5	- 22pF ceramic disk

Miscellaneous:

X1	- 230V AC primary to 7.5V, 300mA secondary transformer
RL1	- 6V, 1C/O relay
S1, S2	- On/off switch
S3	- Push-to-on switch
BATT.	- 6V battery
	- Solenoid (operated with 6V)

Fig. 1: Installation of the automatic flush system

A solenoid is used to actuate the flush from a 6V power supply with battery backup inside the unit that also powers the sensor circuit. This flush system is fully controlled by a microcontroller. It also flushes before the person departs if the person is present for more than the preset time (5 minutes).

Installation of this microcontroller-based automatic flush system is shown in Fig. 1. The IR transmitter LED and the IR receiver modules are mounted side by side such that when the user approaches the mechanism, the IR receiver module receives the IR signal reflected off the person. A solenoid-operated water valve is used in the system.

Circuit description

Fig. 2 shows the circuit of the microcontroller-based flush control system. It is built around Atmel 89C2051 microcontroller that controls the process of automatically flushing the toilet.

The AT89C2051 is an 8-bit microcontroller with 2 kB of flash-based program memory, 128 bytes of RAM, 15 input/output lines, two 16-bit timers/counters, on-chip oscillator and clock circuitry. A 6MHz crystal is used for providing clock. Port pins P1.0 through P1.4 of the microcontroller are connected to buffers N1 through N5 of CD4050 via 10-kilo-ohm pull-up resistors,

respectively.

All the input/output (I/O) pins are reset to '1' as soon as RST (pin 9) goes high on pressing switch S3. Holding the RST pin high for two machine cycles while the oscillator is running resets the device. Power-on-reset is achieved by capacitor C2 and resistor R9.

Pin 12 (P1.0) of microcontroller IC2 provides the 38kHz clock frequency, which is buffered by N1 to drive the two parallel IR-LEDs. These IR-LEDs act as the infrared signal transmitter. Resistor R10 limits the current through the LEDs. Port pins P1.1, P1.2, P1.3 and P1.4 are used for indication of standby, alert, active and flush, respectively. Port pin P1.4 also drives relay RL1 through transistor T1. Diode D5 acts as a free-wheeling diode. The solenoid coil operated off 6V is connected to the contacts of relay RL1.

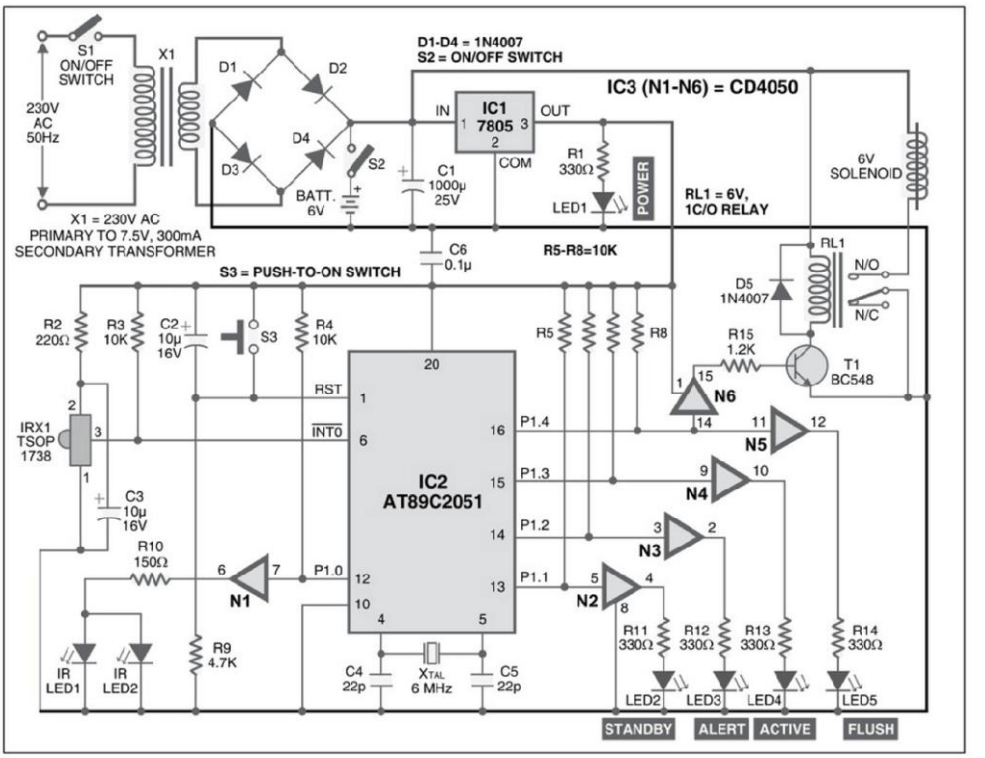


Fig. 2: Circuit of microcontroller-based flush control system

External interrupt 0 (INT0) is used to receive the reflected IR signal. INT0 (pin 6) of the microcontroller is pulled up with resistor R3 and connected to pin 3 of TSOP1738 IR receiver module.

Pin 2 of TSOP1738 is pulled high with resistor R2, while pin 1 is grounded. In the IR receiver module TSOP1738, the PIN diode and the preamplifier are assembled on the lead frame, and the epoxy package is designed as an IR filter. The demodulated output from the receiver module can be directly decoded by the microcontroller.

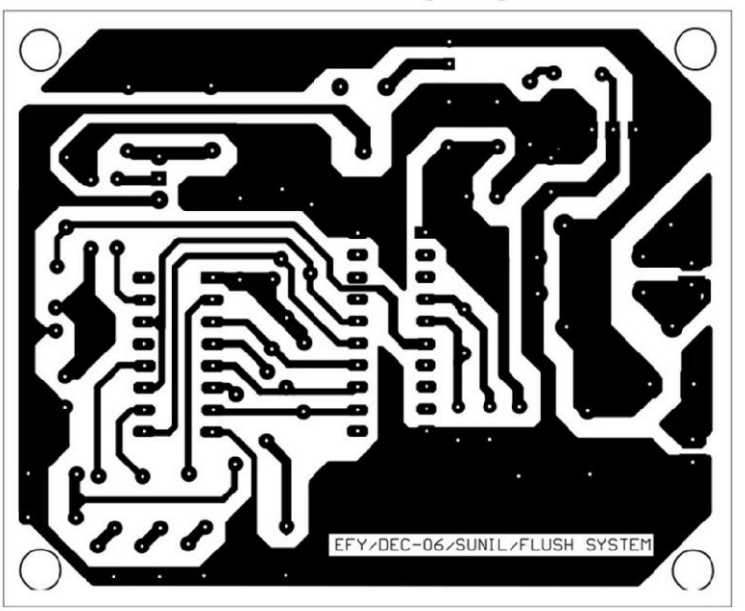


Fig. 3: Actual-size, single-side PCB of microcontroller-based flush control system

The IR-LEDs continuously transmit the IR signal and standby LED2 is always 'on.' When any person comes near the IR-LEDs, the IR receiver module receives the reflected IR signal and alert LED3 lights up. If the alert LED glows for 5 seconds, the active LED (LED4) lights up, indicating that the circuit is now ready to flush. This 5-second time allows for validation of the pot use by the person. When the person goes away, the flush is activated for 10 seconds, which is indicated by LED5. If the person is there for more than 5 minutes, the system flushes once and the software goes back to waiting for the object to move away.

The 5V regulated power supply for the circuit is provided by a conventional circuit. The AC mains is stepped down by transformer X1

to deliver a secondary output of 7.5V, 300mA, which is rectified by a full-wave rectifier comprising diodes D1 through D4, filtered by capacitor C1 to eliminate ripples and regulated by IC 7805 (IC1) to provide regulated 5V power supply for the circuit. LED1 acts as the power indicator. Relay coil and solenoid coil are powered by 6V unregulated power supply. A 6V rechargeable battery is used for power backup.

An actual-size, single-side PCB for the microcontroller-based automatic flush system (Fig. 2) is shown in Fig. 3 and its component layout in Fig. 4.

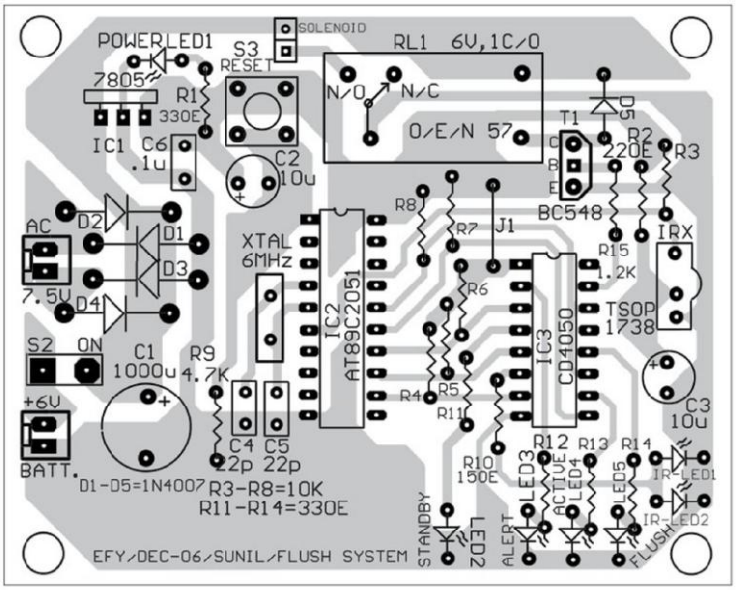


Fig. 4: Component layout for the PCB

The software

The software for flush system is written in 'Basic' language and compiled using Bascom-8051 version. The demo version of Bascom-8051 is available on website 'www.mcselec.com/index.php?option=com_docman&task =doc_download&gid=166&Itemid=54.'

First, instruct the compiler to use 89C2051.dat for microcontroller AT89C2051 by statement '$regfile.' After this, instruct the compiler to override the crystal frequency options setting by statement '$crystal.' Then declare the variables as bits, bytes and words. Initialise port-1 to '0' and port-3 to '1.' (Port-3 acts as the input port.) Enable the interrupt after initialisation. Now write the subroutine 'Fn38K6' to generate 38kHz frequency for transmission of the IR signal.

Standby LED glows when external interrupt 'INT0' is high, i.e., there is no interruption of IR transmission. When 'INT0' goes low, i.e., the transmission is interrupted, alert LED glows. After 5 seconds, active LED lights up. When the person moves away (no interrupt) within 5 minutes, the system flushes for 10 seconds. Otherwise, it flushes every 5 minutes if the person is there. 'Wait' and 'waitms' statements provide the delay in seconds and milliseconds, respectively. Delay time basically depends on the crystal frequency.

Download Source Code: http://www.efymag.com/admin/issuepdf/Flush%20System.zip

FLUSH.BAS

```
$regfile = "89c2051.dat"
' the micro controller's include file
$crystal = 6000000
'6 mhz crystal  used
' define variables below
Dim J As Byte
Dim Irrecd As Bit , I As Byte
Dim K As Byte
Dim L As Bit
Declare Sub Fn38k6(period As Byte)
Dim Period As Word
Irrecd = 0
' another name for port p1.0
Irport Alias P1.0
' make all ports 0
P1 = 0
P3 = 255
' make port-3 high for interrupt to work
' on interrupt - call int0_int fuction
On Int0 Int0_int
Set Tcon.0
' int enabled
Enable Interrupts
Begl:
' show standby mode on port-1 (yellow led on)
P1.1 = 1
' enable the int0 to work now
Enable Int0
Call Fn38k6 100
' call the subroutine to send out a beam
'  of IR at 38khz freq
' int would have taken place if ir recd
Disable Int0
' disable the int now
' check if int occured
If Irrecd = 0 Then
```

```
' no int occured
' so go back to standby mode
P1.1 = 0
' flash standby led
Waitms 100
' wait for ( 1/10th of a second)
Goto Beg1
End If
' here int recd
' wait for about 5 secs to get into alert mode
' ir beam should be reflected for this period of 5
secs
Irrecd = 0
P1.2 = 1
' alert led on now
'below for loop will work for approx 5 seconds
For J = 1 To 30
Irrecd = 0
Enable Int0
Call Fn38k6 100
' call the subroutine to send out a beam of IR at
38khz freq
' int would have taken place if ir recd
Disable Int0
' check if int occured
If Irrecd = 0 Then
' no int occured
' so out of loop - reflective object gone
Goto Nothing
End If
Waitms 100
Next J
' here ir has been recd for 5 secs
' so turn on flush for 10 seconds
'After Object Has Moved Away
' wait for obect to move away
'below for loop will work for approx 5 minutes
P1.3 = 1
For I = 1 To 35
' active led on
For J = 1 To 60
Irrecd = 0
Enable Int0
Call Fn38k6 100
' call the subroutine to send out a beam of IR at
38khz freq
' int would have taken place if ir recd
Disable Int0

' check if int occured
If Irrecd = 0 Then
' no int occured
' so out of loop - reflective object gone
Exit For
' get out of the FOR loop
End If
Waitms 100
Next J
Next I
' time period over so flush
' or object has moved away within 5 minutes
P1.4 = 1
' flush led and buzzer on for 10 seconds
Wait 5
Wait 5
P1 = 0
' all leds off
' get back to start
Waitms 100
Goto Beg1
Nothing:
' no ir recd during the 5 min alert period
' so object has moved away
' go back to start
P1 = 0
Waitms 100
Goto Beg1
' =-=-=-=-subroutines below =-=-=-=-=-=-=
Sub Fn38k6(period As Word)
' parameter 1000 = 1 second approx
' function to oscillate a port pin at 38,000 times
a sec
Dim Ii As Byte , Jj As Byte , Kk As Byte
Dim Periods As Word
Periods = Period / 100
Ii = 0
While Ii < Periods
Incr Ii
Jj = 0
While Jj < 5
Incr Jj
Kk = 0
While Kk < 255
Incr Kk
Irport = 1
NOP
Irport = 0
NOP
NOP
Wend
Wend
Wend
End Sub
Rem The Interrupt Handler For The Int0 Interrupt
Int0_int:
' program comes here if int0 occurs
Irrecd = 1
' just set a flag and get back
' let the main program handle the flag condition
Return
```

MSP430G2231-BASED TEMPERATURE INDICATOR AND CONTROLLER

■ SANI THEO AND LALIT PRAKASH VATSAL

The ambient temperature must be within certain limits for instruments to work properly. Therefore temperature is the most measured process variable in industrial automation. Temperature indicators and controllers are now becoming common even in home appliances. For instance, air-conditioners have built-in temperature indicators.

PARTS LIST

Semiconductors:

IC1	- MSP430G2231 microcontroller
IC2, IC3	- CD4511 BCD-to-7-segment display driver
IC4	- 7806, 6V regulator
T1	- SL100 npn transistor
ZD1	- 3.3V zener diode
LED1, LED2	- 5mm light-emitting diode
D1	- 1N4007 rectifier diode
DIS1	- LT542 common-anode, 7-segment display
DIS2, DIS3	- LT543 common-cathode, 7-segment display
BR1	- 1A bridge rectifier module

Resistors (all ¼-watt, ±5 per cent carbon):

R1	- 680-ohm
R2	- 47-ohm
R3	- 1-kilo-ohm
R4-R18	- 680-ohm
R19, R20	- 470-ohm

Capacitors:

C1	- 1000µF, 35V electrolytic
C2	- 0.1µF ceramic disk
C3	- 10µF, 16V electrolytic

Miscellaneous:

X1	- 230V AC primary to 12V, 250mA secondary transformer
S1	- Push-to-on switch
RL1	- 12V, 1C/O relay

The ambient temperature at any place keeps varying during different times of the day and night. So here we describe a project based on MSP430 microcontroller that indicates the ambient temperature and controls home appliances such as a cooler or a fan at a predetermined temperature. Three 7-segment displays are used to display the temperature.

Main features of this controller are:

1. It also indicates the temperature.
2. It employs MSP430 with internal temperature sensor.
3. The MSP430 is a small but 16-bit device.
4. Low power consumption
5. Development tools are easily available.

This temperature indicator and controller uses minimal components. It is to so simple that even a hobbyist can build it without much effort. No special tools are needed to build this project. All the ICs used here are available in DIP packages. So no special placement tools are needed. The cost of components is minimal and parts are commonly accessible.

Circuit description

Fig. 1 shows the block diagram of the temperature indicator and controller using MSP430G2231 microcontroller, 7-segment displays, CD4511 display drivers, power supply, relay and relay driver. The circuit is shown in Fig. 2.

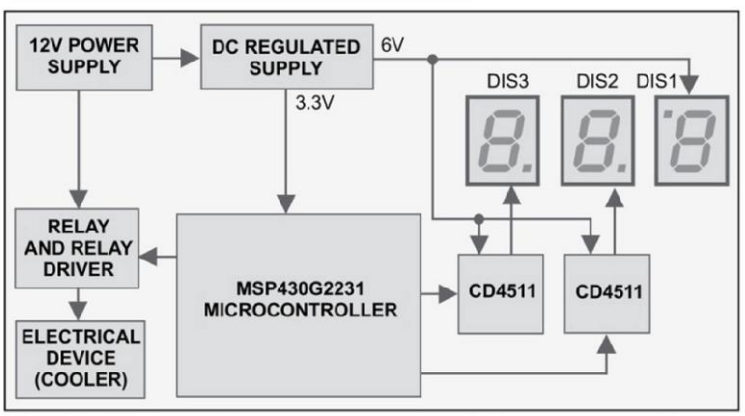


Fig. 1: Block diagram of low-power temperature indicator and controller using MSP430G2231 microcontroller

MSP430G2231 microcontroller. The heart of the circuit is a 14-pin MSP430G2231 microcontroller. Texas Instruments' MSP430 family of ultra-low-power microcontrollers consists of several devices featuring different sets of peripherals targeted for various applications. The architecture, combined with five low-power modes, is optimised to achieve extended battery life in portable measurement applications. The device features a powerful 16-bit RISC CPU, 16-bit registers and constant generators that contribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows wake-up from low-power modes to active mode in less than 1 µs.

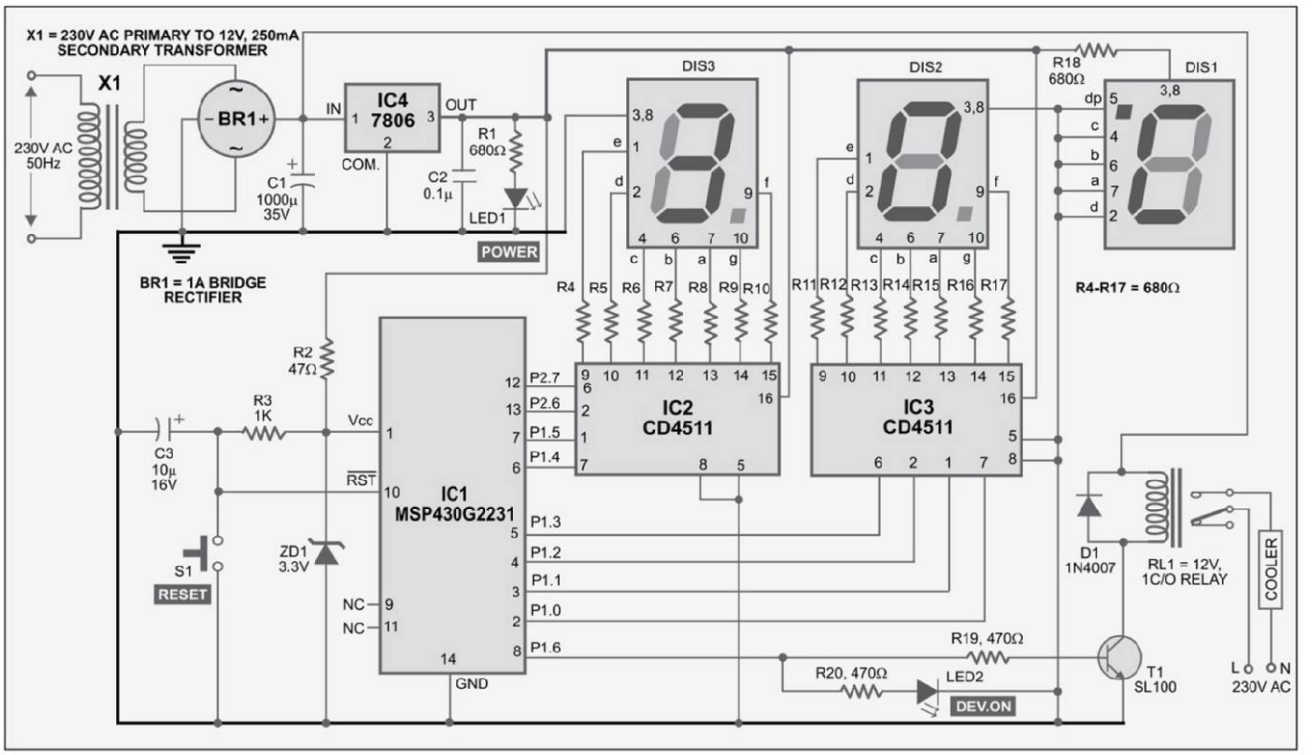


Fig. 2: Circuit diagram of the low-power temperature indicator and controller

The MSP430G2x21/G2x31 series is an ultra-low-power mixed-signal microcontroller with a built-in 16-bit timer and ten input/output (I/O) pins. The MSP430G2x31 family members have a 10-bit analogue-to-digital converter (ADC) and built-in communication capability using synchronous protocols (SPI or I^2C). For configuration details you may refer to the datasheet. Typical applications include low-cost sensor systems that capture analogue signals, convert them into digital values and then process the data for display or transmission to a host system.

Power supply. The circuit requires a 12V DC supply. For this, a 230V AC primary to 12V, 250mA secondary transformer is used. The bridge rectifier rectifies the AC signal and the capacitor further filters it before feeding to 7806 regulator IC. The 12V supply is directly given to the relay for switching on/off the device. The 6V output from 7806 voltage regulator is used to drive the display section. This 6V is also reduced to 3.3V using zener diode ZD1 to drive the MSP430G2231. LED1 indicates the presence of power supply in the circuit.

Relay. As the I/O pin of the MCU cannot drive a relay, a transistor is used for this purpose. LED2 provides a visual indication of the relay status. Electrical appliances such as coolers or fans can be switched on/off through the relay. Glowing of LED2 will indicate the 'on' status of the electrical device connected across relay RL1.

Input switch. Here reset switch S1 is the only input switch. Press it momentarily whenever there is incorrect data display. Note that there is no control input switches for the temperature setting. The program code is fixed at 29°C. That is, if the temperature reaches 30 degrees, your electrical appliance (cooler, fan or AC) connected to this circuit will turn on automatically. If you want to change the temperature setting, you need to change the code as explained below in the software section.

Display. DIS2 and DIS3 display the temperature in digits from 00 to +99. Temperature is commonly measured either in degree centigrade or Fahrenheit. This project displays temperature in degree centigrade. DIS1 is wired for displaying the degree centigrade symbol (°C). CD4511 is a BCD-to-7-segment display driver. IC2 and IC3 drive DIS3 and DIS2, respectively.

Software program

The code for the microcontroller is written in 'C' language in Code Composer Studio version 5 (CCS v5) integrated development environment (IDE). CCS is a complete, Eclipse-based IDE that supports all the MSP430

microcontroller devices.

Code Composer Studio. When using CCS with an MSP430 MCU, a unique and powerful set of plug-ins and tools are made available to fully leverage the MSP430 microcontroller. Free and paid versions are available for download from Texas Instruments' website www.ti.com/tool/ccstudio-msp430. There are two options in the free version, each of which has some limitation:

1. 16kB code-limited version. It has no time limit but supports firmware up to 16kB size only. All the MSP430 devices are supported.

2. 180-day time-limited version. It has no code-size limit but is available only for 180 days (with registration). All the MSP430 devices are supported.

The code for this project is 3kB only, so you can use any of the above free versions.

Operation of CCS. 1. Start Code Composer Studio (CCS) by double-clicking the icon on the desktop or selecting it from the Windows Start menu. When CCS loads, a dialogue box will prompt you for the location of a workspace folder. Browse to "C:\MSP430_LaunchPad\WorkSpace" and do not tick the "Use this as the default ..." checkbox. Click 'OK.' The workspace is saved automatically when CCS is closed.

2. Click 'New Project' option. A project contains all the files you will need to develop an executable output file (.out) which can be run on the MSP430 hardware. To create a new project, click "File→New→CCS Project." In 'Project Name' field, type 'Temperature_Indicator.' Uncheck "Use default location" box.

3. Next, select the appropriate device family, variant and connection type from the pull-down list. This will select the appropriate linker command file and runtime support library. Set the basic build options for the linker and compiler, and set up the target configuration.

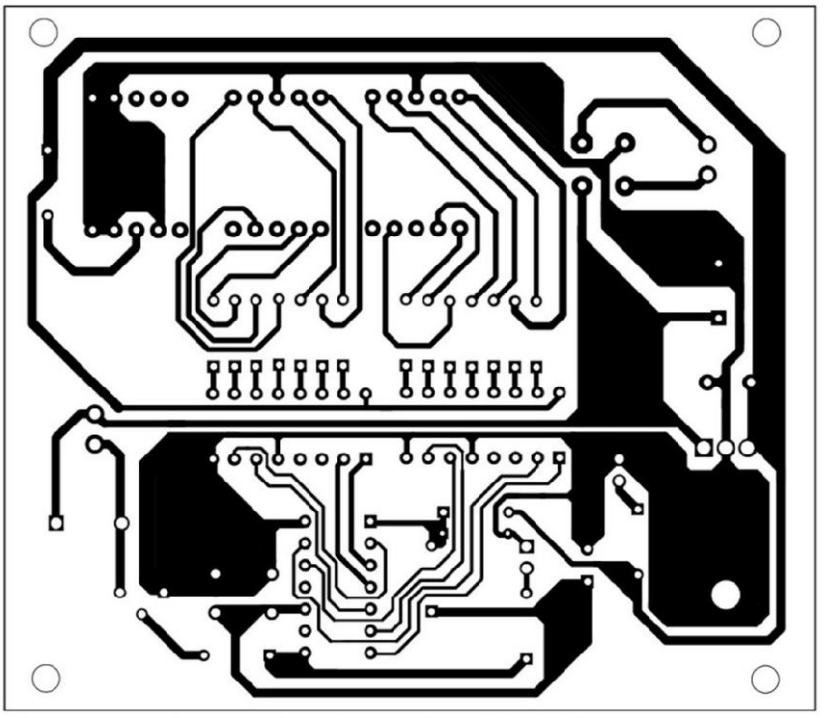


Fig. 3: An actual-size, single-side PCB for the MSP430G2231-based temperature indicator and controller

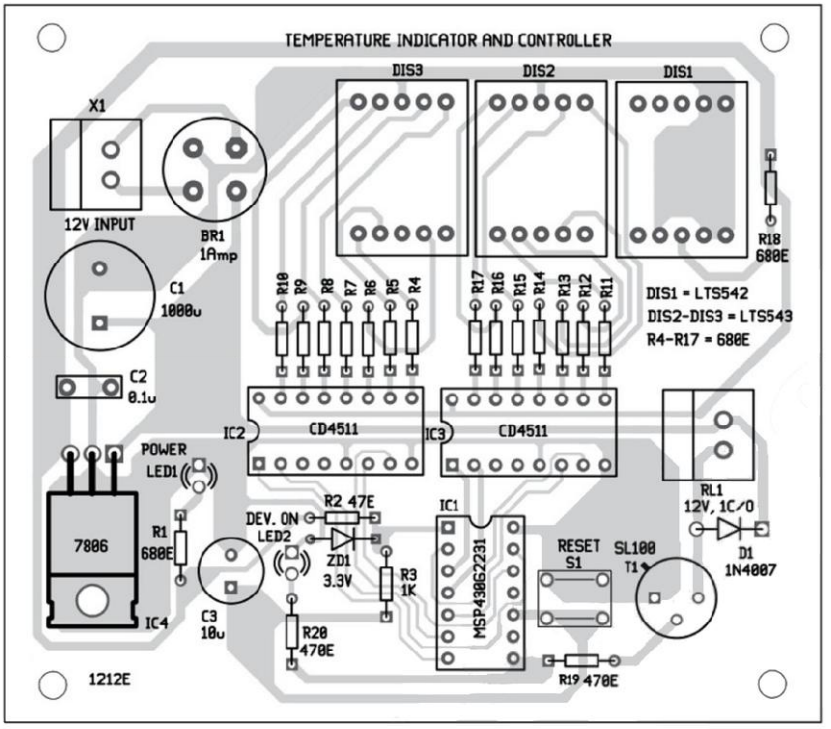


Fig. 4: Component layout for the PCB

4. Click 'Finish.' A new project has now been created. The 'C/C++ Projects' window contains Temperature_Indicator. The project is set as 'Active' and the output files are located in 'Debug' folder. At this point, the project does

not include any source file. The next step is to add the source files to the project.

5. To add a source file to the project, right-click 'Temperature_Indicator' in 'C/C++ Projects' window and select New→Source File. Name the source file as main.c and click 'Finish' button. An empty window will open for the main.c code. You can enter the code here. We have used the original source code that is designed for the MSP430G2231 and included in the link given below.

6. Click 'File→Open File' and navigate to 'C:\MSP430\EFY\Files.' Open the 'Temperature_Indicator.txt' file. Copy and paste its contents into main.c. Save main.c by clicking 'Save' button in the upper left side of the CCS window.

7. Click 'Build' and watch the tools run in 'Console' window. Make sure that no errors are listed in 'Problems' window. The program output with '.out' extension will be generated in 'Debug' folder.

Programming hardware tool setup. In this step, you flash/program the MCU. You need a programmer for that. Either use the TI LaunchPad or the TI EZ430 dongle as a programmer. The Web link to LaunchPad is:

http://processors.wiki.ti.com/index.php/MSP430_LaunchPad_ %28MSP-EXP430G2%29?DCMP=launchpad&HQS=Other+PR+launchpadwiki-pr

The link to EZ430 is:

http://focus.ti.com/docs/toolsw/folders/print/ez430-f2013.html

Burning the code into MSP430 is straightforward with MSP430 LaunchPad and CCSv5 as explained below.

MSP430 LaunchPad setup. The LaunchPad board includes a preprogrammed MSP430G2231 device that is already located in the target socket. When the LaunchPad is connected to your PC via USB, the demo starts with an LED toggle sequence. The on-board emulator generates the supply voltage and all of the signals necessary to start the demo. The driver installation starts automatically. If prompted, allow Windows to install the software automatically.

At this point, the on-board red and green LEDs should be in a toggle sequence. This lets you know that the hardware is working and has been setup correctly. Now, copy and paste the temperature_Indicator.txt file as explained above in Step 6 under operation of CCS section.

Note that CCS can automatically save the modified source files. Once the code is compiled successfully as mentioned in the step above, you can load the code into the MCU using the LaunchPad board.

Open 'Debug' from 'Run' menu, connect the board and download the code to the target (flash device). Click 'Debug' button (green bug). The 'Debug' icon in the upper right-hand corner indicates that you are now in 'Debug Perspective' view. If download is successful, carefully pull up the chip with forceps from the board and place it in your circuit.

Temperature conversion. The conversion and calculation are mainly based on the ADC resolution and reference voltage. The microcontroller used here has a 10-bit ADC, which means there are total 1024 divisions for the entire span.

The MSP430G2231 contains a temperature-sensitive resistor (thermistor) connected to the ADC. The thermistor changes resistance with temperature, changing the voltage input to the ADC. The ADC compares the variable voltage with the reference voltage. The 10-bit ADC on the MSP430G2231 returns a number between 0 and 1023. For example, with an analogue input of 1 volt and the LaunchPad's reference voltage of 1.5 volts, the ADC returns 682 because 1/1.5 = 0.666 = 682/1024. The same method is applied in the project.

Construction and testing

An actual-size, single-side PCB layout of the MSP430G2231-based temperature indicator and controller is shown in Fig. 3 and its component layout in Fig. 4.

The ambient temperature is captured by the inbuilt temperature sensor of MSP430G2231. When the ambient temperature reaches 30 degrees, the cooler connected across relay RL1 is switched on automatically and the displays shows 30°C. If you want to change the temperature setting, change the value '29' in the code 'if(InCelsius>29)' to the value as per your requirement:

```
{
 P2OUT = (BIT7);
 P1OUT |= BIT4;
```

```
}
if(InCelsius>29)
P1OUT |= BIT6;
else
{
P1OUT &= (~BIT6);
}
```

For example, if you want the cooler to turn on at 25 degrees, change the value in the code to '24.' Compile it and burn the code into the MCU using the LaunchPad. Next time, your appliance will turn on when the temperature reaches 25 degrees.

Download source code: http://www.efymag.com/admin/issuepdf/MSP430%20Temperature%20Indicator-Controller.zip

SUN TRACKER WITH POSITION DISPLAY

■ **PRINCE GUPTA AND SANI THEO**

The power generated from solar panels cannot be directly used in many applications. One of the key reasons is that the sun is not stationary as it keeps moving from east to west. The solar panels are able to receive peak sunlight only for a short time period of the day when the sun is directly facing the solar panels. During rest of the day, they get only partial sunlight.

To address this problem, here is a sun tracker device that allows the solar panel to track the sun's position, ensuring maximum power generation.

Circuit description

The block diagram of the sun tracker with position display is shown in Fig. 1. The circuit (shown in Fig. 2) is built around ATmega16 microcontroller, common-cathode 7-segment displays to indicate the position of the panel, ULN2003 high-voltage and high-current Darlington array IC, bipolar stepper motor and CD4511 display driver.

The circuit requires a 12V battery or 12V DC power supply. The working of the circuit is simple. A sensor is used to receive the sun rays and the signal is processed through a microcontroller to rotate the solar panel towards the maximum sunlight using a stepper motor. A light-detecting resistor (LDR) acts as the sensor. The resistance of the LDR varies from zero to a few mega-ohms depending upon the sunlight intensity falling on it. In absolute darkness, it offers the highest resistance, while the resistance dramatically drops when the LDR is exposed to a light source such as sunlight. Using this feature, the microcontroller has been programmed to control the stepper motor such that the solar panel rotates towards the maximum availability of the sun rays.

The main controlling device is the ATmega16 microcontroller. It is used to process the signal received from the sensor. It has an in-built analogue-to-digital-converter (ADC) available through port A. LDR1 is connected to port PA0 as shown in Fig. 2. The in-built ADC converts an analogue input voltage received from LDR1 into a 10-bit digital value through successive approximations. The conversion starts by writing a logical '1' to the ADC start conversion bit. This bit stays high as long as the conversion is in progress and will be cleared by the hardware when the conversion completes.

The digital values can be anywhere between 0 and 1023 depending upon the analogue input received from the sensor. A threshold value is set by the user using hit-and-trial method such that the stepper motor is stationary when the sun is directly facing towards the panel. This is the point where the solar panel gets maximum energy from the sun. The analogue input is detected from time to time. Whenever it reaches below a threshold level, the program instructs the stepper motor to advance by one step so that the panel gets a much broader view

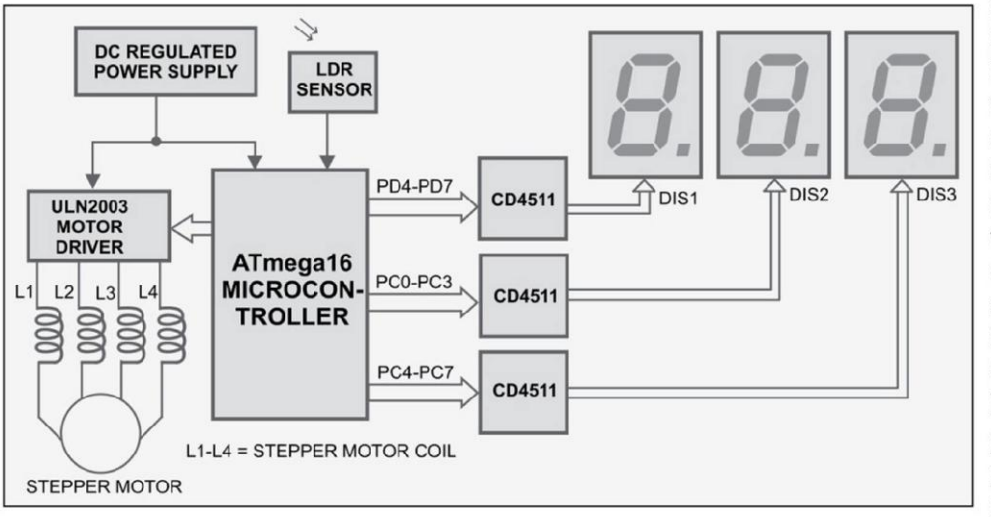


Fig. 1: Block diagram of sun tracker with position display

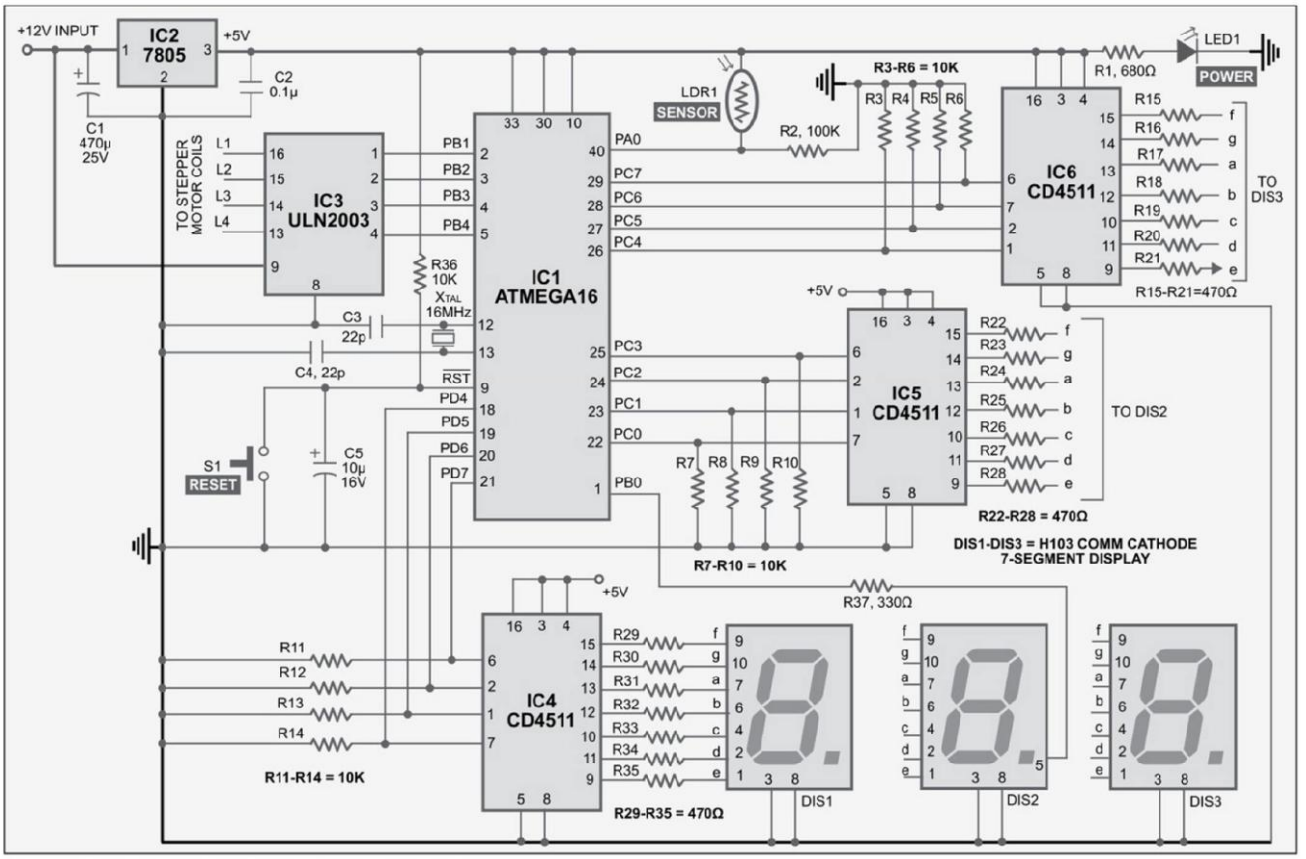


Fig. 2: Circuit of sun tracker with position display

of the sun. As the sun moves from east to west, so does the panel. But after nine hours, on the next day, the panel will return to the same position facing towards east, waiting for the sun.

In this project we have used a stepper motor with step angle of 7.5 degrees per step. The torque of the motor depends on the size of the solar panel you are going to use. High torque is used for heavy load. IC ULN2003 is used to drive the stepper motor. As the motor advances, the position angle of the panel that is being displayed on the 7-segment displays is also incremented. IC CD4511 acts as a BCD (binary-coded-decimal) to 7-segment decoder. To display the digit properly, you need to have at least nine pins. But using a CD4511, we required just four pins of the MCU.

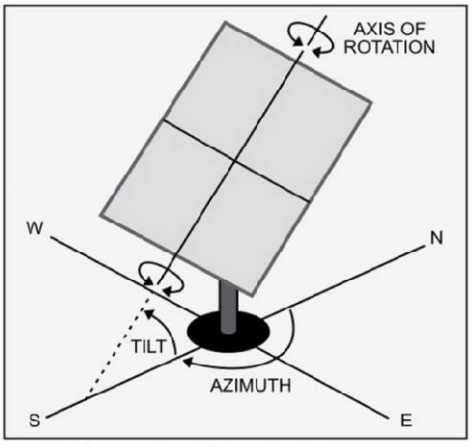


Fig. 3. Polar-type single-axis system

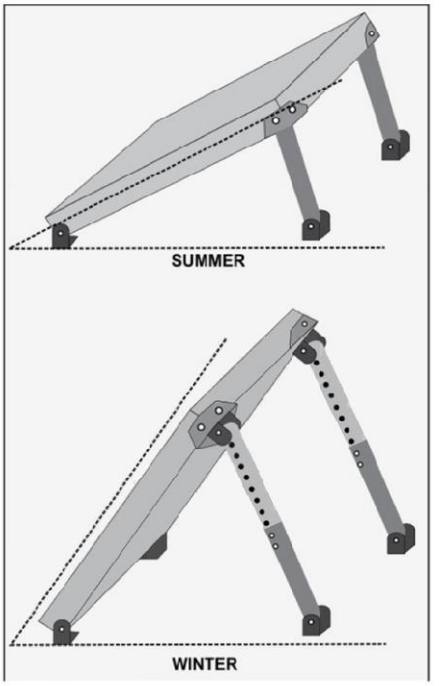


Fig. 4. Panel tilt and orientation during summer and winter

Mounting the solar panel

The panel is mounted on a single-axis system. It is called a single-axis tracker as the mechanism rotates in only one plane around a single axis. The axis can be oriented such that the panels stand up at a tilt (called a polar axis) or lie flat (called a horizontal axis). Horizontal axis is more suitable for small latitudes (locations in the tropics and closer to the equator, i.e., southern India), whilst polar axis is more suitable for larger latitudes (locations far from the equator, i.e., northern India). This system tracks the sun from east to west during the day. The project described is a polar-type single-axis tracking system as shown in Fig. 3.

The solar panel is tilted upward toward the south, at an angle approximately equal to the latitude of the location, to capture more energy from the sun. The correct tilt angle varies with the time of the year when the system is being used, and the latitude of the site (refer Fig. 4).

Latitude can be found in a standard map or from the Internet. But how will you know at which angle you will get the maximum output? There are free software available on the Internet to determine the tilt angle for your site. One such software is PVWatts available on www.nrel.gov/rredc/pvwatts/site_specific.html. It allows installers to easily develop estimates of the performance of panel installations. It allows you to select a location and choose your own system parameters like size, electric cost, array type, tilt angle and azimuth angle.

Fig. 5. Solar module (without LDR) mounted on the support system (courtesy: www.livingonsolar.com)

The panel can be mounted on a diagonal support pipe along with a television antenna rotator using U-bolts (refer Fig. 5). Antenna rotors are usually powered by DC stepper motors. The stepper motor is a special type of motor designed to move slowly and precisely. The motor has a gear-shaped metal disk in the middle surrounding the rotor and several electromagnets surrounding the rotor.

The LDR sensor is mounted on the panel itself. You need to make a contraption to hold the LDR and fix it on the panel (refer Fig. 6). For this, you can use a pen cap or any hollow opaque object of suitable size to hold and cover the LDR. You can also use a marker pen for the same. Remove the felt inside it and insert the LDR. Make sure that both ends of the marker pen are open. Insert the LDR into the pen from one end and cover the other end with a transparent material. Mount the contraption firmly on the solar panel using glue such that the LDR receives the sun rays through the transparent material. Note that proper contraption is required to protect the LDR from rain and also allow the sun rays to fall on the LDR from top only. This ensures proper working of the tracking system.

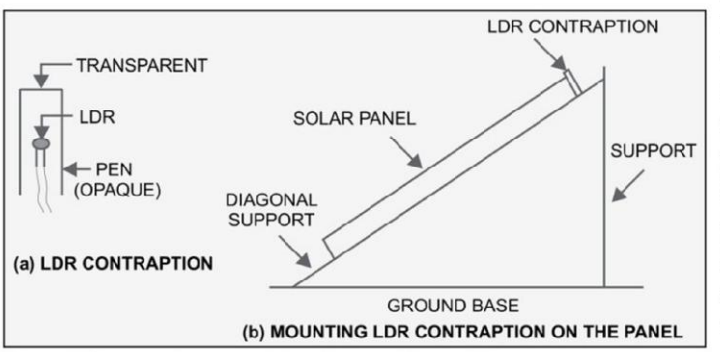


Fig. 6: LDR sensor is mounted on the panel by making a contraption to hold the LDR

An actual-size, single-side PCB layout of the sun tracker with position display is shown

PARTS LIST

Semiconductors:

IC1	- ATmega16 microcontroller
IC2	- 7805, 5V regulator
IC3	- ULN2003 high-current Darlington transistor array
IC4-IC6	- CD4511 BCD-to-7-segment driver
LDR1	- 5mm light-dependent resistor
LED1	- 5mm light-emitting diode
DIS1-DIS3	- H103 common-cathode 7-segment display

Resistors (all ¼-watt, ±5 per cent carbon):

R1	- 680-ohm
R2	- 100-kilo-ohm
R3-R14, R36	- 10-kilo-ohm
R15-R35	- 470-ohm
R37	- 330-ohm

Capacitors:

C1	- 470µF, 25V electrolytic
C2	- 0.1µF ceramic
C3-C4	- 22pF ceramic
C5	- 10µF, 16V electrolytic

Miscellaneous:

S1	- Tactile switch
X_{TAL}	- 16MHz crystal oscillator
	- 7.5-degree bipolar stepper motor
	- Contraption for LDR1
	- Solar module
	- Solar module mounting system
	- 12V battery

in Fig. 7 and its component layout in Fig. 8. Note that the LDR should not be mounted on the PCB. Extend two shielded wires from the PCB to the contraption for the LDR.

Software program

The 'C' code is written using AVR Studio and the hex code is burnt into the MCU using PonyProg2000.

The 'C' code is designed with the reference angle set to 30 degrees with respect to the diagonal support axis. That means the panel starts to rotate from 30 degrees in the morning till 150 degrees in the evening. At 30 degrees, which is displayed on the 7-segment display, the panel should face towards the east when the sun is just above the horizon. As the sun moves from east to west, the panel rotates from east to west in sync with sun movement.

The panel stops at 150 degrees when the sun is at dusk. Thereafter, the panel remains at this position for nine hours. After nine hours, the

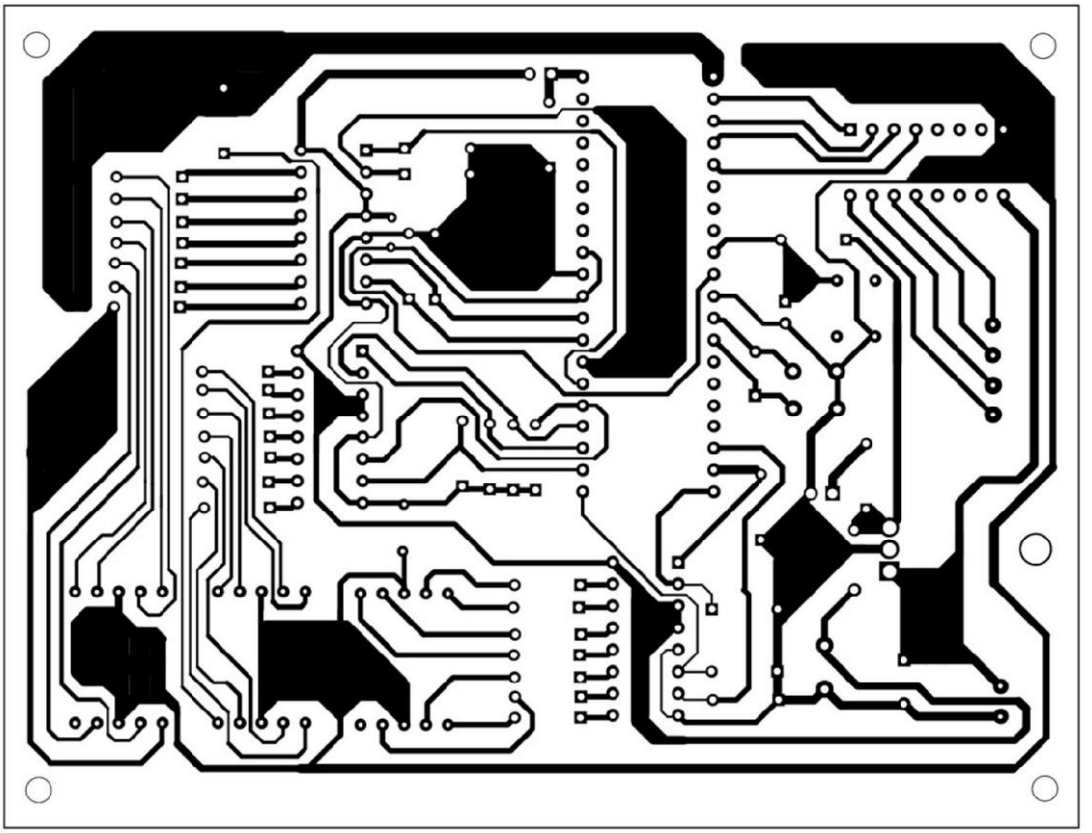


Fig. 7: An actual-size, single-side PCB for the sun tracker with position display

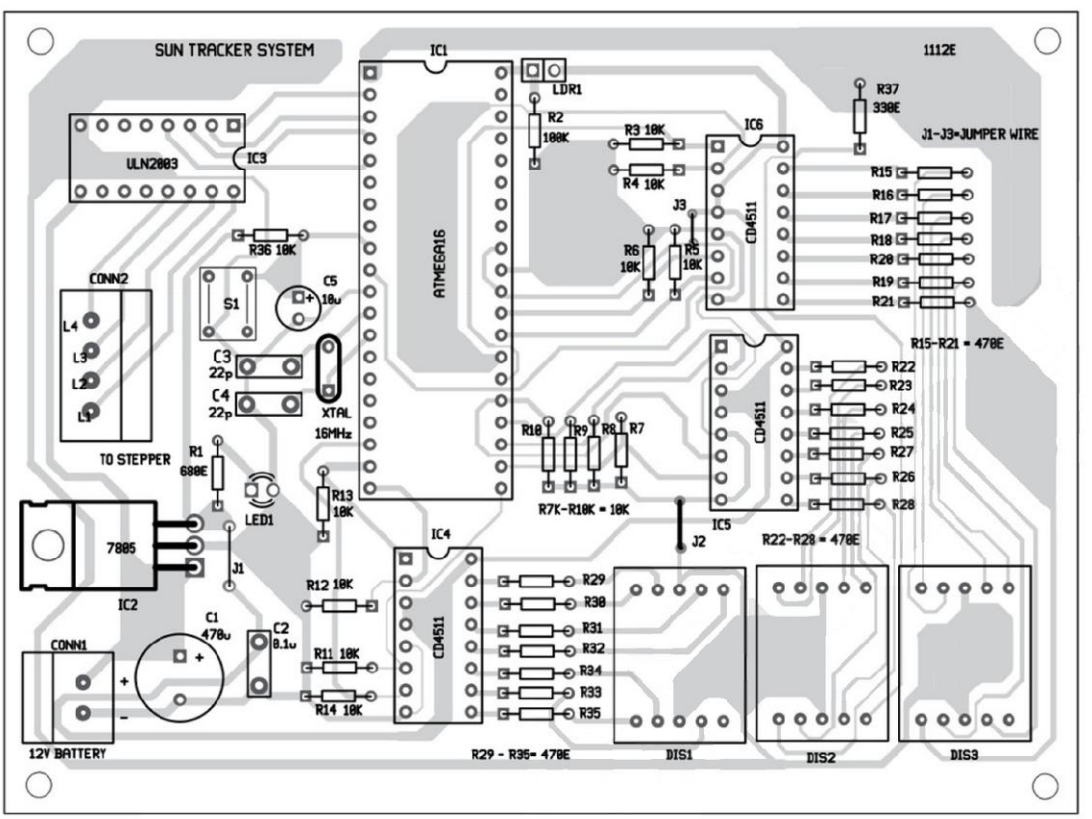


Fig. 8: Component layout for the PCB

program instructs the motor to return to 30 degrees position facing east. When sun rises in the morning, the system begins to receive signal through the sensor again, the motor rotates and the same process repeats.

Note that the angle displayed is just to let you know the position of the panel. For example, at 5 pm, the panel should be at an angle of about 140 degrees, but if the display shows 60 degrees, there is something wrong in the tracking mechanism and the panel is not facing the sun. This will alert you to check the problem and correct it.

On cloudy days, the solar panel still produces energy as there is still light, although its performance is obviously reduced. In such situations, the solar tracker will orient itself optimally to pick up the most reflected ambient light. Once the sun starts shining again, it will re-orient itself to face it.

The project can be further modified to use two stepper motors. The first motor will drive the panel. The second motor should be mounted with the sensor on the shaft rather than on the panel as in this case. The motor with the sensor can be made to rotate continuously and track the sun. The advantage is that there will be automatic tracking of the sun in a more precise way within few seconds on a cloudy day.

Download source code: http://www.efymag.com/admin/issuepdf/Sun%20Tracker%20With%20Position%20Display.rar

PRESENCE SENSING LIGHTS CONTROLLER

■ SANJIB BHUIYA

Many a times we forget to switch off appliances like lights, fans and air-conditioner before leaving home. This leads to a considerable wastage of electricity apart from reducing the life of the appliances.

Here is a circuit that solves this problem by sensing the absence of occupants in a room and automatically shutting the power 'off.' It turns on the power again when someone enters the room. The same circuit can be used for a particular appliance also, say, air-conditioner.

The circuit presented here is a microcontroller-based automatic room light controller. The microcontroller along with op-amp LM324 is wired with two IR sensor units to count the number of persons going inside the room and the number of persons coming out. When the number of persons inside the room is zero, it automatically disconnects the power. When someone enters the room, the counter increments and the power to the room is restored. The number of persons inside the room is displayed on a seven-segment display.

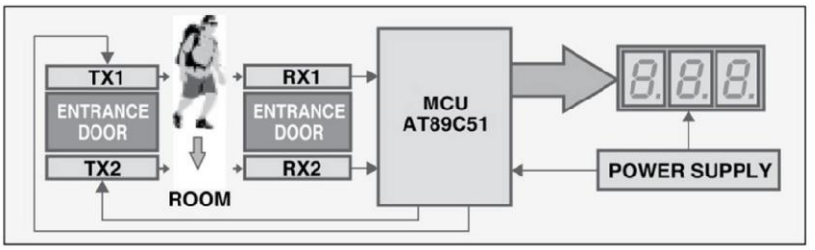


Fig. 1: Block diagram of presence sensing lights controller

Circuit description

The block diagram of the presence sensing lights controller is shown in Fig. 1 and the circuit in Fig. 2. To derive the power supply for the circuit, the 230V, 50Hz AC mains is stepped down by transformer X1 to deliver a secondary output of 12V, 500 mA. The transformer output is rectified by a full-wave rectifier comprising diodes D1 through D4, filtered by capacitor C1 and regulated by IC 7805 (IC1). Capacitor C2 bypasses the ripples present in the regulated supply. Regulated 5V is used to power the circuit, except relay RL1.

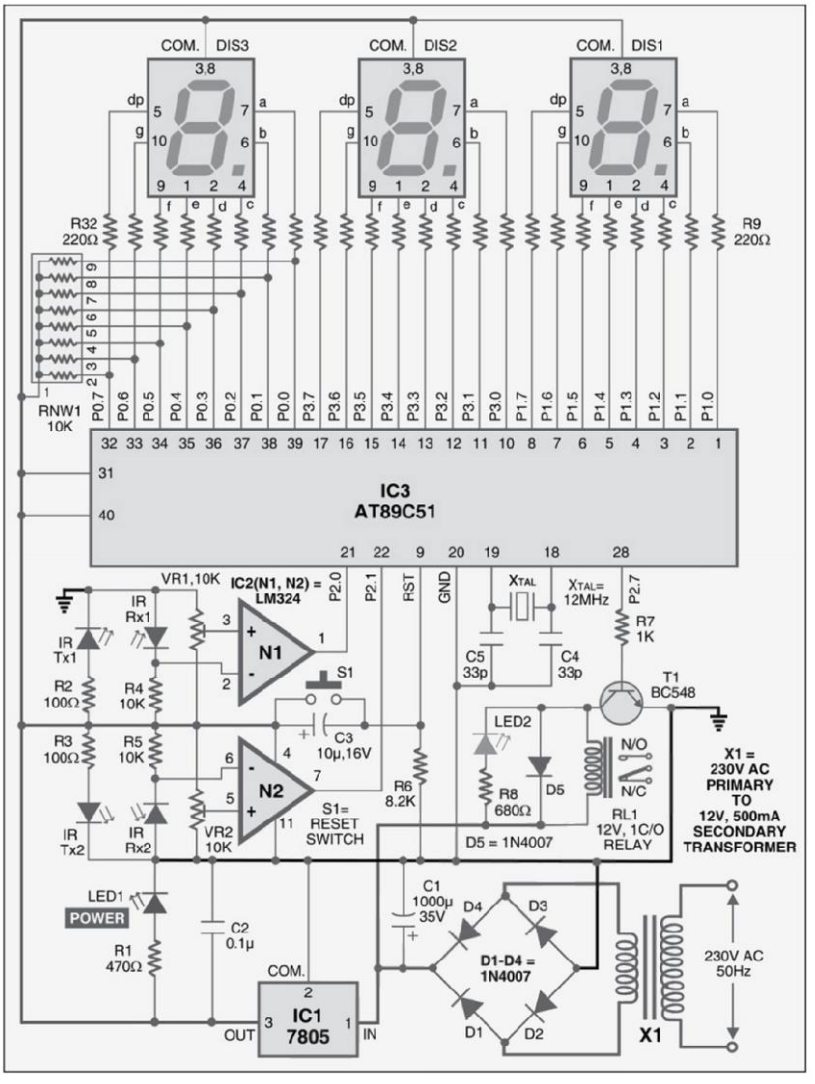


Fig. 2: Circuit of presence sensing lights controller

Infrared transmitter-receiver pairs

TX1-RX1 and TX2-RX2 are mounted on the sides of the entrance door—one pair inside and the other pair outside the room. The transmitters and the receivers are to be mounted on the opposite frames of the door such that light from the infrared transmitter falls directly on the infrared detector on the other side (refer Fig. 1). Cover the infrared receivers with a mask to protect these from ambient light.

Infrared signals from IR TX1 and IR TX2 continuously fall on IR RX1 and IR RX2, respectively. The signals detected by RX1 and RX2 are fed to inverting input pins 2 and 6 of comparators N1 and N2, respectively. Normally, the comparator outputs at pins 1 and 7 are high, which are given to microcontroller port pins P2.0 and P2.1, respectively. When someone passes through the door, the infrared beams are interrupted and the comparator outputs go low. Microcontroller AT89C51 increments/decrements the count depending on the direction of movement.

Microcontroller AT89C51 is the heart of the automatic room light controller. It is an 8-bit microcontroller with 4 kB of flash programmable and erasable read-only memory (PEROM), 128 bytes of RAM, 32 input/output (I/O) lines, two 16-bit timers/counters, a five-vector two-level interrupt architecture, a full-duplex serial port, on-chip oscillator and clock circuitry. Power-on-reset is provided by the combination of resistor R6 and capacitor C3. Switch S1 is used for manual reset.

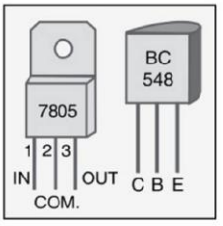


Fig. 3: Pin details of 7805 and BC548

A 12MHz crystal along with two 33pF capacitors provides the basic clock frequency to microcontroller AT89C51. Three seven-segment displays (DIS1 through DIS3) are interfaced with the microcontroller through Port 0, Port 1 and Port 3, which are used to display the number of persons inside the room. Port 0 is pulled high with resistor network RNW1. Port pin P2.7 drives relay RL1 to control the power.

When somebody enters the room, the comparator outputs go low and port pin P2.7 goes high. Transistor T1 drives into saturation to energise relay RL1. Diode D5 acts as a

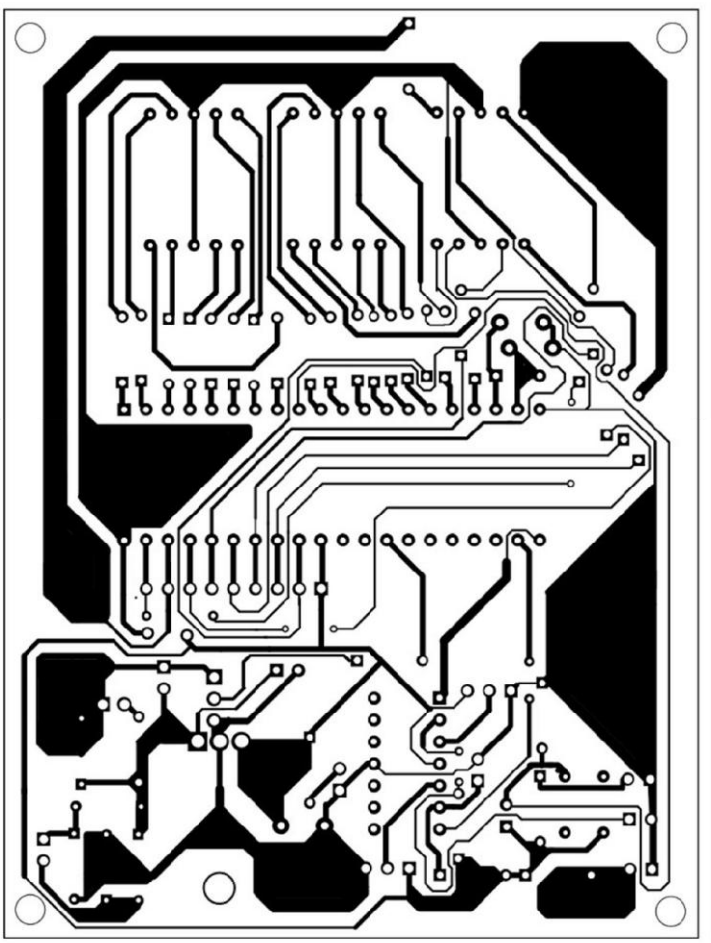


Fig. 4: An actual-size, single-side PCB for the presence sensing lights controller

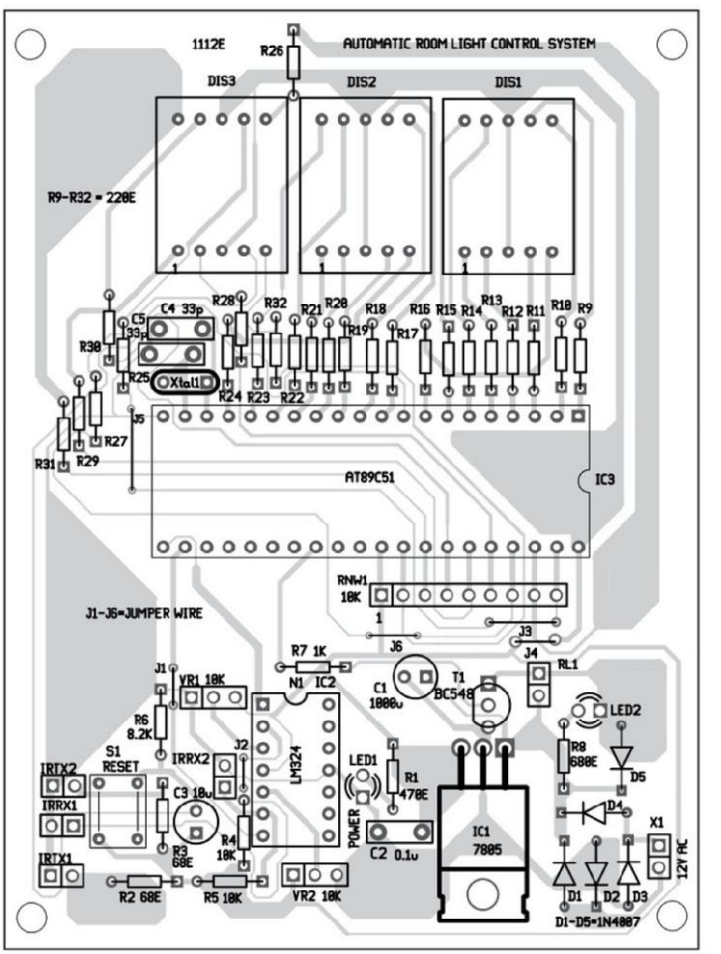


Fig. 5: Component layout for the PCB

free-wheeling diode. Resistors R9 through R32 are used to limit the current through segments of the 7-segment display. Presets VR1 and VR2 are used to set the threshold voltage of comparators N1 and N2, respectively.

Construction

An actual-size, single-side PCB for the automatic room light controller is shown in Fig. 4 and its component layout in Fig. 5. Assemble the circuit on the PCB to save time and minimise assembly errors. Carefully assemble the components and double-check for any overlooked error. Use IC bases for microcontroller AT89C51 and op-amp LM324. Before inserting the ICs, check the supply voltage.

Both the IR transmitter-receiver pairs should be placed approximately 50 cm apart. Align the IR transmitter and IR receiver such that these directly face each other.

Software

The software for the automatic room light controller is written in 'C' language and compiled using the Keil µVision4 compiler. Burn the generated hex code into the microcontroller by using a suitable programmer. It is well commented and easy to understand.

Download source code: http://www.efymag.com/admin/issuepdf/Auto%20Room%20Light%20Controller.rar

PARTS LIST

Semiconductors:

IC1	- 7805, 5V regulator
IC2	- LM324 quad operational amplifier
IC3	- AT89C51 microcontroller
T1	- BC548 npn transistor
D1-D5	- 1N4007 rectifier diode
DIS1-DIS3	- LTS 542 common-anode 7-segment display
IR TX1, IR TX2	- Infrared transmitter
IR RX1, IR RX2	- Infrared detector
LED1, LED2	- 5mm LED

Resistors (all ¼-watt, ±5% carbon):

R1	- 470-ohm
R2, R3	- 100-ohm
R4, R5	- 10-kilo-ohm
R6	- 8.2-kilo-ohm
R7	- 1-kilo-ohm
R8	- 680-ohm
R9-R32	- 220-ohm
VR1, VR2	- 10-kilo-ohm preset
RNW1	- 10-kilo-ohm resistor network

Capacitors:

C1	- 1000µF, 35V electrolytic
C2	- 0.1µF ceramic disk
C3	- 10µF, 16V electrolytic
C4, C5	- 33pF ceramic disk

Miscellaneous:

X1	- 230V AC primary to 12V, 500mA secondary transformer
S1	- Push-to-on tactile switch
X_{TAL}	- 12MHz crystal
RL1	- 12V, 1C/O relay

LIGHT.C

```
#include <REGX51.H>
void display(unsigned int);
sbit control=P2^7;
sbit in=P2^0;
sbit out=P2^1;
void main(void)
 {
  unsigned int z=0;  //set counter at zero
  P1=0XC0;           //display zero in
                  all 7segment display
  P3=0XC0;
  P0=0XC0;
  P2=0X03;
  control=0;
    while(1)         //repeat forever
     {
      if(in==0) // is anybody going
                        inside
       {
         while(out==1);
                while(out==0);
                z++;  //increase the
                          counter
                display(z);
                //display no of person
                control=1;     //turn
                               on light
       }
       else if(out==0) //is anybody
                         coming outside
        {
                while(in==1);
                while(in==0);
                z--;  //decrease the
                             counter
                display(z);
                //display no of person
                if(z==0)
                      // counter=0?
                 {
                  control=0;
                   //turn off light
                 }
                }
     }
 }
void display(unsigned int m)
 {
   unsigned int digit[10]={0XC0,0XF9,0XA4,0XB0,0X99
,0x92,0x82,0XF8,0X80,0X90};
   unsigned int i,j,k;
   i=m%10;
   j=m/10;
   k=m/100;
   j=j-k*10;
   P1=digit[i];  //display lsb
   P3=digit[j];  //display middle digit
   P0=digit[k];   //display msb
  }
```

TOUCHSCREEN CONTROL FOR WHEELCHAIR

■ SACHIDANANDA SAHU

A wheelchair is a chair with wheels, designed to be a replacement for walking. The device comes in variations where it is propelled by motors or by the seated occupant turning the rear wheels by hand. Wheelchairs are used by people for whom walking is difficult or impossible due to illness, injury or disability.

Here we describe a microcontroller-based wheelchair the speed and direction of which can be controlled from a touchscreen. The wheelchair moves by means of a geared motor. Fig. 1 shows the block diagram of the touchscreen-controlled wheelchair. The system includes a resistive touchscreen, microcontroller and motor driver circuit.

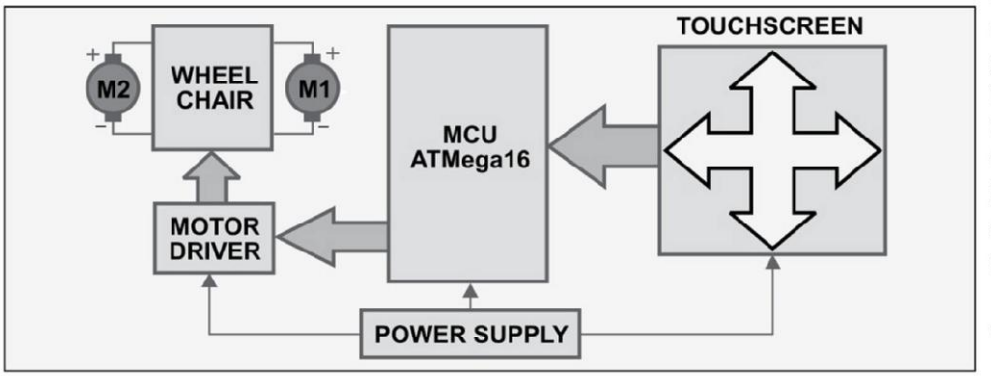


Fig. 1: Block diagram for the touchscreen control for wheelchair

The touchscreen

The touchscreen is an electronic visual display that can detect the presence and location of a touch within the display area. It is sensitive to the touch of a human finger, hand and passive objects like stylus (refer Fig. 2). It is a two-dimensional sensing device which is made of two sheets of material separated by small spacers.

There are three main touchscreen technologies: resistive, capacitive and surface acoustic wave.

Resistive touchscreen. The resistive touchscreen consists of a flexible top layer made of polyethylene and a rigid bottom layer made of glass. Both the layers are coated with a conducting compound of indium-tin oxide (ITO) and then spaced with spacers. When the monitor is operational, an electric current flows between the two layers.

When a touch is made, the flexible screen presses down to touch the bottom layer. A change in electrical current is hence detected and the coordinates of the touch point are calculated by the controller and parsed into readable signals for the operating system to react accordingly.

The 4-wire resistive touchscreen uses

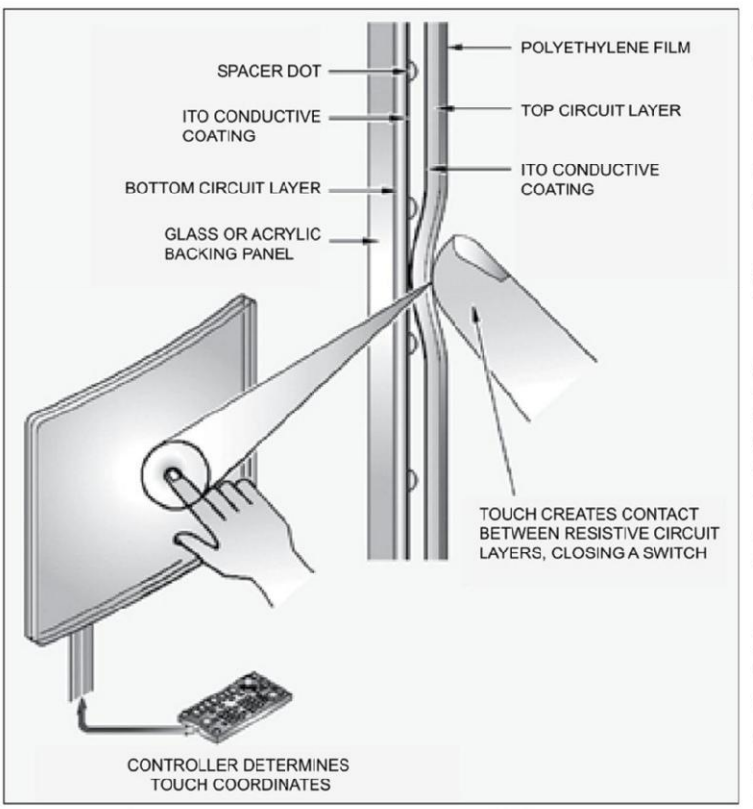


Fig. 2: Typical touchscreen layers (Courtesy: www.tci.de)

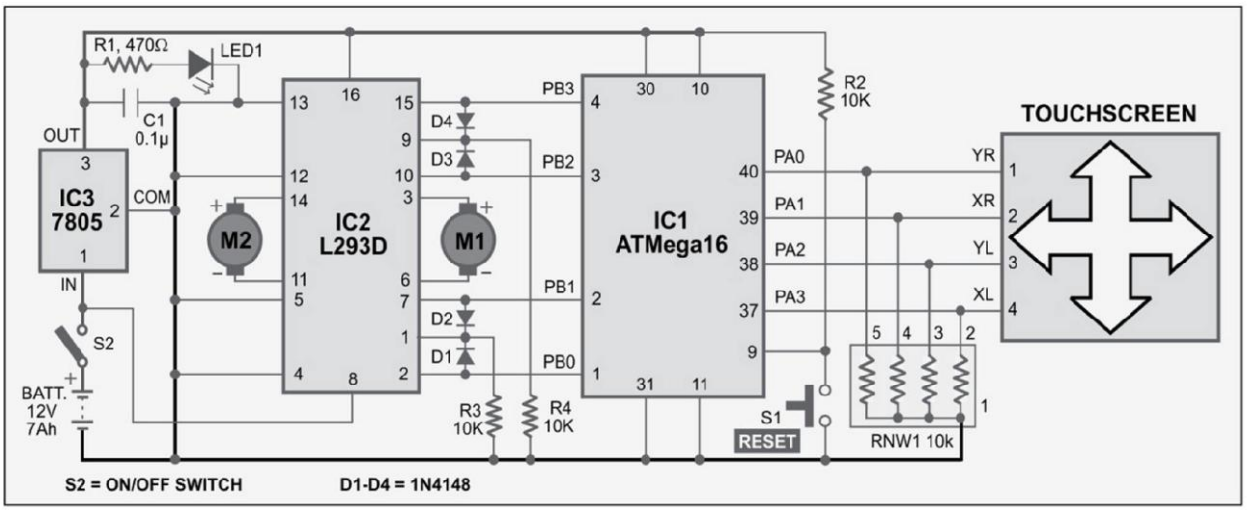


Fig. 3: Circuit for touchscreen control for wheelchair

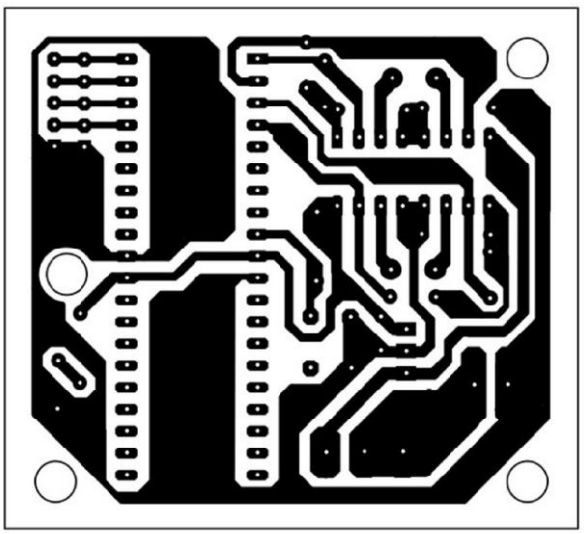


Fig. 4: An actual-size, single-side PCB for the touchscreen control for wheelchair

both the layers to calculate the axes information of the touch. Touch measurement is a two-step process. The x-coordinate of the touch point is calculated by creating a voltage gradient on the flexible layer. The y-coordinate is determined by creating a voltage gradient along the bottom layer.

Capacitive touchscreen. Capacitive touchscreen is the most popular and durable touchscreen technology used all over the world. It consists of a glass panel coated with indium-tin oxide—a capacitive (conductive) material. Capacitive systems transmit almost 90 per cent of light from the monitor. In surface-capacitive screens, only one side of the insulator is coated with a conducting layer.

When the monitor is operational, a uniform electrostatic field is formed over the conductive layer. Whenever the user touches the screen with a finger, conduction of electric charges over the uncoated layer results in the formation of a dynamic capacitor. The computer or the controller then detects the position of touch by measuring the change in capacitance at the four corners of the screen.

Surface acoustic wave touchscreen. Surface acoustic wave touchscreen contains two transducers (transmitting and receiving) placed along the X-axis and Y-axis of the monitor's glass plate along with some reflectors. The waves propagate across the glass and reflect back to the sensors.

When the screen is touched, the waves are absorbed and a touch is detected at that point. These reflectors reflect all the electrical signals sent from one transducer to another. This technology provides excellent throughput and image clarity.

Circuit description

Fig. 3 shows the circuit for the touchscreen control for wheelchair. It comprises microcontroller ATMega16 (IC1), motor driver L293D (IC2), regulator 7805 (IC3), resistive touchscreen and a few discrete components.

The ATmega16 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. The AVR core combines a rich instruction set with 32 general-purpose working registers. All the 32 registers are directly connected to the arithmetic logic unit, allowing two independent registers to be accessed in one single instruction executed in one clock cycle. This architecture is more code-efficient.

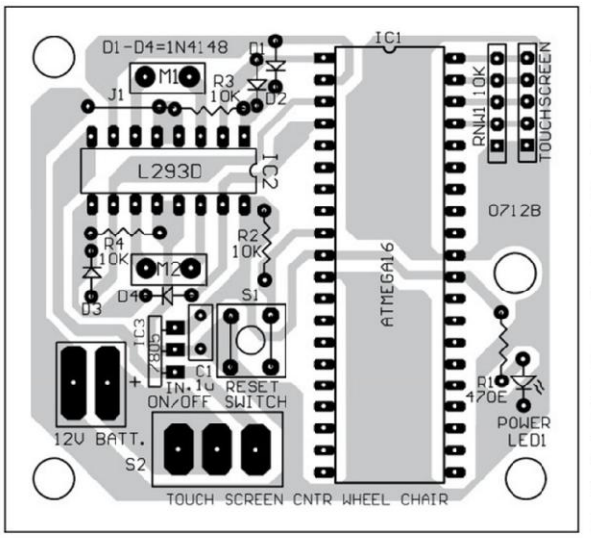


Fig. 5: Component layout for the PCB

Fig. 6: Author's prototype

PARTS LIST

Semiconductors:

IC1	- ATMega16 microcontroller
IC2	- L293D motor driver
IC3	- 7805, 5V regulator
D1-D4	- 1N4148 switching diode
LED1	- 5mm LED
	- Touchscreen module

Resistors (all ¼-watt, ±5 per cent carbon):

R1	- 470-ohm
R2-R4	- 10-kilo-ohm
RNW1	- 10-kilo-ohm

Capacitors:

C1	- 0.1µF ceramic disk

Miscellaneous:

S1	- Push-to-on tactile switch
S2	- On/off switch
M1, M2	- 50-rpm geared DC motor
BATT.	- 12V, 7Ah battery

The ATmega16 has the following features: 16 kB of in-system programmable Flash program memory, 512 bytes of EEPROM, 1kB SRAM, 32 general-purpose input/output (I/O) lines, 32 general-purpose working registers, three flexible timers/counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented two-wire serial interface, an 8-channel 10-bit analogue-to-digital converter (ADC), an SPI serial port, and six software-selectable power-saving modes.

The microcontroller has an 8-channel ADC. Port A of the microcontroller is multiplexed with the 8-channel ADC. Port pins PA0 through PA3 of microcontroller are interfaced with pins 1 through 4 of touchscreen. The touchscreen uses four channels (ADC0 through ADC3). The resistive-type touchscreen used here is quite cheaper than capacitive touchscreens and also easy to use.

When the touchscreen is touched, its resistance changes depending on the position of the touch. The touch input is in the form of analogue values of the voltage relative to the point of touch. This is processed in microcontroller ATMega16 using the ADC.

The ADC converts the analogue values of the touchscreen into 10-bit digital equivalent form. This digital data is then processed to find the exact position of the touch and perform the corresponding action like left turn, right turn, forward movement and reverse movement with the help of motor driver L293D. Switch S1 is used for manual reset.

Motor driver L293D is interfaced with the microcontroller's port B. Port pins PB0 and PB1, and PB2 and PB3 of the microcontroller control motors M1 and M2, respectively. Motor drivers are enabled in pairs. When enable input pins 1 and 9 of IC2 are high, the associated drivers are enabled and their output pins 3 and 6 and pins 11 and 14, respectively, are active and in phase with the inputs. With the proper data inputs, each pair of drivers forms an H-bridge reversible drive, suitable for motor applications.

Motors M1 and M2 rotate in forward direction (clockwise) when port pins PB1 and PB3 are high. Motors M1 and M2 rotate in reverse direction (anti-clockwise) when port pins PB0 and PB2 are high.

A 12V DC battery is used to power the circuit. IC 7805 provides regulated 5V supply to the circuit. Capacitor C1 bypasses ripple from the regulated supply. LED1 acts as the power-'on' indicator and resistor R1 limits the current through LED1.

Construction and testing

An actual-size, single-side PCB for the touchscreen control for wheelchair is shown in Fig. 4 and its component layout in Fig. 5. Assemble the circuit on a PCB as it minimises time and assembly errors. Carefully assemble the components and double-check for any overlooked error. Use IC bases for IC1 and IC2. Before putting the microcontroller and other ICs on the PCB, check the correct supply voltage. Suitable connectors are provided on the PCB to connect geared motors M1 and M2.

Connect the motors and power the circuit with 12V battery. Now the circuit is ready for use. For instance,

when you touch the forward arrow on the touchscreen, both the motors rotate in forward (clock-wise) direction simultaneously. Fit the motors properly with the wheels and the touchscreen near the hand to control the movement of the wheelchair.

Software

The source program is written in 'C' language and compiled using AVR Studio to generate the Intel hex code. The generated hex code is programmed into the chip using a suitable programmer. The code is well commented and easy to understand.

The device is shipped with CKSEL="0001" and SUT="10". The default clock source setting is therefore 1MHz internal RC oscillator with longest startup time. To activate the internal oscillator, program the fuse bytes of the microcontroller as follows:

Fuse low byte = D4

Fuse high byte = 99

Download source code: http://www.efymag.com/admin/issuepdf/Touchscreen%20Controlled%20Wheelchair.zip

TOUCH.C

```
#include<avr/io.h> //header file
#include<util/delay.h> //header file
void Drive_Motor(unsigned char LEFT,unsigned char
RGHT)//For MOTOR
{
          if(RGHT==0)//if right == 0 then right motor
will stop
          {
          PORTB&=~_BV(1);
          PORTB&=~_BV(0);
          }
          if(RGHT==1)//if right == 1 then right motor
will go forward
          {
          PORTB&=~_BV(1);
          PORTB|=_BV(0);
          }
          if(RGHT==2)//if right == 2 then right motor
will go backward
          {
 PORTB|=_BV(1);
          PORTB&=~_BV(0);
          }
          if(LEFT==0)//if left == 0 then left motor
will stop
          {
          PORTB&=~_BV(2);
          PORTB&=~_BV(3);
          }
          if(LEFT==1)//if left == 1 then left motor
will go forward
          {
          PORTB&=~_BV(2);
          PORTB|=_BV(3);
          }
          if(LEFT==2)//if left == 2 then right motor
will go backward
          {
 PORTB|=_BV(2);
          PORTB&=~_BV(3);
          }
}
void adc_init(void) //for adc initilization
{
ADCSRA=(1<<ADEN)|(1<<ADSC)|(1<<ADATE)|(1<<ADPS2);
SFIOR=0x00;
}
unsigned char read_adc_channel(unsigned char channel)//
for channel selection
{
 ADMUX=(1<<REFS0)|(1<<ADLAR)|channel;
 _delay_ms(10);
 return(ADCH);
}
int main()// Main function
{
DDRB=0x0f;//for motor PB -(0,1,2,3) connected to motor
driver
adc_init();//adc initilized
unsigned char x,y;//for storing value of x & y co-
ordinate
while(1)
{
 _delay_ms(20);
 DDRA=0x05;//00000101
 PORTA=0x01;//00000001for x init
 x=read_adc_channel(3);
 _delay_ms(20);
 _delay_ms(20);
 DDRA=0x0a;//00001010
 PORTA=0x08;//00001000//for y init
 y=read_adc_channel(2);
 _delay_ms(20);
if(x>57 && x<140 && y>150)//forward
{
 Drive_Motor(1,1); //PORTB=0b00001001;
}
else if(x>57 && x<140 && y<60)//backward
{
 Drive_Motor(2,2); //PORTB=0b00000110;
}
else if(x<70 && y>80 && y<130)//left
{
 Drive_Motor(2,1); //PORTB=0b00000101;
}
else if(x>150 && y>80 && y<130)//right
{
 Drive_Motor(1,2); //PORTB=0b00001010;
}
else//stop
{
 Drive_Motor(0,0); //PORTB=0b00000000;
}
}
}
```

RF-BASED MULTIPLE DEVICE CONTROL USING MICROCONTROLLER

■ AZARUDEEN ANIFA

Here we describe how to control electrical and electronic gadgets from a remote location using radio frequency (RF) transmission. An RF interface is used instead of infrared (IR) to avoid the drawbacks of an IR interface. Besides, RF has a longer range. The signal is transmitted by an RF transmitter and received by an RF receiver to switch on or switch off the desired device. This system can be used to control up to fifteen devices.

Fig. 1 shows the block diagram for RF-based multiple device control using microcontroller. Signals from the keypad are fed to microcontroller AT89C2051, which, in turn, is interfaced to the RF transmitter through encoder HT12E. The microcontroller continuously reads the status of the keys on the keypad.

When any key is pressed, data is passed to the encoder and then to the RF transmitter from where it is transmitted. The RF receiver receives this data and gives it to the RF decoder. The

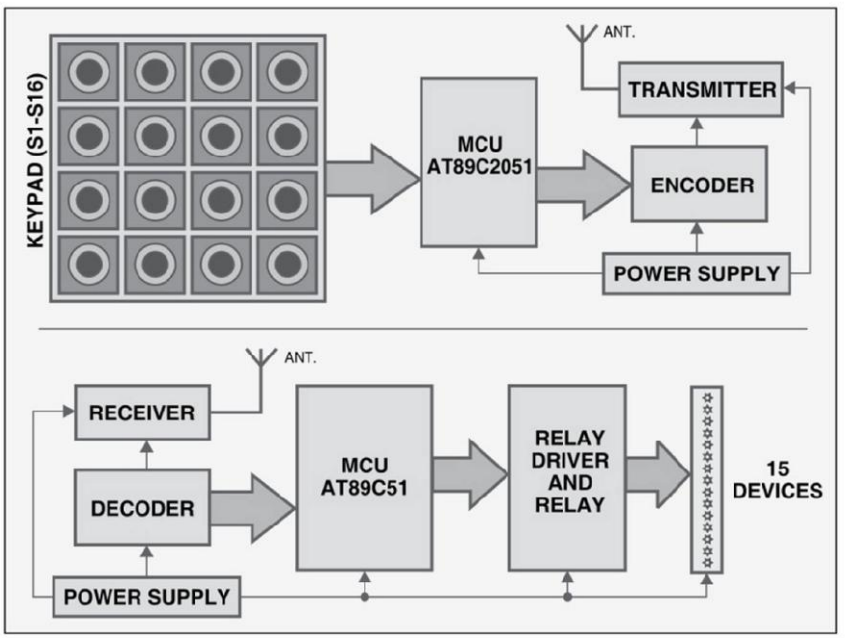


Fig. 1: Block diagram for RF-based multiple device control using microcontroller

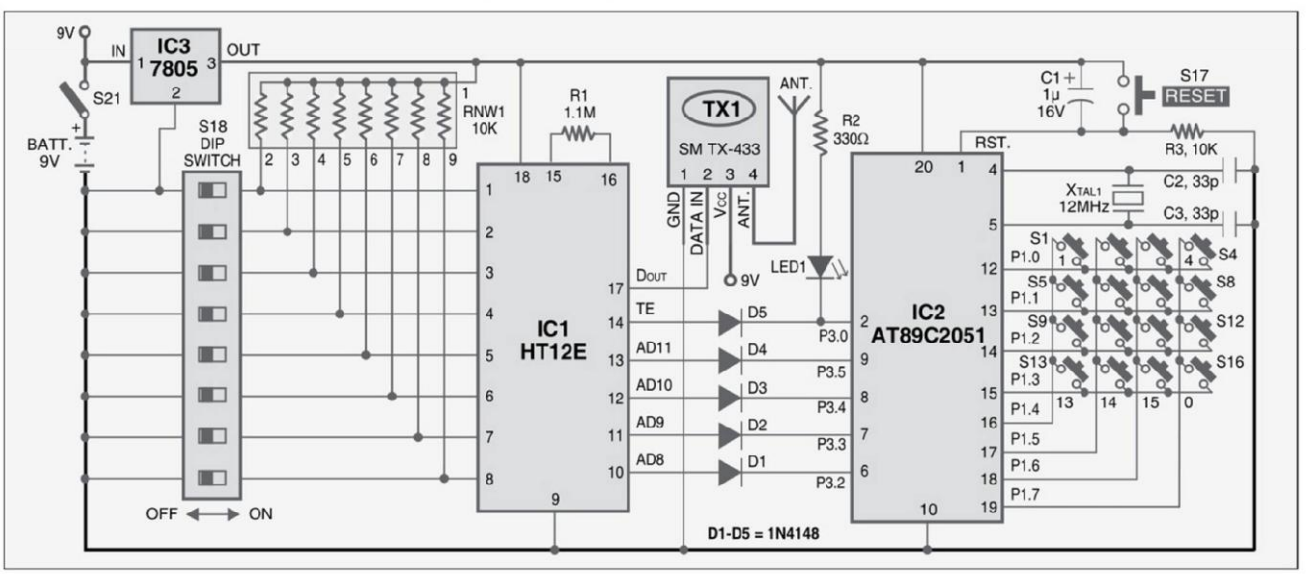


Fig. 2: Transmitter circuit with SM TX-433 RF module (TX1)

may lie between 1.5 kHz and 7 kHz depending on the resistor value used between oscillator pins 15 and 16.

The internal oscillator frequency of decoder HT12D is 50 times the oscillator frequency of encoder HT12E. The values of timing resistors connected between pins 15 and 16 of HT12E and HT12D, for the given supply voltages, can be determined from the graphs given in the datasheet of the respective chips. The resistor values used in the circuit here are chosen for approximately 3kHz frequency for encoder HT12E and 150 kHz for decoder HT12D at a V_{DD} of 5V.

Decoder HT12D receives data from HT12E on its D_{IN} pin serially. If the transmitted address matches the address of the decoder four times in succession, valid transmission pin (V_T) becomes high. The data from pins AD8 through AD11 of the HT12E appears on pins D8 through D11 of the HT12D.

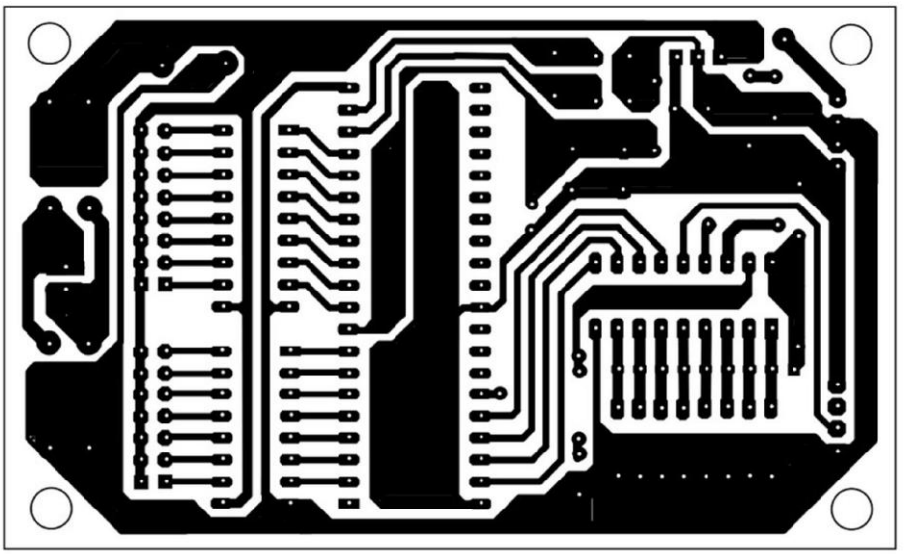


Fig. 6: An actual-size, single-side PCB for the receiver circuit (Fig. 3)

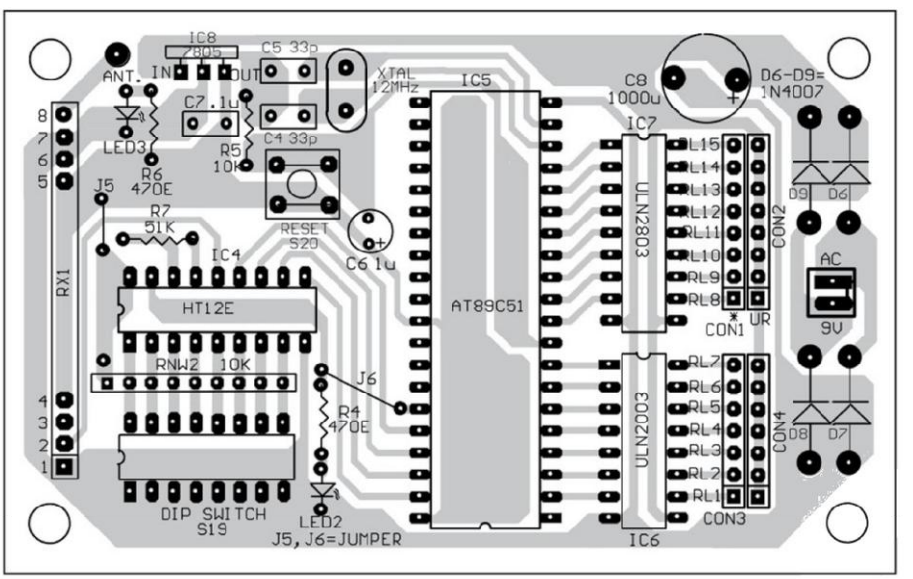


Fig. 7: Component layout for the PCB in Fig. 6

Transmitter unit

Fig. 2 shows the transmitter circuit with SM TX-433 RF module (TX1). TX1 is an AM/ASK transmitter module operating at 433 MHz. AT89C2051 is a low-voltage, high-performance CMOS 8-bit microcontroller. It has 2 kB of Flash, 128 bytes of RAM, 15 input/output (I/O) lines, two 16-bit timers/counters, a five-vector two-level interrupt architecture, a full-duplex serial port, a precision analogue comparator, on-chip oscillator and clock circuitry.

Power-on reset is provided by the combination of resistor R3 and capacitor C1. Switch S17 is used for manual reset. A 12MHz crystal along with two 33pF capacitors provides the basic clock frequency for operation.

The receiver address to be transmitted can be set with the help of 8-way DIP switch S18. Port pins P1.0 through P1.7 of the microcontroller are interfaced with the keypad. Pins P3.0 and P3.2 through P3.5 are interfaced with TE pin and data inputs AD8 through AD11 of encoder HT12E.

When all switches (S1 through S16) are opened on the keypad, the microcontroller pulls the TE pin as well as data input pins AD8 through AD11 to logic 1. If any switch is closed, the microcontroller pulls the corresponding data pin along with TE pin to logic 0. When switch S1 is closed, the microcontroller makes pin 10 (AD8) and pin 14 (TE) of encoder HT12E low, and logic 0 is transmitted through TX1. The other data pins of encoder HT12E will be in logic 1 state in this case. LED1 glows to indicate transmission enabled.

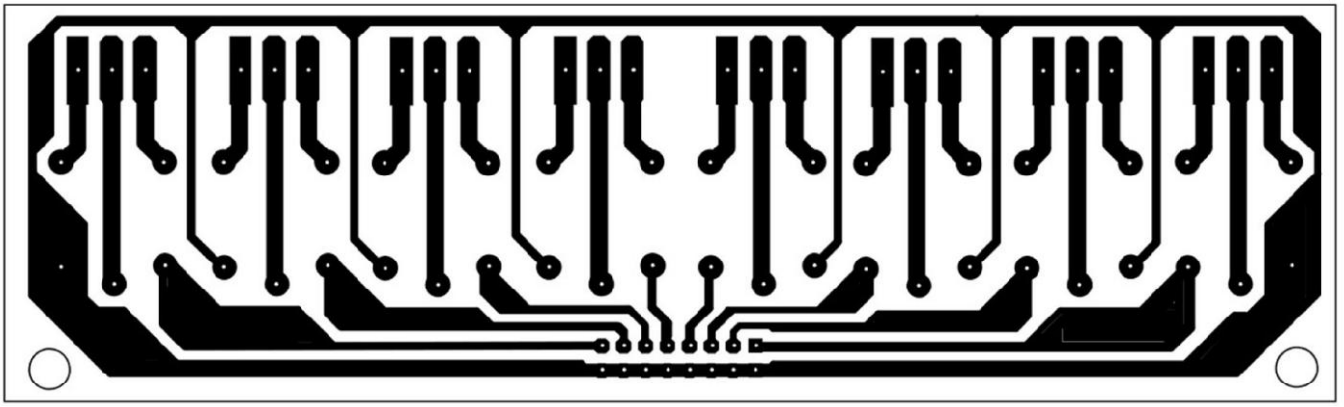


Fig. 8: An actual-size, single-side PCB for relay section

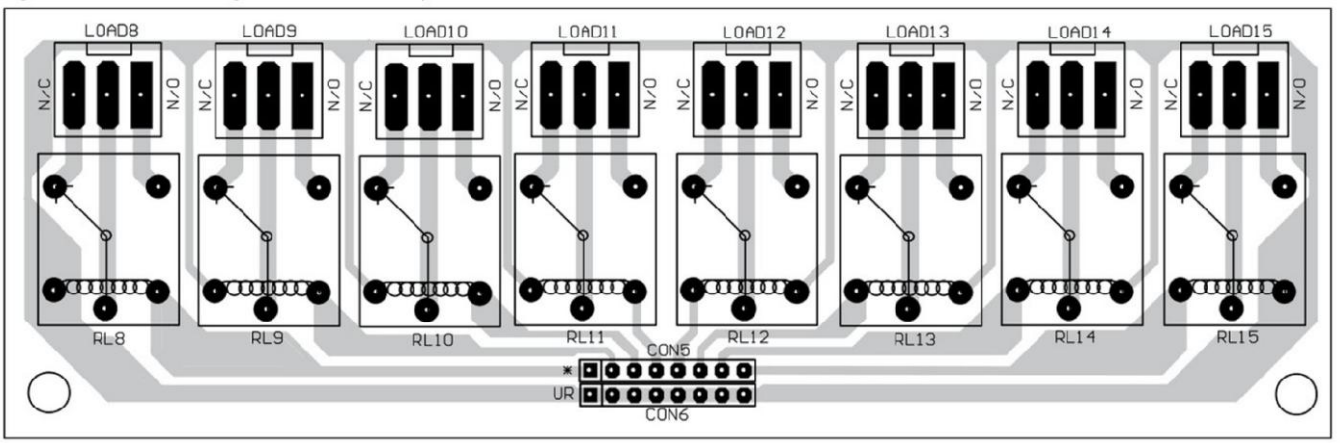


Fig. 9: Component layout for the PCB in Fig. 8

Keys and Corresponding Device Activated

Keypad	Binary	Port pin	Device number
S1	0001	P1.1	1
S2	0010	P1.2	2
S3	0011	P1.3	3
S4	0100	P1.4	4
S5	0101	P1.5	5
S6	0110	P1.6	6
S7	0111	P1.7	7
S8	1000	P3.0	8
S9	1001	P3.1	9
S10	1010	P3.2	10
S11	1011	P3.3	11
S12	1100	P3.4	12
S13	1101	P3.5	13
S14	1110	P3.6	14
S15	1111	P3.7	15
S16	0000	P1.0	Reset all devices

Receiver unit

Fig. 3 shows the receiver circuit with SM RX-433 RF module. AT89C51 is a low-power, high-performance CMOS 8-bit microcontroller. It has 4 kB of Flash, 128 bytes of RAM, 32 I/O lines, two 16-bit timers/counters, five-vector two-level interrupt architecture, a full-duplex serial port, on-chip oscillator and clock circuitry.

Power-on reset is provided by the combination of resistor R5 and capacitor C6. Switch S20 is used for manual reset. A 12MHz crystal along with two 33pF capacitors provides the basic clock frequency to microcontroller AT89C51.

Address lines of the encoder (IC1) and the decoder (IC4) should be identical for data reception in the receiver. Here addresses are made identical through switches S18 and S19. When any of the keys on the keypad is closed, the corresponding data pin of the decoder goes low. When any data is received, valid transmission pin (V_T) goes high as indicated by LED2.

Data outputs D8 through D11 of HT12D are connected to port pins P0.0 through P0.3 of the microcontroller. The microcontroller receives the decoded data and controls the corresponding relay through relay drivers ULN2003 and ULN2803. The device to be controlled is connected to the relay contacts. Unregulated power supply is used for relays.

Power supply

The 230V AC mains is stepped down by transformer X1 to deliver a secondary output of 6V, 500 mA. The transformer output is rectified by a full-wave rectifier comprising diodes D6 through D9, filtered by capacitor C8 and regulated by IC 7805 (IC8). Capacitor C7 bypasses the ripples present in the regulated supply. LED3 acts as the power indicator and R6 limits the current through LED3.

Software

Programs for the microcontrollers are written in BASIC and compiled using BASCOM Basic compiler for 8051 family. These are supported by Windows OS. The microcontrollers of the transmitter and receiver units are programmed with source programs 'Remote.bas' and 'Receiver.bas,' respectively. The bas codes are converted into hex codes using the above compiler. The hex codes are burnt into the respective microcontrollers using a suitable programmer.

Construction and testing

An actual-size, single-side PCB for the transmitter circuit (Fig. 2) is shown in Fig. 4 and its component layout in Fig. 5. The PCB for the receiver circuit excluding relay section (Fig. 3) is shown in Fig. 6 and its component layout in Fig. 7. The PCB for relays RL8 through RL15 connected to load 8 through load 15 is shown in Fig. 8 and its component layout in Fig. 9. You can use another PCB for relays RL1 through RL7 to connect load 1 through load 7. Suitable connectors are provided on the PCB. The receiver PCB is interfaced with the relay PCB by connecting CON1 to CON5, and CON2 to CON6.

PARTS LIST

Semiconductors:

IC1	- HT12E encoder
IC2	- AT89C2051 microcontroller
IC4	- HT12D decoder
IC5	- AT89C51 microcontroller
IC6	- ULN2003 relay driver
IC7	- ULN2803 relay driver
IC3, IC8	- 7805, 5V regulator
D1-D5	- 1N4148 switching diode
D6-D9	- 1N4007 rectifier diode
LED1-LED3	- 5mm LED

Resistors (all ¼-watt, ±5% carbon):

R1	- 1.1-mega-ohm
R2	- 330-ohm
R3, R5	- 10-kilo-ohm
R4, R6	- 470-ohm
R7	- 51-kilo-ohm
RNW1, RNW2	- 10-kilo-ohm network resistor

Capacitors:

C1, C6	- 1µF, 16V electrolytic
C2-C5	- 33pF ceramic disk
C7	- 0.1µF ceramic disk
C8	- 1000µF, 25V electrolytic

Miscellaneous:

X1	- 230V AC primary to 6V, 500mA secondary transformer
RL1-RL15	- 6V, 1C/O relay
S1-S17, S20	- Push-to-on tactile switch
S18, S19	- 8-way DIP switch
S21	- On/off switch
X_{TAL1}, X_{TAL2}	- 12MHz crystal
TX1	- 433MHz RF transmitter module (ASK)
RX1	- 433MHz RF receiver module (ASK)

After assembling the transmitter, receiver and relay sections on the respective PCBs, pull pin 1 of both HT12E and HT12D to ground. LED2 connected to V_T pin of the decoder should glow, indicating that a valid signal has been received. 255 sets of transmitter-receiver pairs can be used within the same area, each with a unique address. Alternately, we can also control 255 receivers with a remote control by changing the address. Now your RF-based multiple device control system is ready for use.

Download source code: http://www.efymag.com/admin/issuepdf/RF%20Based%20Multi%20Device%20Control.zip

GSM-BASED BOREWELL WATER-LEVEL MONITOR

■ **GURUNATH REDDY**

If the water level in a borewell drops below the threshold level for pumping, its pump motor may get air-locked or even burn out due to dry running. It is inconvenient for farmers to walk all the way to their fields at night just to switch the pump motor 'off.' Besides, he may never get to know the problem.

This problem can be solved by using this GSM-based system that will automatically give the user a call on his mobile phone when the water level in the borewell drops below or rises to the threshold level for pumping. The user can also remotely switch on or switch off the pump motor by sending an SMS from

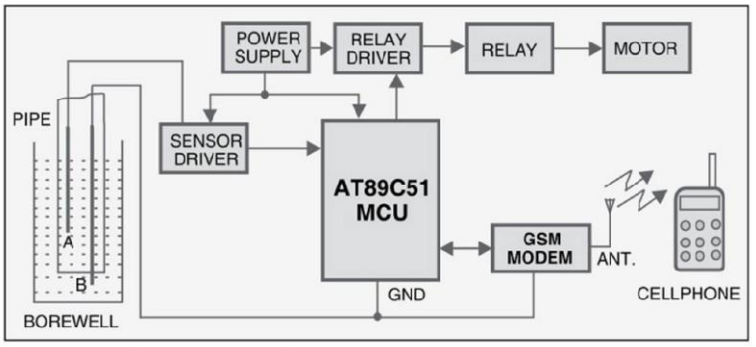


Fig. 1: Block diagram of GSM-based borewell water-level monitoring system

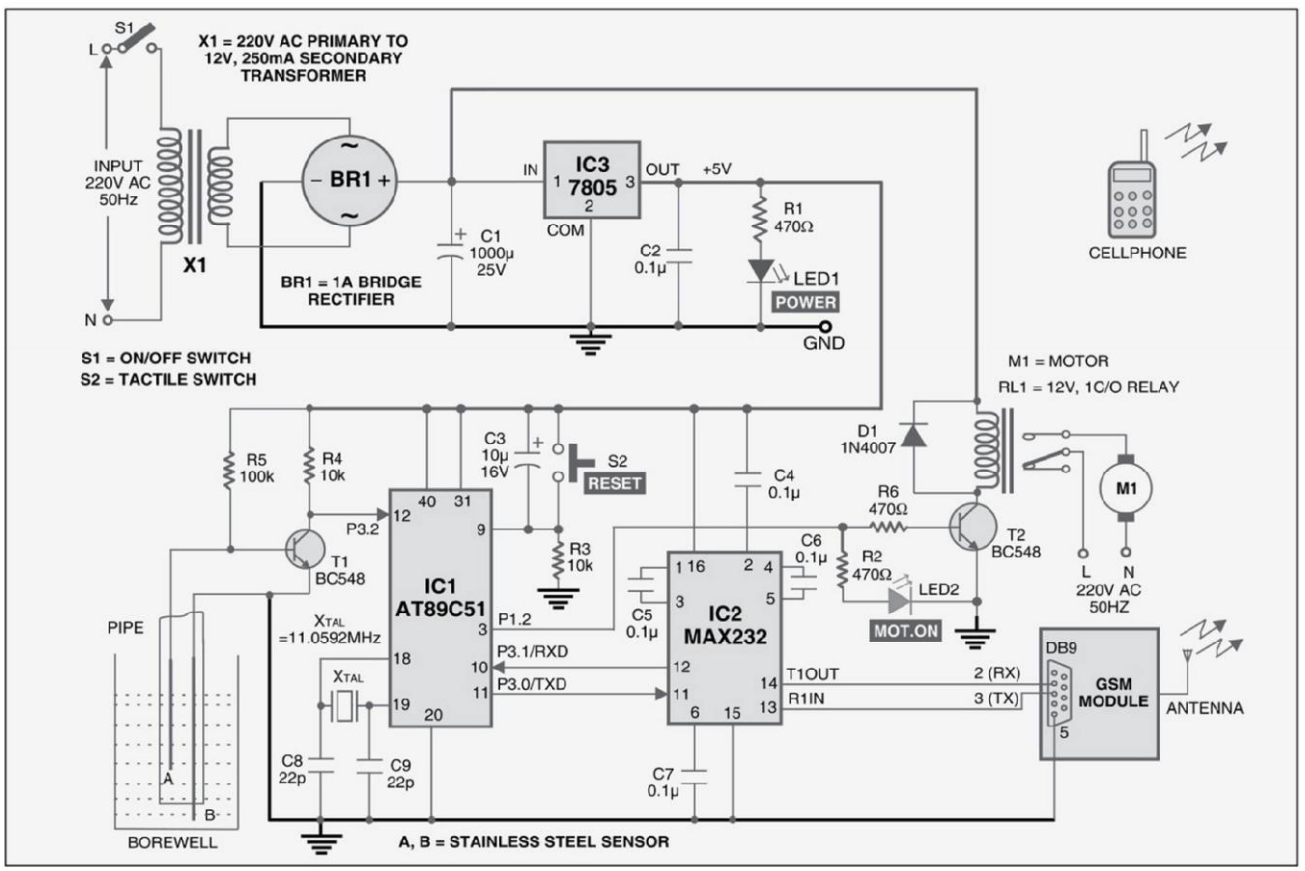


Fig. 2: Circuit of GSM-based borewell water-level monitoring system

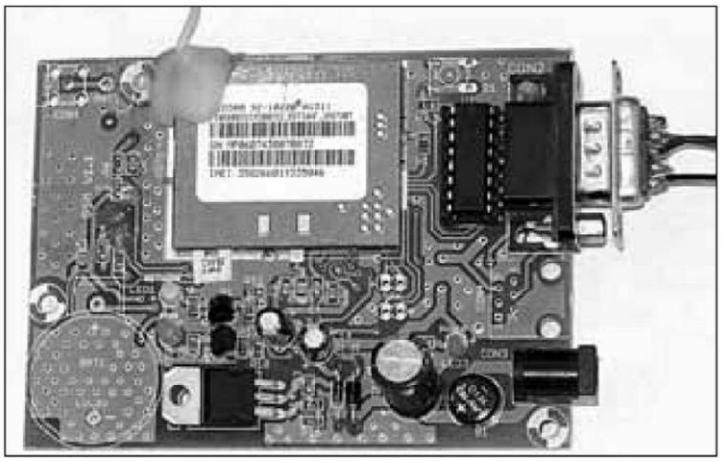


Fig. 3: GSM modem

his mobile phone. The system is simple, reliable, portable and affordable.

Circuit description

Fig. 1 shows the block diagram of the GSM-based borewell water-level monitoring system. Fig. 2 shows the complete circuit. It comprises the power supply section, water-level sensor circuit, microcontroller, MAX232 driver, relay driver and GSM modem. The GSM board has a valid SIM card with sufficient recharge amount to make outgoing calls.

The circuit is powered by regulated 5V DC. The 220V, 50Hz AC mains is stepped down by transformer X1 to deliver a secondary output of 12V, 250 mA. The transformer output is rectified by bridge rectifier BR1, filtered by capacitor C1 and regulated by IC 7805 (IC3). Capacitor C2 bypasses ripples from the regulated supply. LED1 acts as the power-'on' indicator. Resistor R1 limits the current through LED1.

The AT89C51 microcontroller is connected to the water-level sensor circuit, relay driver and MAX232. The microcontroller is programmed to take necessary actions. The mobile number used in the GSM modem is included in the code before burning the code into the microcontroller.

The water-level sensor circuit comprises transistor T1 (BC548) as sensor driver and water sensors A and B dipped into the borewell along with the pipe. Sensor A is dipped to the threshold point for pumping and sensor B is dipped below the pipe to the bottom of the borewell.

TABLE I

Motor, Mobile and LED Status for Different Water Levels

Borewell	Water level	Motor	Mobile	LED2	Remarks
Full	Above point A	On	Call from modem	On	SMS operation to turn on/off the motor
Empty	Below point A	Off	Call from modem	Off	SMS operation not allowed

TABLE II

Key Features of SIM300 Series

Features	Implementation
Power supply	Single supply voltage of 3.4V-4.5V
Power saving	Typical power consumption in SLEEP mode: 2.5 mA
Frequency bands	➤ SIM300 tri-band (EGSM 900, DCS 1800, PCS 1900). The band can be set by AT COMMAND, and default band is EGSM 900 and DCS 1800 ➤ Compliant to GSM Phase 2/2+
SMS	➤ MT, MO, CB, text and PDU mode ➤ SMS storage: SIM card ➤ Supports transmission of SMS alternatively over CSD or GPRS. User can choose the preferred mode
SIM interface	Supported SIM card: 1.8V, 3V
External antenna	Connected via a 50-ohm antenna connector or antenna pad
Two serial interfaces	➤ Serial port 1: Seven lines on the serial port interface ➤ Serial port 1 can be used for CSD FAX, GPRS and sending AT command of controlling module ➤ Serial port 1 can use multiplexing function, but you cannot use serial port 2 at the same time ➤ Autobauding supports baud rate of 1200 to 115,200 bps ➤ Serial port 2: Two lines on serial port interface, /TXD and /RXD ➤ Serial port 2 used only for transmitting AT command

When water in the borewell fills to the threshold level, it is sensed by sensor A and you get a call on your mobile phone. Now you can turn the motor 'on' by sending the SMS "motor11 on" from your mobile phone to the SIM number in the GSM modem. You can also turn the motor 'off' by sending the SMS "motor11 off".

Sensor A is connected to the base of transistor T1 (BC548). When there is a high voltage at the base, T1 conducts and a low voltage is available at its collector. This low signal is fed to pin 12 (port pin p3.2) of the MCU. Similarly, for a low voltage input at the base, T1 stops conducting and a high voltage signal is available at its collector. So pin 12 of the MCU gets a high signal input. The high or low voltage signal at pin 12 is monitored and processed by the program in the MCU, and decision to turn the motor 'off' taken when the water level dips below sensor A.

Pin 3 (port pin p1.2) of the MCU is the output pin. It is connected to relay-driver transistor T2 (BC548) and LED2. T2 drives relay RL1, which, in turn, activates the motor. LED2 glows to indicate the motor-'on' status.

When water level in the borewell dips below sensor A, the conducting path between sensors A and B breaks. Hence a signal is received by the microcontroller. The microcontroller turns the running motor 'off' and makes a call to the user's cell phone through a GSM modem to indicate that the water level is too low to pump the water and the motor has been switched off. In this way, the motor is protected from airlocks and burnouts due to dry running. The status of motor, water level and LED2 are shown in Table I. The GSM modem used in this project is SIM300 V7.03 (refer Fig. 3). Its key features are listed in Table II.

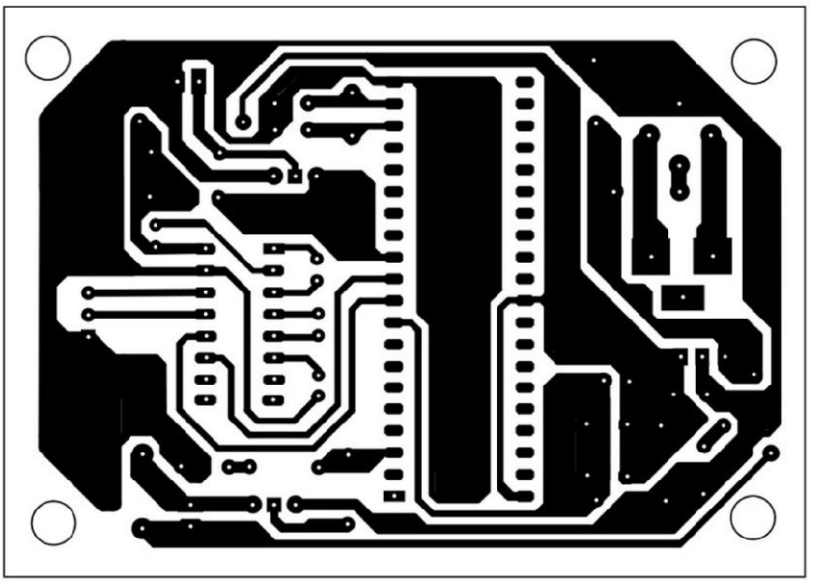


Fig. 4: An actual-size, single-side PCB for the GSM-based borewell water-level monitoring system

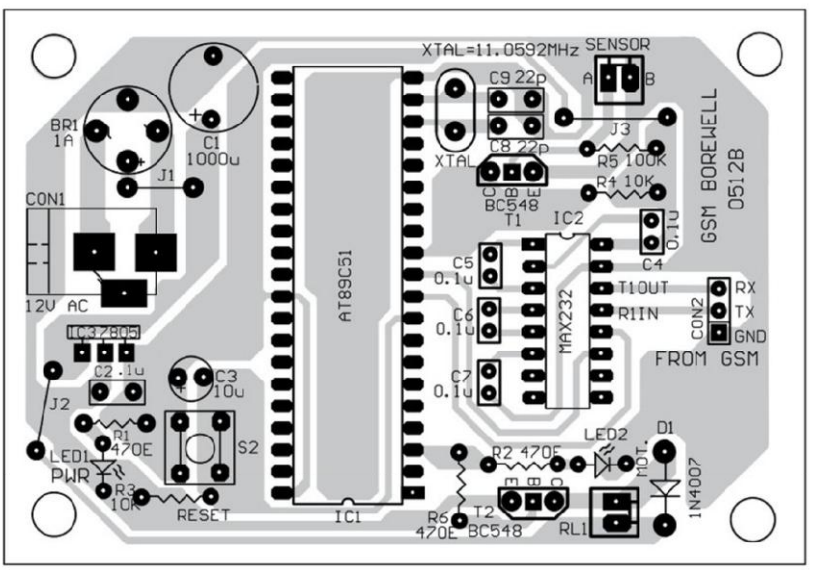


Fig. 5: Component layout for the PCB

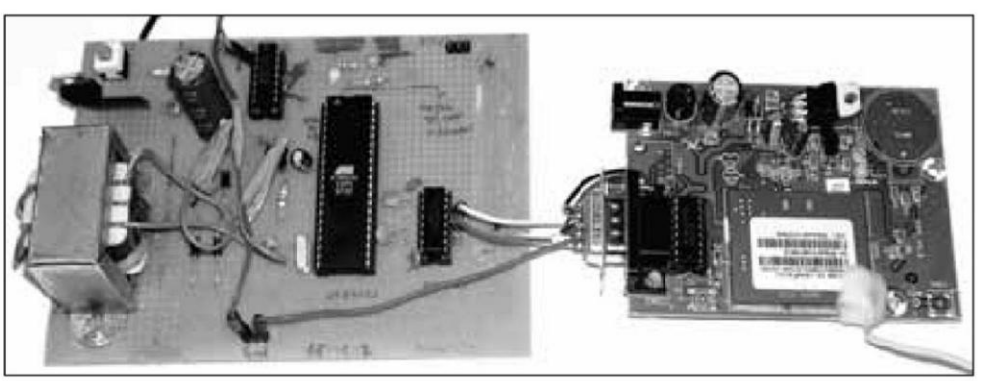


Fig. 6: Author's prototype

TABLE III

List of Commands

Command	Description
AT	Check whether the serial interface and GSM modem are working
ATE0	Turn echo 'off' when there is less traffic on serial line
AT+CNMI	Display the new incoming SMS
AT+CPMS	Select SMS memory
AT+CMGF	SMS string format—how they are compressed
AT+CMGR	Read the new message from a given memory location
AT+CMGS	Send message to a given recipient
AT+CMGD	Delete message

PARTS LIST

Semiconductors:

IC1	- AT89C51 microcontroller
IC2	- MAX232 driver
IC3	- 7805, 5V regulator
BR1	- 1A bridge rectifier
T1, T2	- BC548 npn transistor
LED1, LED2	- 5mm light-emitting diode
D1	- 1N4007 rectifier diode

Resistors (all ¼-watt, ±5 per cent carbon):

R1	- 1-kilo-ohm
R2, R6	- 470-ohm
R3, R4	- 10-kilo-ohm
R5	- 100-kilo-ohm

Capacitors:

C1	- 1000µF, 25V electrolytic
C2, C4-C7	- 0.1µF ceramic
C3	- 10µF, 16V electrolytic
C8, C9	- 22pF ceramic

Miscellaneous:

X1	- 220V AC primary to 12V, 250mA secondary transformer
RL1	- 12V, 1C/O relay
S1	- Tactile switch
X_{TAL}	- 11.0592MHz crystal
GSM modem	- SIM300 V7.03 modem
	- Two water-level steel sensor rods

GSM modem SIM300 V7.03

The GSM module is a specialised type of modem which accepts a SIM card and operates on a subscriber's mobile number over a network, just like a cellular phone. Basically, it is a cellphone without display. Modem SIM300 is a triband GSM/GPRS engine that works on EGSM 900MHz, DCS 1800MHz and PCS 1900MHz frequencies.

GSM modem is RS232-logic-level compatible, i.e., it takes -3V to -15V as logic 'high' and +3V to +15V as logic 'low'. MAX 232 is used to convert TTL into RS232 logic level and vice versa. Hence MAX232 is a voltage-level converter used between the microcontroller and the GSM board.

The signal at pin 11 of the microcontroller is sent to the GSM modem through pin 11 of MAX232. This signal is received at Pin 2 (RX) of the GSM modem. The GSM modem transmits the signal from Pin 3 (TX) to the microcontroller through MAX232, which is received at pin 10 of IC1.

Software program

The software program is written in 'C' language and compiled using Keil software. The AT commands listed in Table III are used in the code to receive the mobile signal. The hex code of the program is burnt into the MCU using Flash Magic software.

Construction and testing

An actual-size, single-side PCB layout of the GSM-based borewell water-level monitoring system is shown in Fig. 4 and its component layout in Fig. 5.

For testing the circuit, proceed as follows:

1. After assembling all the components on the PCB, connect TX and RX pins of the GSM modem to pins 13 and 14 of MAX232, respectively. Insert a valid SIM in the card holder of the GSM modem.

2. Connect ground pin of the GSM modem to the ground rail of the circuit.

3. Use two single-strand (hook-up) wires as sensor A and sensor B. (In the actual application, use of steel rods as sensors is recommended.) Hang the sensors into a bucket or mug such that sensor A is above sensor B.

4. Pour water into the bucket until the water level reaches sensor A.

5. Now switch on the circuit. You should get a call on your mobile phone. This indicates that you can turn the motor 'on.'

6. Send SMS "motor11 on" from any mobile phone to the SIM in the modem to turn the motor 'on'. You can also turn the motor 'off' by sending "motor11 off" message from any mobile phone.

7. Now remove water from the bucket until the water level in the bucket dips below sensor A. The motor should automatically switch off and you should receive a call from the modem simultaneously alerting you that the borewell (bucket in this case) is empty and the motor has been switched off.

After testing the above steps, you can install the system in the borewell by inserting sensors A and B into the pipe with sensor B placed at the bottom of the borewell as shown in the circuit. Your borewell monitoring system is now ready for use. The author's prototype is shown in Fig. 6.

Download source code: http://www.efymag.com/admin/issuepdf/GSM%20Based%20Borewell%20Water%20Level%20Monitor_May12.zip

WIRELESS WATER-LEVEL INDICATOR

■ ROBIN CHALANA

Using this system, you can remotely monitor the water level of an overhead tank that is placed up to 30 metres away. The system features an RF transmitter-receiver pair, doing away with the need to run wires from the roof to ground. The transmitter is placed near the tank with sensors inside the tank to monitor the level of water. The sensed level is streamed wirelessly through the RF transmitter. This is received by the receiver unit placed remotely and decoded to indicate the water level on an LCD. It also has a buzzer that beeps when the water level drops below one-fourth level or when the tank is about to overflow.

When the tank is quarter-, half- or three-quarters-full, the percentage of the water level is flashed on the LCD. The system is developed using an EFY-KnS 8051 development board which is available from EFY associates Kits'n'Spares (KnS) for Rs 500.

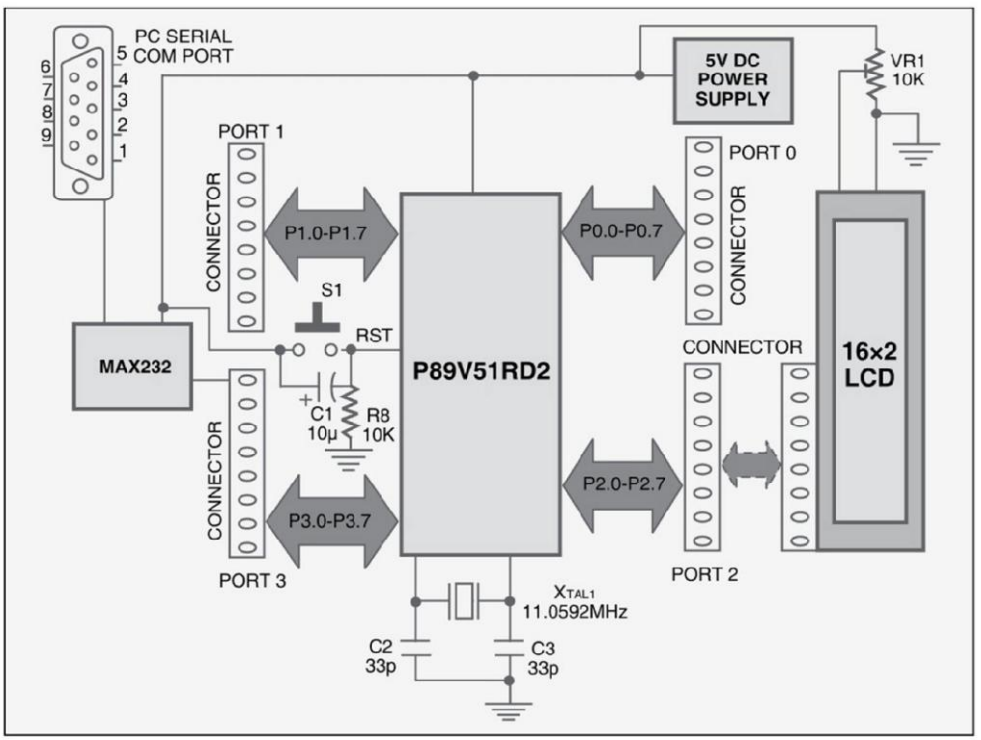


Fig. 1: Block diagram of EFY-KnS 8051 development board

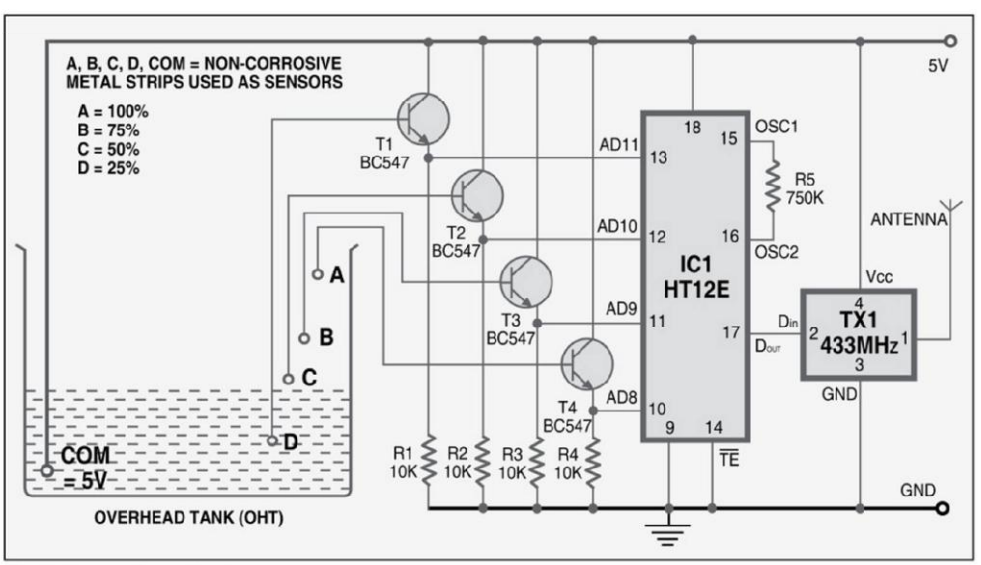


Fig. 2: Transmitter circuit

EFY-KnS' 8051 development board

Fig. 1 shows the block dia-

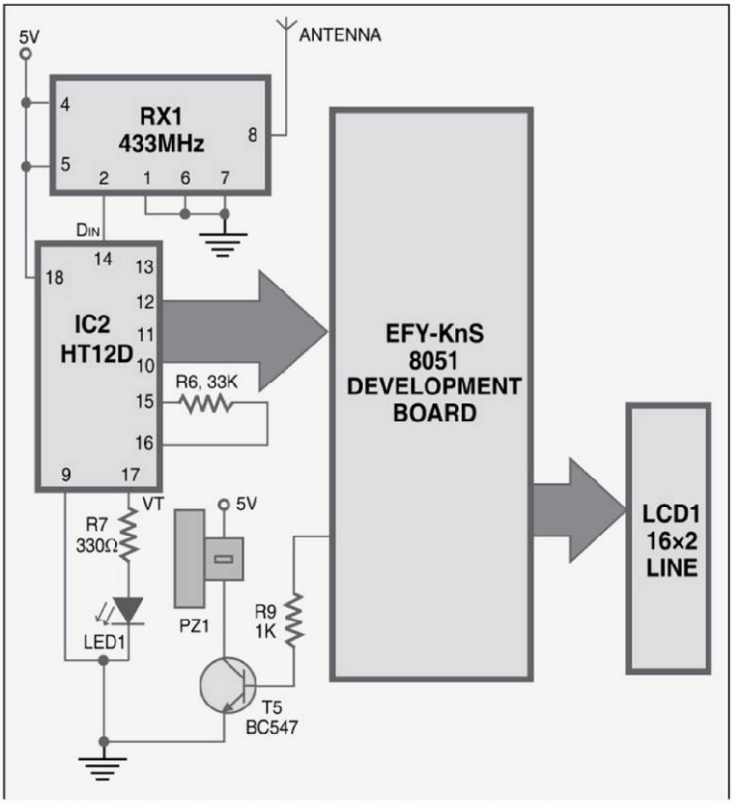


Fig. 3: Testing of receiver circuit using development board

gram of EFY-KnS 8051 development board. It consists of a 40-pin IC socket, four 8-pin berg-strip male connectors, 9-pin serial COM port connector, MAX232 driver, 16-pin connector for the LCD and 5V regulator. Some other features of the board include:

1. Power LED with an on/off switch
2. Reset LED with a reset switch. This LED is also used as a programming-status indicator.
3. A 5-pin male connector (not shown here) for 5V DC supply. The voltage is regulated to 5V using voltage regulator IC 7805.
4. Eleven jumper wires (16cm each) for connecting the LCD

Data and control pins of the LCD are not permanently connected to the microcontroller. So you can use any of the controller ports to connect the LCD using jumper wires.

Circuit description

Fig. 2 shows the circuit of the transmitter. It operates off 5V DC and consists of a sensor assembly, encoder HT12E (IC1) and RF transmitter module (TX1). The sensor assembly consists of four BC547 npn transistors (T1 through T4), each connected to a water-level-sensor metal strip corresponding to one of the four water levels—25 per cent, 50 per cent, 75 per cent and 100 per cent. The sensors are non-corrosive stainless-steel metal strips.

The receiver circuit too operates off 5V DC. It is assembled and tested on EFY-KnS 8051 development board as shown in Fig. 3. Mount the buzzer (PZ1), HT12D and RX1 on a general-purpose PCB or breadboard. Connect decoder HT12D (IC2), piezobuzzer (PZ1) and LCD1 to the development board. The output of the RF receiver (RX1) is fed to data input D_{in} (pin 14) of decoder HT12D. A red LED (LED1) is connected to VT pin (pin 17) of the decoder through R7.

Data is processed by P89V51RD2 microcontroller mounted on the development board. The program is written in Assembly language and assembled using ASM51 cross-assembler. Burn the code into the microcontroller using the on-board RS-232 serial COM port.

4-bit Code, Message and Buzzer Status for Different Water Levels

Water level	4-bit code (DCBA)	Message displayed on the LCD	Buzzer status
Less than 25 per cent	0000	Water level low	Buzzer rings for two minutes
25 per cent	1000	Water level 25 per cent	—
50 per cent	1100	Water level 50 per cent	—
75 per cent	1110	Water level 75 per cent	—
100 per cent	1111	Water level 100 per cent	Buzzer rings for two minutes

How it works?

The water level in the tank is sensed by the sensor assembly, which is connected to 4-bit data lines (AD8 through AD11) of encoder HT12E (IC1) through transistors. Depending on the water level in the tank, BC547 transistors (T1 through T4) conduct to generate a 4-bit code (refer the table). The 4-bit code so generated is encoded by en-

coder HT12E. The encoded data is fed to pin 2 of RF transmitter TX1, which transmits it serially at 433 MHz through the antenna connected to its pin 1. The transmission range of TX1 module is about 30 metres.

The 4-bit signal from the transmitter is received by the antenna of the RF receiver (RX1). LED1 glows to indicate that a valid signal is received. The 4-bit output from decoder HT12D is processed by the microcontroller to generate an 8-bit code. The microcontroller's output is fed to data input lines of LCD1, which, in turn, shows the water level in percentage.

At an intermediate water level, say, 25 per cent of the tank capacity, LCD1 shows the message "Water level 25 per cent. When the tank is full, the buzzer rings for two minutes, while LCD1 shows "Water level 100 per cent. The 4-bit code, message and buzzer status for different water levels are shown in the table.

PARTS LIST

Semiconductors:

IC1	- HT12E encoder
IC2	- HT12D decoder
T1-T5	- BC547 npn transistor
LED1	- 5mm light-emitting diode
LCD1	- 16×2 line LCD module
TX1, RX1	- 433MHz RF module

Resistors (all ¼-watt, ±5 per cent carbon):

R1-R4, R8	- 10-kilo-ohm
R5	- 750-kilo-ohm
R6	- 33-kilo-ohm
R7	- 330-ohm
R9	- 1-kilo-ohm
VR1	- 10-kilo-ohm preset

Capacitors:

C1	- 10nF ceramic disk
C2, C3	- 33pF ceramic disk

Miscellaneous:

X_{TAL1}	- 11.0592MHz crystal
S1	- Push-to-on switch
PZ1	- Piezobuzzer
Board	- EFY-KnS 8051 development board

Software

In the program, first LCD1 and buzzer are initialised followed by the reset, read-write and enable pins of LCD1. Then LCD1 shows 'EFY' in the first line and 'Water Level' in the second line. The program further enters a loop to check which of the five values is true—less than 25 per cent, 25 per cent, 50 per cent, 75 per cent or 100 per cent. The percentage is displayed in the second line after 'Water Level.' The piezobuzzer is timed to sound for two minutes for the 100 per cent full condition and less than 25 per cent full condition.

This circuit can also be modified to work as a water-level controller system. When the water level is low, the microcontroller can be programmed to start the motor pump. When the tank is full, the same can be made to stop the motor.

Download source code: http://www.efymag.com/admin/issuepdf/Wireless%20Water%20Level%20Indicator.zip

MAKE YOUR OWN DIGITAL ALARM CLOCK

■ ATISH GUPTA

Time management is very important in today's fast-paced life. An alarm clock helps you manage your time effectively by alerting you of appointments or other important tasks.

Here we have described a microcontroller-based digital alarm clock. The time and alarm can be customised by the user, and are shown on the liquid crystal display (LCD) of the system.

Circuit description

Fig. 1 shows the circuit of the microcontroller-based digital alarm clock. It comprises microcontroller AT89C51, an LCD module, regulator 7805 and a few discrete components.

Microcontroller AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer with 4 kB of Flash programmable and erasable read-only memory (PEROM). It has the following standard features: 4 kB of Flash, 128 bytes of RAM, 32 I/O lines, two 16-bit timers/counters, five-vector two-level interrupt architecture, a full-duplex serial port, and on-chip oscillator and clock circuitry. In addition, the AT89C51 is designed with static logic for operation down to zero frequency and supports two software-selectable power-saving modes. The idle mode stops the CPU while allowing the RAM, timers/counters, serial port and interrupt system to continue functioning. The power-down mode saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next hardware reset.

The system clock also plays a significant role in operation of the microcontroller. An 11.0592MHz quartz

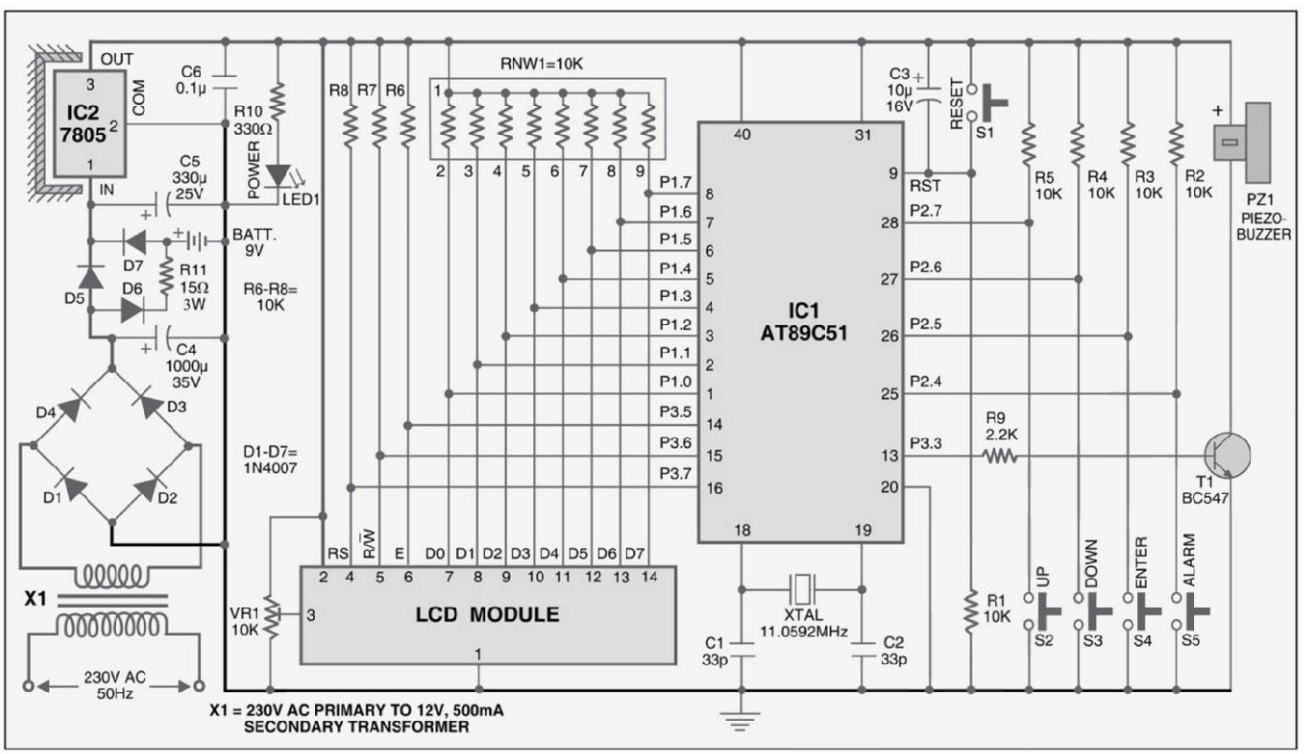


Fig. 1: Circuit of microcontroller-based digital alarm clock

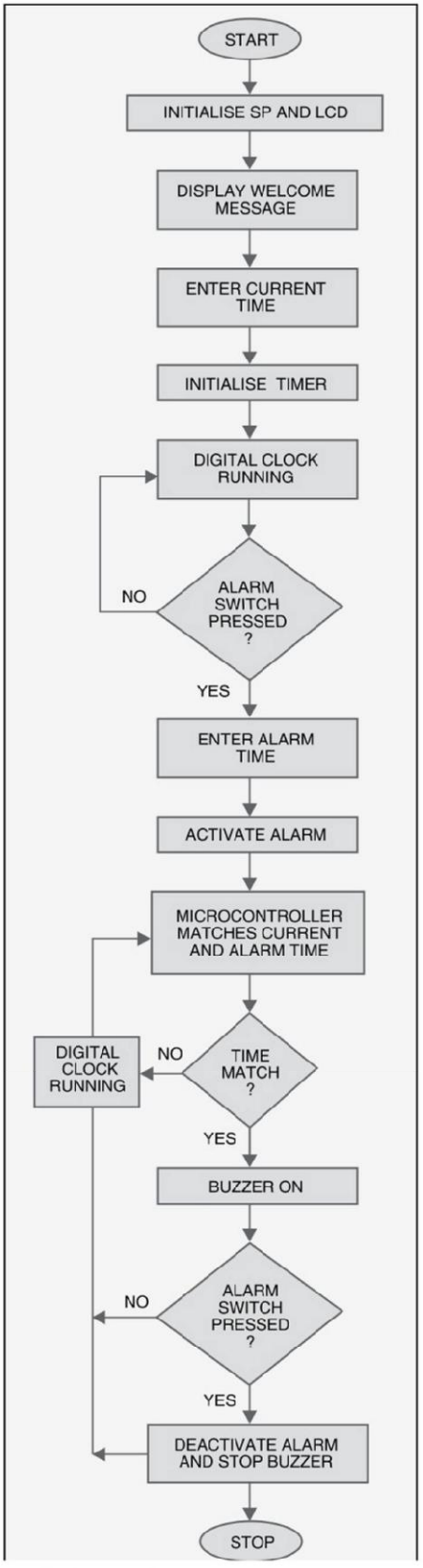


Fig. 2: Flow chart

crystal is connected to pins 18 and 19, to provide basic clock to the microcontroller. Power-on reset is provided by electrolytic capacitor C3 and resistor R1. Switch S1 is used for manual reset. The clock time is shown on the LCD.

Port pins P1.0 through P1.7 of the microcontroller are connected to data pins D0 through D7 of the LCD, respectively. Port pins P3.7, P3.6 and P3.5 are connected to register-select RS, read-write R/$\overline{W}$ and enable E of the LCD, respectively. Port pins P1.0 through P1.7 are pulled high by resistor network RNW1, while port pins P3.5 through P3.7 are pulled high by resistors R6 through R8, respectively.

All the data is sent to the LCD in ASCII format for display. Only the commands are sent in hex form to the LCD. Register-select RS signal is used to distinguish between data (RS=1) and command (RS=0). Using preset VR1 you can control the contrast of the LCD.

Switch on the power to the system and enter the current time with the help of switches S2 through S4. 'Up,' 'down' and 'enter' switches are connected to pins P2.7, P2.6 and P2.5 of microcontroller AT89C51, respectively. Using 'up' switch, you can increase the numeral values of hours, minutes and seconds, respectively, for time or alarm setting, while 'Down' switch can be used to decrease the values. The current time or alarm time set using up/down switches is accepted on pressing 'enter' switch. The time is set and displayed in 12-hour (a.m./p.m. format).

Using 'alarm' switch connected to pin P2.4 of the microcontroller you can activate or deactivate the alarm function. When the alarm is activated, enter the new alarm time in the same way as the current time. The message "Alarm Active" is displayed on the second line of the LCD with current running time on first line of the LCD. Every second, current time is compared with the alarm time. When current time matches with alarm time, pin P3.3 of the microcontroller goes high to drive the transistor into saturation. Piezobuzzer PZ1 sounds until you press the alarm switch (S5) to deactivate the alarm.

The 230V, 50Hz AC mains is stepped down by transformer X1 to deliver secondary output of 12V, 500 mA. The transformer output is rectified by a full-wave rectifier comprising diodes D1 through D4, filtered by capacitor C4 and regulated by IC 7805 (IC2). A

PARTS LIST

Semiconductors:

IC1	- AT89C51 microcontroller
IC2	- 7805, 5V regulator
T1	- BC547 npn transistor
D1-D7	- 1N4007 rectifier diode
LED1	- 5mm LED

Resistors (all ¼-watt, ±5% carbon, unless specified):

R1-R8	- 10-kilo-ohm
R9	- 2.2-kilo-ohm
R10	- 330-ohm
R11	- 15-ohm, 3-watt
RNW1	- 10-kilo-ohm resistor network
VR1	- 10-kilo-ohm preset

Capacitors:

C1, C2	- 33pF ceramic
C3	- 10µF, 16V electrolytic
C4	- 1000µF, 35V electrolytic
C5	- 330µF, 25V electrolytic
C6	- 0.1µF ceramic

Miscellaneous:

X1	- 230V AC primary to 12V, 500mA secondary transformer
S1-S5	- Push-to-on tacktile switch
X_{TAL}	- 11.0592MHz crystal
PZ1	- Piezo-electric buzzer
LCD module	- 16×2 line LCD

9V rechargeable battery is used for battery backup. The battery is charged through diode D6. Diode D7 is reverse-biased when the supply is present. The battery powers regulator 7805 to provide regulated 5V supply for the circuit in absence of mains supply. Capacitor C6 bypasses the ripples, if any, in the regulated supply. LED1 acts as the power indicator and resistor R10 limits the current through LED1.

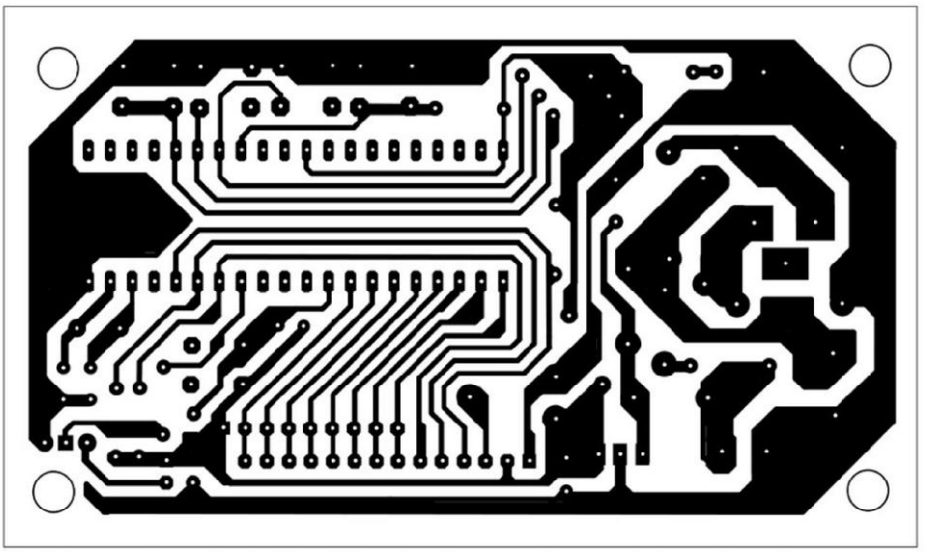


Fig. 3: An actual-size, single-side PCB for the microcontroller-based digital alarm clock

Software

The software program is written in Assembly language and assembled using cross-assembler ASM51. Burn the generated Intel hex code into the microcontroller using a suitable programmer. The software works as per the flow-chart shown in Fig. 2.

When the microcontroller starts on pressing reset switch S1, it initialises the stack pointer and the LCD to display the message. The stack pointer, as a part of the RAM, stores the return address of the main program from subroutines. To initialise the LCD, the microcontroller sends a set of commands to the LCD. The lcd_initialize subroutine in the software is used to initialise the LCD. The delays for the LCD are provided by nested loops instead of the timer. These delays are used to provide timing delays at various locations of the software. The values for the loops are calculated according to the crystal frequency and the machine cycles taken by the respective instructions.

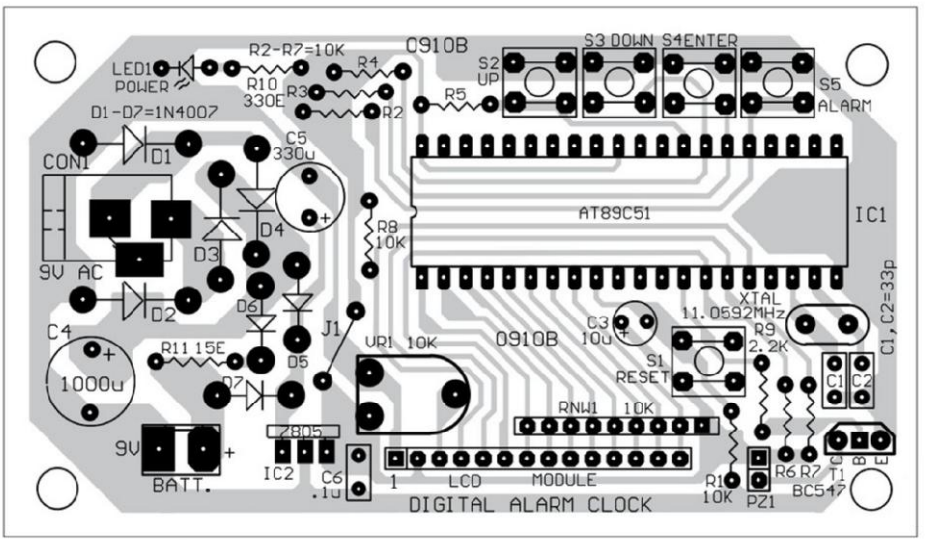


Fig. 4: Component layout for the PCB

Initially, set the current time using switches. The internal timers of the microcontroller are used to provide one-second delay for the digital clock. The hex values fed to timer-1 registers TH1 and TL1 make timer-1 provide a delay of 10ms according to 11.0592MHz crystal. With some internal software logic, one-second delay is formed using 10ms delay of timer-1. Timer-1 is configured in 16-bit mode for the desired operations through TMOD register. Thus current time is updated every second.

The software of the digital clock is interrupt-based. Whenever timer-1 overflows, an interrupt is generated by the microcontroller. This interrupt is provided by the interrupt service routine at vector 001BH. The jump command written at this vector initiates the intr_service routine. All the functioning of the digital clock with alarm is controlled by this service routine.

By default, all the ports act as output. The software makes port pins P2.4 through P2.7 of the microcontroller

act as input. The software also handles the functioning of all the switches—up, down, enter and alarm. Pooling and identification of switches and limits for up and down too are provided by the software.

Construction and testing

An actual-size, single-side PCB of the microcontroller-based digital alarm clock is shown in Fig. 3 and its component layout in Fig. 4. Use IC base for microcontroller AT89C51. Also, use a heat-sink with voltage regulator 7805 (IC2) to avoid any damage to the circuit. Check continuity between respective connections using a multimeter.

Initially, by varying preset VR1, set the contrast level for proper display on the LCD. If the reading or display is not steady, check for loose connections or dry soldering joints.

Download source code: http://www.efymag.com/admin/issuepdf/Make_Your_Own_Alarm_Clock.zip

WIRELESS EQUIPMENT CONTROL USING AT89C51

■ DR A.A. BHASKAR, DR H.N. PANDYA AND S.J. OZA

Here is a microcontroller-based wireless equipment controller that can switch on or switch off up to four devices at a desired time interval set by the user in the transmitter. The devices can be controlled remotely from a distance of up to 30 metres from the transmitter. In the transmitter, an LCD module is used to show the device numbers and preset control time for the devices (00 to 99 seconds). Concepts of wireless RF communication and automation with AT89C51 microcontroller are used here.

The system is small, simple, cost-effective and good for wireless control of home appliances or industrial instrumentation.

RF Module Specifications

Parameter	Value
Frequency of operation	434 MHz
Modulation	ASK
Range	9.14 metres
Power Supply	5V (RX) 3V to 12V (TRX)

Block diagram

The system comprises a transmitter and a receiver as described below.

Transmitter section. Fig. 1 shows the block diagram of the transmitter section.

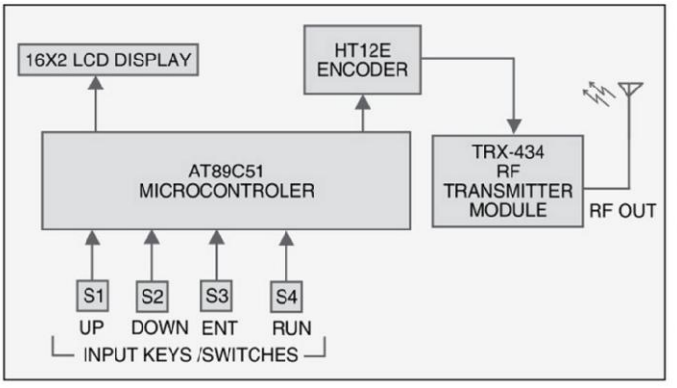


Fig. 1: Block diagram of transmitter section for wireless equipment control

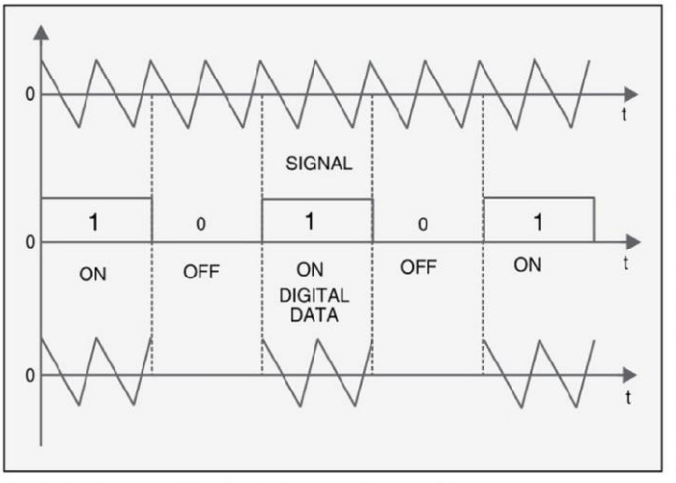


Fig. 2: ASK concept for the RF transmitter module

Four pushbutton switches (S1 through S4) are used as inputs to select the devices and set the time-out in the transmitter section. These are designated as up, down, enter and run keys, respectively. The time-out data is transferred over the RF wireless link to the receiver section.

The 8-bit AT89C51 microcontroller is the main controlling part of the transmitter section. It is connected to the LCD module, input switches and encoder IC (HT12E). The device control program is stored in the memory of the microcontroller to control the devices as per the time-out settings done through input switches S1 through S4.

A two-line, 16-character LCD module shows the status of the main program that is running inside the microcontroller.

The HT12E is an 18-pin DIP package encoder IC that encodes 4-bit data and sends it to TRX-434 RF transmitter module.

The TRX-434 RF transmitter module uses a digital modulation technique called amplitude-shift keying (ASK) or on-off keying. In this technique, whenever logic '1' is to be sent, it is modulated with carrier signal (434MHz). This modulated signal is then transmitted through the antenna. The waveforms in Fig. 2 depict the ASK concept. The main specifications of the RF module are shown in the table.

Receiver section. Fig. 3 shows the block diagram

of the receiver section.

The 12V DC supply, used along with a 5V regulator, can be provided by a 12V battery or power adaptor.

The RX-434 radio receiver module receives the ASK signal from TRX-434. The HT12D decoder demodulates the received address and data bits. IC CD4519 is a quadruple two-input multiplexer that selects the appropriate data bits to control the devices.

The ULN 2003 relay driver consists of seven npn Darlington pairs that feature high-voltage outputs with common-cathode clamp diodes for switching the inductive loads. The collector-current rating of a single Darlington pair is 500 mA.

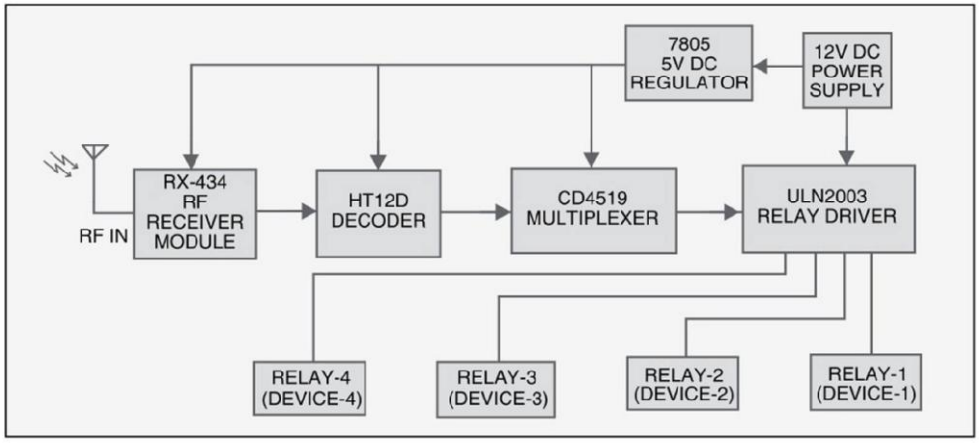


Fig. 3: Block diagram of receiver section for wireless equipment control

Circuit description

Transmitter circuit. Fig. 4 shows the transmitter circuit. The microcontroller reads the input data from switches S1 through S4 at its port-2 pins 21 through 24 and displays it on the LCD. Port 3 provides read data to the encoder IC HT12E at pins 10 through 13. The microcontroller is programmed to control input

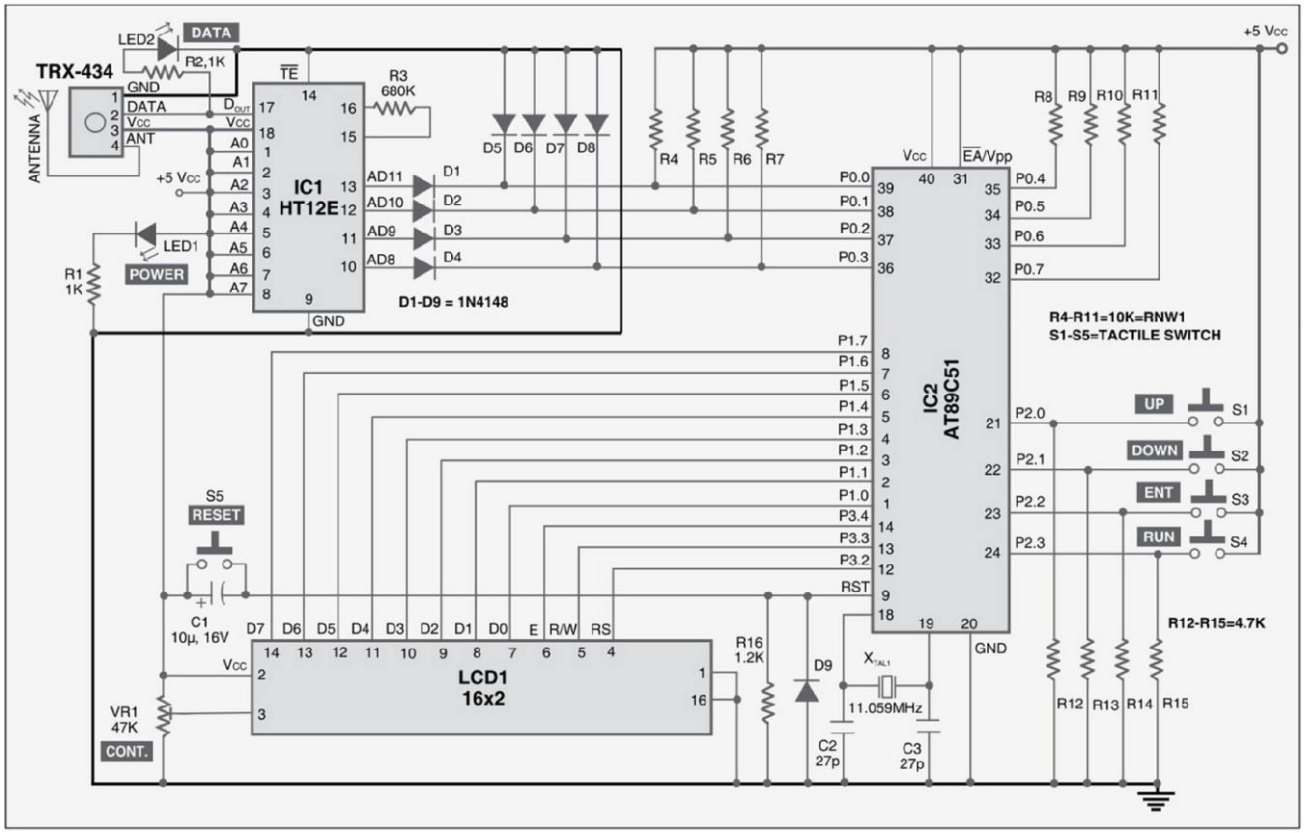


Fig. 4: Transmitter circuit

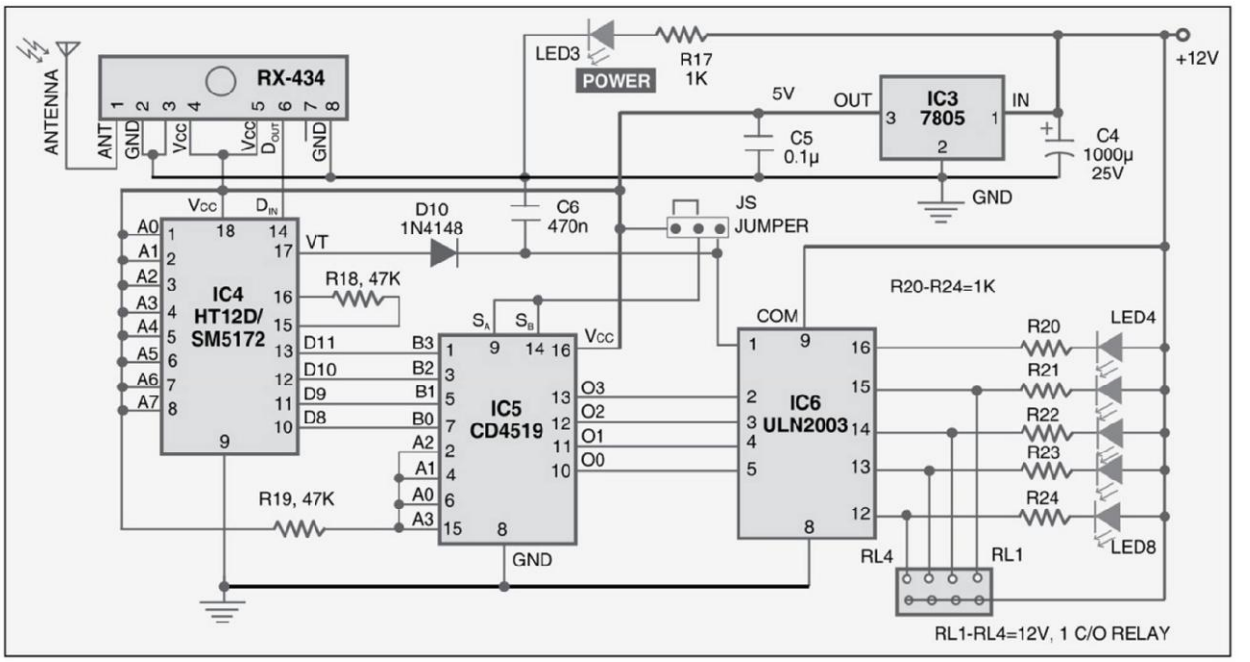


Fig. 5: Receiver circuit

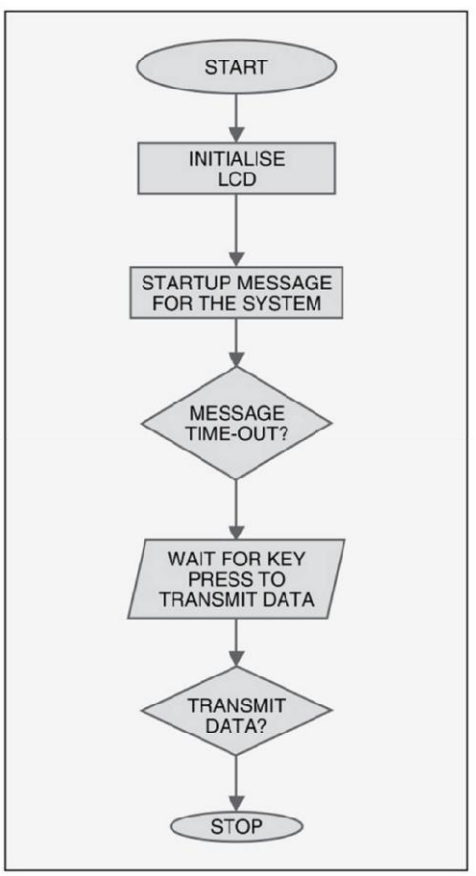


Fig. 6: Software flow-chart

and output data.

When the pushbutton switches (S1 through S4) are open, logic '0' is constantly fed to the respective port pins of the microcontroller. When any of the buttons is pressed, logic '1' is fed to the respective port pin of the microcontroller.

The device control program stored in the memory of the microcontroller activates and executes as per the functions defined in the program for respective input switches.

Data inputs AD8 through AD11 (pins 10 through 13) of HT12E are connected to the microcontroller. Pins 1 through 8 (A0 through A7) of the IC are address inputs. Shorting the address pins using switches to either Vcc or Gnd enables different address selections for data transmission. Here we have connected them to 5V. Since address pins are connected to 5V, the address is set to 255d (in decimal). If you were to connect all the address pins to ground, the address would be 000d. Thus there are 256 possible addresses available. So you can set up switches to control one or more of the encoder address pins.

Pin 14 is a transmit-enable (TE) input pin. The encoder will send data only when pin 14 is connected to ground. Whenever a button is pressed, logic '0' is sent to this pin through the microcontroller, thus activating it and enabling transmission.

Pin 17 is the data-out (D_{out}) pin that sends a serial stream of pulses containing the address and data. It is connected to the data input pin of the TRX RF module.

The time-out control is set using input keys S1 through S4 to turn on/off the devices at predetermined time. The default time for all the devices is '00' seconds. So using 'up' key you can increment time by one second, and using 'down' key you can decrement time by one second down. At the same time, the LCD module shows the current status of increments and decrements.

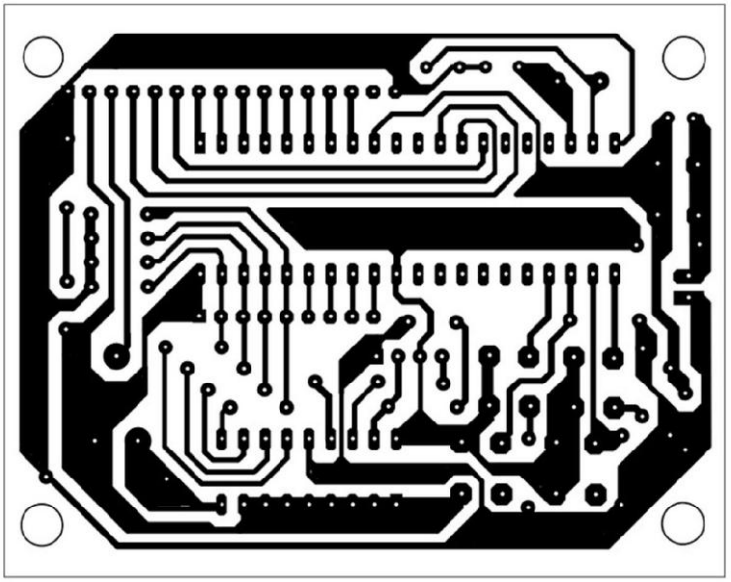


Fig. 7: An actual-size, single-side PCB layout of transmitter circuit for wireless equipment control

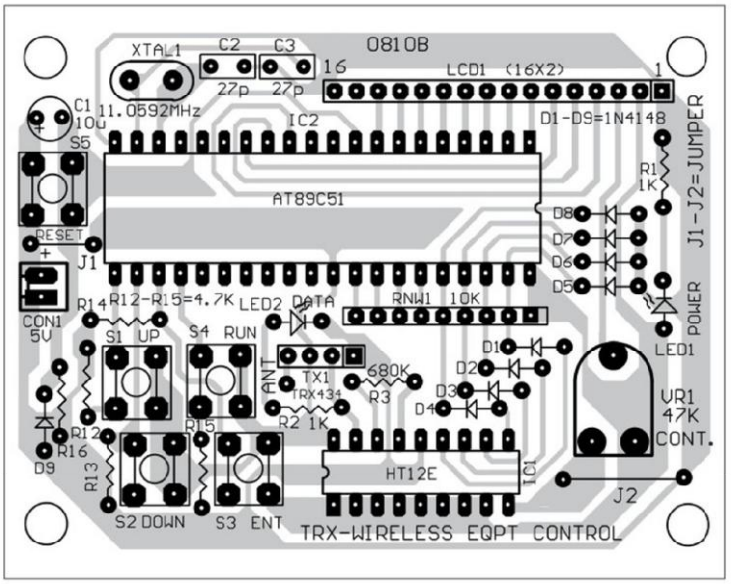


Fig. 8: Component layout for the PCB in Fig. 7

When the time-out for a device is set, press 'ent' key so that the program control transfers to the next device for time-out settings. In the same way, the remaining three time-out settings must be done before pressing 'run' key. When 'run' key is pressed, it executes the device control program subroutine in the microcontroller and the program automatically collects the time-out information entered by the user and sends the processed data to encoder IC HT12E. The encoder IC sends the data to D_{in} (pin 2) of the RF transmitter module. The data is transmitted by the TRX-434 module to the receiver section through the antenna.

Receiver circuit. Fig. 5 shows the receiver circuit. The RF receiver (RX-434) module can receive the signal transmitted by the transmitter from a distance of up to 9 metres (30 feet). The range can be increased up to 30 metres using a good antenna.

D_{out} pin of RX-434 RF module is connected to D_{in} pin of decoder IC HT12D (IC4). D_{in} pin of IC4 receives address and data bits serially from the RF module. Decoder IC4 separates data and address from the received information. It accepts data only if the received address matches with the address assigned to encoder IC1 (HT12E). We have used '1111' as the permanent address for communication. Pins 1 through 8 of IC4 are address pins and therefore 256 possible addresses are available. The address on the encoder and decoder ICs must match for the data to be valid.

The HT12D decoder receives serial addresses and data from the encoder that are transmitted by a carrier signal over RF medium. The decoder compares the serial input data three times continuously with its local addresses. If no error or unmatched codes are found, the input data codes are decoded and transferred to the output pins. VT pin (valid transmission) goes high to indicate a valid transmission. The HT12D provides four latch-type data pins whose data remains unchanged until new data is received.

Data pins D8 through D11 (pins 10 through 13) of the decoder send 4-bit data to CD4519 multiplexer IC5.

CD4519 multiplexer. This IC provides four multiplexing circuits with common select inputs (S_A and S_B); each circuit contains two inputs (An, Bn) and one output (On). It may be used to select 4-bit information from one of the two sources.

There are eight input lines (A0 through A3 and B0 through B3), of which four (A0 through A3) are permanently connected to Vcc through resistor R19, while the rest four (B0 through B3) are connected to the data output lines of the decoder (IC4).

The select inputs can be connected to either Vcc or VT pin (pin 17) for latch or momentary mode-selection section. Jumper switch (JS) is used to select between latch and momentary operation. When latch mode is selected,

data present at the output pins is latched, i.e., they remain the same and the respective relay energises until the next change is made in the mode selection. When momentary mode is selected, data present at the output pins is available as long as VT pin remains active-high. As soon as VT pin becomes active-low, the respective relay de-energises.

The latched output data from multiplexer CD4519 is fed to relay driver IC ULN2003, to control up to four devices through the relays (RL1 through RL4). VT pin is connected to LED4 through IC6 to indicate the status of VT signal when it is active-high.

Software program

The software flowchart programmed in the microcontroller of the transmitter section is shown in Fig. 6. It is written in Assembly language and compiled using ASM51 software to generate the hex code. The hex program can be burnt into the AT89C51 microcontroller by using any standard programmer available in the market. We have used TopView programmer from Frontline Electronics to program the microcontroller.

The software program is designed to accept the input from the user as well as control the devices. It identifies the key pressed and displays the key code on the LCD module.

In the program, the LCD module is initialised first. As soon as the time-out is set, all the four devices turn on initially, then a particular device turns off at preset time. In this project, the time-out range is 00 to 99 seconds, which can be easily modified to extend the time duration

PARTS LIST

Semiconductors:

IC1	- HT12E encoder
IC2	- AT89C51 microcontroller
IC3	- 7805 5V regulator
IC4	- HT12D/SM5172 decoder
IC5	- CD4519 multiplexer
IC6	- ULN2003 relay driver
TRX-434	- 434MHz RF transmitter module
RX-434	- 434MHz RF receiver module
LED1-LED8	- 5mm light-emitting diode
D1-D10	- 1N4148 signal diode

Resistors (all ¼-watt, ±5% carbon):

R1, R2, R17, R20-R24	- 1-kilo-ohm
R3	- 680-kilo-ohm
R4-R11 (RNW1)	- 10-kilo-ohm network resistor
R12-R15	- 4.7-kilo-ohm
R16	- 1.2-kilo-ohm
R18, R19	- 47-kilo-ohm
VR1	- 47-kilo-ohm preset

Capacitors:

C1	- 10µF, 16V electrolytic
C2, C3	- 27pF ceramic
C4	- 1000µF, 25V electrolytic
C5	- 0.1µF ceramic
C6	- 470nF ceramic

Miscellaneous:

LCD1	- 16×2 line LCD display
S1-S5	- Tactile switches
X_{TAL1}	- 11.0592MHz crystal
JS	- Jumper switch
	- Antenna
	- 4×2-pin berg strip male and female connectors
	- 12V DC supply
	- 5V DC supply

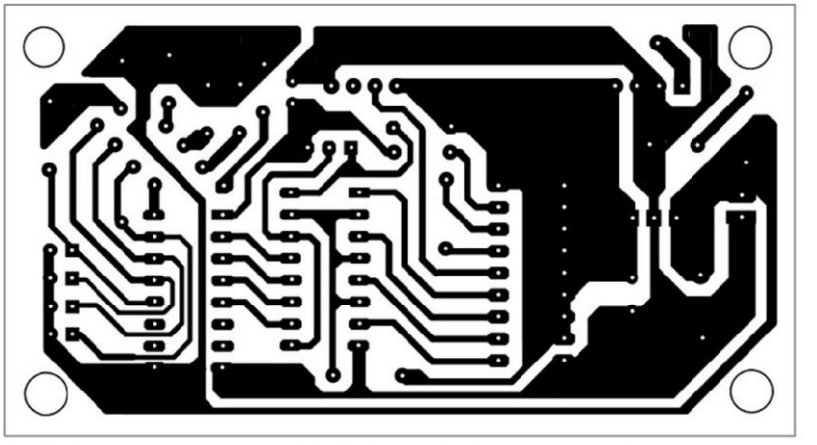


Fig. 9: An actual-size, single-side PCB layout of receiver circuit for wireless equipment control

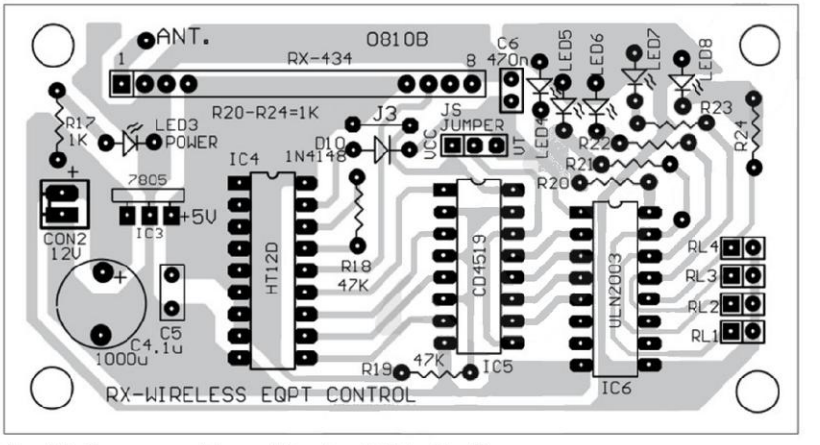


Fig. 10: Component layout for the PCB in Fig. 9

in the delay subroutine of Assembly code.

Port 0 is configured as output port and interfaced with the RF module through encoder IC1. Port 1 is used for LCD interface and port 2 is used for the input from push-to-on switches.

Circuit operation

When the system is switched on, the startup message "press any key" appears on the LCD screen. When any key is pressed by the user, the LCD displays the message "to set time out press ent!". Pressing 'ent' key displays the following messages on the LCD with a cursor blinking near the first device 'D1_T':

D1_T= D2_T=
D3_T= D4_T=

Use 'up' and 'down' keys to set the time for controlling the devices. The set time for each device on the LCD screen looks like this:

D1_T=10 D2_T=20
D3_T=30 D4_T=40

Now press 'ent' key followed by 'run' key. A device control subroutine executes and sends the data to the RF module, which transmits the data through ANT antenna. You can set maximum of 99 seconds as the control time for the device. If you set it to 00, a particular device is turned on for infinite time.

Construction

An actual-size, single-side PCB layout of the transmitter for wireless equipment control using microcontroller is shown in Fig. 7 and its component layout in Fig. 8. The actual-size, single-side PCB layout for the receiver circuit is shown in Fig. 9 and its component layout in Fig. 10.

Download source code: http://www.efymag.com/admin/issuepdf/Wireless%20Equipment%20Control.zip

Industrial Applications

TRIGGERING CIRCUIT FOR SCR PHASE CONTROL

■ **SUKESH RAO M.**

Controllable triggering circuits are often needed to control the output voltage of SCR-/triac-based converters. An SCR-/triac-based converter can handle DC voltages as high as 300V, which can be obtained by direct rectification of the mains AC voltage. The output voltage of such a converter can be controlled by controlling the phase angle of conduction by adjustable delay of the firing/triggering voltage to the SCRs during each half cycle of the input mains supply.

In the present circuit, the firing angle of SCRs is manually controllable through two pushbutton switches, however the same could be changed to programmable control by making use of the feedback obtained by sampling the output voltage across the load. This would need addition of a sampling-cum-feedback circuit and corresponding changes in the software.

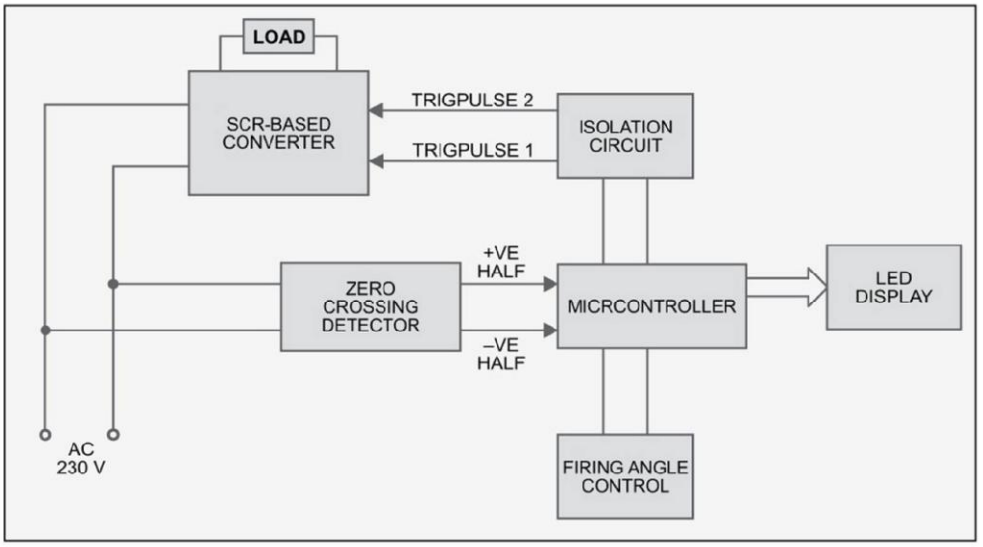


Fig. 1: Functional block diagram of the microcontroller-based single-phase SCR-based trigger controller

The principle

The phase control of the firing angle is referenced to zero-crossing point of each half of the input 50Hz AC mains waveform. A pulse is generated at zero-crossing instant of each half cycle. The duration of half cycle for 50Hz mains is 10 ms (corresponding to 180° traverse time). By delaying the triggering/firing instant of each of the two SCRs by a maximum duration of 10 ms with respect to the zero-crossing pulses, we can control the output of the converter as desired.

Circuit description

Fig. 1 shows the functional block diagram of the microcontroller-based single-phase SCR-based trigger controller. Single-phase AC mains supply is connected to the SCR-based converter and zero-crossing detector blocks. The output pulses corresponding to zero-crossing points of both the positive and negative half cycles of mains 50Hz supply from the zero-crossing detector block are fed to the microcontroller. The delayed triggers from the microcontroller after isolation via the isolation block are provided to the converter block for phase control of SCRs. The

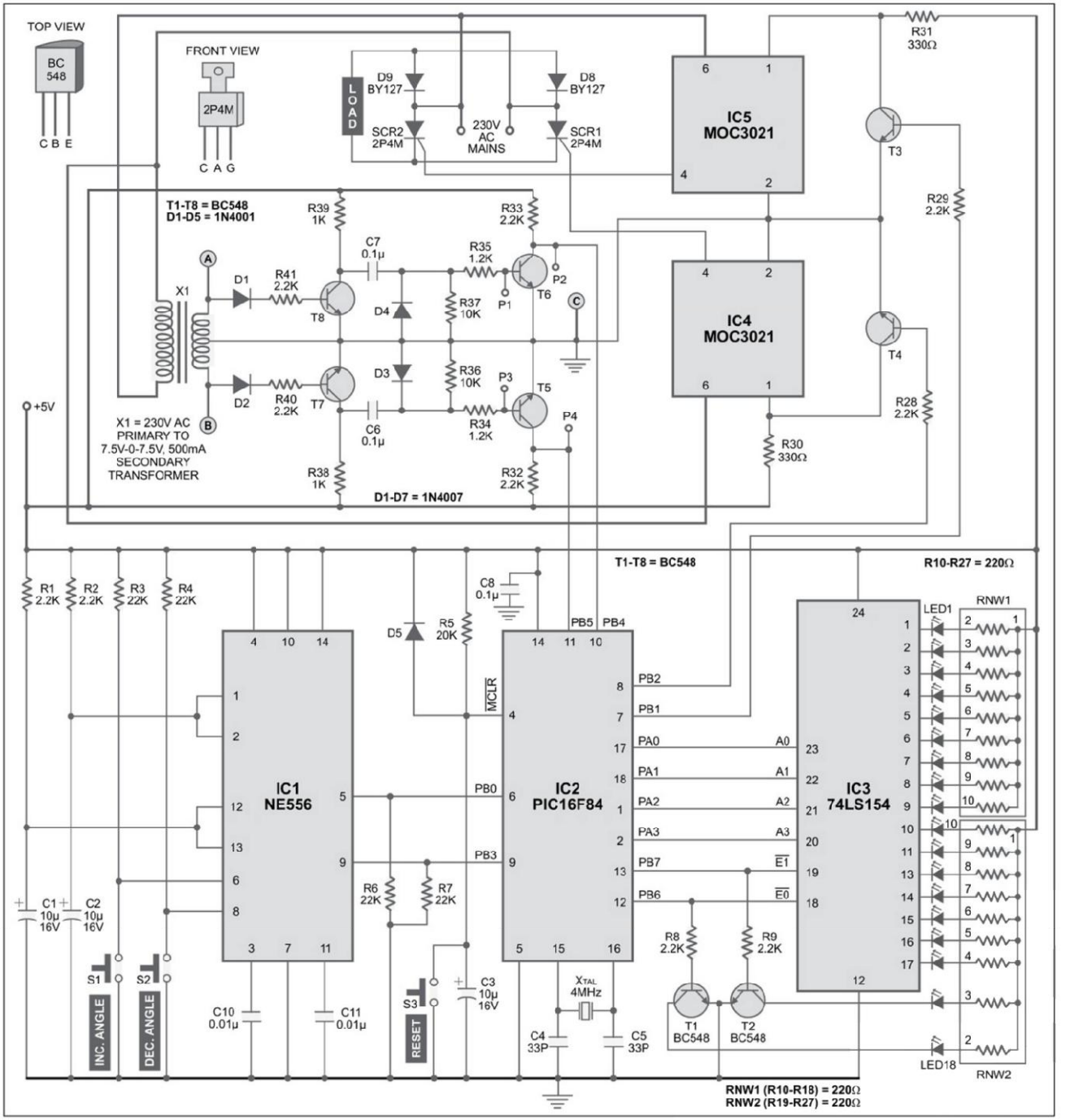


Fig. 2: Circuit of the microcontroller-based single-phase SCR-based trigger controller

output of the converter block is fed to the load. The firing-angle-control block increases or decreases the delay in generation of the triggering levels with respect to a preset (middle) position.

The microcontroller additionally provides the information relating to trigger delay for display purposes. The complete schematic of the microcontroller-based triggering circuit for SCR phase control is shown in Fig. 2. The triggering waveforms for SCR1 and SCR2 are shown in Fig. 5.

The complete circuit can be divided into zero-crossing detector, microcon-troller, trigger isolation, AC-to-DC converter and power supply sections.

Zero-crossing detector section. This section comprises step-down transformer X1, diodes D1 through D4,

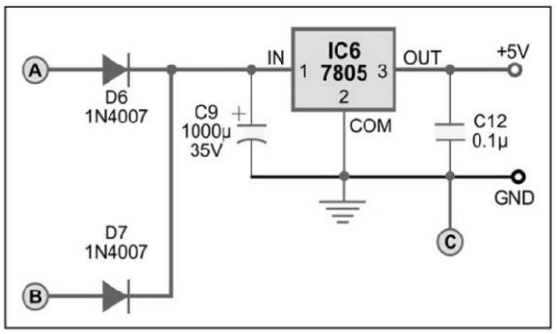


Fig. 3: Power supply circuit

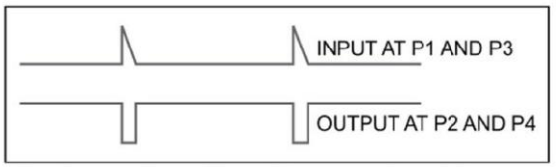


Fig. 4: Input at P1 and P3 and output at P2 and P4

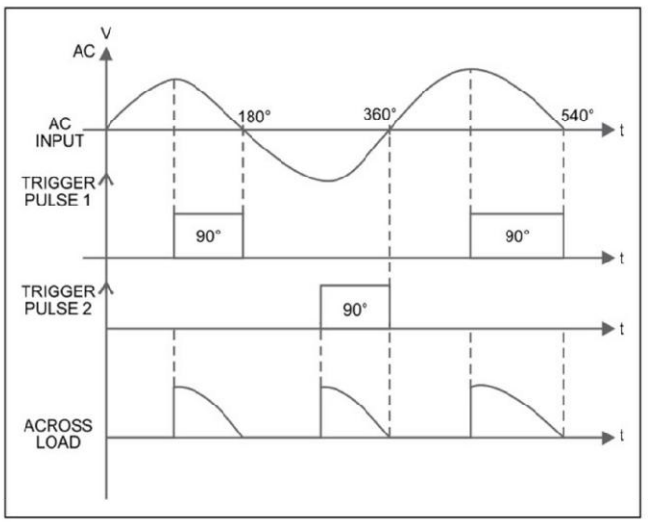


Fig. 5: Triggering waveforms for SCR1 and SCR2

transistors T5 through T8 and a few other passive components. While top half of this circuit detects the zero-crossing point of one half of the input signal, the bottom half section detects the zero-crossing point of the other half cycle.

When the top side of X1 secondary goes positive, transistor T8 conducts and its collector voltage falls. Capacitor C7 charges up towards this low voltage almost instantly via diode D4 and hence no change is noticed at the base/collector of transistor T6.

Towards the end of the half cycle, the collector voltage of transistor T8 rises towards the positive supply rail. Now diode D4 acts as almost open and the capacitor charges via resistor R37. Thus C7-R37 combination acts as a differentiating network to produce about 1ms pulse at the collector of T6 towards the end of the half cycle. Similar pulse is produced at the collector of transistor T5 towards the end of the next half cycle.

PARTS LIST

Semiconductors:

IC1	- NE556 dual timer
IC2	- PIC16F84 microcontroller
IC3	- 74LS154 4:16 decoder
IC4, IC5	- MOC3021 opto-isolator
IC6	- 7805 5V regulator
T1-T8	- BC548 npn transistor
SCR1, SCR2	- 2P4M silicon-controlled rectifier
D1- D7	- 1N4007 rectifier diode
D8, D9	- BY127 rectifier diode
LED1-LED18	- 5mm red LED

Resistors (all ¼-watt, ±5% carbon):

R1, R2, R8, R9, R28, R29, R32, R33, R40, R41	- 2.2-kilo-ohm
R3, R4, R6, R7	- 22-kilo-ohm
R5	- 20-kilo-ohm
RNW1 (R10-R18)	
RNW2 (R19-R27)	- 220-ohm
R30, R31	- 330-ohm
R34, R35	- 1.2-kilo-ohm
R36, R37	- 10-kilo-ohm
R38, R39	- 1-kilo-ohm

Capacitors:

C1-C3	- 10µF, 16V electrolytic
C4, C5	- 33pF ceramic disk
C6-C8, C12	- 0.1µF ceramic disk
C9	- 1000µF, 35V electrolytic
C10, C11	- 0.01µF ceramic disk

Miscellaneous:

X_{TAL}	- 4MHz crystal
X1	- 230V AC primary to 7.5V-0-7.5V, 500mA secondary transformer
S1-S3	- Push-to-on switch

The positive-going differentiated pulse at the base of transistors T6 and negative-going square-wave at their collectors are shown in Fig. 4. These zero-crossing pulses are used as reference for generation of delayed triggers by the microcontroller.

Note that the step-down transformer is common to the zero-crossing detector section and the power supply section.

Microcontroller section. The microcontroller used here is PIC16F84 from Microchip. The 4MHz crystal connected across its pins 15 and 16 provides the clock. The pins are used for input/output as follows:

1. PB0, PB3 (input): Used for sensing the output from increase and decrease pushbuttons.
2. PB1, PB2 (output): Used for triggering pulses for SCR1 and SCR2 (via the isolating section).
3. PB4, PB5 (input): From the zero-crossing section.
4. PB6, PB7 (output): Inhibit/enable signal for the 4:16 decoder and enable/inhibit signal for the phase-angle-indicating LEDs (LED17 and LED18).

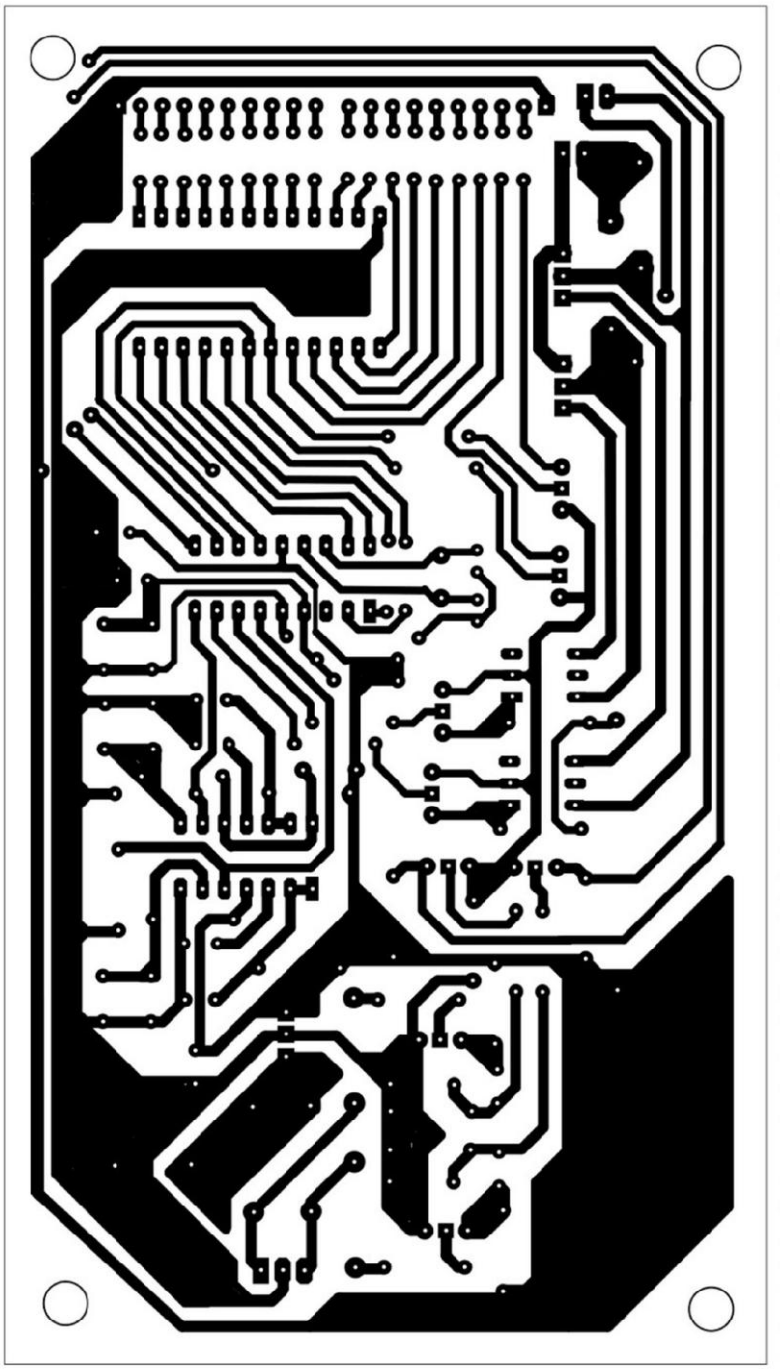


Fig. 6: Actual-size, single-side PCB of the microcontroller-based single-phase SCR-based trigger controller

5. PA0, PA1, PA2, PA3 (all output): Used as inputs for the 4:16 decoder for phase-angle-indicating LEDs (LED1 to LED16).

The microcontroller is programmed to detect the zero-crossing instants of the two halves of the mains input cycles as well as the signals received via pushbuttons labeled as 'Inc Angle' and 'Dec Angle.' After detecting these signals, the microcontroller outputs properly delayed trigger levels to control the triggering/firing angle of the SCRs via the isolation section. These delayed trigger levels will trigger SCR1 during the first half cycle and SCR2 during the second half cycle to provide the desired phase control.

The 'Inc Angle' and 'Dec Angle' buttons make use of NE556 dual timer configured as dual monostable. Each of the two buttons, when depressed momentarily, triggers the respective monostable multivibrator associated with the depressed button to provide a pulse of 1ms duration. The monostables take care of the switch debouncing problems.

The output of one of the monostables is connected to PB0 line of the PIC. Similarly, the output of the other mono is connected to PB3 line of the PIC. If the PIC senses logic '1' at PB0, the 'incfir' function (as shown in the source code of the program) is made to run. Similarly, the 'decfir' function is called when PB1 senses logic '1.' After completing either of these routines, the microcontroller does the job of indicating the firing angle via LEDs connected at the output of 4:16 decoder IC3 as explained in the succeeding paragraph.

A 4-bit BCD value is passed through lines PA0 through PA3, which is decoded using the 4:16 bit decoder. Since the output of the decoder is active-low, the output pins of the decoder are connected to the cathodes of the respective LEDs via current-limiting resistors, while the anodes are all strapped to Vcc (5V) to light up the LED connected to the active output line of the decoder.

As the firing angle difference between the LEDs is ten degrees, we require 18 LEDs to represent firing angle from 0 to 180°. The output of the 4:16 decoder is connected to 16 LEDs and the rest two LEDs are connected to lines PB6 and PB7 of the PIC. When the latter LEDs (LED17 or LED 18) glow, the 4:16 decoder is inhibited (disabled) to avoid glowing of multiple LEDs at the same time for the same angle. Hence E0 and E1 of the 4:16 decoder are controlled by PB6 and PB7 lines. So when any of these two port lines is high, it deactivates the decoder. Transistor is used to augment the signal from PB6 and PB7 lines to light up the additional LEDs (LED17

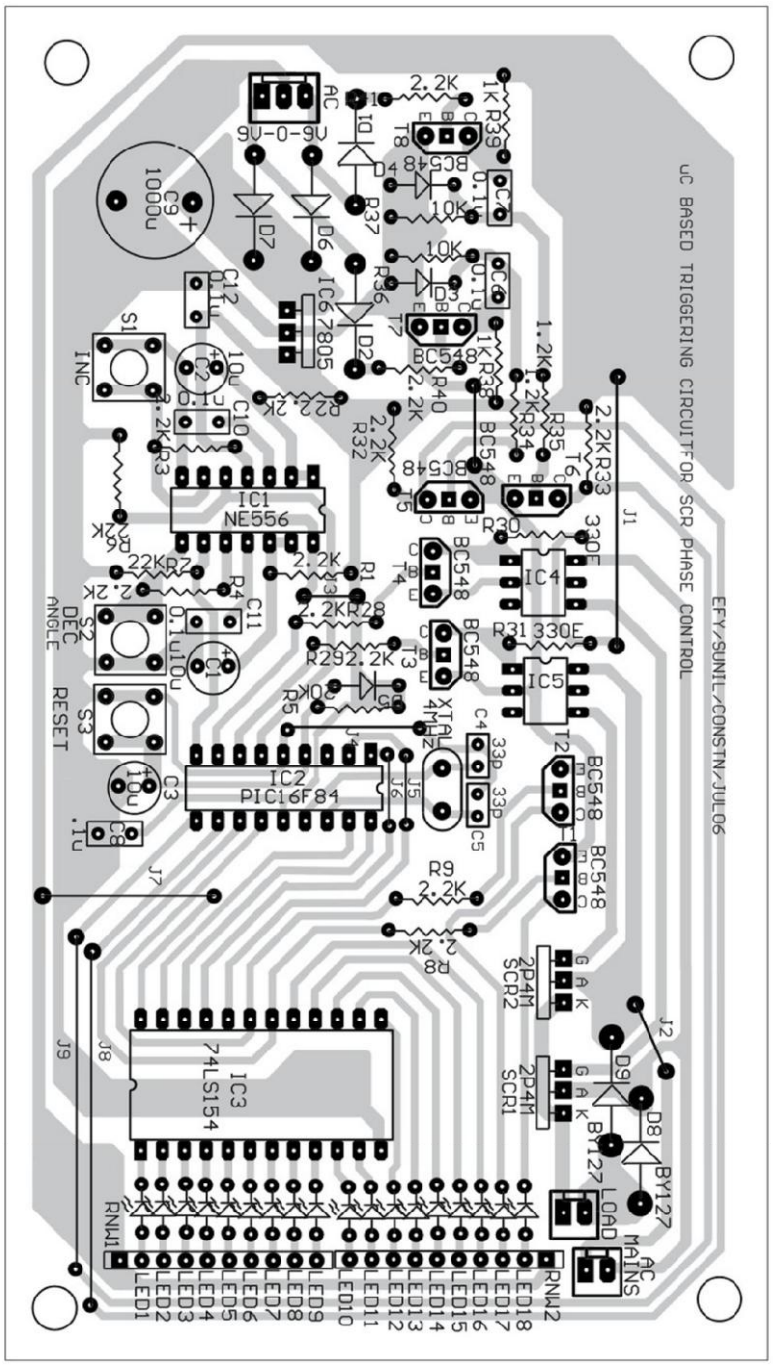


Fig. 7: Component layout for the PCB

and LED18).

Trigger isolation section. The AC-to-DC converter section comprising the SCRs and diodes employs mains 230V AC supply, while the maximum permissible voltage levels for the microcontroller pins are limited to only a few volts. Hence the lines carrying trigger pulses from the microcontroller to the converter section must be isolated to avoid high voltages reaching the microcontroller in the event of SCRs' failure. As a simple and effective isolating device, opto-SCR driver MOC3021 is used here, whose output can directly drive an SCR. Two such isolators are used for the two triggering signals (one for each SCR).

AC-to-DC converter section. This section employs two diodes (BY127) and two SCRs (2P4M) arranged as a rectifier bridge. At any given time (after firing angle is reached), only one SCR and one diode (in cross formation) will conduct to provide the rectified output across the load.

Power supply section. The regulated 5V supply for the circuit is provided by the conventional regulator circuit shown in Fig. 3. As stated earlier, step-down transformer X1 is common for zero-crossing and power supply sections. It steps down AC mains to deliver the secondary output of 7.5V-0-7.5V AC. The transformer output is rectified by the full-wave rectifier, filtered by capacitor C9 and regulated to 5V by regulator 7805. Capacitor C12 bypasses any ripple in regulated output.

A single-side, actual-size PCB layout of the main circuit, including its power supply, is shown in Fig. 6 and its component layout in Fig. 7.

The software

The source code for the program (Firing.asm) is appended at the end of article. The configuration word that determines the device configuration is set to select the low-power (LP) mode for the crystal oscillator, power-up timer is enabled (_PWRTE_ON) to provide a 72ms delay at power-on, code-protect bits are enabled (_CP_ON) and watch-dog timer is disabled (_WDT_OFF) since its application is not required in this project.

Ports are initialised to be input or output ports as indicated in the description of the microcontroller section. All the port lines (except PA4, pin 3) are fully utilised for different purposes as described under the microcontroller section.

The program performs the functions of sensing the zero-crossing instances at PB4 and PB5 lines and delaying

the control signals at PB1 and PB2 for firing of SCR1 and SCR2. It also checks lines PB0 and PB3 for depression for incrementing and decrementing the firing angle and takes action for incrementing or decrementing the stored value by a value that corresponds to 10° step-size. The program displays the current firing angle through lines PA0 through PA3, PB6 and PB7 via 18 LEDs to accommodate 180°.

Download Source Code: http://www.efymag.com/admin/issuepdf/Firing%20circuit.zip

FIRING.ASM

```
Include <p16f84A.INC>
LIST P=pic16f84A ; Processor type PIC16F84A

__CONFIG _LP_OSC & _PWRTE_ON & _CP_ON & _WDT_OFF
cblock 0x10
temp
store
str
stk
endc
org 0x00
goto start
org 0x05
start:
movlw            0x04
movwf store
movlw            0x00
bsf STATUS,RP0
movwf            TRISA   ;ALL PORTA
          LINES ARE
                                                    MADE
OUTPUT
movlw            0x39                          ;PORTB
LINES ARE CONFIGURED AS DESCRIBED
movwf TRISB                                ; IN THE ARICLE
bcf  STATUS,RP0
bcf  PORTB,1
bcf  PORTB,2
bcf  PORTB,6
bcf  PORTB,7
up:  nop
up1:             btfsc   PORTB,4     ;ZERO CROSSING
                          DETECTION OF AC
                          HALF CYCLE

goto up1
bsf  PORTB,2
call delay1             ;FIRING ANGLE DELAY
bcf  PORTB,1     ;SIGNAL TO OPTCOUPLER
up2:             btfsc    PORTB,5   ;ZERO CROSSING DETEC-
TION OF ANOTHER AC HALF CYCLE
goto up2
bsf PORTB,1
call delay1
bcf  PORTB,2
btfsc            PORTB,0 ;CHECKING THE
        LOGIC HIGH FOR
                  INCREMRNTING ANGLE
call             inctime
btfsc            PORTB,3 ;CHECKING THE
                         LOGIC HIGH FOR
                      DECREMRNTING ANGLE
call             dectime
movf             store,0
bcf  PORTB,6
bcf  PORTB,7
btfsc            store,4
call             conv
movwf PORTA
goto             up


conv:               ;CONTROLLING THE FIRING
             ANGLE OF 170 AND 180 DEGREE
bsf  PORTB,6       ;DISABLING THE 4:16 DECODER
             AND INDICATIONG 170th
             DEGREE
btfsc            store,0
goto             bel
return
bel: bcf         PORTB,6
bsf   PORTB,7      ; DISABLING THE 4:16 DECODER
          AND INDICATIONG 180th DEGREE
return


delay1:                                              ;DELAY
PROGRAM FOR FIRING
           ANGLE
movf             store,0
incfsz           store,1
decfsz           store,1  ;TO CHECK
                          WHETHER
                    ANGLE IS ZERO OR NOT.
goto skip1
goto skip
skip1:           movwf                    stk
back:            movlw                    0xbb  ;DELAY OF
0.55mSEC
movwf temp
loop:            decfsz                   temp,1
goto loop
decfsz           stk,1
goto back
skip:            return

inctime:
incf             store,1            ;INCREAMENT FIRING
                 ANGLE BY 10 DEGREE.
movlw            0x13
movwf str
movf             store,0
subwf            str,1
decfsz           str,1
goto next
movlw    0x11       ;KEEPING THE ANGLE SAME
                 WHEN IT REACHES 180 DEGREE
movwf store
next:            return


dectime:
movf   store,0    ;DECREAMENTING THE ANGLE
            BY 10 DEGREE.
movwf   str
incfsz           str,1
decfsz           str,1 ;CONDITION FOR
                       KEEPING ANGLE
                 AT ZERO IF IT STILL
                   DECREASED
goto bb
goto decs
bb: decf  str,0
goto dwn
decs:movlw       0x00
dwn:             movwf                    store
return

end
```

PHASE-ANGLE CONTROL OF SCR USING AT89C51

■ A.M. BHATT

Silicon-controlled rectifiers (SCR) are solidstate semiconductor devices that are usually used in power switching circuits. SCR controls the output signal by switching it 'on' or 'off,' thereby controlling the power to the load in context. The two primary modes of SCR control are phase-angle fired—where a partial waveform is passed every half cycle—and zero-crossing fired—where a portion of the complete wave-

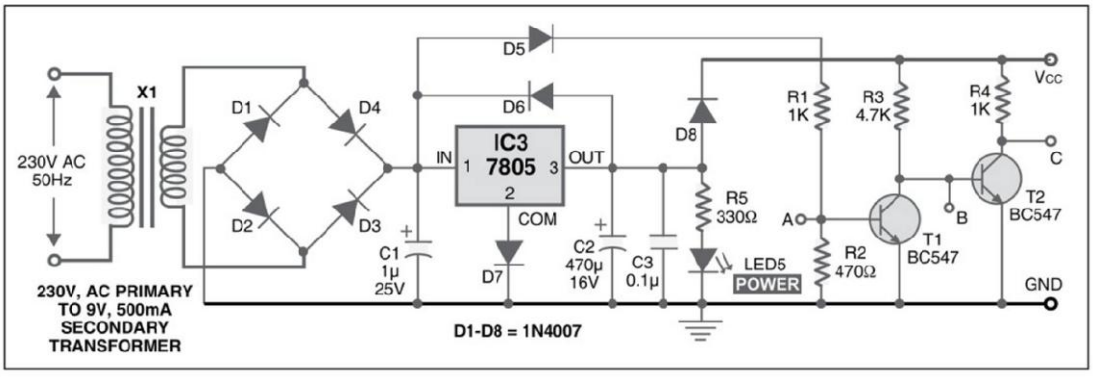


Fig. 1: Power supply and zero-crossing detector circuits

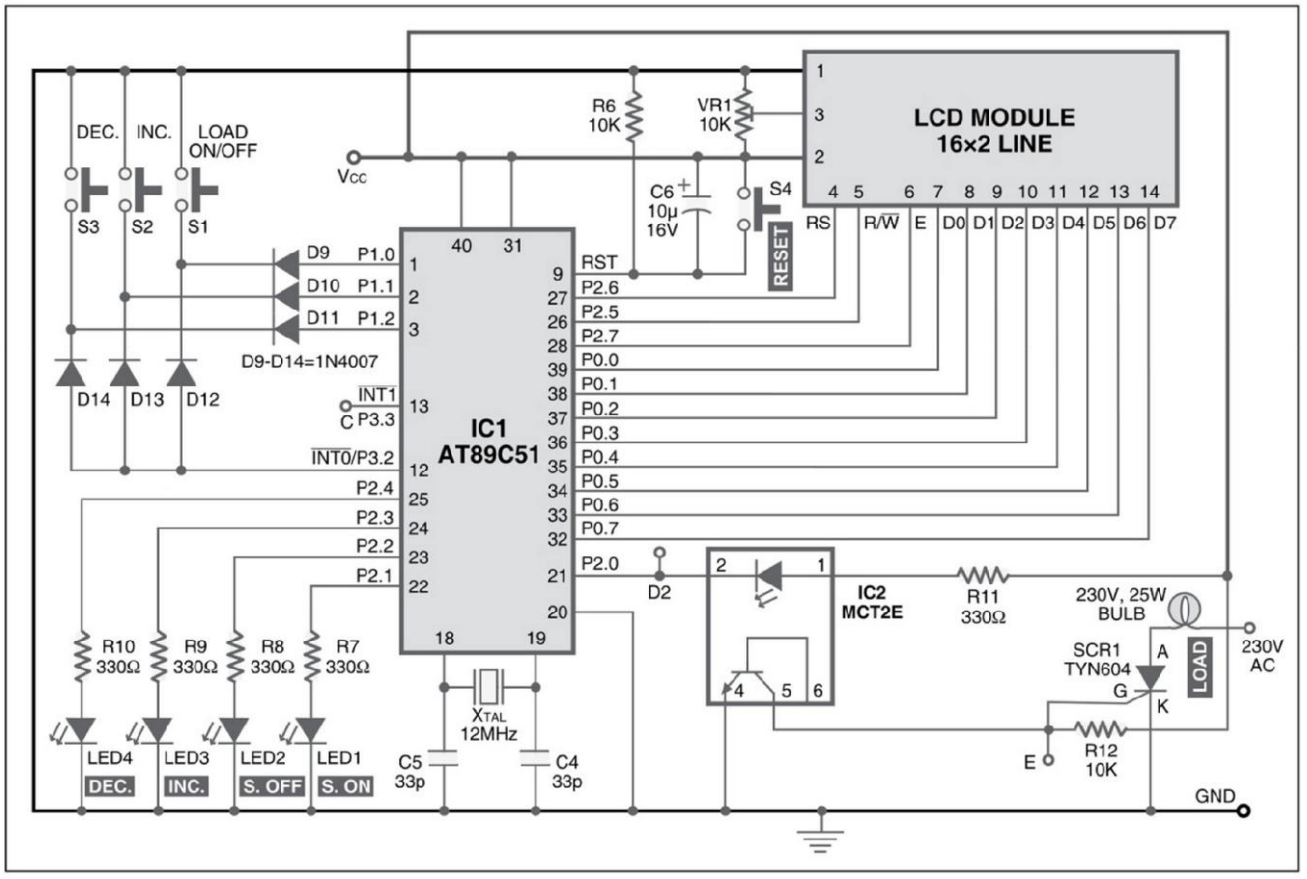


Fig. 2: Circuit diagram of phase angle control of SCR using AT89C51

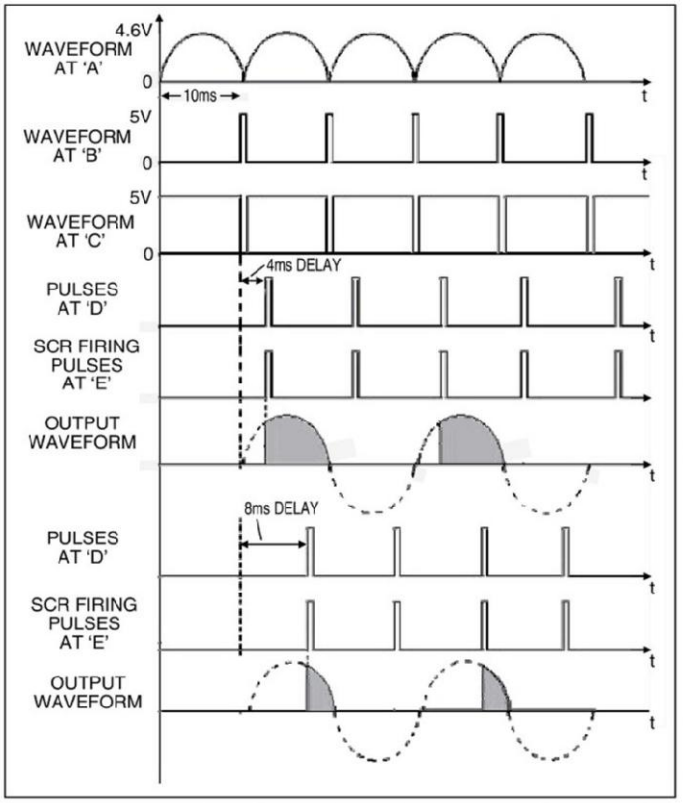


Fig. 3: Waveforms observed at various points in Fig. 1 and Fig. 2 and SCR output waveforms

forms is passed to regulate the power.

In the phase-angle controller, the firing pulse is delayed to turn on the SCR in the middle of every half cycle. This means that every time a part of an AC cycle is cut, the power to the load also gets cut. To deliver more or less power to the load, the phase angle is increased or decreased, thereby controlling the throughput power.

There are several ways to control the firing angle of SCR. This article describes a microcontroller AT89C51-based phase-angle controller. A microcontroller can be programmed to fire SCR over the full range of half cycles—from 0 to 180°—to get a good linear relationship between the phase angle and the delivered output power.

Some of the features of this microcontroller-based phase-angle controller for SCR are:

1. Utilises the zero-crossing detector circuit
2. Controls the phase angle from 0–162°
3. Displays the phase angle on an LCD panel
4. LED indicators are used for displaying the status of SCR
5. Increases or decreases the phase angle with intervals of 18°

Basically, the zero-crossing detector circuit interrupts the microcontroller after every 10 ms. This interrupt commands the microcontroller to generate some delay (in the range of 1 ms to 9 ms). The user can increase or decrease the delay in intervals of 1 ms using switches. The SCR is then fired through the opto-coupler. This repeats after every 10 ms.

TABLE I
Role of Different Switches

Switch	Function
S1	To switch SCR on/off
S2	Increase delay by 1 sec
S3	Decrease delay by 1 sec

Circuit description

The complete circuit is divided into two sections:

1. The zero-cross detector section
2. The control section

The zero-cross detector section. Fig. 1 shows the circuit diagram of the zero-crossing detector and the power supply. The main sections of the circuit are a rectifier, regulated power supply and zero-crossing detector. The 230V AC mains is stepped down by transformer X1 to deliver the secondary output of 9V, 500 mA. The transformer output is rectified by a full-wave bridge rectifier comprising diodes D1 through D4 and then regulated by IC 7805 (IC3). Capacitors C2 and C3 are used for bypassing the ripples present in the regulated 5V power supply. A capacitor above 10µF is connected across the output of the regulator IC, while diode D6 protects the regulator IC in case their input is short to ground. LED5 acts as the power-on indicator and resistor R5 limits the current through LED5.

This regulated 5V is also used as biasing voltage for both transistors (T1 and T2) and the control section. A pulsating DC voltage is applied to the base of transistor T1 through diode D5 and resistors R1 and R2. When the pulsating voltage goes to zero, the collector of transistor T1 goes high. This is used for detecting the pulse when the voltage is zero. Finally, the detected pulse from 'C' is fed to the microcontroller of the control section.

The control section. Fig. 2 shows the circuit diagram of the control section for the phase-angle control of SCR. It comprises a microcontroller AT89C51, opto-coupler MCT2E, LCD module and a few discrete components. Port 0 (P0.0 through P0.7) of AT89C51 is used for interfacing data input pins D0 through D7 of the LCD

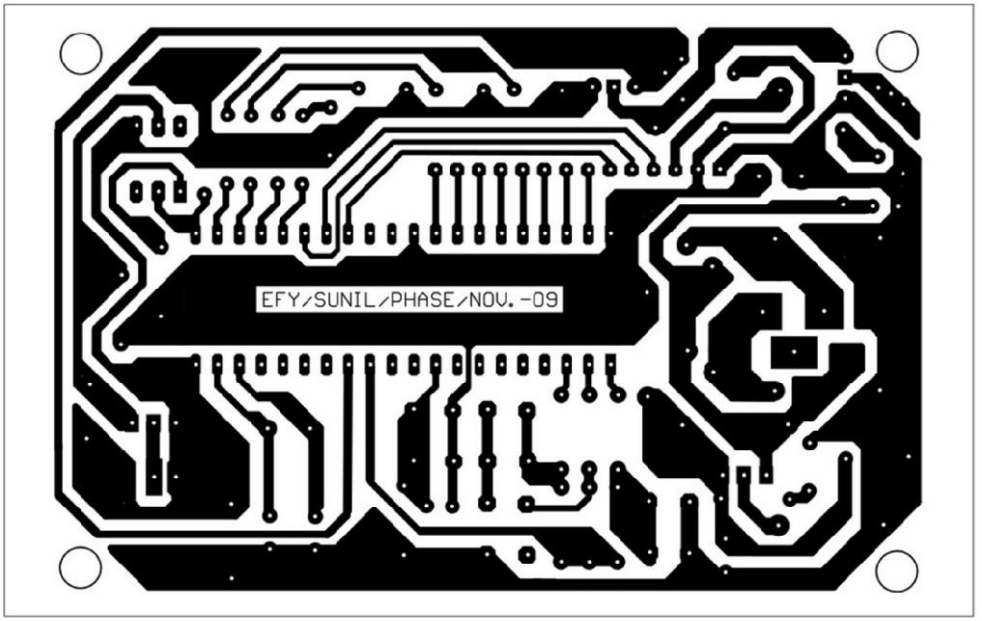


Fig. 4: Actual-size, single-side PCB for the phase angle control of SCR using AT89C51

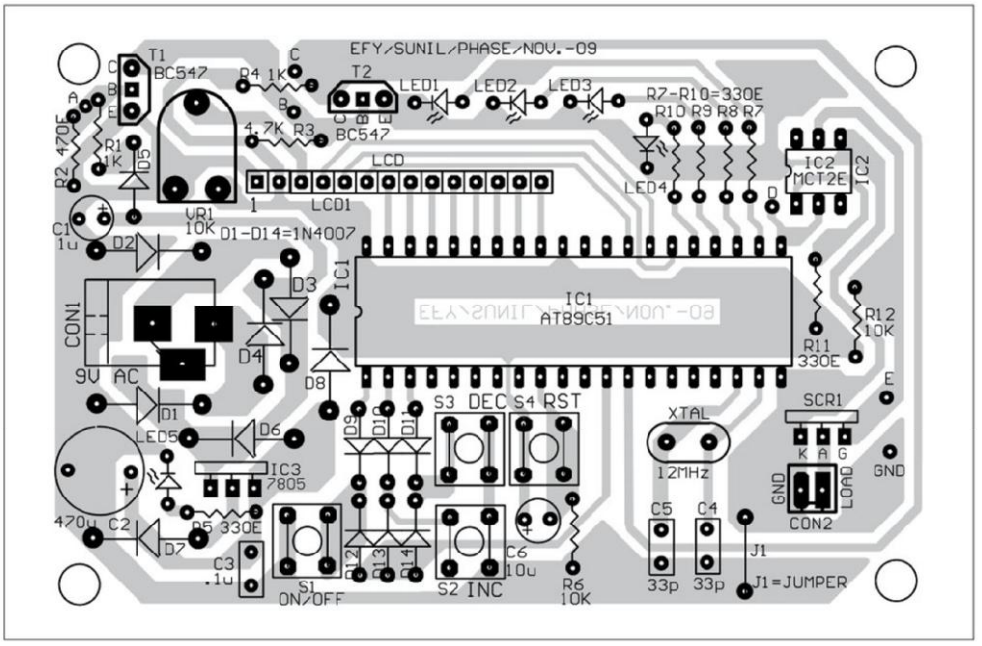


Fig. 5: Component layout for the PCB

module. Port pins P2.6, P2.5 and P2.7 of the microcontroller control the registers select (RS), read/write (R/W) and enable (E) input pin of the LCD module, respectively. Preset VR1 is used for controlling the contrast of the LCD module. Push-to-on switches S1, S2 and S3 are connected with the pins P1.0, P1.1 and P1.2 through diodes D9, D10 and D11, respectively. External interrupt pin $\overline{\text{INT0}}$ (P3.2) of the microcontroller is connected to S1, S2 and S3 through D12, D13 and D14, respectively. The role of different switches is shown in Table I.

The output of the zero-crossing detector from 'C' is fed to the external interrupt pin $\overline{\text{INT1}}$ (P3.3) of the microcontroller.

Port pin P2.0 is connected with pin 2 of the opto-coupler (MCT2E). The output pin 5 of MCT2E is used for triggering the gate of SCR TYN604. The anode of SCR is connected to the load (bulb) with the 230V AC supply.

A 12MHz crystal along with capacitors C5 and C4 are connected to the microcontroller pins 18 and 19 to provide the basic clock to the microcontroller. Power on reset is derived by using capacitor C6 and resistor R6. Switch S4 is used for a manual reset.

The operation

The complete operation can be well understood with the help of waveforms in Fig. 3.

1. The waveform at point 'A' is a fully rectified wave that is fed to the base of T1.
2. When the base voltage falls below 0.7V, transistor T1 is switched off, pulling the output higher. This

PARTS LIST

Semiconductors:

IC1	- AT89C51 microcontroller
IC2	- MCT2E opto-coupler
IC3	- 7805, 5V regulator
T1, T2	- BC547 npn transistor
SCR1	- TYN604 SCR
D1-D14	- 1N4007 rectifier diode
LED1-LED5	- 5mm LED

Resistors (all ¼-watt, ±5% carbon):

R1, R4	- 1-kilo-ohm
R2	- 470-ohm
R3	- 4.7-kilo-ohm
R5, R7-R11	- 330-ohm
R6, R12	- 10-kilo-ohm
VR1	- 10-kilo-ohm preset

Capacitors:

C1	- 1µF, 25V electrolytic
C2	- 470µF, 16V electrolytic
C3	- 0.1µF ceramic disk
C4, C5	- 33pF ceramic disk
C6	- 10µF, 16V electrolytic

Miscellaneous:

X1	- 230V AC primary to 9V, 500mA secondary transformer
S1-S4	- push-to-on switch
X_{TAL}	- 12MHz crystal
LCD Module	- 16×2 line LCD

TABLE II

Change in Delay Using Push-to-on Switches

LED	Indication
LED1	SCR on
LED2	SCR off
LED3	Blinks when delay is increased
LED4	Blinks when delay is decreased

results in a very short positive pulse, which is available at the collector, (at point 'B') as shown in the second waveform.

3. As this positive pulse is inverted by transistor T2, it produces one negative pulse of the same width at 'C.' This is shown as the third waveform.

4. This negative pulse is fed to the interrupt pin ($\overline{INT1}$) of the microcontroller, which acts as an interrupt for the microcontroller. The microcontroller then generates a positive pulse on P2.0 (at point 'D') after some delay. This turns 'off' the internal LED of the opto-coupler (MCT2E) and a positive pulse is produced at output 'E'. This is used for triggering (fire) SCR1.

5. Depending on the time delay in between the interrupt and the pulse on port pin P2.0 of the microcontroller, the SCR is fired in the middle of the half-wave cycle.

6. Two different waveforms—one for 4 ms delay and the other for 8 ms delay—are shown in Fig. 3. In the case of 4ms delay, the output positive cycle of the AC wave is 60 per cent of the input. Therefore, nearly 60 per cent of the power is delivered to the load (the dotted line shows part of waveform that has been cut). In the second case of 8 ms delay, the output cycle is 20 per cent of the input cycle, so only 20 per cent of the power is delivered to the load.

This change in delay is done using switches S1 and S2. Different LEDs are used for indicating different functions as shown in Table II.

The diodes D12 through D14 are connected in such a manner that whenever any of the three push-to-on switches are pressed, it generates an external interrupt $\overline{INT0}$.

When switch S1 is pressed for the first time, it enables external interrupt $\overline{INT1}$ and displays the message 'SCR on.' So after every 10 ms, external interrupt $\overline{INT1}$ is generated which starts the entire operation. Pressing switch S1 again disables external interrupt 0 and the message 'SCR off' is displayed. The complete SCR operation gets shut off.

On pressing S2, the delay increases by 1 ms (firing angle will shift by 18°) and firing of SCR is delayed by 1 ms. The power delivered to the load is also decreased by 10 per cent. The maximum delay that can be applied is 9 ms which will delay firing by an angle of 162°. When the limit is reached, it is indicated by LED3 and a message 'Max. phase angle' is displayed on the LCD. The glowing of the bulb goes off.

Similarly, when S3 is pressed, the delay is decreased by 1 ms and the load current increases by 10 per cent. The minimum delay is 0 ms, which means a full positive cycle is applied. However, when the limit is reached, it is indicated by LED4 and a message 'Min. phase angle' is displayed.

An actual-size, single-side PCB for phase-angle control using SCR is shown in Fig. 4 and its component layout in Fig. 5.

Software program

The software code for this project is written in 'C' programming language and compiled using the Keil µVision3 compiler. After compilation, the final .hex code is downloaded to the microcontroller using a suitable programmer. The source program is well commented and easy to understand.

The main function initialises the timer, ports and LCD. Finally, after enabling the external interrupt 0, it enters

into a continuous loop.

Int0 function is an interrupt function and is automatically called when any of the three switches S1 through S3 is pressed.

1. If switch S1 is pressed, it checks if it is pressed for an even/odd number of times. Accordingly, it either switches 'on' or switches 'off' the SCR. Basically, it enables/disables external interrupt 1. The state of the SCR is displayed by a message on the LCD and an indication comes on LED1 and LED2 also.

2. If switch S2 is pressed, the delay is increased by 1 ms and the angle is increased by 18°. The light intensity of the bulb also increases. If the limit is reached, the message is displayed on the LCD.

3. For switch S3, the operation remains the same as with S2, but the delay is decreased by 1 ms and the angle is decreased by 18°.

Int1 function is also an interrupt function and is automatically called when the zero-crossing detector gives the pulse after every 10 ms. It feeds one pulse to the gate of the SCR after the desired delay (set by switch S2 and S3). The pulse applied is indicated on LED1.

writecmd function sends the command byte to the LCD. It takes one argument byte and sends it to port P1

writedata function sends data bytes to be displayed on the LCD. It also takes one argument byte and sends it to port P1.

writestr function writes a whole string (message) on the LCD. It takes the pointer as an argument that points the address of the first character of the string. Then through the pointer, it sends all the characters, one by one, to port P0.

busy function checks the status of the busy flag of the LCD. If the flag is set, it means the LCD is not ready and the programs remain within the loop. When the flag is reset, it means the LCD is ready and the program comes out of the loop.

keydly function, used for key debouncing, is the fix delay by approximately 100 ms.

delay function is a variable delay generated by timer 0. The basic delay is of 1 ms, which is rotated in the loop from 1 to 9 times to generate a minimum of 1 ms and a maximum of 9 ms delay.

display function separates each digit of the angle and converts them into an equivalent ASCII number, before sending it to the LCD, one by one, for display.

Download Source Code: http://www.efymag.com/admin/issuepdf/Phase%20Angle%20Control%20Using%20AT89C51.zip

SCR.C

```
#include<reg51.h>
#include <string.h>
sbit rs = P2^6;                                 // rs pin of LCD
sbit en = P2^7;                                 // en pin of LCD
sbit rw = P2^5;                                 // rw pin of LCD
sbit b = P0^7;                                  // busy flag
sbit led1 = P2^1;
sbit led2 = P2^2;
sbit led3 = P2^3;
sbit led4 = P2^4;
sbit pulse = P2^0;

unsigned int d1=0;
unsigned int d2=0;
unsigned int c=0;

void writecmd(unsigned char a);      // function initializations
void writedata(unsigned char b);
void busy(void);
void writestr(unsigned char *s);
void dely(void);
void incangle(void);
void decangle(void);
void delay(int d);
void display(unsigned int z);

void keydly(void)                               //key debounce delay
  {
            unsigned int x,y;
            for(x=0;x<100;x++)
                for(y=0;y<1000;y++);
  }
void decangle()                         //decrease delay by 1ms
  {
            EX1=0;                      //first disable all interrupts
            led3=1;                                 // indication on LED
            writecmd(0xC0);             // set next line of LCD
            if(d1>0)
              {
            d1--;                       // decrease delay
            d2-=18;      // decrease display angle by 18
            writestr("angle:");         // show it on LCD
                        display(d2);
                        writestr(" deg.  ");
              }                         // display message
            else if(d1==0) writestr("Min phase angle ");
// if max limit is reached
            led3=0;                     // indication OFF
             EX1=1;      // enable interrupts before leaving
  }
void incangle()          // increase delay by 1 ms
  {
            EX1=0;      // all other things remains same as above
            writecmd(0xC0);
            led4=1;
             if(d1<9)
              {
                        d1++;
```

```
                        d2+=18;
                        writestr("angle:");
                        display(d2);
                        writestr(" deg.  ");
                    }
            else if(d1==9) writestr("Max phase angle ");
            led4=0;
            EX1=1;
  }
void delay(int d)
// generates delay from 1 to 9 ms
 {
            int k;
            TL0 = 0x17;             // load timer 0 with 64535 = FC17h
            TH0 = 0xFC;             // so it will overflow after 1000
counts
            TR0 = 1;                // start timer
            for(k=0;k<d;k++)        // count overflows of timer 0 till
              {
// desire delay is required
                        while(TF0==0);
                        TF0 = 0;
                        TL0 = 0x17;
                        TH0 = 0xFC;
              }
            TR0 = 0;
]

void writecmd(unsigned char a)      // send command to LCD
  {
            busy();                 // check busy flag
            rs = 0;                 // select command register
            rw = 0;                 // write enable
            P0 = a;                 // send byte to LCD
            en = 1;                 // apply strobe pulse
            en = 0;
  }
void writedata(unsigned char b)                 // send data to LCD
  {
            busy();                             // check busy flag
            rs = 1;                 // select data register
            rw = 0;                             // write enable
            P0 = b;                             // write enable
            en = 1;                             // send byte to LCD
            en = 0;                                 // apply strobe
pulse
  }
void busy()                         // check busy flag of LCD
  {
            en = 0;                             // disable display
            P0 = 0xFF;                          // P0 as input
            rs = 0;                 // select command register
            rw = 1;
// read enable
            while(b==1)             // if busy bit is 1
            {
             en=0;                  // remain within loop
             en=1;
            }
            en=0;


  }
void writestr(unsigned char *s)     // send string message to LCD
 {
            unsigned char l,i;
            l = strlen(s);          // get length of string
            for(i=0;i<l;i++)

            {
                        writedata(*s);
// till the length of string
                        s++;
// send characters one by one
            }
 }

void display(unsigned int z)        // convert decimal number to ASCII
  {
            int z1,a,ASCII[3];
            if(z>=100)              // if number is greater then 100
              {                     // take 3 digits
                        a=2;
                        while(z>=10)
                        {
                                    z1=z%10;
                                    ASCII[a]=z1+0x30;
// convert all three into ASCII
                                    z=z/10;
                                    a--;
                        }
                        ASCII[0]=z+0x30;
              }
            else                    // otherwise take 2 digits
              {
                        z1=z%10;
                        ASCII[2]=z1+0x30;
// convert them in to ASCII
                        z=z/10;
                        ASCII[1]=z+0x30;
                        ASCII[0]=0x30;
// take first digit as 0
              }
            writedata(ASCII[0]);    // display them one by one
            writedata(ASCII[1]);
            writedata(ASCII[2]);
  }
void int0(void) interrupt 0         // external interrupt 0 subroutine
  {
    switch(P1)
            {
               case 0xFE:           // for first key
                                    keydly();
// after key debounce delay
                                    c++;
// increase counter
                        if((c%2)==1)            // check even or odd
                                      {
                                                writecmd(0x01);
// for odd times
                                                writestr("SCR ON ");
// switch on SCR
                                                writecmd(0xC0);
                                                writestr("angle:");
// display current angle
                        display(d2);
                        writestr(" deg.  ");
                        led2=0;                 // make led2 off
            EX1 = 1;                // enable external interrupt 1
                                      }
                                    else
                                      {
                                                writecmd(0x01);
// for even times
                                                writestr("SCR OFF ");
// SCR is off
                                                led2=1;
// led2 is on
                                    EX1 = 0;    // disable external
interrupt 1
                                      }
                                    break;
                        case 0xFD:              // for second key
                        keydly();   // after key debounce
                        incangle(); // increase phase angle
                        break;
                        case 0xFB:  // for third key
                        keydly();   // after key debounce
                        decangle(); // decrease phase angle
                        break;
              }
   }

 void int1(void) interrupt 2        // external interrupt 1 subroutine
   {
                        int t;
                        led1=1;     // led1 is on
                        delay(d1);  // after desired delay
                        pulse=1;    // send pulse on p2.0
                        for(t=0;t<200;t++);
                        pulse=0;
                        led1 = 0;   // led1 off
   }

void main()
```

```
  {
            TMOD = 0x01;
// initialize timer0 as 16 bit timer
            P2=0x00;     // P2 and P0 as output ports
            P0=0x00;
            P1=0xFF;     // P1 as input port
            writecmd(0x3C); // initialize LCD
            writecmd(0x0E);
            writecmd(0x01);
             writestr("SCR Phase angle");
// display message at the center
            writecmd(0xC4);
            writestr("Control");     // of LCD
            led2=1;                              // scr off led is on
            IE=0x81;                 // enable external interrupt 0
            while(1);                // continuous loop

  }
```

BEVERAGE VENDING MACHINE CONTROLLER

■ VINAY CHADDHA

Beverage vending machines are commonplace at railway stations, airports, fast-food restaurants and even in companies. Press a switch and the machine delivers a hot cup of your favourite drink.

This looks quite a simple operation but has a very complex logic behind it: It involves use of twelve precision timers and four counters apart from physical devices like display, solenoid and motor to deliver water and premixed tea/coffee/soup powder in exact quantity for better taste and in exact sequence.

This has become possible because of the use of microcontrollers, which allow compact size, higher reliability, lower cost and multiple functionalities.

The prototype of beverage vending machine controller developed by the author

This tea/coffee/soup vending machine controller uses Freescale's latest MC908JL16 microcontroller chip. The controller is programmable and user-friendly. You can set the quantity of the beverages through a button switch provided on the front panel of the controller as per your requirements. Thus, cups of any size can be filled at any time.

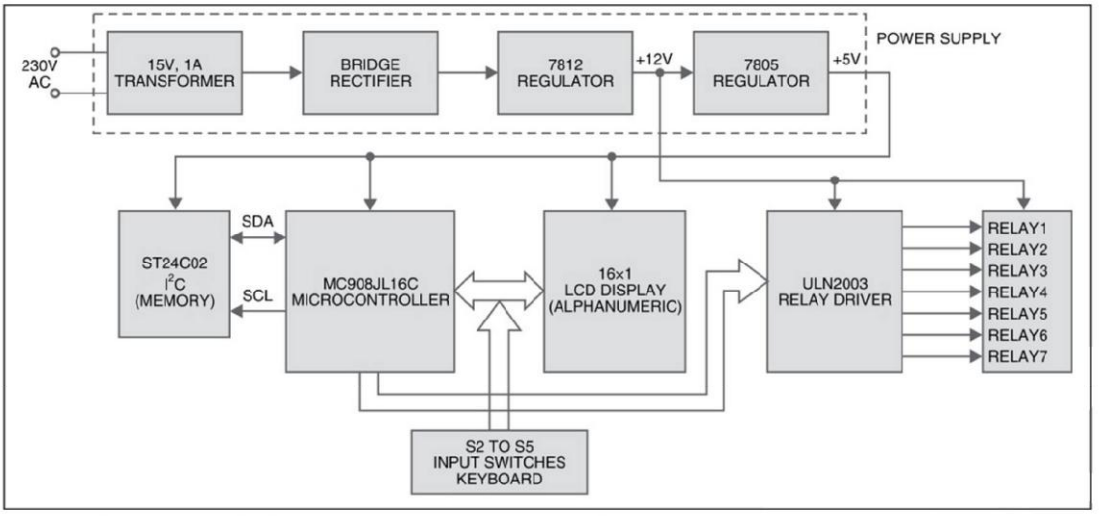


Fig. 1: Block diagram of the beverage vending machine controller

The hardware

Fig. 1 shows the block diagram of the

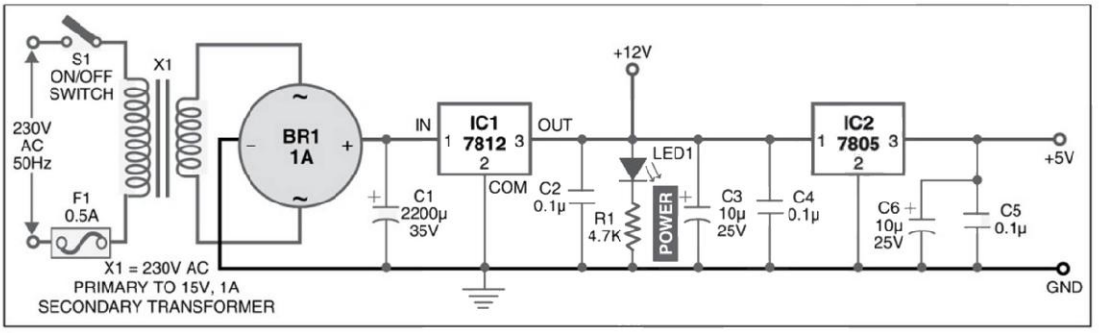


Fig. 2: Power supply circuit

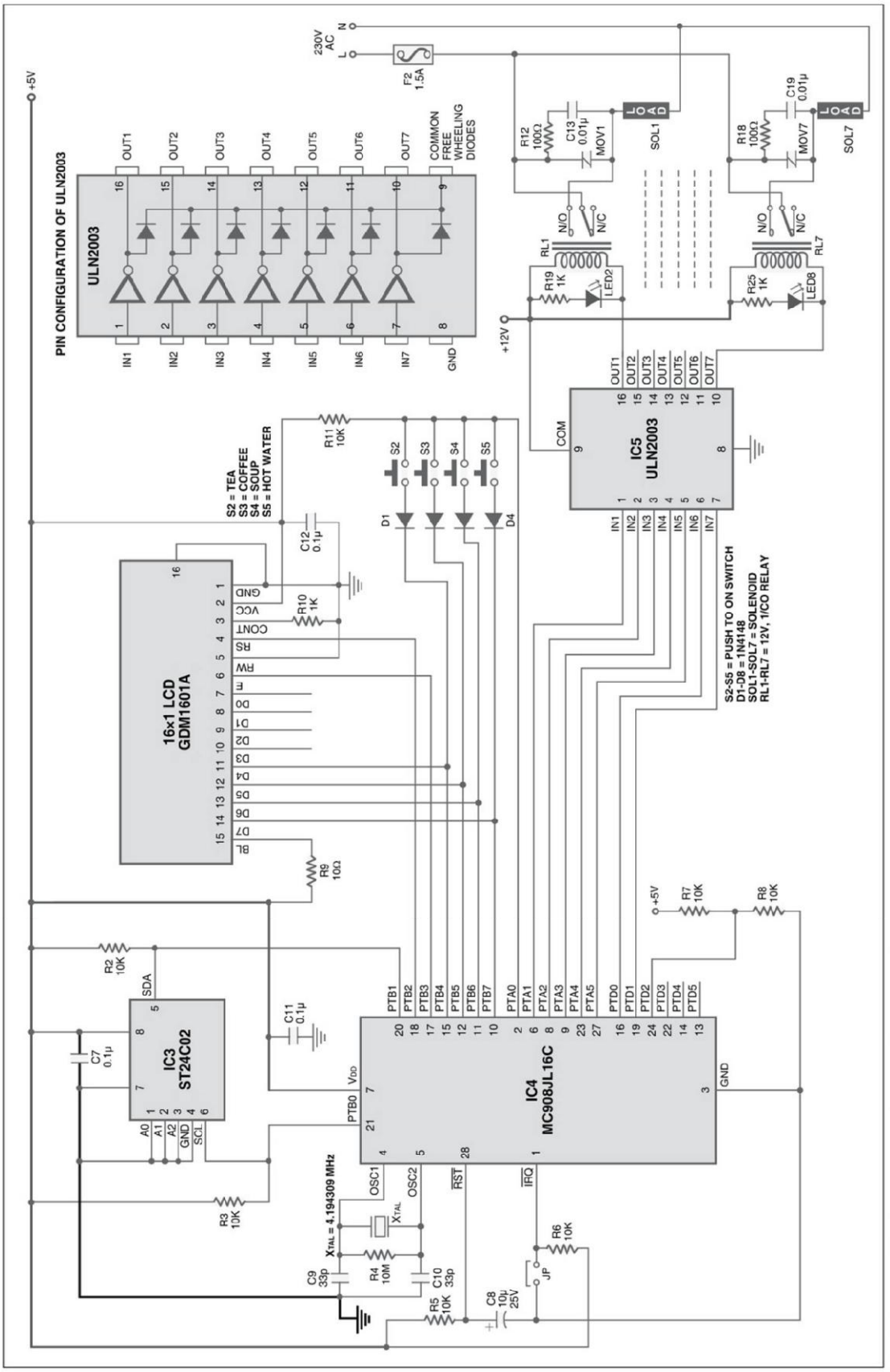
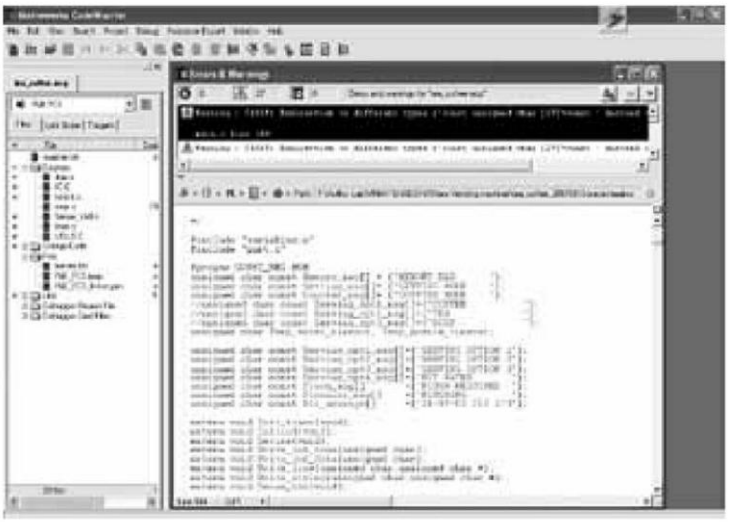


Fig. 3: Circuit of the beverage vending machine controller

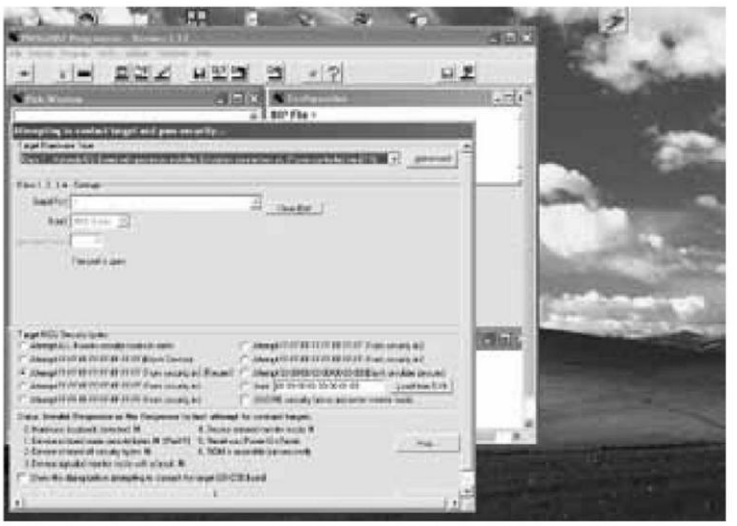
vending machine controller. It comprises the following sections: power supply, microcontroller, relays, relay driver, alphanumeric display, keyboard and memory. The power supply circuit is shown in Fig. 2.

The control unit uses low-cost, readily available components. The temperature control section has not been included in the design as the parts required are expensive and not easily available. However, a low-cost thermostat used in water heaters can be used in the unit.

Power supply. The relays need 12V DC and the microcon-



Screenshot 1: The CodeWarrior 5.2 integrated development environment



Screenshot 2: The PROG08SZ programming environment

PARTS LIST

Semiconductor:	
IC1	- 7812, 12V regulator
IC2	- 7805, 5V regulator
IC3	- 24C02 I²C memory
IC4	- MC908JL16C microcontroller
IC5	- ULN2003 relay driver
D1-D8	- 1N4148 switching diode
LED1-LED8	- 5mm light-emitting diode
BR1	- 1A bridge rectifier
Resistors (all ¼-watt, ±5% carbon):	
R1	- 4.7-kilo-ohm
R2, R3, R5, R6, R7, R8, R11	- 10-kilo-ohm
R4	- 10-mega ohm
R9	- 10-ohm
R10	- 1-kilo-ohm
R12-R18	- 100-ohm
Capacitors:	
C1	- 2200µF, 35V electrolytic
C2, C4, C5, C7, C11, C12	- 0.1µF ceramic
C3, C6, C8	- 10µF, 25V electrolytic
C9, C10	- 33pF ceramic
C13-C19	- 0.01µF ceramic
Miscellaneous:	
X1	- 230V AC primary to 15V, 1A secondary transformer
S1	- On/off toggle switch
S2-S5	- Push-to-'on' switch (4-pin)
RL1-RL7	- 12V, 1/CO relay
X_{TAL}	- 4.1943MHz crystal oscillator
F1	- 0.5A fuse
F2	- 1.5A fuse
SOL1-SOL7	- Solenoid valve
LCD	- 16×1 liquid-crystal display

troller and memory need 5V DC for operation. Bridge rectifier, capacitive filter and regulator ICs 7812 and 7805 are the standard parts used in the power supply.

Microcontroller. We need individually settable timings for the seven relays and multiple time delays between the operation of these relays, i.e., the water inlet relay should operate only when water has to be dispensed. All these functions, though possible using discrete components, are best handled by Freescale's MC908JL16 microcontroller.

Relays. Selection of each beverage requires two solenoid valves, one for premix powder and another for water outlet. The solenoid valves are operated through relays. That is, for three beverages, we need six solenoids and six relays to operate them. The water heater tank is not connected to the source of water as this may increase the power consumption. The water inlet is opened for some time after one cup is filled to allow small quantity of water to be filled again in the tank. This requires one more solenoid. So a total of seven solenoids are required.

To activate these high-current solenoids, sugar-cube type relays are used. These are cheap and used in low-cost UPS for PCs. The relays need around 100mA, 12V supply to operate.

Relay driver ULN2003. The microcontroller cannot drive 12V, 100mA relays directly and needs a buffer. This can be easily achieved using BC547 transistors and a few resistors. However IC ULN2003 has been used to drive

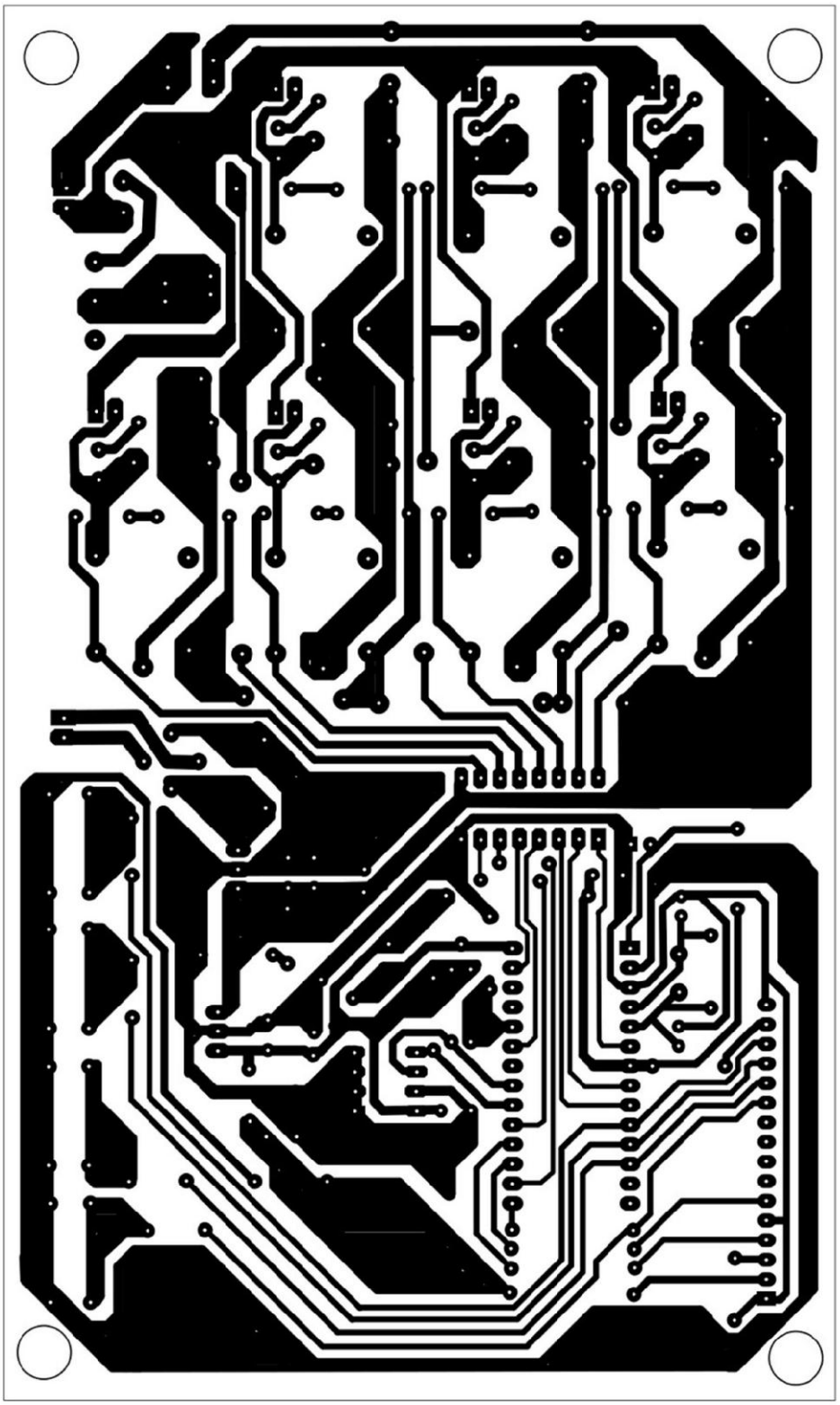


Fig. 4: An actual-size, single-side PCB layout for the beverage vending machine controller

the relays.

Alphanumeric display. The operator needs a visual interface for setting the various parameters and the status of the input switches pressed. LED displays will not be visible in brightly-lit places, so LCDs are the best choice. Character size is not a constraint as the machine is to be operated from a close distance. A standard-size, 16-character, single-line alphanumeric LCD has been used here. It is readily available and commonly used in industrial applications and PCO monitors.

Keyboard. We need at least four keys: two for beverages (one each for tea and coffee), one for half or full cup, and one for hot water. Tactile key switches have been used here. These are low-cost and readily available.

Memory. The control unit doesn't have a battery backup system and will lose the set data when power fails. So a provision has to be made to ensure that the unit recalls the set parameters when power resumes. This is achieved by using a small-size 24C02 memory, which is commonly used in electronic goods like TV sets.

Circuit description

Fig. 3 shows the circuit of the vending machine controller.

Power supply. The power supply circuit comprises a 15V, 1A step-down transformer, filters, 12V DC regulator 7812 (IC1) for relays and 5V DC regulator 7805 (IC2) for microcontrollers (see Fig. 2). Fuse F1 protects against surge current in the event of short circuit.

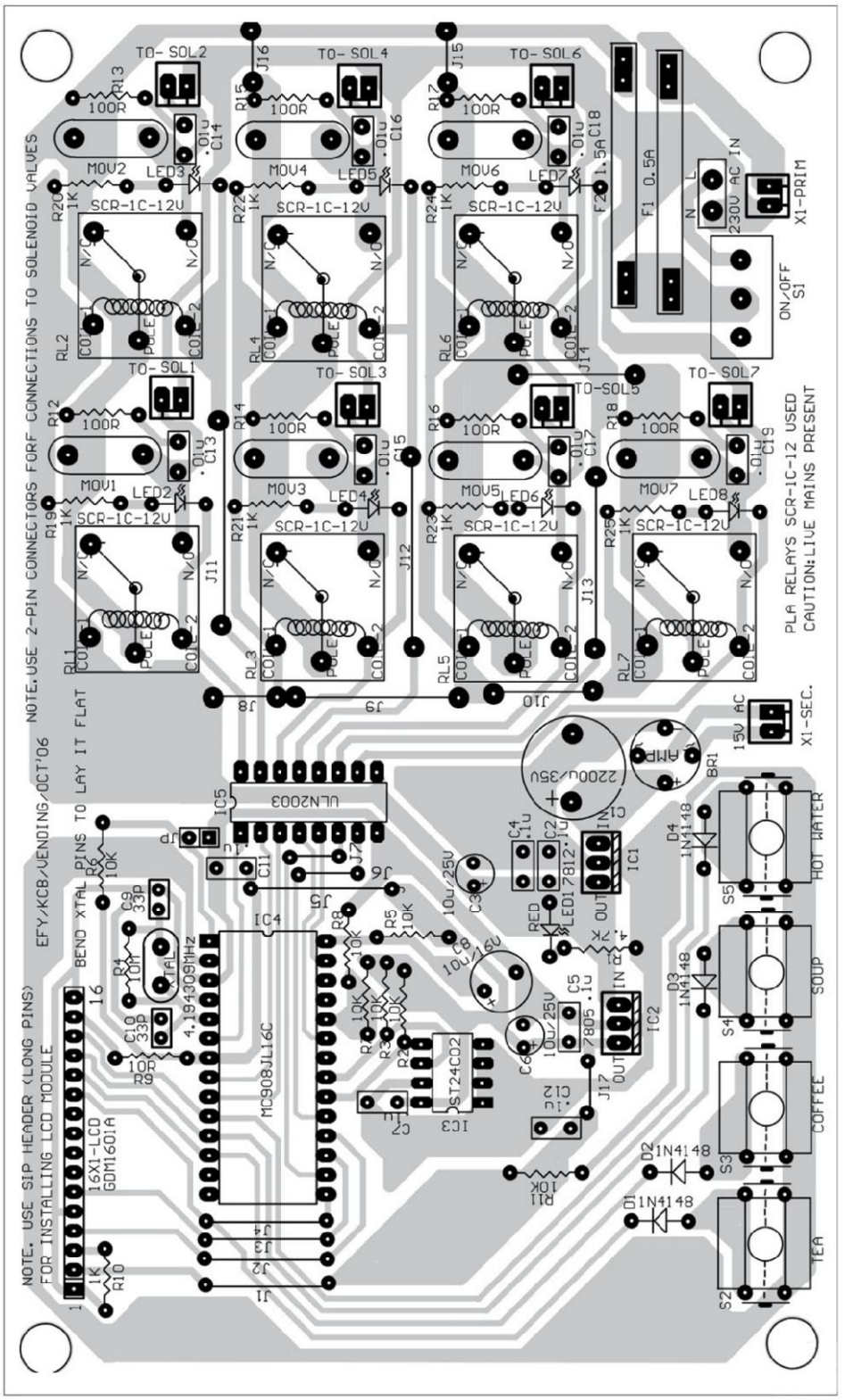


Fig. 5: Component layout for the PCB

Microcontroller. The MC908JL16 microcontroller (also called the MC68HC908JL16) from Freescale Semiconductor, Inc. (formerly Motorola) has 28 pins with 23 general-purpose I/O lines, 16 kB of flash memory to store program and 512 bytes of RAM. It utilises an HC08 CPU core and provides a cost-effective reprogrammable flash memory. It also has two 16-bit timers and other standard functionalities, making it an all-in-one control IC.

The device is a part of the growing JL Family that includes multiple clock options, keyboard interrupts, low-voltage inhibit and a watchdog timer. In particular, the MC908JL16 has a built-in serial communications interface module, master inter-integrated circuit (I^2C) interface and 10-bit analogue-to-digital converter (not used in this program).

The MC908JL16 is a low-cost, high-performance, 8-bit microcontroller unit. It uses the enhanced M68HC08 central processing unit (CPU08) and is available with a variety of modules, memory sizes, package types, etc. Its datasheet is available on Freescale's website 'www.freescale.com.'

Relays. The 12V, 1/CO relays are capable of switching up to 7A, 220V AC loads. Across each relay contact, metal-oxide varistor (MOV) and resistor-capacitor (RC) snubber circuit have been added to reduce noise generated by sparking at relay contacts at the time of switching. Fuse F2 used for protection against surge current can be replaced with another higher-current fuse as per the requirement of

the solenoid used.

Relay driver. The relay driver ULN2003 with seven outputs is capable of sinking 500mA on each output. Inputs are TTL CMOS-compatible and outputs are fed to relay coils directly. With this driver, no free-wheeling diode is required across the relay coils. Datasheets are included in the CD.

Display. The display GDM1601A used here is a 16x1 alphanumeric LCD based on Hitachi HD44780 LCD controller. It is interfaced to the microcontroller using four data lines (D7 through D4) and two control signals (RS and E). Complete details of HD44780 are available on the Internet.

Keyboard. It has four push-to-on switches (S2 through S5) connected to the microcontroller using five lines, of which four are shared with LCD data lines as shown in Fig. 3.

Memory. EEPROM 24CO2 permanently stores information like different timings, options for switches and count of cups filled. It is connected to the microcontroller using two lines, namely, serial data (SDA) and serial clock (SCL). Technical details of I^2C bus and the memory have been covered earlier in EFY. (Refer to articles 'Access Control' in EFY Sept. 2002, 'Set-Top Converter' in EFY June '97, 'Caller ID' in EFY April '99 and 'Remote-Controlled Audio Processor Using Microcontroller' in EFY Sept. '99).

An actual-size, single-side PCB layout for the beverage vending machine controller is shown in Fig. 4 and its component layout in Fig. 5.

The software

The software has been developed in 'C' language using Metrowerks' Code Warrior 'C' compiler. A 16kB free version of the compiler is available on the website 'www.metrowerks.com'

The program in 'C' is written on the MC68HC908JL8, which is equivalent to the MC68HC908JL16, except that the MC68HC908JL8 has only 8kB flash memory and 256 bytes of RAM. A specially developed programmer board along with PROG08SZ software (www.pemicro.com) has been used for programming the microcontroller chip.

Seven program modules have been used for this project: 'disp.c,' 'iic.c,' 'main.c,' 'initlcd.c,' 'sense_kbd.c,' 'timer.c' and 'utils.c.'

The software is developed in two stages: First, basic common input/output (I/O) routines are developed to display information on the LCD, read or write data from a specific memory location, scan the keyboard input (check which key has been pressed) and switch on the relays. Next, logic/program specific for this function is developed. This is more or less a translation job where you convert the end-product requirement in 'C' language so that it performs as expected.

CodeWarrior. The CodeWarrior 5.2 integrated development environment (IDE) includes compilers, linkers, source-code browser, debugger, editor and rapid application development tool set (see Screenshot 1). You can use it to edit, navigate, examine, compile and link code throughout your software development process. You can also configure options for code generation, project navigation and other operations.

If you have this IDE tool, just copy the relevant files from the EFY-CD into your PC. Open the project file 'tea_coffee.mcp.' When the source code is compiled, a '.s19' file is generated in 'bin' folder of the directory where the original source code is located. This file is equivalent to the hex code in other programming tools.

Programmer. The PROG08SZ version 2.12 is a programmer for EEPROM/EPROM modules internal to a Motorola HC08 processor. It communicates to the processor's monitor mode (MON08) via one of P&E Microcomputer Systems' hardware interfaces that is designed to work with the monitor mode. Alternatively, the MON08 circuitry can be built directly into the end user's hardware. The PROG08SZ programming environment is shown in Screenshot 2.

The connection dialogue appears initially. We have used 'class-3 direct serial-to-target W/Mon08 serial port circuitry' option for the programmer board used for programming. Next, select the serial port, tick the box corresponding to the 'ignore security failure and enter monitor mode' and then click 'contact target with this setting...' The 'power cycle dialogue' box appears. Now switch off the programmer board.

Switch on the programmer board again and press 'ok.' Select the device and open '.s19' file under 'File' menu to program into the chip.

Operations of the unit

Each switch performs dual functions as follows:

Switch	Normal mode	Setting mode
S2	Tea	Reset
S3	Coffee	Increment
S4	Soup or half/full	Decrement
S5	Hot water	Setting/next/OK

Setting mode. Normally, this mode is used only once to set various parameters as per the specific requirement, say, the size of the cup. This can be done by following a simple procedure. Switch off the controller unit. Press switch S5 (marked as 'hot water') for the settings mode and switch on the unit. Follow the instructions on the LCD. Press switch S5 again for the desired option. Pressing switches S3 and S4 will increment or decrement the value (in seconds), respectively.

Message Displayed on Pressing Switches S2 through S5

Switch pressed	Message displayed
S2 for dispensing tea	Serving tea
S3 for dispensing coffee	Serving coffee
S4 for dispensing soup	Soup or half/full depending on the setting
S5 for dispensing hot water	Hot water

The program has an option for a half or full cup of the beverage. S4 can be used for either 'soup' or 'half/full' option in the setting mode. When 'OPT3' is selected, S4 will function as a switch for dispensing the soup. When 'half/full' option is selected, S4 will function as a switch for filling up the cup to half or full. For dispensing tea and coffee, two time settings are required, i.e., premix-dispensing time and water-dispensing time. These can be adjusted as per specific requirements, say, quantity and strength (flavour) of the beverage.

Dispensing of tea has an extra function of brewing (optional). In this mode, water is dispensed for a second, then stopped to allow mixing of the powder with water, and dispensed again after some time. This timing can be set through the 'brew time' option.

Common setting for tea/coffee/soup. The time for activation of the water solenoid valves can be set to allow hot water to be dispensed through the common outlet.

After dispensing, some water remaining in the pipes cools off. In the next dispensing, this water is dispensed along with hot water from the water heater. So a provision is made to automatically flush the cold water out from the pipes at regular intervals. The time interval to flush out the cold water can be set by the user. For example, if you set 'Flush To' as '010 m,' i.e., 10 minutes as the time duration for which nobody uses the machine, message "flush required" will be displayed. The user will have to press switch S5 to flush the water out.

Apart from this, there are time settings for delay, refill, etc. The delay time is the time interval between the consecutive dispensing of water and premix powder. Refill time is the time for refilling water into the water heater. Relay RL7 energises through pin 10 of IC5 to refill water in the heater.

After you are done with settings, switch off the power supply for about 30 seconds to allow complete discharge of the filtering capacitor used in the power supply section. Switch on the unit again and it is now ready for use. The display will show "have a nice day."

Download Source Code: http://www.efymag.com/admin/issuepdf/Vending%20Machine.zip

AT89C51-BASED DC MOTOR CONTROLLER

■ A.M. BHATT

Motion control plays a vital role in industrial automation. Manufacturing plants in industries like chemical, pharmaceutical, plastic and textile, all require motion control. And it may be a flat-belt application, flow-control application or mixing of substances. Different types of motors—AC, DC, servo or stepper—are used depending upon the application. Of these, DC motors are widely used because controlling a DC motor is somewhat easier than other kinds of motors.

The motion of a DC motor is controlled using a DC drive. DC drive changes the speed and direction of motion of the motor. Some of the DC drives are just a rectifier with a series resistor that converts standard AC supply into DC and gives it to the motor through a switch and a series resistor to change the speed and direction of rotation of the motor. But many of the DC drives have an inbuilt microcontroller that provides programmable facilities, message display on LCD, precise control and also protection for motors. Using the DC drive you can program the motion of the motor, i.e., how it should rotate.

Here are some of the features of this DC motor controller:

1. Controlled through microcontroller AT89C51
2. Message displayed on the LCD module
3. Start, stop and change of direction of the motor controlled by pushbutton switches and indicated by LED
4. Changes the running mode of the motor to continuous, reversible or jogging
5. Changes the speed of the motor
6. Time settings are possible for forward and reverse running of the motor

PARTS LIST

Semiconductors:	
IC1	- AT89C51 microcontroller
IC2	- 74LS04 hex inverter
IC3	- NE555 timer
IC4	- L293D motor driver
IC5	- 7805, 5V regulator
IC6	- 7806, 6V regulator
T1, T2	- BC548 npn transistor
D1-D6	- 1N4007 rectifier diode
LED1-LED5	- 5mm LED
Resistors (all ¼-watt, ±5% carbon):	
R1	- 10-kilo-ohm
R2-R5	- 2-kilo-ohm
R6-R10, R12, R13	- 220-ohm
R11	- 1-kilo-ohm
VR1, VR2	- 10-kilo-ohm preset
Capacitors:	
C1	- 10µF, 16V electrolytic
C2, C3	- 33pF ceramic disk
C4, C7, C8	- 0.1µF ceramic disk
C5	- 0.01µF ceramic disk
C6	- 1000µF, 25V electrolytic
Miscellaneous:	
X1	- 230V AC primary to 9V, 500mA secondary transformer
X_{TAL}	- 12MHz crystal
RL1	- 6V, 1C/O relay
RL2	- 6V, 2C/O relay
M	- 6V DC motor
S1-S9	- Push-to-on switch
LCD module	- 16×2 line

Circuit description

Fig. 1 shows the circuit of the microcontroller-based DC motor controller. At the heart of the DC motor controller is microcontroller AT89C51. Port pins P0.0 through P0.7 of the microcontroller are interfaced with data pins D0 through D7 of the LCD module, respectively. Port pins P3.0, P3.1 and P3.2 control the LCD operation through enable (E), register-select (RS) and read/write (R/W) pins, respectively. Contrast of the LCD is set by preset VR1. Port pins P1.0 through P1.7 are connected to switches S1 through S8 for performing the various operations.

Power-on reset signal for the microcontroller is generated by the combination of capacitor C1 and resistor R1. Switch S9 provides manual reset to the microcontroller. A 12MHz crystal provides the basic clock frequency to the microcontroller. Capacitors C2 and C3 provide stability to the oscillator. EA pin (pin 31) of the microcontroller is connected to 5V to enable internal access. Port pins P2.0 through P2.3 of the microcontroller are used for LED indication of run, stop, clockwise and anti-clockwise rotation. Port pins P2.4 through P2.6 are connected to the inputs of inverters N3, N2 and N1 of 74LS04 (IC2). The output of inverter N3 is used to trigger pin 2 of NE555 timer.

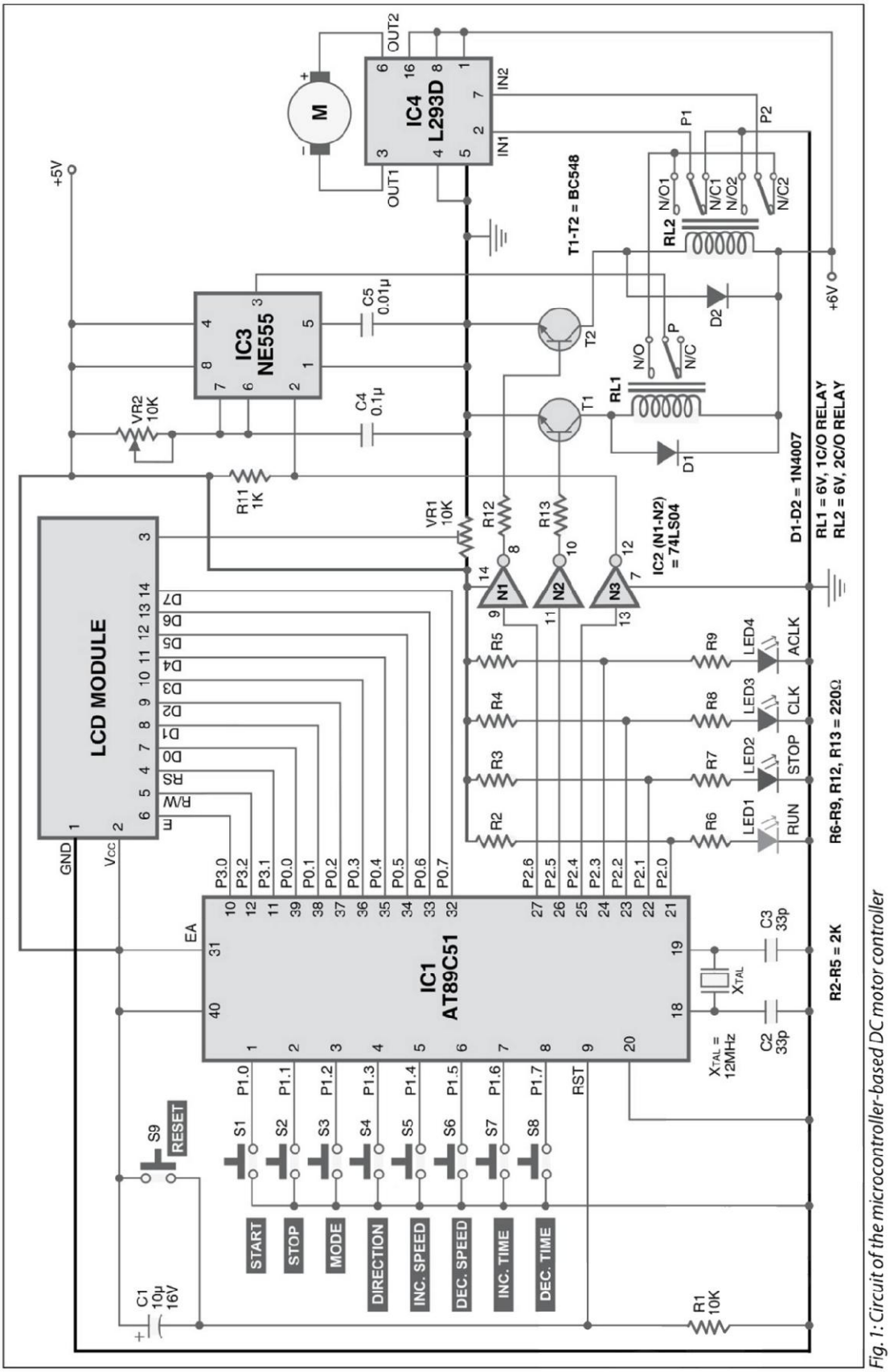


Fig. 1: Circuit of the microcontroller-based DC motor controller

Timer NE555 is configured as a monostable and its time period is decided by preset VR2 and capacitor C4. When pin 2 of NE555 goes low, output pin 3 becomes high for the predetermined period.

The output of NE555 is connected to pole P of relay RL1. Normally-open (N/O) contacts of relay RL1 are connected to N/O1 and N/C2 contacts of relay RL2. N/C1 and N/O2 contacts of RL2 are connected to ground. The outputs of inverters N2 and N1 drive relays RL1 and RL2 with the help of transistors T1 and T2, respectively. D1 and D2 act as free-wheeling diodes. Poles P1 and P2 of RL2 are connected to IN1 and IN2 pins

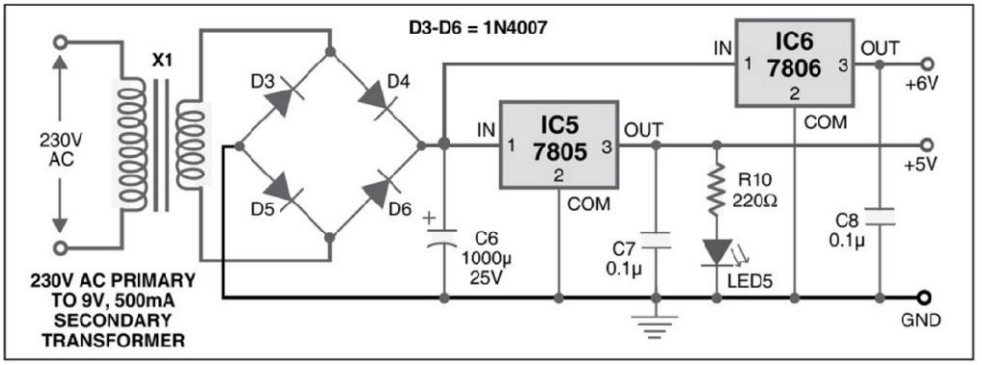


Fig. 2: Circuit of the power supply

Functions of Different Switches

Switch	Function
S1	To start motor
S2	To stop the motor
S3	Change the direction
S4	Change the mode
S5	Increase speed
S6	Decrease speed
S7	Increase time
S8	Decrease time

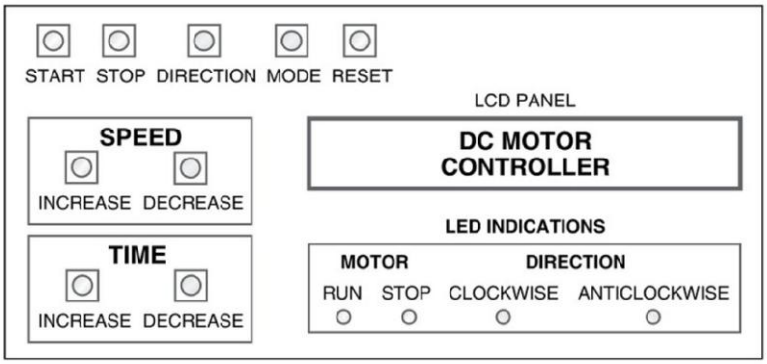


Fig. 3: Proposed panel arrangement for the DC motor controller

of motor driver L293D. OUT1 and OUT2 of L293D drive motor M.

Fig. 2 shows the power supply circuit. The 230V AC mains is stepped down by transformer X1 to deliver the secondary output of 9V, 500 mA. The transformer output is rectified by a full-wave bridge rectifier comprising diodes D3 through D6, filtered by capacitor C6 and then regulated by ICs 7805 (IC5) and 7806 (IC6). Capacitors C7 and C8 bypass the ripples present in the regulated 5V and 6V power supplies. LED5 acts as a power-'on' indicator and resistor R10 limits the current through LED5.

The proposed panel arrangement for the microcontroller-based DC motor controller is shown in Fig. 3.

An actual-size, single-side PCB for the microcontroller-based DC motor controller is shown in Fig. 4 and its component layout in Fig. 5.

Operation

The eight pushbutton switches are connected for eight different functions as shown in the table.

When S1 is pressed, the microcontroller sends low logic to port pin P2.5. The high output of inverter N2 drives transistor T1 into saturation and relay RL1 energises. So the output of NE555 is fed to inputs IN1 and IN2 of L293D through both the contacts of relay RL2. Now at the same time, after RL1 energises, the microcontroller starts generating PWM signal on port pin P2.4, which is fed to trigger pin 2 of NE555 through inverter N3. The base frequency of the generated PWM signal is 500 Hz, which means the time period is 2 ms (2000µs). The output pulse width varies from 500 µs to 1500 µs. The R-C time constant of the monostable multivibrator is kept slightly less then 500 µs to generate exactly the same inverted PWM as is generated by the microcontroller.

When switch S2 is pressed, port-pin P2.5 goes high and RL1 de-energises to stop the motor.

When switch S3 is pressed, relay RL2 energises. Pin IN1 of motor driver L293D receives the PWM signal and pin IN2 connects to ground. As a result, the motor rotates in one direction (say, clockwise).

When switch S4 is pressed, relay RL2 de-energises. Pin IN2 of motor driver L293D receives the PWM signal and pin IN1 connects to ground. The motor now rotates in opposite direction (anti-clockwise).

When switch S3 is pressed, different modes are selected in cyclic manner as given below:

1. *Continuous mode.* The motor rotates continuously with the set speed in either direction
2. *Reversible mode.* The motor reverses automatically after the set time
3. *Jogging mode.* The motor rotates for the set time in either direction and then stops for a few seconds and again rotates for the set time. It is also called 'pulse rotation'

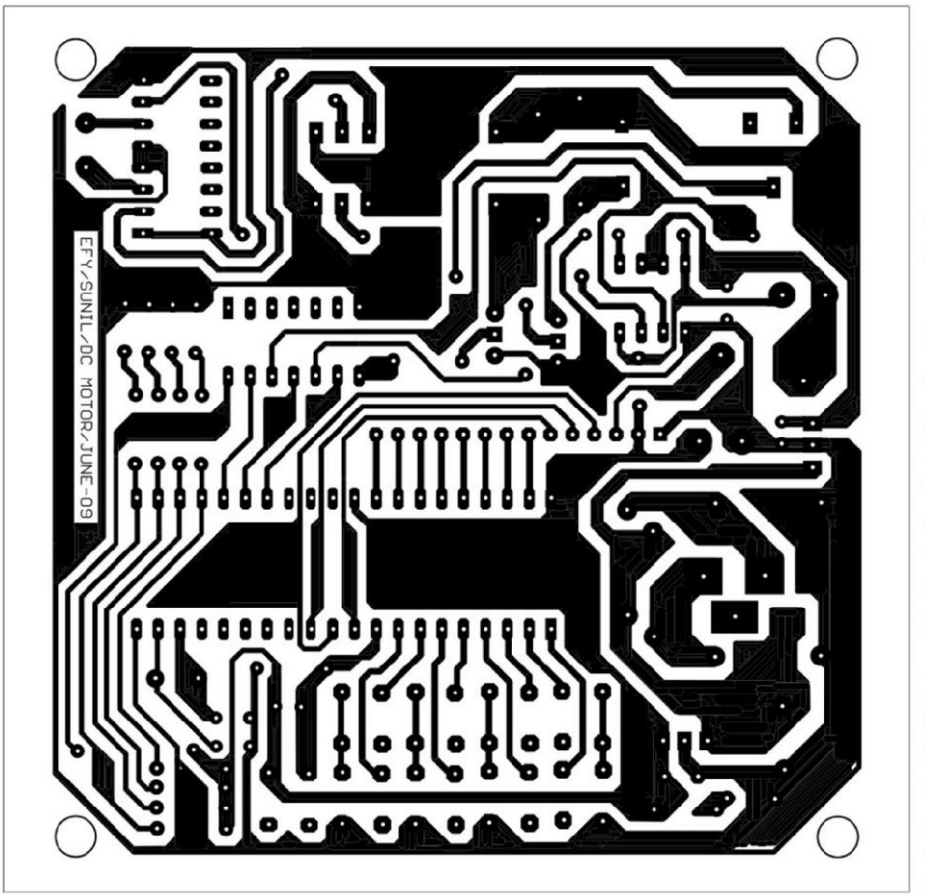


Fig. 4: A single-side, actual-size PCB layout for the microcontroller-based DC motor controller

Switches S5 and S6 are used to set the speed of the motor, either in increasing order or decreasing order, in continuous mode only.

Switches S7 and S8 are used to set the time either in increasing order or decreasing order.

Software

The program is written in 'C' language and compiled using Keil µVision3 compiler. It is well commented and easy to understand. The program has three major sections: initialisation and key press, display and motor control.

Initialisation and key press. It consists of the main function that initialises timers, ports, LCD module and LED indication and then waits for key press. When any key is pressed, the program calls that particular function.

In the main function, first the ports are initialised as input/output, then the LCD is initialised and cleared. At power-on reset or manual reset, message "DC motor controller" is displayed on the LCD and LED1 through LED4 turn off. When start switch S1 is pressed, message "Motor Start" is displayed for a second, and then the mode and current speed are displayed. Run LED and clockwise-direction LED are 'on.' Now the program waits for key press. When any key is pressed, the program jumps to one of the functions (start, direction, mode, etc).

Display. It uses the following functions:

1. 'writecmd' function sends command byte to the LCD. It takes one argument byte and sends it to P0.
2. 'writedata' function sends data byte to be displayed on the LCD. It also takes one argument byte and sends it to P0.
3. 'writestr' function writes the whole string (message) on the LCD. It takes pointer as an argument that points address of the first character of the string, then through the pointer it sends all the characters one by one to P0.
4. 'busy' function checks the status of the busy flag of the LCD. If the flag is set, the LCD is not ready and the program remains within the loop. When the flag is reset, the LCD is ready and the program comes out of the loop.

Motor control. It uses the following functions:

1. 'start' function starts motor rotation in one of the modes (continuous, reversible or jogging). The mode is selected by mode-selection flag 'm.' As explained earlier, in continuous mode the motor keeps rotating with the set speed. In reversible mode, the motor changes direction automatically after the set time. In jogging mode, the motor rotates for the set time, stops for a few seconds and then rotates.
2. 'stop' function stops rotating the motor and displays the message on the LCD module. This is indicated by glowing of the stop LED.
3. 'direction' function increments the counter by '1' every time and checks whether the count is even or odd. If the count is even, it selects clock-wise direction, and if the count is odd, it selects anticlockwise direction. This is

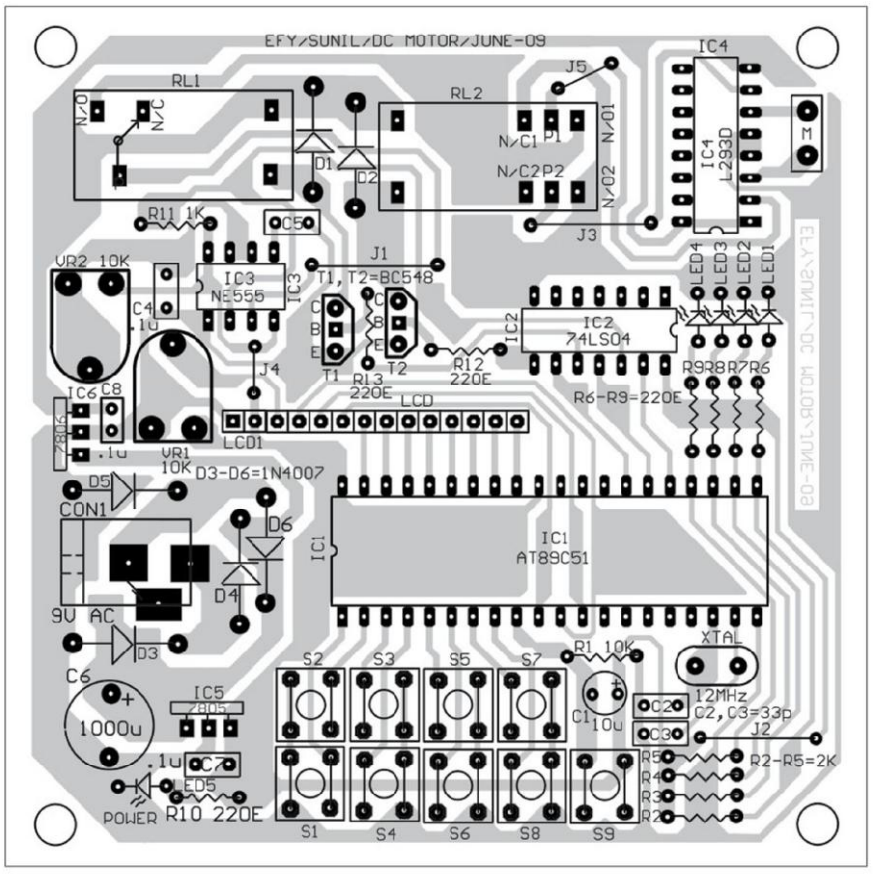


Fig. 5: Component layout for the PCB

also indicated on direction CLK and ACLK LEDs.

4. 'mode' function changes modes in cyclic manner. It increases mode-selection flag 'm' every time. If m=0 it selects continuous mode, if m=1 it selects reversible mode and if m=2 it selects jogging mode. If m=3 it is reset to '0' again and selects continuous mode and likewise.

5. 'incspeed' function increases the width of the pulse by 100 µs. The generated PWM is of 500 Hz. That means total time is 2 ms=2000 µs. The width of the pulse is varied from 500 µs to 1500 µs in steps of 100 µs. To display the speed, the variable is first converted into speed factor '1' to '9' and then into ASCII.

6. 'decspeed' function is the same as 'incspeed' but here the width of the pulse is decreased by 100 µs.

7. 'inctime' function increases the reversible time of the motor by one second. It increases the variable in multiples of 20. To display it on the LCD module, it is first divided by 20 and then converted into ASCII.

8. 'dectime' function is the same as 'inctime' but it decreases reversible time by one second.

Delay. It uses the following functions:

1. 'keydly' function generates a fix delay of around 50 ms for key debouncing.

2. 'delay' function generates a variable delay by timer 0. The basic delay is of 100 µs. It is rotated in loop from five to 15 times to generate minimum 500µs and maximum 1500µs delay.

3. 'time' function again generates a variable delay by timer 1. The basic delay is of 50 ms. It is rotated in multiples of 20 from 20 to 180 to generate minimum 1-second and maximum 9-second delay.

Download source code: http://www.efymag.com/admin/issuepdf/Microcontroller%20Based%20DC%20Motor%20Controller.zip

DCMC.C

```
#include<reg51.h>
#include <string.h>
sbit rs = P3^1;                         // rs pin of LCD
sbit en = P3^0;                         // en pin of LCD
sbit rw = P3^2;                         // rw pin of LCD
sbit b = P0^7;                          // busy flag
sbit led1=P2^0;                         // run indicator
sbit led2=P2^1;                 // stop indicator
sbit led3=P2^2;                 //  clockwise  direction
indicator
sbit led4=P2^3;                 // anticlockwise direc-
tion indicator
sbit PWM=P2^4;                          // PWM output
sbit RL1=P2^5;                          // relay 1 pin
sbit RL2=P2^6;                          // relay 2 pin

unsigned int x=10;                      // ontime
unsigned int y=10;                      // offtime
unsigned int m=0;                       // mode
unsigned int d=0;                       // direction
unsigned int t=100;                     // time
unsigned int r=0;                       // run flag
void start(void);               // function initilization
void mode(void);
void direction(void);
void incspeed(void);
void decspeed(void);
void inctime(void);
void dectime(void);
void time(unsigned int);
void delay(unsigned int);
void keydly(void);
void busy(void);
```

```
void writecmd(unsigned char a)        // send command
to LCD
  {

         busy();              // check busy flag
         rs = 0;              // select command regis-
ter
         rw = 0;              // write enable
         P0 = a;              // send byte to LCD
         en = 1;              // apply strobe pulse
         en = 0;
  }
void writedata(unsigned char b)       // send data to
LCD
  {

         busy();              // check busy flag
         rs = 1;              // select data register
         rw = 0;              // write enable
         P0 = b;              // write enable
         en = 1;              // send byte to LCD
         en = 0;              //  apply  strobe  pulse

  }
void busy()                   // check busy flag of LCD
  {
         en = 0;              // disable display
         P0 = 0xFF;           // P0 as input
         rs = 0;              // select command regis-
ter
         rw = 1;              // read enable
         while(b==1)          // if busy bit is 1
         {
          en=0;               // remain withine loop
          en=1;
         }
         en=0;

  }
void writestr(unsigned char *s)       // send string
message to LCD
 {
         unsigned char l,i;
         l = strlen(s);       // get length of string
         for(i=0;i<l;i++)

         {
                  writedata(*s);
// till the length of string
                  s++;
         // send characters one by one
         }
 }
void start()                  // start rotating motor
  {
         if(m==0)
// for m=0 start continuous mode
           {
                  RL1=0;     // switch on RL1
                  r=1;       // set run flag
                  P1=0xFF;   // send all 1's to P1
                  while(P1==0xFF)
// till no key is pressed
                    {
                           led1=1;
// indication on run LED
                           PWM=1;
// send high logic to PWM pin
                           delay(x);
// on time delay
                           PWM=0;
// now send low logic to PWM
                           delay(y);
// off time delay
                    }
                  }
                  else if(m==1)                // for
m=1 start reversible mode
                     {
                              r=1;
// set run flag
                              P1=0xFF;
// send all 1's to P1
                              while(P1==0xFF)
// till no key is pressed
                                                   {

         led1=1;  // run LED=1
         led3=1;
         led4=0;
         PWM=1;   // send high on PWM pin
         RL2=1;   // select one direction
         RL1=0;   // switch on RL1
         time(t); // wait for desired time
         RL1=1;   // switch off RL1
         led1=0;  // run LED=0;
         time(20);// wait for 1 sec
         led1=1;  // again run LED=1
         led3=0;
         led4=1;
         RL2=0;   // select other direction
         RL1=0;   // switch on RL1
         time(t); // wait for desire time
         RL1=1;   // switch off RL1
         led1=0;  // run LED=0
         time(20);// wait for 1 sec
                                     }
                                              PWM=0;


                              }
                        else if(m==2)
// for m=2 start jogging mode
                          {
                                              r=1;
// reset run flag
                                              P1=0xFF;
// send all 1's to P1
                              while(P1==0xFF)
// till no key is pressed
                                                   {
                                              led1=1;
                                              PWM=1;
// send high on PWM pin
                                              RL1=0;
// switch on RL1
                                              time(t);
// wait for 1 sec
                                              RL1=1;
// switch off RL1
                                              PWM=0;
// send low on PWM pin
                                              led1=0;
                                              time(20);
                                    }

                              }
  }
void direction()              // alter the direction
  {
         keydly();            // key debounce delay
         d++;                 // increment count
         if((d%2)==0)         // check for even or odd
           {
                  led3=1;     // indicate on LEDs
                  led4=0;
                  RL2=1;      // switch ON / OFF RL2
           }
         else
           {
                  led3=0;
                  led4=1;
                  RL2=0;
```

```
          }
  }
 void mode()       // change mode of rotation
   {
         keydly();          // key debounce delay
         writecmd(0x80);
// display message on first line first column
         m++;
// increment count
         if(m==3) m=0;      // if it is 3 reset it
         if(m==0)
           { writestr("mode:continuous ");
// otherwise display mode
           time(15);
           }
         else if(m==1)
    {writestr("mode:reversible ");
           time(15);
         }
         else if(m==2)
         {writestr("mode:jogging    ");
         time(15);
   }
   }
 void decspeed()            // increase speed
   {
         int z;
         keydly();          // key debounce
         writecmd(0xC0);    // select second line on
LCD
         if(y<14)           //  if  not  max  pulse
width
         {
          x--;
          y++;              // increase it convert
it in to                     z=y-5+0x30;
// 1 to 10 scale and ASCII
          writestr("speed: ");
// diaplay speed on LCD
          writedata(z);
          writestr("       ");
         }
 else if(y==14)
         writestr("min speed: 9  ");
// if max speed display message
         }
void incspeed()             // increase speed
   {
         int w;
         keydly();
         writecmd(0xC0); // key debounce
                                      if(y>6)
// if not minimum width
         {
          x++;
          y-- ;             // decrease it

          w=y-5+0x30;       // do same as above

          writestr("speed: ");
          writedata(w);
          writestr("       ");
         }
   else if(y==6)
         writestr("max speed: 1  "); // if min speed
display message
         }
void inctime()              // increase time
  {
         int p;
         keydly();          // key debounce delay
         writecmd(0xC0);

         if(t<180)          // if not max time
           {
                  t+=20;    // increase it by 1 sec

                  p=t/20;
                  p=p+0x30;
         // convert it in to ASCII
         writestr("time: "); // display it
                  writedata(p);
                  writestr(" sec    ");
           }
         else if(t==180)
          writestr("max time: 9 sec");
// if max time display message
  }
void dectime()              // decrease time
  {
         int q;
         keydly();          // key debounce delay
         writecmd(0xC0);
         if(t>20)           // if not min time
           {
                  t-=20;    //  decrease  it

                  q=t/20;
                  q=q+0x30;
// do same as above
                  writestr("time: ");
                  writedata(q);
                  writestr(" sec    ");
           }
         else if(t==20)
         writestr("min time: 1 sec");
// if min time display message
  }
void keydly()               // key debounce delay
  {
         int a,b;
         for(a=0;a<50;a++)
           for(b=0;b<1000;b++);
  }
void time(unsigned int c)
// change time in seconds
 {
         int k;
         TL1 = 0xAF;                // use timer 1
         TH1 = 0x3C;                // to generate
50 ms delay
         TR1 = 1;                   // start timer
         for(k=0;k<=c;k++)          // rotate loop
in multiples of 20
           {
                  while(TF1==0);    //  wait  till
timer overflow
                  TF1 = 0;          //  reset  the
flag
                  TL1 = 0xAF;       // reload it
                  TH1 = 0x3C;
           }
         TR1 = 0;                   // stop timer
  }
void delay(unsigned int c1) // change time in micro
seconds
 {
         int a;
         TH0=0x9B;          // select timer 0
         TL0=0x9B;          // to generate 100 micro
second delay
         TR0=1;             // start timer

         for(a=0;a<c1;a++)  // rotate loop between 5
to 15
           {
                  while(TF0==0);    // wait  until
timer overflow
                  TF0=0;    // reset the flag
           }
```

```
        TR0=0;                          // stop timer
  }
void main()
  {
        TMOD=0x12; // timer1 in 16 bit,timer 0 in 8
bit auto reload mode
        P2=0xE0;                        //  LEDs  off,
relays OFF
        P0=0x00;                        // P0, P3 output
ports
        P3=0x00;
        writecmd(0x3C);                 // initilize LCD
        writecmd(0x0E);
        writecmd(0x01);
        writecmd(0x84);                 // display mes-
sage
    writestr("DC Motor");   // DC motor controller in
        writecmd(0xC3);                 // center of LCD
        writestr("Controller");
agin:P1=0xFF;                   // P1 as input port
        while(P1==0xFF);                //  wait  until
any key press
loop:switch(P1)
        {
         case 0xFE:                     // for first key
         keydly();                      // key debounce
         writecmd(0x01);
         writestr("motor start");
         time(50);          // wait for 2.5 sec
         writecmd(0x80);
          writestr("mode:continuous  ");//  display
current mode and speed
         writecmd(0xC0);
         writestr("speed: 5       ");
         led1=1;                        // Run LED ON
         led2=0;                        // stop LED OFF
         led3=1;                        //  clockwise
direction ON
         led4=0;                        // anticlockwise
direction OFF
         start();                       // sart rotating
motor
         break;
         case 0xFD:         // for second key
         keydly();          // key debounce
         r=0;               // run flag reset
         writecmd(0x01);
         writestr("motor stop");// display message
         led1=0;                        // Run OFF
         led2=1;                        // stop LED ON
         led3=0;            //  clockwise  direction
OFF
         led4=0;            // anticlockwise direc-
tion OFF
         PWM=0;             // low logic to PWM pin
         RL1=1;                         // relay1 off
         break;
         case 0xFB:         // for third key

         mode();            // select mode

        if(r==1) start();           // jump to start
if run flag is set
                            break;
         case 0xF7:         // for fourth key

         direction();       // change direction

         if(r==1) start();
// jump to start if run flag is set
         break;
         case 0xEF:         // for fifth key

         incspeed();        // increase speed
         if(r==1) start(); // jump to start if run
flag is set
         break;
         case 0xDF:         // for sixth key

         decspeed();        // decrease speed

         if(r==1) start();
// jump to start if run flag is set
         break;
         case 0xBF:         // for seventh key

         inctime();         // increase time
         if(r==1) start(); // jump to start if run
flag is set
         break;
         case 0x7F:         // for eigth key

         dectime();         // decrease time

         if(r==1) start(); // jump to start if run
flag is set
         break;
         }
        if(r==1) goto loop;         // if run flag is
set jump of key detect
        else goto agin;     // if not jump to again
  }
```

GPS- AND GSM-BASED VEHICLE TRACKING SYSTEM

■ LALIT PRAKASH VATSAL, PRINCE GUPTA AND SANI THEO

Presented here is a microcontroller-based project for tracking a vehicle using global positioning system (GPS) and global system for mobile communication (GSM).

This is a cheaper solution than a two-way GPS communication system wherein communication is done in both ways with GPS satellites. This project uses only one GPS device and two-way communication is achieved using a GSM modem. GSM modem with a SIM card used here implements the same communication technique as in a regular cellphone.

The system can be mounted or fitted in your vehicle in a hidden or suitable compartment. After this installation, you can easily track your vehicle using your mobile phone by dialling the mobile number of the SIM attached to the GSM modem. You will automatically get the location of the vehicle in the form of an SMS (short message) on your mobile phone.

This system allows you to track your vehicle anytime and anywhere. Whether you own a company with a fleet of hundreds of vehicles or you have expensive piece of equipment and you want to keep an eye on them, this tracking system can inform you of the status without you having to be actually present on the site.

Fig. 1 shows the block diagram of the GPS- and GSM-based vehicle tracking system.

PARTS LIST

Semiconductors:

IC1	- 7805, 5V regulator
IC2	- ATMega16 microcontroller
IC3	- MAX232 converter
LED1	- 5mm light-emitting diode

Resistors (all ¼-watt, ±5% carbon):

R1	- 680-ohm
R2	- 10-kilo-ohm

Capacitors:

C1	- 0.1µF ceramic
C2, C3	- 22pF ceramic
C4-C8	- 10µF, 16V electrolytic

Miscellaneous:

S1	- Tactile switch
X_{TAL}	- 12MHz crystal
BATT.	- 9V PP3 battery
GSM modem	- SIM300
GPS module	- iWave

Applications and benefits

1. You can locate your stolen vehicle easily using your mobile without any extra cost.

2. It can be used for trucks carrying valuable goods, to keep track of the status of delivery and location of the truck at all times.

3. The device ensures vehicle security and smooth fleet management.

4. You can easily install it in any vehicle such as cars, boats and motorbikes. An SMS will inform you whether the vehicle is stationary or on the move.

5. You can also use it to keep tab on your driver. It reduces vehicle abuse and ultimately results in significant cost-savings for

EFY note. *Since we have started using gEDA Open Source software to draw circuit diagrams and PCB patterns, the diagrams may look slightly different.*

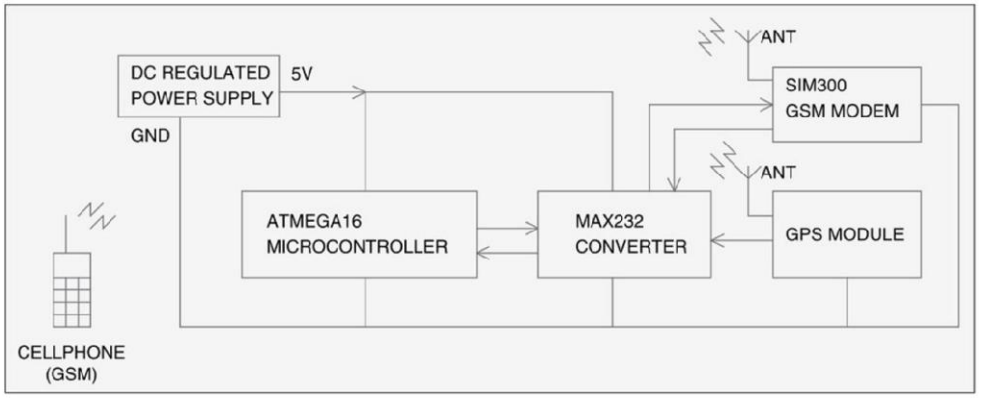


Fig. 1: Block diagram of the GPS- and GSM-based vehicle tracking system

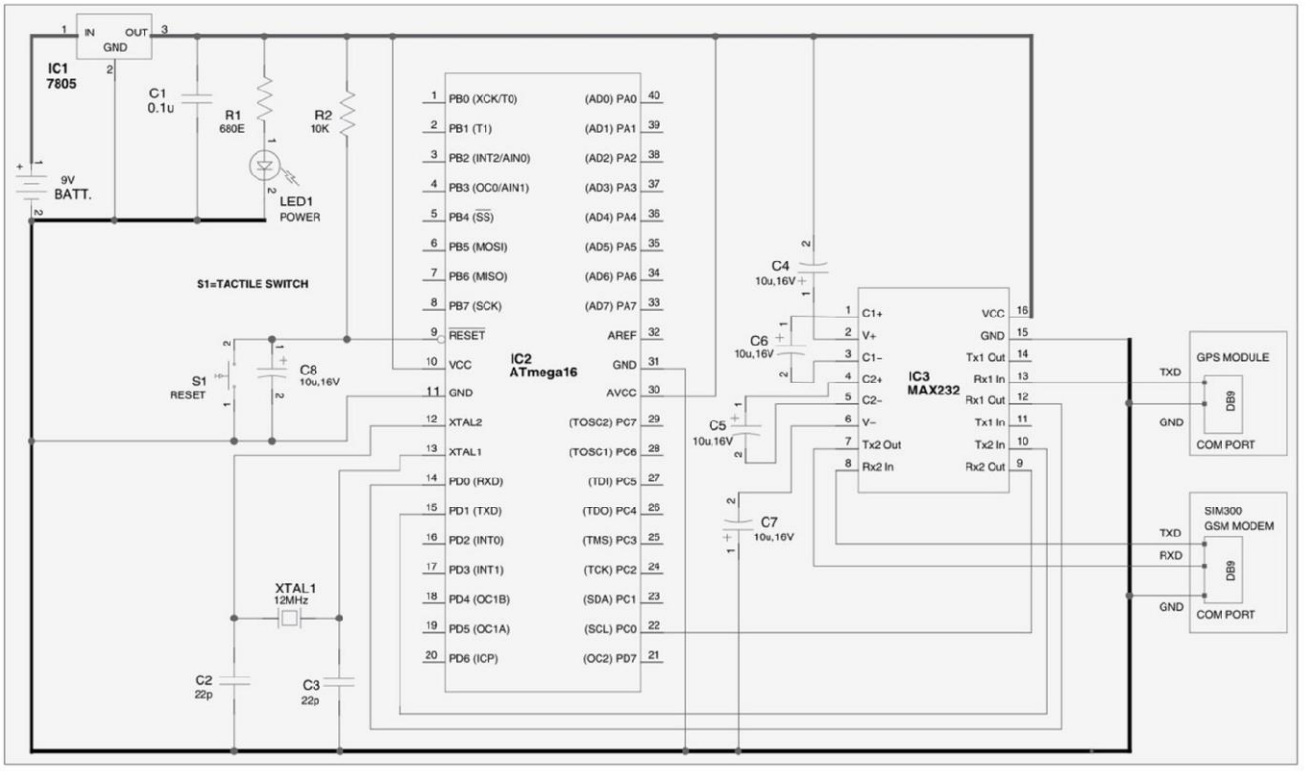


Fig. 2: Circuit of a GPS- and GSM-based vehicle tracking system

individuals, fleet owners and the like.

Before delving into the detailed working of the project, let's discuss some basics of GPS and GSM technologies.

GPS

GPS is a space-based satellite navigation system. It provides location and time information in all weather conditions, anywhere on or near the Earth. GPS receivers are popularly used for navigation, positioning, time dissemination and other research purposes.

The GPS consists of satellites that orbit the earth. These satellites are geosynchronous with an orbital period that is the same as the Earth's rotation period. So they maintain exactly the same position with respect to the earth below them. All the GPS satellites transmit radio signals, which are then captured by a GPS receiver and used to calculate its geographical position. A minimum of four satellites may be required to compute the four dimensions of X, Y, Z (latitude, longitude and elevation) and time. GPS receiver converts the received signals into position and estimates time and some other useful information depending on the application and requirements.

GPS determines the distance between a GPS satellite and a GPS receiver by measuring the amount of time taken by a radio signal (the GPS signal) to travel from the satellite to the receiver. To obtain accurate information, the satellites and the receiver use very accurate clocks, which are synchronised so that they generate the same code at exactly the same time.

If accuracy is important, you need GPS with a wide-area augmentation system (WAAS) capability. This is a satellite service providing additional correction information to the GPS receiver in order to increase its accuracy.

Before purchasing a GPS receiver, it's good to know the protocols supported by it. Some popular protocols for GPS receivers are:

NMEA 0183. An industry-standard protocol common to marine applications defined by National Marine Electronics Association (NMEA), USA. NMEA provides direct compatibility with other NMEA-capable devices such as chart plotters and radars.

TSIP (trimble standard interface protocol). A binary packet protocol that allows the designer to configure and control the GPS receiver for optimal performance in any number of applications.

TAIP (trimble ASCII interface protocol). Designed specifically for vehicle tracking applications. It is a bidirectional protocol using simple ASCII commands with the associated ASCII responses.

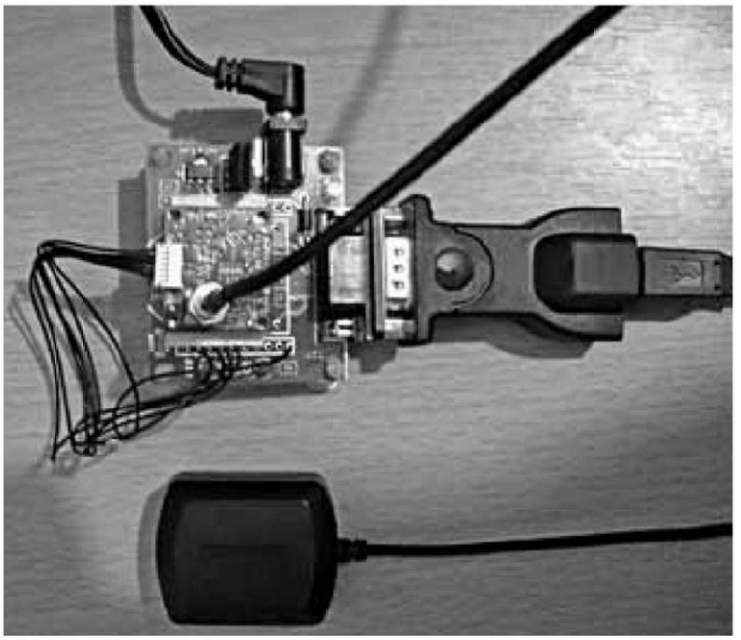


Fig. 3: iWave GPS module

GSM modem

GSM is a standard set developed by the European Telecommunications Standards Institute (ETSI) to describe technologies for second-generation (2G) digital cellular networks.

A GSM modem is a specialised type of modem that accepts a SIM card and operates over a subscription to a mobile operator just like a mobile phone.

GSM modems are a cost-effective solution for receiving SMS messages because the sender is paying for the message delivery. To perform these tasks, a GSM modem must support an extended AT command set for sending and receiving SMS messages, as defined in the ETSI GSM 07.05 and 3GPP TS 27.005 specifications.

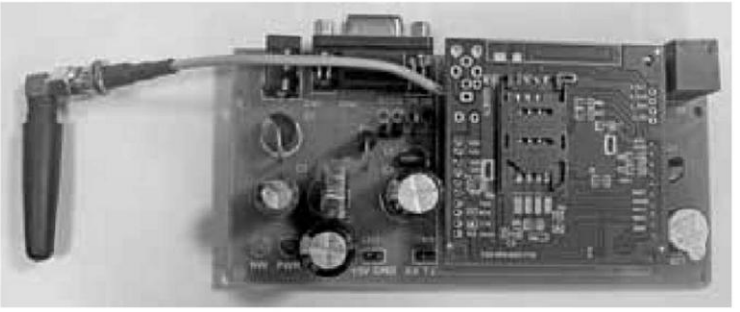


Fig. 4: SIM300 GSM modem

It should also be noted that not all phones support this modem interface for sending and receiving SMS messages, particularly most smartphones like the Blackberry, iPhone and Windows mobile devices.

Circuit description

Fig. 2 shows the circuit of a GPS- and GSM-based vehicle tracking system. It consists of a microcontroller, GPS module, GSM modem and 9V DC power supply. GPS module gets the location information from satellites in the form of latitude and longitude. The microcontroller processes this information and sends it to the GSM modem. The GSM modem then sends the information to the owner's mobile phone.

ATmega16 microcontroller. ATmega16 microcontroller (IC2) is the heart of the project that is used for interfacing to various hardware peripherals. It is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture.

ATmega16 microcontroller is interfaced serially to a GPS module and GSM modem. The GPS module outputs many data but in this project only the NMEA data is read and processed by the microcontroller. The processed data is sent to the user's mobile through a GSM modem.

This project design implements RS-232 protocol for serial communication between the microcontroller, GPS and GSM modem. A serial driver IC MAX232 (IC3) is used for converting RS-232 voltage levels into TTL voltage levels.

The user's mobile number should be included in the source code written for the microcontroller. Thus the user's mobile number resides in the internal memory of the MCU.

iWave GPS module. In this project, we have used the iWave GPS module (refer Fig. 3). GPS always trans-

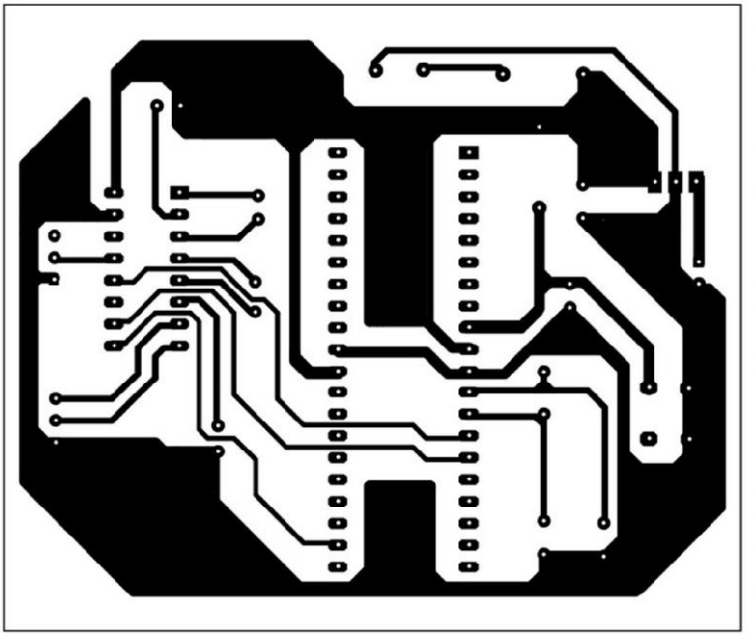


Fig. 5: An actual-size, single-side PCB for the GPS- and GSM-based vehicle tracking circuit

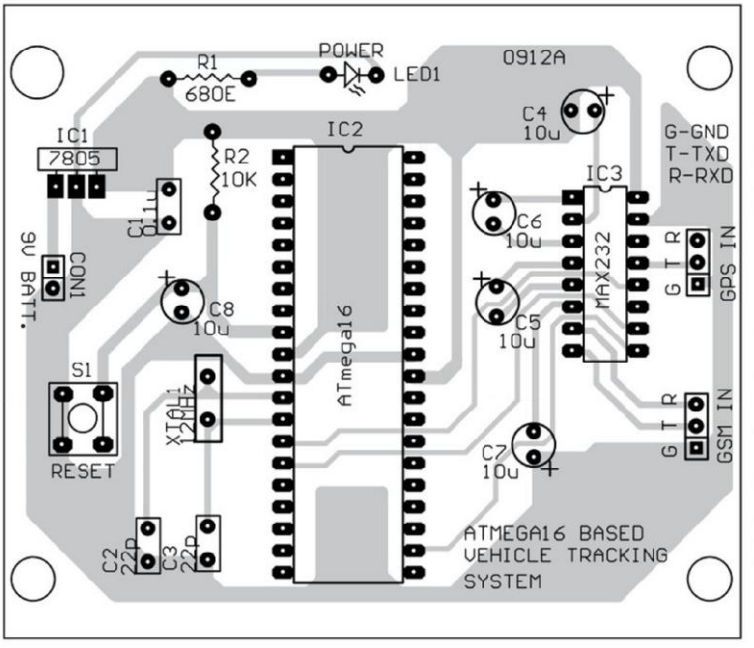


Fig. 6: Component layout for the PCB

mits the data to the microcontroller. Transmit pin TXD of GPS is connected to the microcontroller via MAX232. NMEA defined an RS-232 communication standard for devices that include GPS receivers. The iWave GPS module supports the NMEA-0183 standard that is a subset of the NMEA protocol. It operates in the L1 frequency (1575.42 MHz) and provides information with accuracy of up to 10 metres in open sky. Antenna should be placed in the open space and there should be at least 50 per cent space visibility.

GSM modem. In this project, we have used SIM300 GSM modem (refer Fig. 4). GSM modem transmits and receives the data. Modem SIM300 is a tri-band GSM/GPRS engine that works on frequencies EGSM 900 MHz, DCS 1800 MHz and PCS 1900 MHz.

Transmit pin TXD and receive pin RXD of GSM modem are connected to the microcontroller (IC2) via MAX232 (IC3). Port pin PD0 (RXD) and port pin PD1 (TXD) of the microcontroller are connected to pins 12 and 10 of MAX232, respectively.

Power supply. The circuit is powered off a 9V battery. 7805 regulator (IC1) is used to convert 9V into 5V. The microcontroller and MAX232 are powered by 5V. LED1 indicates the presence of power supply.

Software program

The program for the microcontroller is written in 'C' language and compiled using AVR Studio. The user's mobile number should be included in the source code in order to receive the call from the SIM card used in the GSM modem. The hex code of the program is burnt into the MCU using PonyProg2000 software. You can use any other suitable tool for the same.

GPS module with 9600 baud rate is used to receive the data from the satellites, which is defined in the software. The software is developed to decode the NMEA protocol. This protocol includes a set of messages that use ASCII character set and have a defined format that are continuously sent by the GPS module to the interfacing device.

The GPS module or receiver provides data in the form of ASCII comma-delimited message strings. Each message starts with a dollar sign '$' (hex 0x24) and ends with <CR><LF> (hex 0x0D 0x0A).

The software output protocol message includes global positioning system fixed data **(GGA)** and geographic position latitude/longitude (GLL). In this project, we will use GGA only.

Note that the latitude and longitude information are both represented in the 'degrees, minutes and decimal

minutes' format as ddmm.mmmm. However, most mapping applications require longitude and latitude to be expressed in decimal, degrees, in 'dd.dddddd' format with a corresponding sign (negative for south latitude and west longitude). So some kind of conversion is required in the software if you want a particular format.

The NMEA standard explains how each message string is formed with a dollar sign ($) leading each new GPS message.

For example: $GPGGA,002153.000,3342.6618,N,11751.3858,W
where $GPGGA is the GGA protocol header, 002153.000 is UTC time in hhmmss.ss format, 3342.6618 is the latitude of the GPS position fixed data in ddmm.mmmm format, 11751.3858 is the longitude of the GPS position fixed data in dddmm.mmmm format, and 'N' stands for North and 'W' for West.

With this data you can find out the exact location using a map or you can use freely available software to check the location.

Construction and testing

An actual-size, single-side PCB layout of the GPS and GSM-based vehicle tracking circuit is shown in Fig. 5 and its component layout in Fig. 6.

Assemble the components on the PCB with IC bases for ATmega16 and MAX232. Burn the code into the MCU and mount it on the PCB. Insert the SIM card with sufficient balance in the GSM module. Connect the circuit as shown in Fig. 2.

Testing

1. Connect the circuit to GPS and GSM modem as shown in Fig. 2.
2. Switch on the circuit and you will see LED1 glow.
3. Switch on the GPS module and wait for 10-15 minutes for initialisation.
4. Switch on the GSM modem.
5. Dial the mobile number in the GSM modem. After two rings, the ringing stops automatically. Wait for a few seconds. You will get an SMS alert in your mobile.
6. Check your SMS inbox. You will see the latitude and longitude data in the form of SMS text.
7. Open a standard map and locate the point on the map. You can also enter latitude and longitude values in a software such as on http://www.latlong.net/Show-Latitude-Longitude.html or any other suitable software.

Further applications

This system can also be used where the information is not needed so frequently and the subject has to be tracked at irregular time periods, such as monitoring of adolescents by parents, in research to track animals in the jungle, coordinating search and rescue efforts, and mapping trails and exploring new terrains.

Download source code: http://www.efymag.com/admin/issuepdf/GPS%20and%20GSM%20Vehicle%20Tracking%20System.rar

BATTERY BANK PROTECTOR WITH MULTIPLE FEATURES

■ SRIRAM V. AND ARUNMUTHUPANDIAN C.

The prime function of a battery management system is to provide the necessary monitoring and control to protect the battery bank from out-of-tolerance ambient or operating conditions, saving the user from the consequences of battery failures.

The microcontroller-based battery bank protection system described here can monitor the charge level, voltage, run time and temperature of your battery bank. It is designed around PIC16F877A microcontroller and also provides battery protection against high temperature and dry-run. The system is very useful for IT firms, computer laboratories, colleges and schools.

Circuit description

The block diagram for microcontroller-based battery bank protector is shown in Fig. 1. The microcontroller senses the signals from three battery sensing units, temperature sensor and AC mains sensing unit. It controls the supply to the appliances and displays corresponding information on the LCD. The power supply section powers the complete circuit. The keyboard is used for time setting and can also mute the buzzer. Real-time clock (RTC) provides the timing inputs.

Microcontroller. PIC16F877A is a low-power, high-performance, CMOS 8-bit microcontroller. Its main features are 8kB flash, 256 bytes of EEPROM, 368 bytes of RAM, 33 input/output (I/O) pins, 10-bit 8-channel analogue-to-digital converter (ADC), three timers, watchdog timer with its own on-chip R-C oscillator for reliable operation and synchronous I^2C interface.

RTC DS 1307. Timing inputs are generated by RTC DS1307. It is a low-power, real-time, full binary-coded decimal clock/calendar having 56 bytes of non-volatile static random-access memory (RAM). Address and data are transferred serially via a two-wire, bidirectional bus. The clock/calendar provides seconds, minutes, hours, day, date, month and year information. Battery (BATT) is connected at pin 3 to

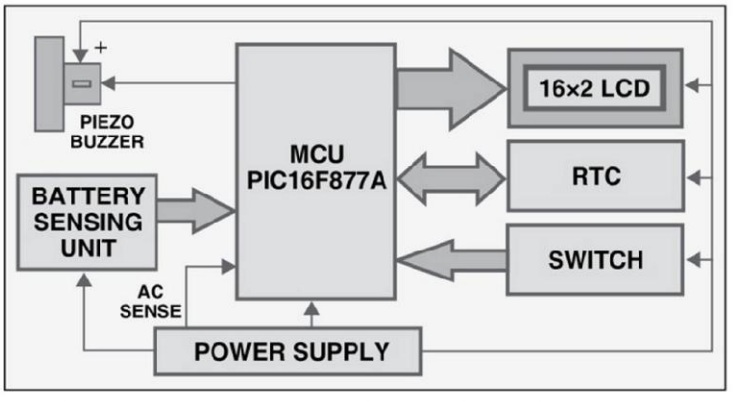


Fig. 1: Block diagram for microcontroller-based battery bank protector

PARTS LIST

Semiconductors:

IC1	- LM35 temperature sensor
IC2	- PIC 16F877A microcontroller
IC3	- DS1307 RTC
IC4	- 7805, 5V regulator
LED1	- 5mm red LED
D1-D20	- 1N4007 rectifier diode
T1-T3	- BC547 npn transistor

Resistors (all ¼-watt, ±5% carbon):

R1, R3, R5, R7	- 33-kilo-ohm
R2, R4, R6, R8, R15-R21, R25	- 10-kilo-ohm
R9, R10, R11, R12, R13	- 5-kilo-ohm
R14	- 1-kilo-ohm
R22, R23, R24	- 2.2-kilo-ohm
R26	- 470-ohm
VR1	- 10-kilo-ohm preset

Capacitors:

C1-C5, C11	- 10µF, 16V electrolytic
C6, C9	- 0.1µF polyester
C7, C8	- 22p ceramic
C10	- 1000µF, 35V electrolytic

Miscellaneous:

S1-S5	- Push-to-on switch
RL1, RL2	- 12V, 1C/O relay
X_{TAL1}	- 4MHz
X_{TAL2}	- 32.768kHz
X1	- 230V AC primary to 15V, 500mA secondary transformer
LCD	- 16×2 LCD
BATT.	- 3V battery
PZ1	- Piezobuzzer

provide battery backup.

Data is transferred between the microcontroller and the RTC using two wires (from the I²C bus), one of which serves as the clock line (SCL) and the other as data line (SDA). The RTC is driven by an external 32.768kHz crystal. Pins 5 and 6 of DS1307 are pulled high by resistors R20 and R21 and connected to pins RC4 and RC3 of the microcontroller, respectively, for serial communication between the RTC and the microcontroller.

LM35. The LM35 is a temperature sensor whose output voltage is linearly proportional to degree Celsius (centigrade) temperature. Its low output impedance and linear output make interfacing easy. It is rated to operate over a temperature range of –55°C to 150°C. Its other important features include linear +10.0mV/°C scale factor and current drain of less than 60 μA.

Power supply. The 230V, 50Hz AC mains is stepped down by transformer X1 to deliver a secondary output of 12V, 500 mA. The transformer output is rectified by a bridge rectifier comprising diodes D14 through D17, filtered

Fig. 2: Circuit of microcontroller-based battery bank protection

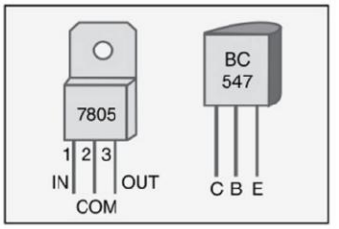


Fig. 3: Pin configurations of 7805 and BC547

by capacitor C10 and regulated by IC 7805 (IC4). Diode D18 is used to isolate the filtered voltage from the mains sensing voltage. Diodes D11, D12 and D13 prevent unregulated supply from passing to the batteries when AC mains is 'on'. Capacitor C9 bypasses the ripples, if any, in the regulated power supply. LED1 acts as the power-on indicator. Resistor R26 limits current through LED1.

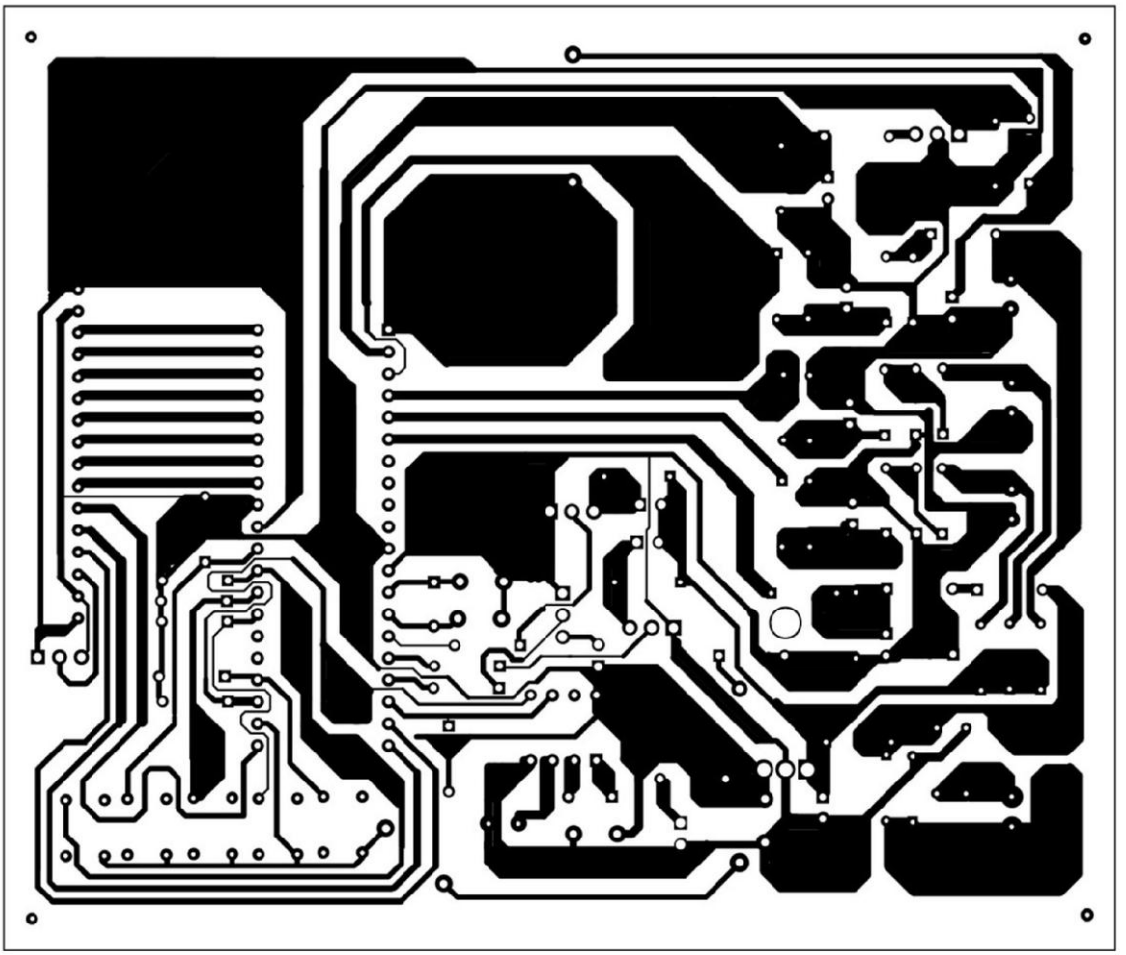


Fig. 4: An actual-size, single-side PCB for the microcontroller-based battery bank protector

Relays RL1 and RL2. Relays RL1 and RL2 are operated through 12V unregulated supply. These are driven by transistors T2 and T3, respectively. Port pins RC0 and RC1 of microcontroller PIC16F877A drive transistors T2 and T3 into saturation to energise relays RL1 and RL2, respectively, to operate the appliances either on AC mains or battery backup.

Buzzer. Port pin RC2 of microcontroller PIC16F877A drives transistor T1 into saturation to ring the piezobuzzer (PZ1).

Battery and mains voltage sensing circuitry. The battery sensing circuit for BANK1 is a voltage divider circuit made by resistors R1 and R9. The output of the voltage divider is filtered by the combination of resistor R2 and capacitor C2. This signal is fed to ADC channel RA1 of microcontroller PIC16F877A. Diodes D1 and D2 protect the microcontroller ports from voltage spikes. Similarly, outputs of the sensing circuits of BANK2 and BANK3, and mains voltage, are fed to ADC channels RA2, RA3 and RA4 of the microcontroller, respectively.

Display. A two-line, 16-character alpha-numeric liquid crystal display (LCD) is used to show the messages. Data lines D0 through D7 of the LCD are connected to port B of PIC 16F877A. Register select (RS), control (R/$\overline{W}$) and enable (E) lines are connected to port pins RD0, RD1 and RD2 of the microcontroller, respectively. Preset (VR1) is used to set the contrast of the LCD for proper display.

Switches. Switches S1 (SCRL), S2 (Inc), S3 (Dec), S4 (Set) and S5 (Mute) are interfaced to port pins RC5, RD7, RD6, RD5 and RD4 of microcontroller PIC16F877A to set the real-time clock and mute the buzzer. To set the time and date, press 'Scroll' switch S1. The display will show 'Hour.' Press 'Set' (S4) switch and change the hours value using increment switch S2 or decrement switch S3. Again, press 'Set' switch (S4). The display will show 'minute.' Change the minutes value by using the increment and decrement keys and then again press 'Set' switch. The display shows AM/PM. Select either AM or PM using increment/decrement switch and press 'Set' (S4). Now

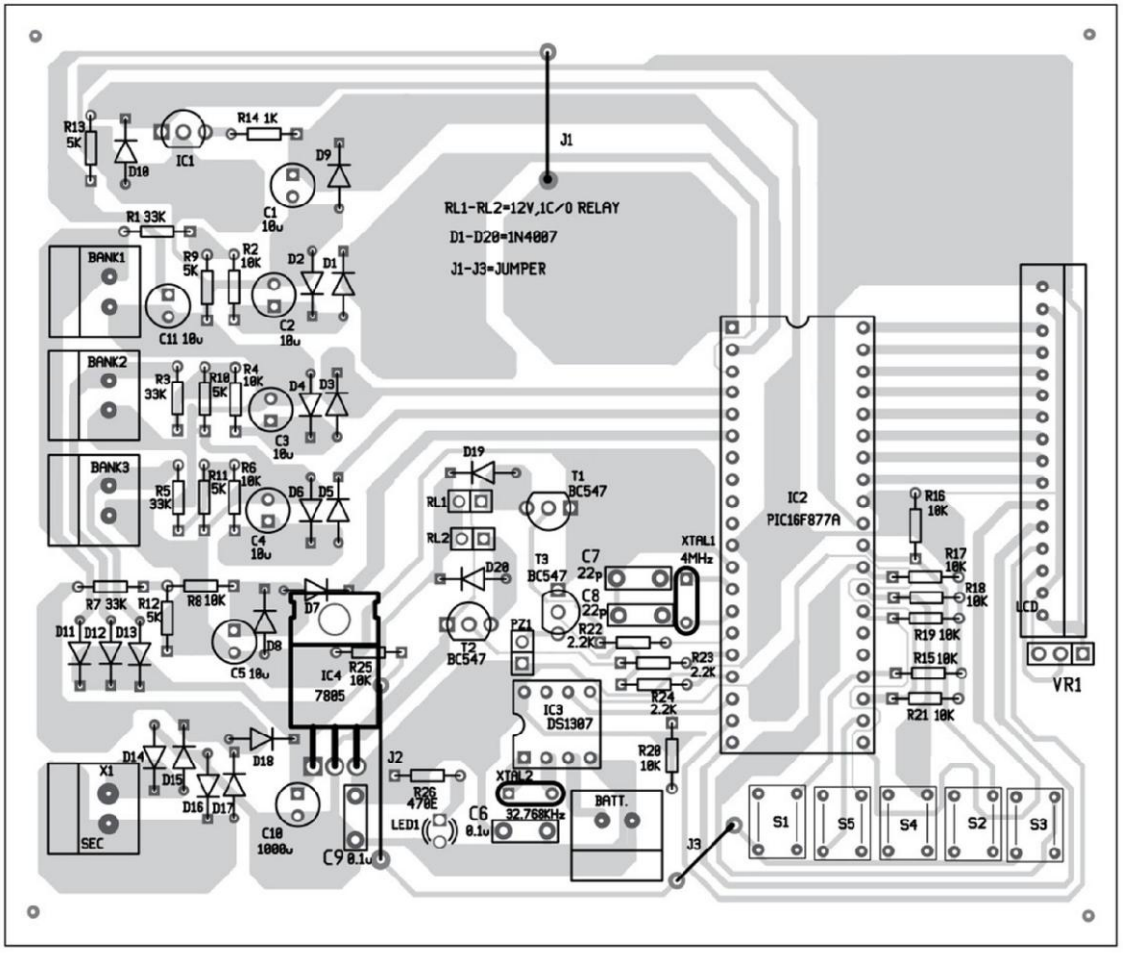


Fig. 5: Component layout for the PCB

the time setting is over and the correct time is displayed on the LCD.

Port pins RC5, RD7, RD6, RD5 and RD4 of microcontroller PIC16F877A are pulled high with resistors R15 through R19, respectively, and RD3 is pulled down by resistor R25 for key return.

Construction and working

An actual-size, single-side PCB for microcontroller-based battery bank protection is shown in Fig. 4 and its component layout in Fig. 5. Assemble the circuit on a PCB as it minimises time and assembly errors. Carefully assemble the components and double-check for any overlooked error. Use proper IC base for the microcontroller and RTC. Before inserting the ICs, check all the supply voltages.

When power is switched on, the "microcontroller-based battery bank protection" message is displayed on the LCD along with a short beep from piezobuzzer PZ1. The battery temperature is sensed by LM35 and displayed on LCD (refer Fig. 6). Simultaneously, the real time is read from RTC IC DS1307 and displayed on LCD (refer Fig. 6).

When the battery temperature rises above 50°C, the piezobuzzer sounds an alarm continuously and both the relays de-energise to cut off the power to appliances.

The display can be scrolled down to check the power of batteries 1 and 2. Powers of batteries 1

```
TEMP: 043 deg C
TIME : 11:43 PM
```

Fig. 6: Temperature and time display on the LCD

```
BAT-1 POWER:099%
BAT-2 POWER:099%
```

Fig. 7: Power indication of battery banks 1 and 2

```
BAT-3 POWER:099%
MAINS VOLT:206 V
```

Fig. 8: Power indication of battery bank 3 and mains voltage

```
BAT1.
     DRY RUNNING
```

Fig. 9: Indication for battery power below 60 per cent

```
BATTERY RUN TIME
000:08  (MMM:SS)
```

Fig. 10: Battery run time

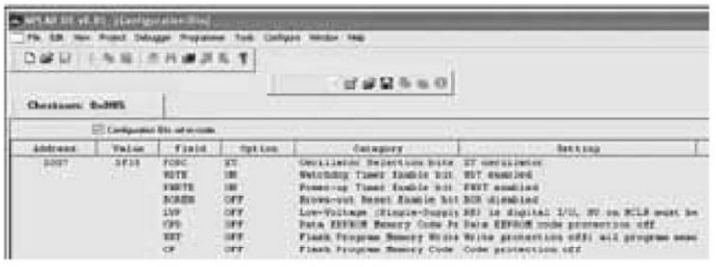


Fig. 11: Configuration bit

and 2 are sensed by the ADC of the microcontroller. The analogue input is converted into 8-bit digital data by the ADC. The sensed battery power is displayed on the LCD in terms of percentage (refer Fig. 7).

The display can be scrolled down again to check the voltages of battery-3 and mains. The analogue input is converted into 8-bit digital data by the ADC. The mains voltage is sensed from the unfiltered output of the bridge rectifier. The sensed battery-3 and mains voltages are displayed on the LCD (refer Fig. 8). If any battery bank (say Battery Bank 1) goes down below 60 per cent, it is displayed on the LCD as indicated in Fig. 9. Piezobuzzer PZ1 sounds continuously if all battery banks go below 60 per cent.

When the mains fail, the microcontroller shifts the appliances from mains to battery supply through relay RL2.

The display scrolls down further to show the changeover time—real time at which the supply changes from mains to battery supply (refer Fig. 10).

Header Files Used in the Program

Files and subroutines	Functionality
#include "delay.c"	Generate a time delay
#include "i2c_bat.h"	Configure master/slave port
#include "lcd_bat.h"	Contains the display subroutine
#include "adc_bat.h"	Contains the analogue-to-digital conversion data
void control(void)	Control real-time monitoring process
void key(void)	Check the status of the pushbutton and set the real time

Software

The program is written in 'C' language and compiled using HI-TECH PICC compiler to generate hex code. The generated hex code is burnt into the microcontroller using a suitable programmer with configuration bit setting. The set configuration bit is shown in Fig. 11.

When the system is switched on, the main program initiates the LCD. The display scans every five seconds, showing all the parameters. The control subroutine monitors all the parameters at the background without interruptions. CLRWDT( ) is used to reset the watchdog timer. If the microcontroller cannot reset the watchdog timer to the particular time period, an internal reset can be generated to reset the microcontroller to avoid malfunction. The header files used in this program are shown in the table.

Download source code: http://www.efymag.com/admin/issuepdf/Microcontroller%20Based%20Battery%20Bank%20Protection.rar

MICROCONTROLLER-BASED INTELLIGENT TRAFFIC LIGHT SYSTEM

■ ATISH GUPTA

Traffic light intervals are fixed independent of the traffic movement. So sometimes large red-light delays cause traffic congestion. Here is a microcontroller-based intelligent traffic light system that reduces the possibility of traffic jams caused by traffic light delays to an extent.

The block diagram of the microcontroller-based intelligent traffic light system is shown in Fig. 1. The system has two infrared (IR) module pairs comprising IR transmitters and receivers (TX1-RX1 and TX2-RX2). The IR transmitters are placed on the roadside facing the IR receivers that are placed on the opposite side of the road. The two IR module pairs are placed about a metre apart so that the system can differentiate the vehicles from smaller-width objects like individuals.

The IR modules are situated far from traffic lights and activate when a vehicle passes the road between the transmitter and the receiver. The microcontroller controls the modules and counts the number of vehicles passing the road. Based on the vehicle count, the microcontroller firmware generates three traffic profiles—low, medium and high.

The microcontroller firmware contains different traffic light intervals (red, yellow and green light delays in seconds) for different traffic profiles. After calculating the traffic profile depending on the number of vehicles passing through the IR system in a predefined controlling interval (e.g., one minute), it further controls the timing of traffic lights. After the controlling interval lapses, the vehicle count is re-initialised to zero to calculate the traffic profile again in the next controlling interval. The firmware of the system also allows the user to change the traffic profiles, vehicle counts, traffic light delays for each profile and controlling interval.

Here the system allows control of only one traffic light. Multiple traffic lights in different areas can be controlled by a centralised system. Further development includes recording of traffic profiles for the whole day, which can be further analysed to determine traffic conditions in different areas of the city at a specific time. Based on the analysis, the centralised system can also update the traffic lights delays, profiles and controlling interval of any traffic light.

PARTS LIST

Semiconductors:

IC1	- 7805 regulator
IC2	- P89V51RD2 microcontroller
T1-T5	- BC337 npn transistor
D1-D7	- 1N4007 rectifier diode
LED1	- 5mm LED
TX1, TX2	- IR LED
T6, T7	- L14F1 photo-transistor
LCD	- 16×2-line LCD

Resistors (all ¼-watt, ±5 per cent carbon)

R1	- 470-ohm
R2-R7	- 10-kilo-ohm
R8-R12	- 1-kilo-ohm
R13, R14	- 47-ohm
R15	- 100-ohm
VR1	- 10-kilo-ohm preset

Capacitors:

C1	- 1000µF, 25V electrolytic
C2	- 0.1µF ceramic
C3, C4	- 33pF ceramic
C5	- 10µF, 16V electrolytic

Miscellaneous:

X1	- 230V AC primary to 6V, 500mA secondary transformer
S1-S4	- Push-to-on tactile switch
X1	- 11.0592MHz crystal
RL1-RL3	- 5V, 1C/O relays
B1	- 60W, 230V green bulb
B2	- 60W, 230V yellow bulb
B3	- 60W, 230V red bulb

Circuit description

Fig. 2 shows the circuit of the microcontroller-based intelligent traffic light system. It comprises microcontroller P89V51RD2 (IC2), regulator

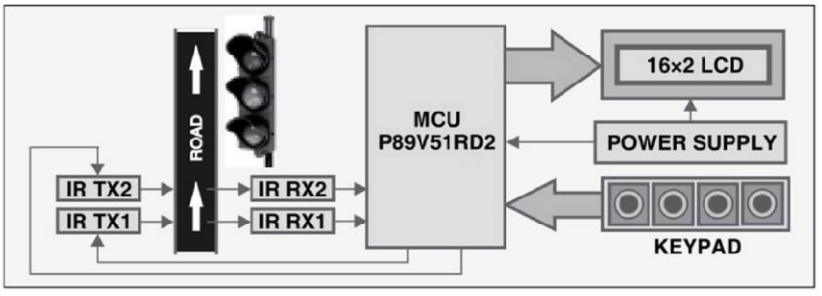


Fig. 1: Block diagram of the microcontroller-based intelligent traffic light system

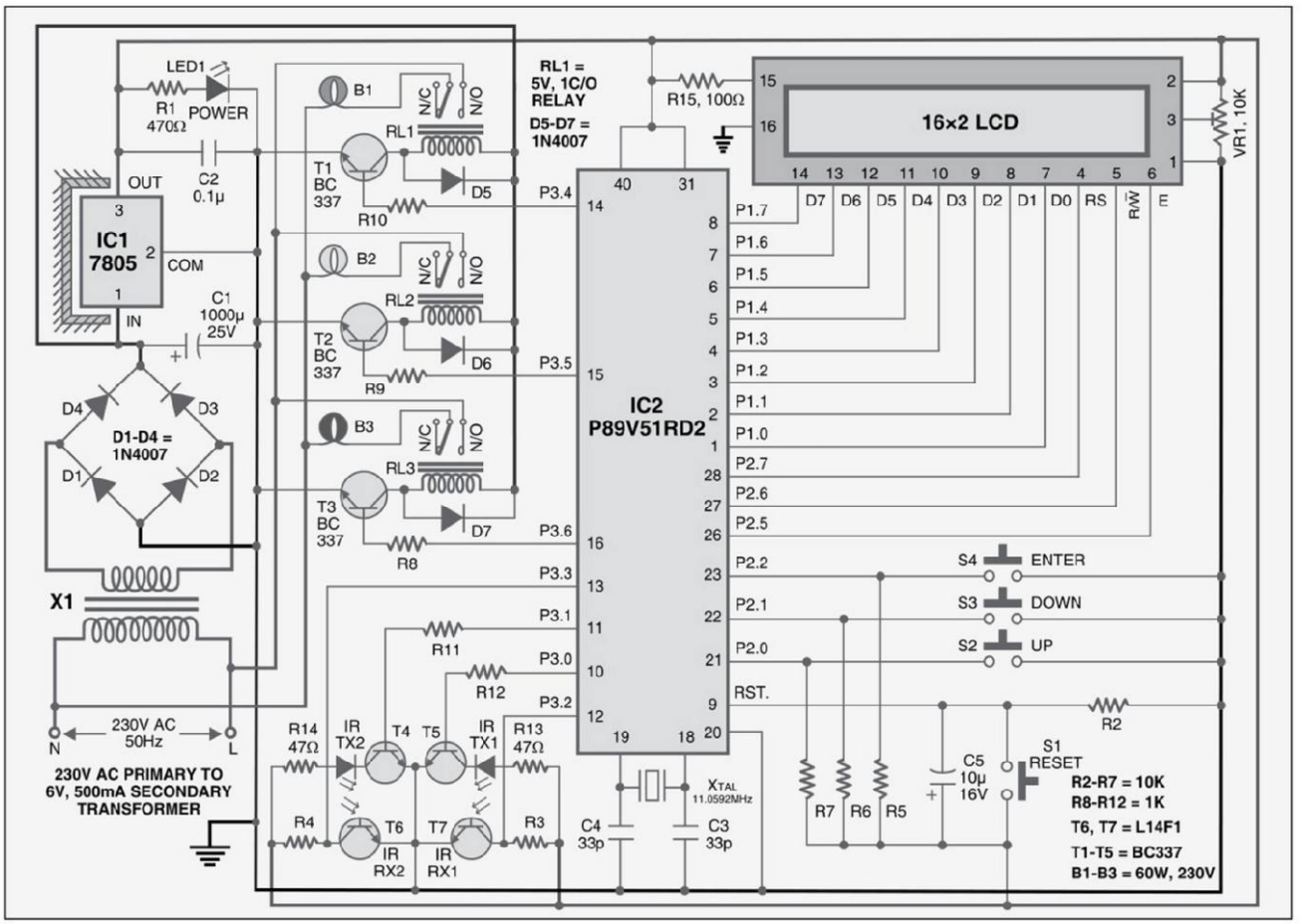


Fig. 2: Circuit diagram of traffic light controller

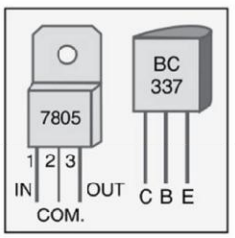


Fig. 3: Pin details of 7805 and BC337

IC 7805 (IC1), five BC337 transistors (T1 through T5), two L14F1 phototransistors (T6 and T7), an LCD module, and a few discrete components. Pin configurations of IC 7805 and transistor BC337 are shown in Fig. 3.

The 230V, 50Hz AC mains is stepped down by transformer X1 to deliver a secondary output of 6V, 500mA. The transformer output is rectified by a full-wave rectifier comprising diodes D1 through D4, filtered by capacitor C1 and regulated by IC 7805. Capacitor C2 bypasses the ripples, if any, in the regulated supply. LED1 acts as the power-on indicator. Resistor R1 limits the current through LED1.

The microcontroller is the heart of the system. It is an 8-bit, 40-pin, low-power, high-performance device with in-system programming (ISP) and in-application programming (IAP) features. It has 64 kB of Flash, 1024 bytes of internal RAM, 32 input/output lines, three 16-bit timers/counters, eight interrupt sources with four priority levels, serial peripheral interface (SPI) and enhanced UART, programmable counter array with pulse-width modulation and capture/compare functions, programmable watchdog timer, on-chip oscillator and clock circuitry. In addition, it supports two software-selectable power-saving modes—idle mode and power-down mode.

The system clock plays a significant role in the microcontroller's operation. A 11.0592MHz quartz crystal is connected across pins 18 and 19 of the microcontroller to provide it basic clock frequency. Capacitors C3 and C4 connected to the crystal maintain the resonance. Power-on reset at pin 9 is provided by the combination of capacitor C5 and resistor R2. Switch S1 is used for manual reset.

Port pins P3.0 and P3.1 of the microcontroller control IR transmitting LEDs TX1 and TX2 through transistors T5 and T4, respectively. Port pins P3.2 and P3.3 receive the detected signal from the collector of

Fig. 4: Display on the LCD

phototransistors T7 and T6, respectively. Phototransistors T6 and T7 conduct on receiving IR signals from TX2 and TX1 and provide a low output at pins P3.3 and P3.2 of the microcontroller, respectively.

Whenever any vehicle comes in between the IR transmitters and receivers, T6 and T7 stop conducting to provide a high signal to port pins P3.3 and P3.2 of the microcontroller, respectively, indicating the presence of a vehicle between the IR modules. The microcontroller counts the number of vehicles between IR transmitting LEDs and phototransistors by detecting the number of high signals at its pins P3.2 and P3.3. Traffic profile is computed by the microcontroller in a predefined delay (controller interval) according to the vehicle count.

The microcontroller controls red, yellow and green lights (connected to its pins P3.6, P3.5 and P3.4, respectively) according to the traffic profile. The traffic lights are connected to the microcontroller's output pins through relays. Transistor T3 drives relay RL3 for the red light. Similarly, yellow and green lights are controlled by pins P3.5 and P3.4 through relays RL2 and RL1 driven by transistors T2 and T1, respectively. D5, D6 and D7 act as free-wheeling diodes for protection of respective relays.

The system information is displayed on the LCD. Port pins P1.0 through P1.7 of the microcontroller are connected to data port pins D0 through D7 of the LCD. Control pins register-select (RS), read /write (R/W) and enable (E) are connected to port pins P2.7, P2.6 and P2.5, respectively. All the data is sent to the LCD in ASCII format and the commands in hex format. Preset VR1 connected to pin 3 of the LCD allows contrast control.

Fig. 4 shows a traffic profile displayed on the LCD. L1 in the first line of the LCD indicates the lane or location where this system has been installed. The next character (R,

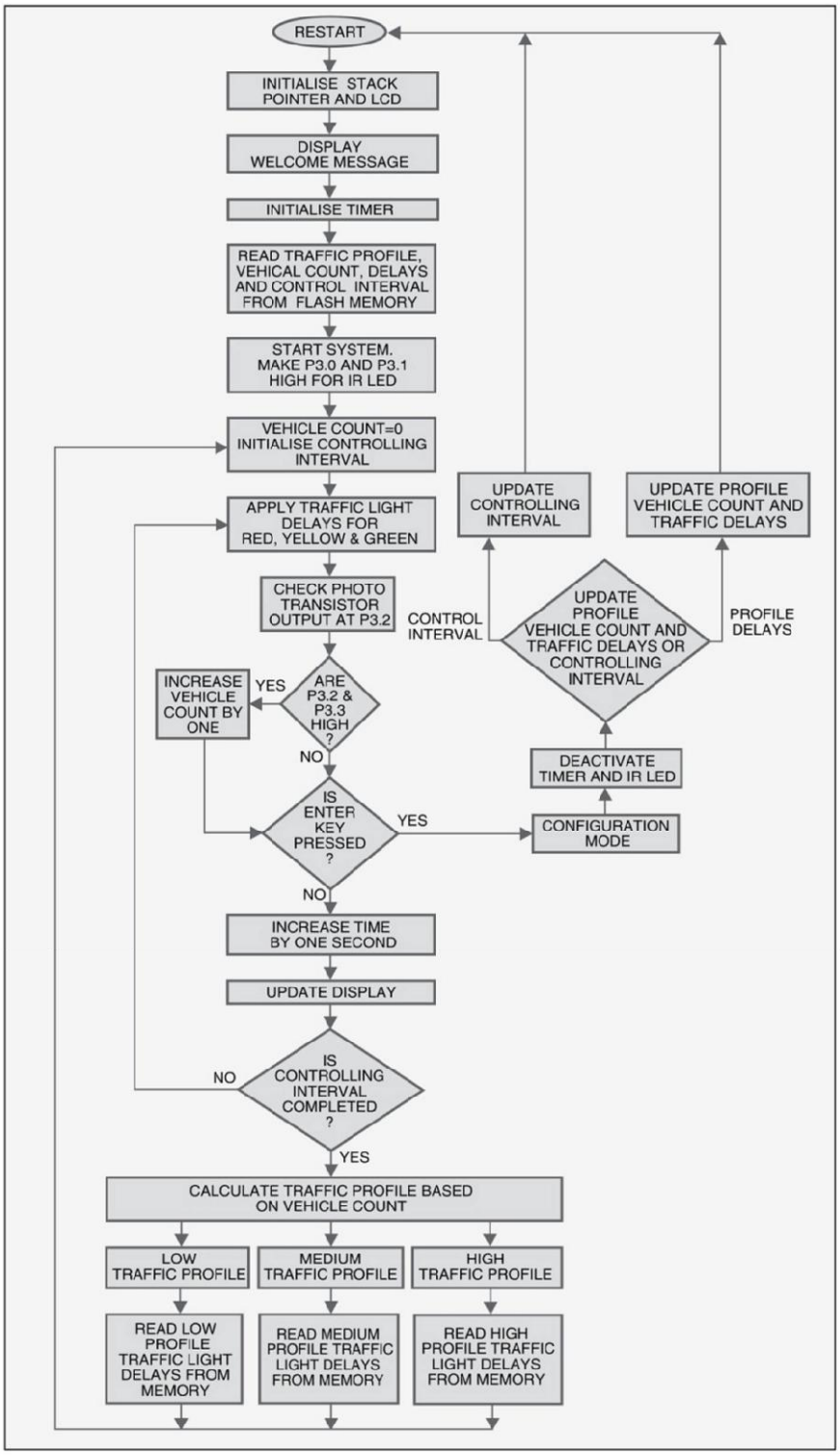


Fig. 5: Software flow-chart

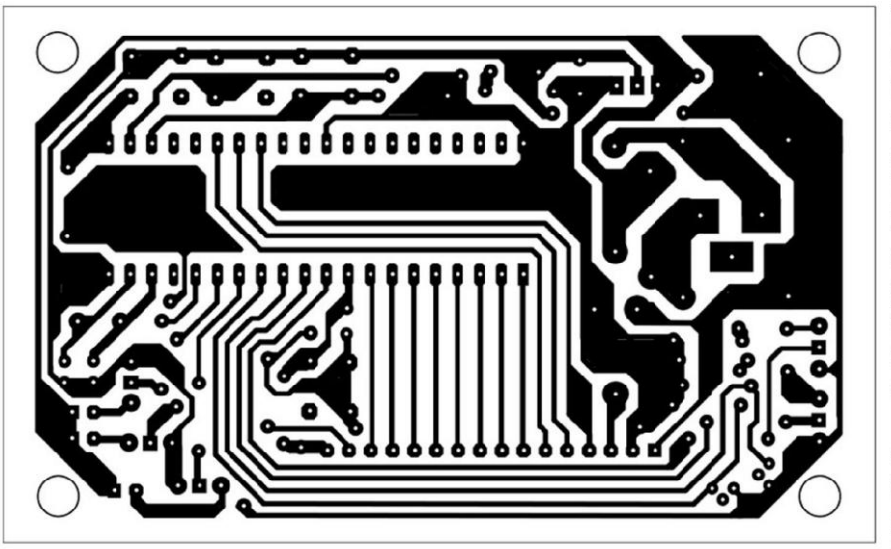


Fig. 6: An actual-size, single-side PCB for the microcontroller-based intelligent traffic light system

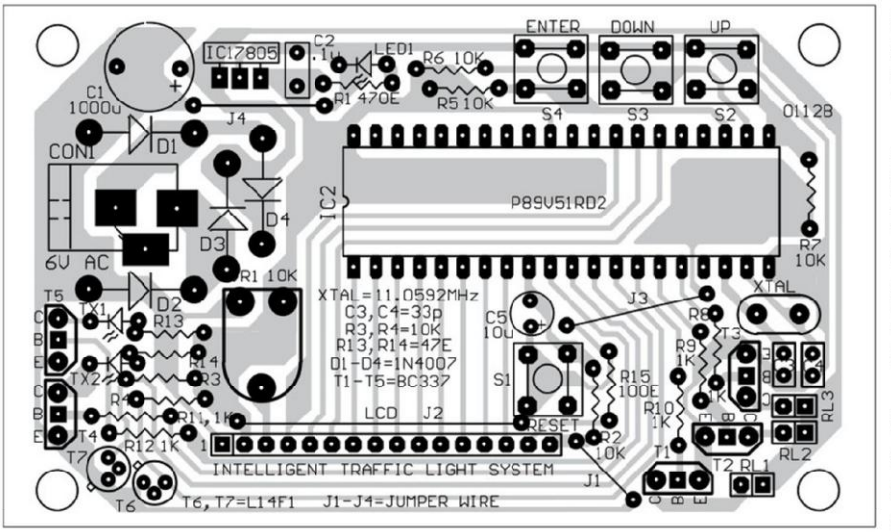


Fig. 7: Component layout for the PCB

Y or G for red, yellow or green) indicates red as the currently-running traffic light. The next three digits indicate the remaining time (in seconds) of the current traffic light. The next three characters (LOW, MED and HIG for low, medium and high) show the current traffic profile based on the number of vehicles that passed through the IR transmitter-receiver pairs in the last controlling interval.

The second line of the LCD starts with the system running time in hours, minutes and seconds. Next is the number of vehicles that have passed through the traffic light's IR sensor system. Based on this vehicle count and traffic profile, the system applies traffic delays for the next controlling interval.

The IR sensor system is installed before the traffic light, say, 500 metres away from the traffic light. Applying the traffic profile and respective traffic delays based on the previous controlling interval helps the system to manage traffic congestion to some extent.

Switches 'Up,' 'Down' and 'Enter' connected to pins P2.0, P2.1 and P2.2 of the microcontroller, respectively, allow the user to change the traffic profile, vehicle count, traffic delays in respective profiles and controlling interval. 'Up' switch is for increasing and 'Down' switch is for decreasing the default values of system parameters. The default parameter values for different traffic profiles and parameter ranges are shown in Tables I and II, respectively. The value set by the user using 'Up' and 'Down' switches is accepted only after pressing 'Enter' switch. The system can be made to enter configuration state by pressing 'Enter' key while running.

Software

The source program for the microcontroller-based intelligent traffic light system is written in Assembly language and assembled using A51 assembler. The generated hex code is burnt into the microcontroller using a suitable programmer. Flash Magic programmer can also be used to burn the hex code through a serial cable. The program works as per the flow-chart shown in Fig. 5.

The default values of system parameters like traffic profile, vehicle count and traffic delays for respective profile are stored at memory location 2000H. The two-digit controlling interval is stored at memory location 2080H. Storing the traffic vehicle count/traffic delays and controlling interval in two separate blocks allows separate

updation of the required parameter. Whenever the configuration parameters are changed through switches and LCD interaction, different memory blocks are erased and updated parameters written to the same.

Sub-routine 'erase_config' erases memory blocks 2000H through 207FH, which are used for storing the traffic profiles, vehicle counts and traffic delays for respective profiles. Sub-routine 'erase_cntrlint' erases memory blocks 2080H through 20FFH, which are used for storing the controlling interval. Sub-routine 'write_flash' handles writing any byte stored in the accumulator to any memory location.

Timer T0 of the microcontroller is configured to roll-over at a delay of 10 milliseconds. The roll-over is handled by 'intr_service' interrupt service routine. One-second delay formed by 10-milliseconds delay of timer T0 is used to manage the functionality of the entire system.

TABLE I
Default Parameter Values

Parameter	Values
Vehicle count for low profile	05 vehicles
Vehicle count for medium profile	10 vehicles
Vehicle count for high profile	15 vehicles
Red light delay for low profile	011 seconds
Yellow light delay for low profile	006 seconds
Green light delay for low profile	009 seconds
Red light delay for medium profile	010 seconds
Yellow light delay for medium profile	006 seconds
Green light delay for medium profile	011 seconds
Red light delay for high profile	008 seconds
Yellow light delay for high profile	005 seconds
Green light delay for high profile	013 seconds
Control interval	01 minute

Construction and testing

An actual-size, single-side PCB for the microcontroller-based intelligent traffic light system is shown in Fig. 6 and its component layout in Fig. 7. Assemble the circuit on a PCB as it minimises time and assembly errors. Carefully assemble the components and double-check for any overlooked error. Use proper IC base for the microcontroller. Before inserting the ICs, check the supply voltage. Align infrared LEDs and photo-transistors such that these directly face each other. Cover the photo-transistor with a mask to protect it from light. Using preset VR1, set the LCD contrast for proper display.

TABLE II
Parameter Ranges

Parameter	Range
Vehicle count	00 to 99 vehicles
Traffic lights delays	001 to 999 seconds
Control interval	01 to 99 minutes

Download source code: http://www.efymag.com/admin/issuepdf/Const-2_Microcontroller%20Based%20Traffic%20Light%20System.zip

RFID-BASED AUTOMATIC VEHICLE PARKING SYSTEM

■ **BIKRAMJEET WARAICH**

Radio-frequency identification (RFID) is an automatic identification method wherein the data stored on RFID tags or transponders is remotely retrieved. The RFID tag is a device that can be attached to or incorporated into a product, animal or person for identification and tracking using radio waves. Some tags can be read from several metres away, beyond the line of sight of the reader.

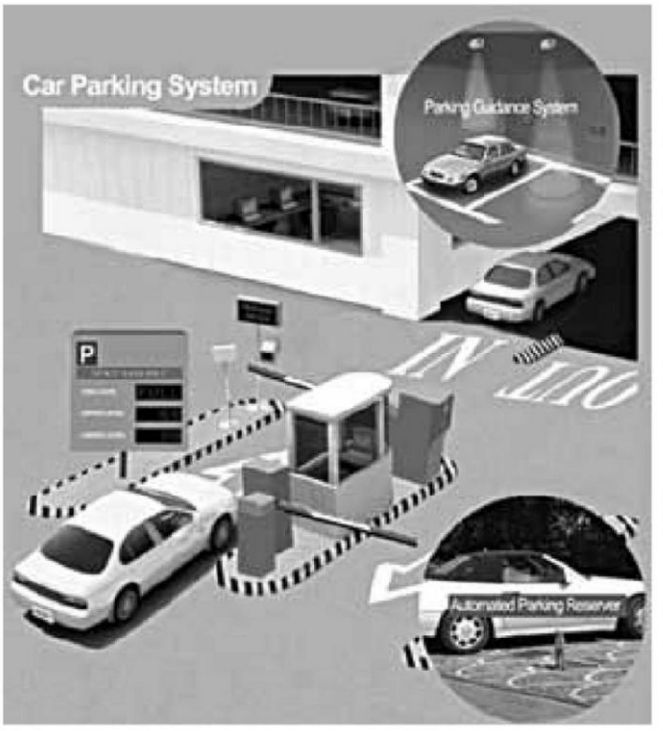


Fig. 1: Automatic vehicle parking system

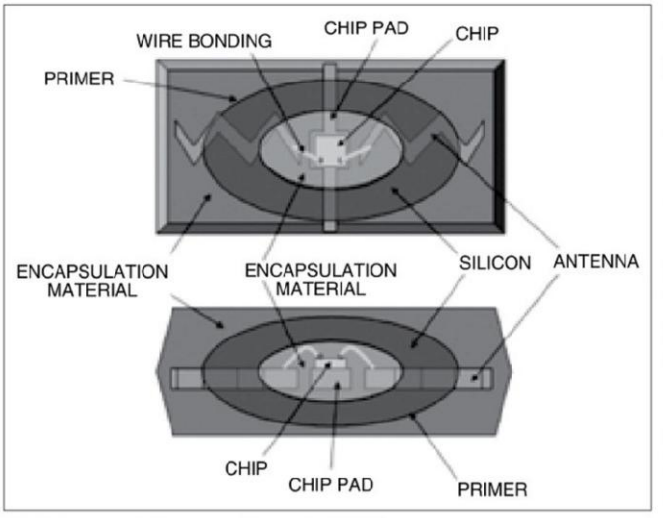


Fig. 2: Internal diagram of a typical RFID antenna

RFID technology is used in vehicle parking systems of malls and buildings (refer Fig. 1). The system normally consists of a vehicle counter, sensors, display board, gate controller, RFID tags and RFID reader. Presented here is an automatic vehicle parking system using AT89S52 microcontroller.

RFID system fundamentals

Basically, an RFID system consists of an antenna or coil, a transceiver (with decoder) and a transponder (RF tag) electronically programmed with unique information. There are many different types of RFID systems in the market. These are categorised on the basis of their frequency ranges. Some of the most commonly used RFID kits are low-frequency (30-500kHz), mid-frequency (900kHz-1500MHz) and high-frequency (2.4-2.5GHz).

RFID antenna. Fig. 2 shows the internal diagram of a typical RFID antenna. The antenna emits radio signals to activate the tag and read/write data from/to it. It is the conduit between the tag and the transceiver, which controls the system's data acquisition and communication.

Antennae are available in a variety of shapes and sizes. These can be built into a door frame to receive tag data from persons or things passing through the door, or mounted on an inter-state tollbooth to monitor the traffic passing by on a freeway. The electromagnetic field produced by the antenna can be constantly present when multiple tags are expected continually. If constant interrogation is not required, a sensor device can activate the field.

Often the antenna is packaged with a transceiver and decoder to act as a reader (interrogator), which can be configured either as a handheld or a fixed-mount device. The reader emits radio waves in the range of 2.5 cm to 30 metres or more, depending upon its

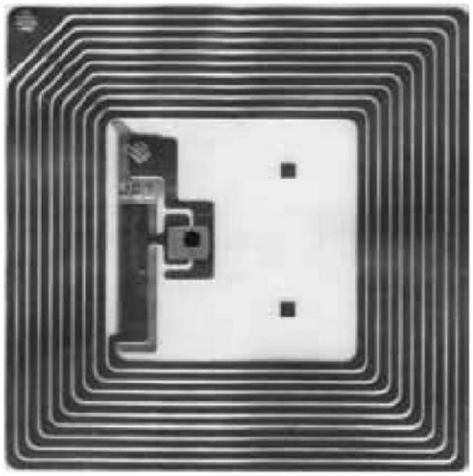


Fig. 3: Internal structure of typical RFID tag

power output and the radio frequency used. When an RFID tag passes through the electromagnetic zone, it detects the reader's activation signal. The reader decodes the data encoded in the tag's integrated circuit (silicon chip) and communicates to the host computer for processing.

Tags (transponders). Fig. 3 shows the internal structure of a typical RFID tag. It comprises a microchip containing identifying information about the item and an antenna that transmits this data wirelessly to the reader. At its most basic, the chip contains a serialised identifier or licence plate number that uniquely identifies that item (similar to bar codes). A key difference, however, is that RFID tags have a higher data capacity than their bar code counterparts. This increases the options for the type of information that can be encoded on the tag; it may include the manufacturer's name, batch or lot number, weight, ownership, destination and history (such as the temperature range to which an item has been exposed). In fact, an unlimited list of other types of information can be stored on RFID tags, depending on the application's requirements.

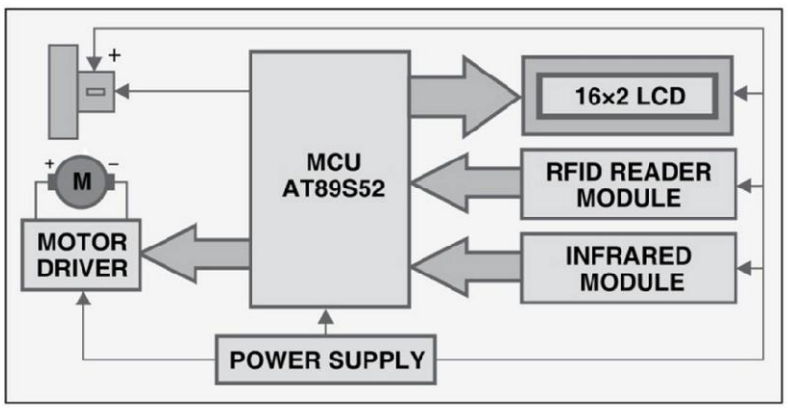


Fig. 4: Block diagram of RFID-based automatic vehicle parking system

RFID tag can be placed on individual items, cases or pallets for identification purposes, as well as fixed assets such as trailers, containers and totes. There are different types of tags with varying capabilities:

1. Read-only tags contain such data as a serialised tracking number, which is pre-written onto these by the tag manufacturer or distributor. These are generally the least expensive tags as no additional information can be included when they move through the supply chain. Any update to the information has to be maintained in the application software that tracks the stock-keeping unit's movement and activity.

2. Write-once tags enable the user to write data once in the production or distribution process. The data may include a serial number or lot or batch number.

3. Full read-write tags allow new data to be written to the tag—even over the original data—when needed. Examples include the time and date of ownership transfer or updating the repair history of a fixed asset. While these are the most costly of the three tag types and impractical for tracking inexpensive items, future standards for electronic product codes (EPCs) appear to be headed in this direction.

Other features of the tag include:

Data capacity. The capacity of data storage on a tag can vary from 16 bits to several thousand bits. Of course, the greater the storage capacity, the higher the price of the tag.

Form factor. The tag and antenna structure can come in a variety of physical form factors and can either be self-contained or embedded as part of a traditional label structure (termed as 'smart label,' it has the tag inside what looks like a regular bar code label).

Passive and active. Passive tags have no battery and broadcast their data only when energised by a reader. It means these must be actively polled to send information. Active tags broadcast data using their battery power. This means their read range is greater than passive tags—around 30 metres or more, versus 5 metres or less for most passive tags.

The extra capability and read range of active tags, however, come at a cost. These are several times more expensive than passive tags. Today, active tags are much more likely to be used for high-value items or fixed assets such as trailers, where the cost is minimal compared to item value and very long read ranges are required. Most traditional supply chain applications, such as the RFID-based tracking and compliance programmes emerging in the consumer goods retail chain, use the less expensive passive tags.

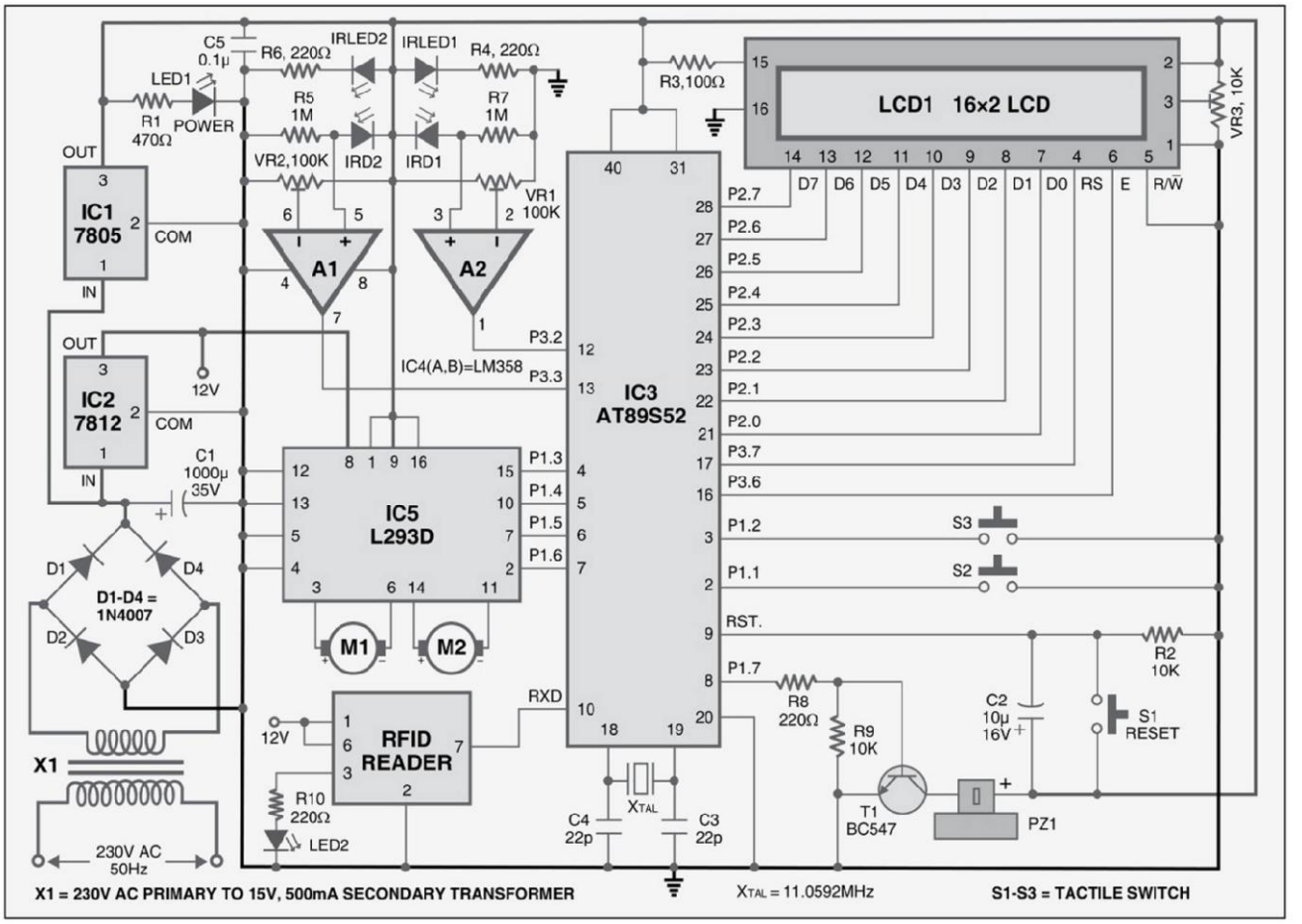


Fig. 5: Circuit of the automatic RFID-based automatic vehicle parking system

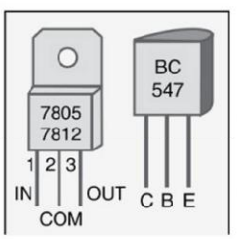


Fig. 6: Pin details of 7805, 7812 and BC547

Frequency range. Like all wireless communications, there are a variety of frequencies or spectra through which RFID tags communicate with readers. Again, there are trade-offs among cost, performance and application requirements. For instance, low-frequency tags are cheaper than ultra-high-frequency (UHF) tags, use less power and are better able to penetrate non-metallic substances. These are ideal for scanning objects with high water content, such as fruit, at close ranges.

UHFs typically offer longer range and can transfer data faster. But these use more power and are less likely to be effective with some materials.

Electronic product code (EPC) tags. EPC is an emerging specification for RFID tags, readers and business applications. It represents a specific approach to item identification, including an emerging standard for the tags—with both the data content of the tag and open wireless communication protocols.

RF transceiver. RF transceiver is the source of RF energy used to activate and power the passive RFID tags. It may be enclosed in the same cabinet as the reader or it may be a separate piece of equipment. When provided as a separate piece of equipment, the transceiver is commonly referred to as an RF module. RF transceiver controls and modulates the radio frequencies that the antenna transmits and receives. The transceiver filters and amplifies the backscatter signal from a passive RFID tag.

How this vehicle parking system works

Fig. 4 shows the block diagram of the RFID-based automatic vehicle parking system.

To get started with RFID-based automatic vehicle parking system, the vehicle owner has to first register

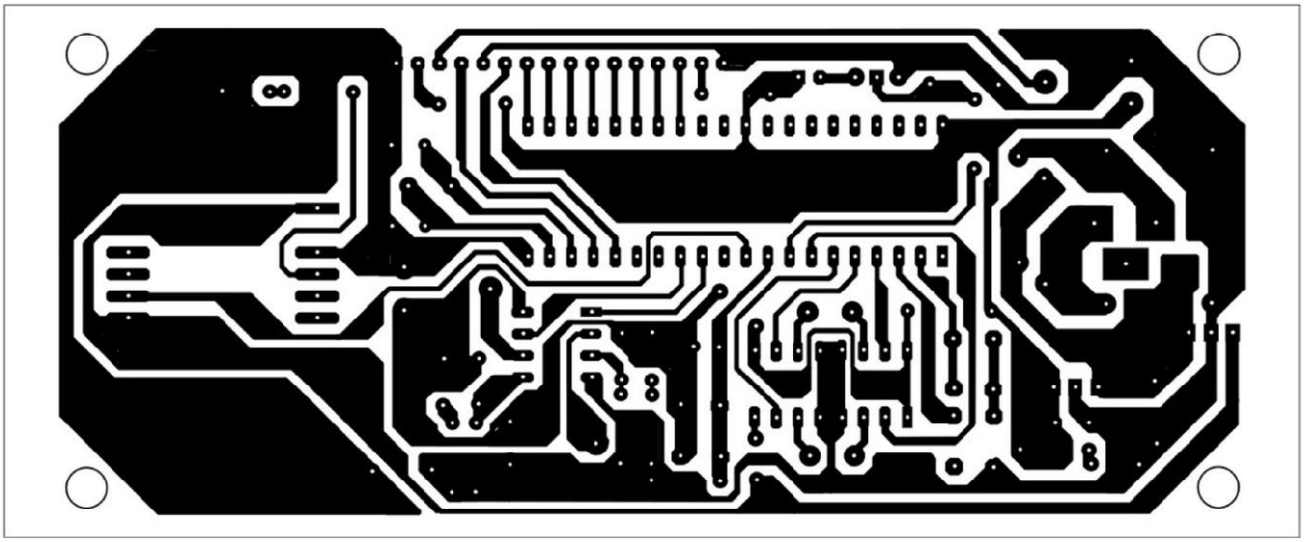


Fig. 7: An actual-size, single-side PCB for the RFID-based automatic vehicle parking system

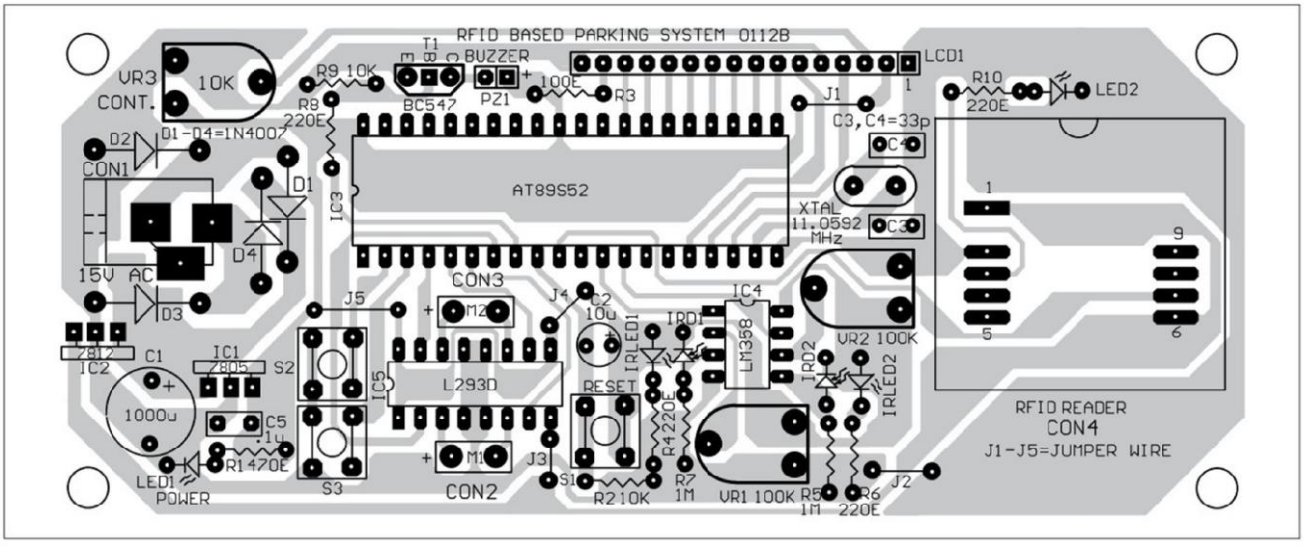


Fig. 8: Component layout for the PCB

the vehicle with the parking owner and get the RFID tag. When the car has to be parked, the RFID tag is placed near the RFID reader, which is installed near the entry gate of the parking lot. As soon as the RFID tag is read by the reader, the system automatically deducts the specified amount from the RFID tag and the entry gate boomer opens to allow the car inside the parking area. At the same time, the parking counter increments by one. Similarly, the door is opened at the exit gate and the parking counter decremented.

The system also offers the facility to recharge the amount for each RFID tag. No manual processing is involved. In addition, the system provides security.

Circuit description

Fig. 5 shows the circuit of the RFID-based automatic vehicle parking system. The circuit can be divided into different sections:

Power supply. Connector CON1 (refer Fig. 8), diodes D1 through D4, capacitor C1, and voltage regulator ICs 7805 (IC1) and 7812 (IC2) form the power supply section of the automatic vehicle parking system. CON1 is a three-pin connector that provides 15V AC or DC power supply to the circuit. In case of 15V AC, diodes D1

through D4 form a bridge rectifier to rectify the AC supply. Capacitor C1 filters out the ripples from the rectified output. ICs 7805 and 7812 provide regulated +5V and +12V, respectively, to the circuit. +5V is used to operate the microcontroller, LCD, RFID and IR sensor circuit and +12V operates the motor.

AT89S52 microcontroller. AT89S52 is a low-power, high-performance CMOS 8-bit microcontroller with 8kB Flash memory. It is compatible with the industry-standard 80C51 instruction set and pin-out. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional non-volatile memory programmer. Other features include 256 bytes of RAM, 32 input/output lines, watchdog timer, two data pointers, three 16-bit timers/counters, a six-vector two-level interrupt architecture, a full-duplex serial port, on-chip oscillator and clock circuitry.

Connectors CON2 through CON4. CON2 and CON3 are two-pin connectors that connect the 12V DC motors to the circuit for controlling the entry and exit gate boomers. CON4 is a ten-pin dual-in-line female connector that connects the RFID reader module to the circuit.

L293D motor driver. H-bridge DC motor driver L293D (IC5) operates the DC motors to open the door or barrier for entry into and exit from the parking lot. Two high-current motor drivers can be used in place of L293D and 12V DC motors to control the entry and exit gates, respectively.

LM358 op-amp. Dual-operational amplifier LM358 (IC4) is used as a voltage comparator to compare the output of the IR sensors with a fixed threshold voltage in order to know whether the IR beam is interrupted or not.

IR transmitter and receiver. Two IR transmitter-receiver pairs are used. The IR LEDs are connected in forward-biased condition to the +5V power supply through 220-ohm resistors. These emit IR light, which is interrupted when an object comes into its way to the IR receiver. The IR receiving photodiodes are connected in reverse-biased condition to +5V power supply through 1-mega-ohm resistors. When the IR light falls on the photodiodes, their resistance changes and so does their output. This output is compared with a fixed voltage to give a digital output to the microcontroller in order to judge the entry and exit of the vehicles.

PARTS LIST

Semiconductors:

IC1	- 7805, 5V regulator
IC2	- 7812, 12V regulator
IC3	- AT89S52 microcontroller
IC4	- LM358 dual-operational amplifier
RFID reader	- 9-pin 125kHz RFID reader
LED1, LED2	- 5mm light-emitting diode
IR LED1, IR LED2	- 5mm infrared transmitter diode
IRD1, IRD2	- 5mm infrared receiver diode
D1-D4	- 1N4007 rectifier diode
T1	- BC547 transistor

Resistors (all ¼-watt, ±5% carbon):

R1	- 470-ohm
R4, R6, R8, R10	- 220-ohm
R2, R9	- 10-kilo-ohm
R3	- 100-ohm
R5, R7	- 1-mega-ohm
VR1, VR2	- 100-kilo-ohm preset
VR3	- 10-kilo-ohm preset

Capacitors:

C1	- 1000µF, 35V electrolytic
C2	- 10µF, 16V electrolytic
C3, C4	- 22pF ceramic
C5	- 0.1µF ceramic

Miscellaneous:

LCD1	- 16×2 LCD display module
CON1	- DC connector
CON2, CON3	- Two-pin berg strip male connector
CON4	- 9-pin berg strip female connector
PZ1	- Piezobuzzer
X_{TAL1}	- 11.0592MHz crystal
S1-S3	- Tactile switch
M1, M2	- DC motor
X1	- 230V AC primary to 15V, 500mA secondary transformer

LCD display. LCD1 is a two-line, 16-character, alpha-numeric liquid crystal display. Data lines D0 through D7 of the LCD are connected to port 2 of AT89S52 (IC3). Reset (RS) and enable (E) control lines are connected to port pins P3.6 and P3.7, respectively. Control lines control data flow from the microcontroller to LCD1.

When power is switched on, LED1 glows to indicate the presence of power in the circuit and LED2 glows to indicate the presence of RFID reader. Simultaneously, the 'Automatic RFID Car Parking' message is displayed on LCD1 along with a short beep from piezobuzzer PZ1. Transistor BC547 drives the buzzer. Pin details of 7805, 7812 and BC547 are shown in Fig. 6.

When a car crosses the IR LED1-D1 pair installed at the entry gate, the gate boomer does not open until an RFID tag is placed near the RFID reader. After the tag is placed near the reader, the gate boomer opens for three seconds and closes automatically. If the initial recharge amount was Rs 900, the LCD display shows 'Vehicle1 Amount' in the first line and 'Deducted 100' in the second line, followed by 'Balance Amount' in the first line and '800' in the second line. It is then followed by display of 'Number of Cars' in the first line and '001' in the second line. If the parking lot is full, the message "Parking is Full, Sorry for Inconvenience" is displayed on LCD1.

When a car leaves the parking area and crosses the IR beam between IR LED2 and D2 at the exit gate, the vehicle count decreases by one. The LCD shows the number of cars in the parking lot along with "Thanks for Visiting" message.

Software

The program (parking.c) for the microcontroller is written in C and compiled using Keil software to generate the hex code. The program coding starts with '#include<reg51.h>' and '#include<string.h>' header files. The microcontroller port pins are defined using 'sbit' function for interfacing with the surrounding peripherals. The entry gate motor is controlled using 'sbit START_POINT=P1^3;' code.

The LCD is initialised using the following code:

```
void lcdinit(void)
{
lcdcmd(0x38);
DelayMs(250);
lcdcmd(0x0E);
DelayMs(250);
lcdcmd(0x01);
DelayMs(250);
lcdcmd(0x06);
DelayMs(250);
lcdcmd(0x80);
DelayMs(250);
}
```

Construction and testing

An actual-size, single-side PCB layout for the RFID-based automatic vehicle parking system is shown in Fig. 7 and its component layout in Fig. 8. Burn the hex code into the AT89S52 microcontroller using a suitable programmer and then mount the microcontroller on the PCB. Install IR LED1-D1 pair at the entry gate such that these face each other. Similarly, install IR LED2-D2 pair at the exit gate.

For testing, switch on the circuit, interrupt the infrared beam between IR LED1 and IR D1 with your hand or some other opaque object and then remove it, and place the tag near the reader. The LCD should show the message as described earlier in 'How this vehicle parking system works' section. An amount of Rs 100 should be deducted for every interruption of the IR beam. The card can be recharged by pressing the pushbutton switches (S2 and S3) provided in the circuit. Pressing switch S2 recharges the card with Rs 900 and pressing switch S3 recharges it with Rs 500.

Similarly, interrupt the IR beam at the exit gate. LCD1 should show the number of cars in the parking lot along with 'Thanks for Visit' message. No amount should be deducted at the time of exit.

Download source code: http://www.efymag.com/admin/issuepdf/Const-1_RFID%20Based%20Automatic%20Vehicle%20Parking%20System.zip

PARKING.C

```
//Automatic RFID Based Vehicle Parking System
#include<reg51.h>
#include<string.h>

sbit START_POINT=P1^3;
sbit TERMINATE_POINT=P1^4;

sbit S1=P3^2;
sbit S2=P3^3;

sbit rc1=P1^1;
sbit rc2=P1^2;

sbit BUZZPORT=P1^7;

sbit RS=P3^7;
sbit EN=P3^6;

void lcdinit();
void lcdData(unsigned char l);
void lcdcmd(unsigned char k);
void buzzer(unsigned int time);
void DelayMs(unsigned int count);
void display(unsigned char s,t,u);
void Welcome(unsigned char c[],unsigned char d[]);
void  ConvertAndDisplay(unsigned  int  value1,unsigned
char c[]);
void dcMotor();

void main()
{
unsigned char i=0,j=0,result=0;
unsigned int count;
unsigned char c[15];
unsigned char d[]="42006B1BB8";

signed int amount2=900;
```

```
TMOD=0x20;                          // To configure the se-
rial port at 9600 baud rate
TH1=0xFD;
SCON=0X50;
TR1=1;

S1=1;
S2=1;

BUZZPORT=0;
START_POINT=0;
TERMINATE_POINT=0;

lcdinit();
Welcome("AUTOMATIC RFID","CAR PARKING");
DelayMs(1000);

while(1)
{
known:
while(S1==1 && S2==1);

if(S2==0)
{
                    while(S2==0);
                    if(count>0)
                    {
                     count--;

                     ConvertAndDisplay(count,"Thanks for
Visit");

                     DelayMs(1000);

                     goto known;

               }
                    else
                    {
                      count=0;
                        ConvertAndDisplay(count,"Thanks
for Visit");
                    DelayMs(1000);

                                        goto known;

               }

}

if(S1==0)
{
while(S1==0);
for(i=0;i<12;i++)
{
c[i]=0xFF;
}

while(RI==0);

for(i=0;i<12;i++)                       //command to
recv data
{
j=0;
while(RI==0)
{
if(j>=1000)
goto timeout;
DelayMs(1);
j++;
}
c[i]=SBUF;
RI=0;
}
timeout:

i=strncmp(c,d,10);
if(result==0 && count<10)
{
lcdinit();

Welcome("VEHICAL1 Amount ","Detected:100");
DelayMs(1000);
amount2-=100;
if(amount2>0)
{
ConvertAndDisplay(amount2,"Balance Amount");
dcMotor();
}
else
{
  amount2=0;
  Welcome("VEHICAL1 Amount ","BALANCE NIL");
  buzzer(500);
  Welcome("Recharge Your","Card Please");
  while(rc1==1&&rc2==1);
  {                 if(rc1==0)
          {
                    while(rc1==0);
                    amount2=900;
                    Welcome("Card  is  recharged","with
amount 900");
    }
  if(rc2==0)
  {
          while(rc2==0);
          amount2=500;
          Welcome("Card  is  recharged","with  amount
500");
  }
}
}
if(amount2==400)
{
  Welcome("VEHICAL1 Amount ","BALANCE LOW");
  buzzer(200);

}
DelayMs(1000);

count++;
lcdcmd(0x01);
DelayMs(10);
ConvertAndDisplay(count,"Number of Cars");
goto known;
}
 else
 {

                    Welcome("Parking is Full","Sorry for
Inconvenience");
                    buzzer(500);
                    DelayMs(1000);
                    goto known;
 }
}
}
}

void Welcome(unsigned char c[],unsigned char d[])
{
unsigned int i=0;

lcdcmd(0x01);
DelayMs(10);
lcdcmd(0x80);
DelayMs(10);

i=0;
while(c[i]!='\0')
{
lcdData(c[i]);
i++;
}

lcdcmd(0xc0);
```

```
DelayMs(10);

i=0;
while(d[i]!='\0')
{
lcdData(d[i]);
i++;
}
}

void  ConvertAndDisplay(unsigned  int  value1,unsigned
char c[])
{
unsigned int i,a=0,j;
unsigned char d1,d2,d3;
for(i=0;i<value1;i++)
a=a+1;
lcdcmd(0x01);
DelayMs(10);
lcdcmd(0x80);
DelayMs(10);


j=0;
while(c[j]!='\0')
{
lcdData(c[j]);
j++;
}
d1=a%10;  //digits before desible point
a=a/10;
d2=a%10;
a=a/10;
d3=a%10;

lcdcmd(0xc0);
DelayMs(10);

display(d1,d2,d3);
}
//--------------------------------------
// Lcd initialization subroutine
//--------------------------------------
void lcdinit(void)
{
lcdcmd(0x38);
DelayMs(250);
lcdcmd(0x0E);
DelayMs(250);
lcdcmd(0x01);
DelayMs(250);
lcdcmd(0x06);
DelayMs(250);
lcdcmd(0x80);
DelayMs(250);
}

//--------------------------------------
// Lcd data display
//--------------------------------------
void lcdData(unsigned char l)
{
P2=l;
RS=1;
EN=1;
DelayMs(1);
EN=0;
return;
}

//--------------------------------------
// Lcd command
//--------------------------------------
void lcdcmd(unsigned char k)
{
P2=k;
RS=0;
EN=1;
DelayMs(1);
EN=0;
return;

}

//--------------------------------------
// Delay mS function
//--------------------------------------
void DelayMs(unsigned int count)
{  // mSec Delay 11.0592 Mhz
    unsigned int i;          // Keil v7.5a
    while(count)
          {
        i = 115;                        // 115   exact
value
                    while(i>0)
                    i--;
        count--;
    }
}

void dcMotor()
{
                    START_POINT=1;
                    TERMINATE_POINT=0;
                    DelayMs(400);
                    START_POINT=0;
                    TERMINATE_POINT=0;
                    DelayMs(2000);
                    START_POINT=0;
                    TERMINATE_POINT=1;
                    DelayMs(400);
                    START_POINT=0;
                    TERMINATE_POINT=0;

}

void display(unsigned char s,t,u)
{
s=s+0x30;                    //convert  each  digit  to
equivalent ASCII value
t=t+0x30;
u=u+0x30;

//Move the cursor to position 5 on LCD
DelayMs(50);

lcdData(u);                             //Display  the
digits one by one on LCD
DelayMs(50);
lcdData(t);
DelayMs(50);
lcdData(s);
DelayMs(50);
}

void buzzer(unsigned int time)
{

          BUZZPORT=1;

          DelayMs(time);

          BUZZPORT=0;
}
```

MICROCONTROLLER-BASED SCIENTIFIC CALCULATOR

■ BODHIBRATA MUKHOPADHYAY

A scientific calculator gives you quick access to certain mathematical functions. Basically, it's an electronic calculator designed to solve problems in science, engineering and mathematics.

Here we present a microcontroller-based scientific calculator. Its main features are:

1. Out of the 20 switches on the 5×4 matrix keypad, seven are for mathematical operations, performing 17 mathematical functions. That is, one switch can perform more than one mathematical function; for example, a single switch for sine, cosine and tangent functions, and a single switch for log and ln functions.

2. It performs operations on signed numbers.

3. The calculator accepts four digits for integer place and two digits for decimal place. The output has a total of eight digits for integer and decimal places but the maximum number of digits in decimal place is limited to four.

4. The program is written such that inputting wrong data is made difficult. For example, while inputting an integer, it will not accept any

PARTS LIST

Semiconductors:

IC1	- PIC18F4580 microcontroller
IC2	- 7805 5V regulator

Resistors (all ¼-watt, ±5% carbon):

R1	- 10-kilo-ohm
R2-R5	- 1-kilo-ohm
R6	- 100-ohm
VR1	- 10-kilo-ohm preset

Capacitors:

C1, C2	- 22pF ceramic disk
C3	- 0.1µF ceramic disk
C4	- 2.2µF, 16V electrolytic

Miscellaneous:

S1-S21	- Push-to-on tactile switch
S22	- On/off switch
LCD	- 16×2 line LCD module
X_{TAL}	- 20MHz crystal
BATT.	- 9V battery

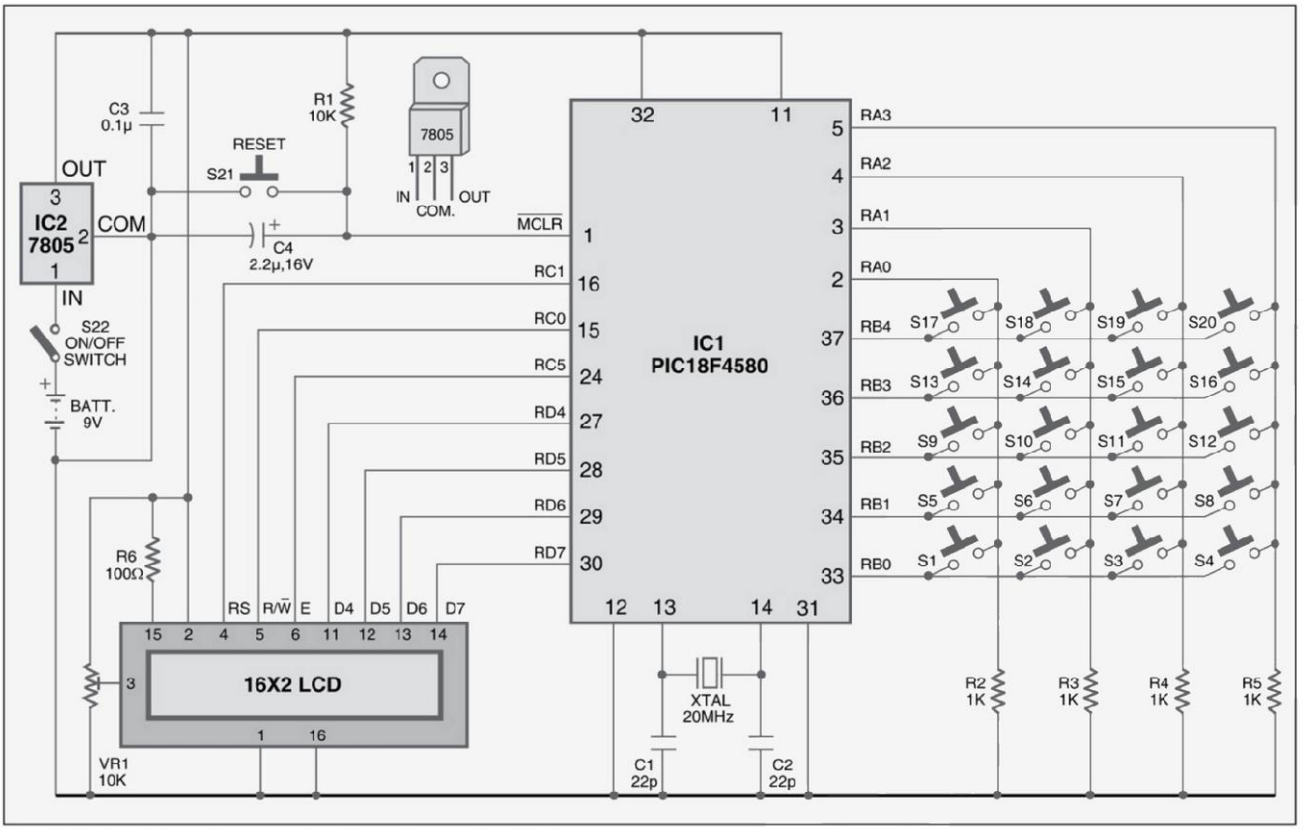


Fig. 1: Circuit of the microcontroller-based scientific calculator

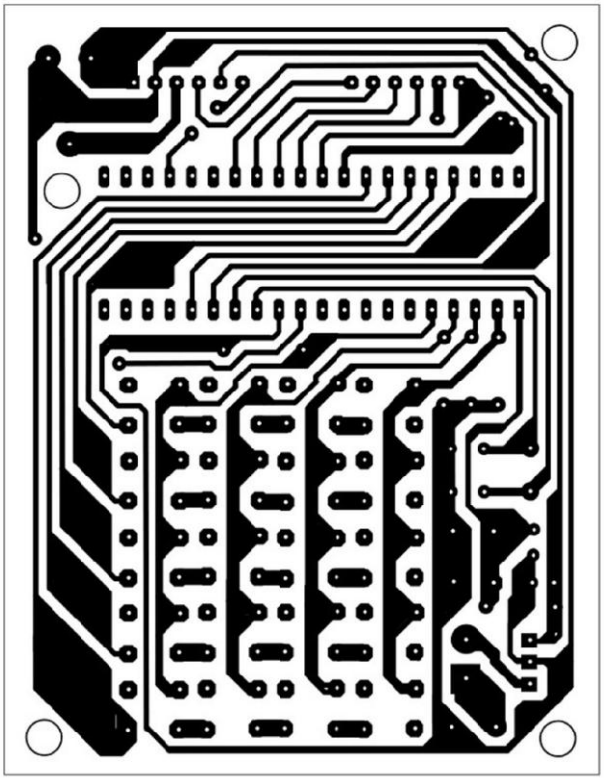


Fig. 2: An actual-size, single-side PCB for the microcontroller-based scientific calculator

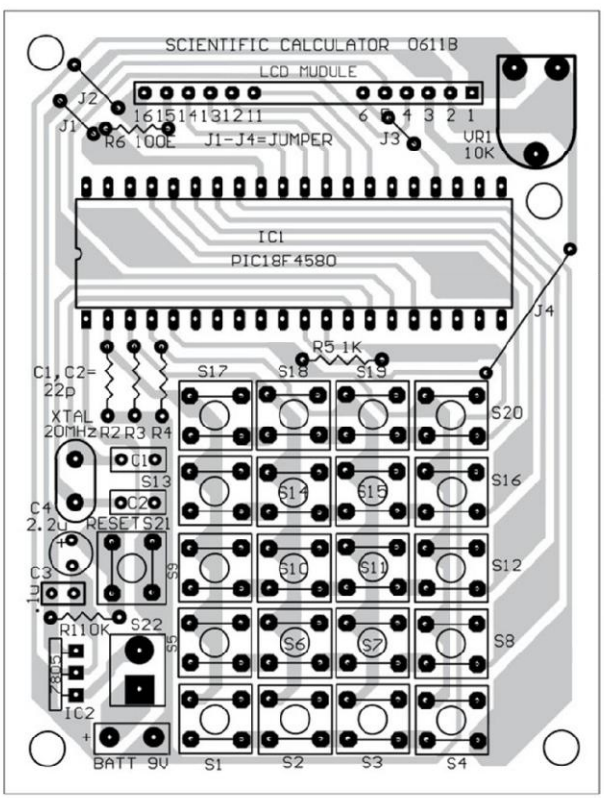


Fig. 3: Component layout for the PCB

operator function even if you press the operator switches. Error will of course occur if the answer is out of range.

Circuit description

Fig. 1 shows the circuit of the microcontroller-based scientific calculator. Microcontroller PIC18F4580 is at the heart of this calculator. It has 32 kB of enhanced Flash programmable memory, five bidirectional input/output ports, four timers, 11-bit analogue-to-digital converter, two comparators and is also capable of serial communication. It offers different oscillator options to provide the basic clock frequency.

Power-on reset is provided by the combination of resistor R1 and capacitor C4. Switch S21 is used for manual reset. A 20MHz crystal connected to pins 13 and 14 provides the basic clock to the microcontroller. Port pins RD4 through RD7 of the microcontroller are connected to data port pins D4 through D7 of the LCD module, respectively.

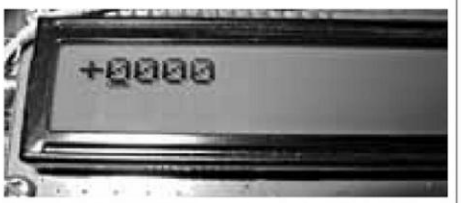
The microcontroller drives the LCD in 4-bit mode and sends the data in two nibbles (4-bit parts) for display on the LCD. Port pins RC0, RC1 and RC5 are connected to read/write (R/$\overline{W}$), register-select (RS) and enable (E) of the LCD, respectively, to control the LCD operation. Preset VR1 is used for contrast control of the LCD.

The 5×4-matrix keypad is interfaced to microcontroller PIC18F4580 through its port pins. Its five rows are connected to five pins of Port B (RB0 through RB4) and four columns to four pins of Port A (RA0 through RA3). Key debounce technique is used to eliminate false key-press detection. The microcontroller scans the keypad continuously to detect and identify the key pressed by the user. Eight out of 20 keys (S1 through S20

Keypad Details

Switch	Function
S1	1
S2	2
S3	3
S4	4
S5	5
S6	6
S7	7
S8	8
S9	9
S10	0
S11	+/-
S12	AC
S13	ADD/SUB
S14	MUL/DIV
S15	POW/SQR/SQRT
S16	INV./EXP.
S17	EXC
S18	LOG/LN
S19	SIN/COS/TAN
S20	ASIN/ACOS/ATAN

(a) Initial stage of the LCD



(b) The sign bit 'S' is replaced with '+' sign

(c) After the first operand is inputted the word 'operator' shows up

(d) The operator is division '/', so it requires the second operand

(e) -23 is the second operand data. The letter 'E' indicates the user to press the execute button now

(f) The result displayed

Fig. 4: User manual of the calculator for division of two numbers

switches) can perform more than one function at a time.

IC 7805 (IC2) provides regulated 5V to the circuit. Capacitor C3 bypasses ripples, if any, in the regulated supply.

An actual-size, single-side PCB for the microcontroller-based scientific calculator is shown in Fig. 2 and its component layout in Fig. 3. Assembling the circuit on a PCB minimises time and assembly errors. Carefully assemble the components and double-check for any overlooked error. Use bases to avoid damage to ICs due to direct soldering and over-heating.

Functions of different keys

Functions assigned to different switches on the keypad are shown in the table. Fig. 4 shows the user manual of the calculator for division of two numbers.

The '+/–' key is used to give sign to the number. It works only when the cursor is below letter 'S' displayed on the LCD. If it is pressed once '+' sign is displayed, and if it is pressed twice '-' sign is displayed.

Digits 0 through 9 on the keypad have the same functions as in a normal calculator. You can enter digits only when the cursor is below digit '0' displayed in the LCD. When you press a number key, '0' displayed on the LCD changes to the number pressed.

After the first data is inputted, the calculator shows word 'operator' in the second line of the LCD. There are seven operator keys. When an operator key is pressed once, its first operator value is displayed. When it is pressed again, the second operator value is displayed. Pressing the key for the third time displays the third operator (if it is present) or again the first operator. So by repeatedly pressing a key, the mathematical operator on the LCD can be changed. The user can enter an operator by pressing the respective key as many times as needed and then leave the keypad for a finite time gap. After that time, the calculator will accept only the operator shown on the LCD and reject any other operator from the keypad.

If the mathematical operator requires only one operand, wait for some time after setting the operator. An 'E' will be displayed in the second row of the LCD. Press the execution key to see the result. If the mathematical operator requires two numbers, you will need to enter the second operand after setting the operator. After inputting the second operand, wait for some time until letter 'E' is displayed in the second row of the LCD. Press the execution key to see the result.

If the answer to any mathematical operation goes out of range, the LCD shows 'Error' message.

The 'AC' key can be pressed at any time to clear the display and reset the whole system.

Software

The source program of this scientific calculator is written in 'C' language and compiled using MPLAB IDE. The program is well commented and easy to understand. MPLAB IDE and MPLAB C18 must be installed before compilation of the program. Fig. 5 shows the configuration

Fig. 5: Configuration bits setting of microcontroller PIC18F4580

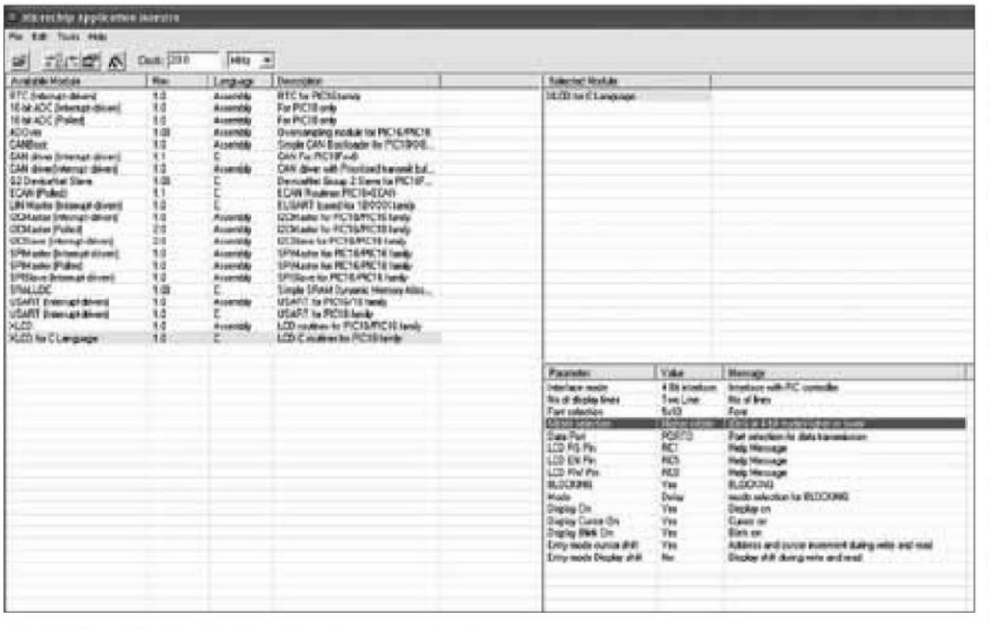


Fig. 6: Microchip Application Maestro screenshot

bits setting of microcontroller PIC18F4580. Set the configuration bits before compiling the source program and check before burning the code into the microcontroller.

A software called 'Microchip Application Maestro' is used to generate the xlcd.h and xlcd.c files to interface the LCD. In the main program, only library routines have been called for writing to the LCD. The Microchip Application Maestro Modules is available on website 'http://ww1.microchip.com/downloads/en/DeviceDoc/mpam-modulesv103.zip'. After installation, run the Microchip Application Maestro application and select 'XLCD for C Language' (refer Fig. 6).

Configure the LCD by selecting different parameters like interfacing mode, data port and control pins (RS, R/W, E). After the configuration is done, click 'Generate Code' icon. Save the generated files in the same directory as the source program MPLAB. Both the xlcd.c and xlcd.h files are added in the workspace of MPLAB. 'XLCDc.readme' and 'XLCD.DEF' are also generated. 'XLCDc.readme' contains names of all the library routines. Before compiling the program, some delay subroutines need to be written in the xlcd.c files in order to add delays required for the working of the LCD.

The program works as per the flow-chart shown in Fig. 7.

In the program, header files for 'timers,' 'delays,' 'lcd' and 'math' have been used. First, configure Port A pins as digital input. The LCD is initialised for 4-bit mode, display on, cursor on, blink off, etc. The first operand is displayed in the first line of the LCD with the help of the function 'input_number.' The word 'operator' displayed in the second line of the LCD means that the program is waiting for an operator from the user. Pressing an operator switch displays the operator on the LCD. At this time, no switch other than the operator switch will work.

Operator switches are time-multiplexed. That means an operator switch allows you to perform more than one function. When an operator switch is pressed once, it displays the first operator function. Pressing it again soon after displays another operator at the same position of the LCD. A switch can have two-three operators.

After you select the desired operator, leave the keypad. After a small period of time, the program will understand which operator has been selected. Flag variable 'f' is used to find out the number of times a switch is pressed. Its initial value is 0. When an operator key is pressed, the flag variable's value changes from 0 to 1. If the same switch is pressed again, the program comes to know through the flag variable that the same switch is pressed for the second time and changes the flag variable value from 1 to 2, depending on the number of operators related to that switch. Now the LCD shows the second operator function of the switch.

A number is given to each operator to understand which operation is selected or which operation has to be performed. This number is stored in variable 'ope.' For example, '+' has 'ope' value of 1, '-' has 'ope' value of 2, 'log' has 'ope' value of 10 and so on. When the program accepts the operator, the word 'operator' in the second line of the LCD is erased.

Now the program checks whether the given mathematical operator requires another operand or not. If it requires another operand, the above process repeats. The only difference is that now the second line of the LCD, not the first line, is used for inputting the data.

After the operand and the operator are entered, the program takes some time to read the data from the LCD and then perform mathematical operation on it. It checks the value of 'ope' to know which mathematical operation has to be performed. After performing the mathematical operation, it shows letter 'E' on the right side of the second line of the LCD. Now if you press the execution key, the result is shown in the second line of the LCD. Pressing the execution key before letter 'E' is displayed in the LCD will not give any result.

If 'AC' key is pressed while executing the code, the LCD gets cleared and the program starts executing from the beginning.

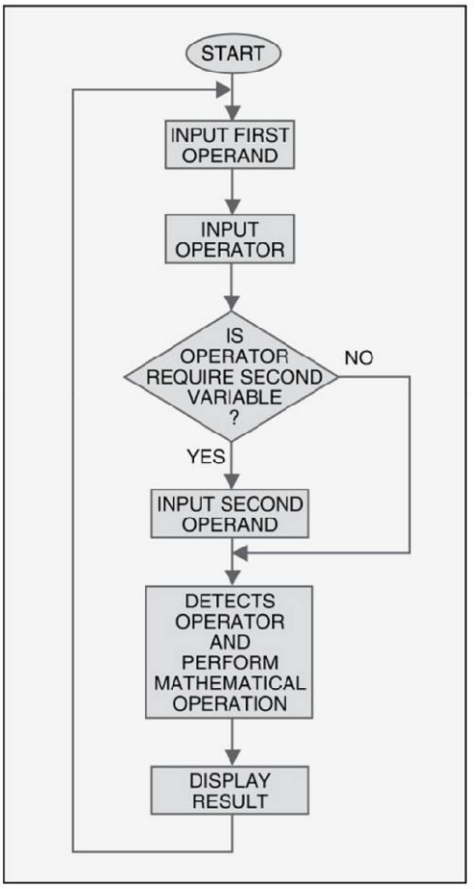


Fig. 7: Flow-chart of working of scientific calculator

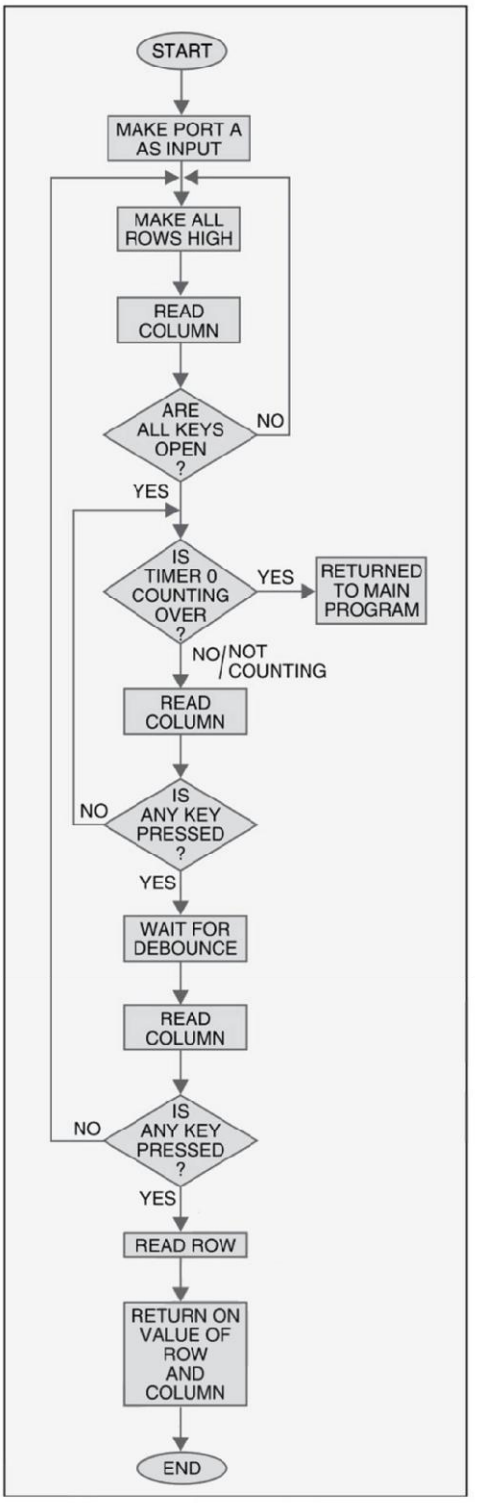


Fig. 8: Flow-chart of keyboard subroutine

(a) Square of a number

(b) Ln of a number

(c) Tan inverse of a number

Fig. 9: User manual of the calculator for square, natural log (Ln) and tan inverse

After letter 'E' is shown on the LCD, only 'EXC' and 'AC' keys work.

Keyboard subroutine. This subroutine uses the keys debounce technique and works as per the flow-chart shown in Fig. 8. When the keyboard function is called, the program waits for any key to be pressed and returns to its position (column and row values). The value of the row and the column matches with the value of the 5x4 matrix of the matrix array 'key_pad [5][4]' declared in the program.

Row reading.

1. Starting with the top row, the microcontroller makes each row high at a time and then reads the columns. If the data read is all zeros, no key in that row is activated and the program moves to the next row.
2. It makes the next row high, reads the columns and checks for any high.
3. This process continues until the row having a key pressed is identified, i.e., reading of the column returns a non-zero value.
4. After identification of the row in which the key has been pressed, the program finds out which column the pressed key belongs to.

Column reading. The microcontroller reads the data at Port A and masks its lower bit.

Variables 'row' and 'column' are global, and they can be used from anywhere in the program. For reading a switch which has more than one operation, timer1 is started before entering the function 'keyboard( ).' The program goes on reading the keyboard until timer1 stops counting. When timer1 finishes counting, it returns from the function.

Key multiplexing is done with the help of the timer. At first, the timer is initialised to 16 bits and 'TMR0IF' is set to 0. The timer starts counting only when a value is written to it. After it finishes counting, 'TMR0IF' becomes 1. This timer is used only for those keys which perform mathematical operation. For keys other than operator keys, it does not start counting. But at the press of any operator key, 'WriteTimer0(50000)' is used to write value to the timer and the timer starts counting. If the same key is pressed again before the timer completes counting, the second value of the key gets accepted and the timer value is again set to 50000. The process goes on. This way, key multiplexing is done and a single key can perform more than one operation.

'Read from LCD' subroutine. This subroutine reads numbers from the LCD and returns them as signed, double-data type. The cursor goes to the LCD location (given by the user) from where the data is to be read. The program reads the number, sign and decimal point from the LCD one by one, and stores them in string array 'ch[i].' Then it converts the string array into signed, double-data type 'ans' using function 'atof( )' and returns the value of 'ans.' Now the microcontroller has the number that was typed into the LCD and any mathematical operation can be performed on it.

'Write result to LCD' subroutine. This subroutine writes the result to the LCD. It receives the LCD location where the data is to be written, the signed floating point number and the number of places to be displayed after the decimal point.

This function also displays word 'Error' on the LCD if the result of any mathematical operation is out of range or mathematically incorrect.

Input number subroutine. This function takes operand inputs for mathematical calculations. For mathematical operations requiring two operands (like addition and subtraction), it is executed twice. For mathematical operations requiring a single operand (like log), it is executed only once. It first takes the input of the sign bit, then the integer part followed by the decimal part.

The function uses time multiplexing of the sign (+/–) switch with the help of timer0. When '+/–' switch is pressed once, the LCD shows '+'. When it is pressed again within a short period of time, the LCD shows '-'. Pressing it further shows '+'.

After reset, 'S0000' is shown on the LCD. Initially, the cursor is at 'S.' At this time, no switch other than '+/-' works. This '+/-' switch is time-multiplexed, and either '+' or '–' can be displayed at the same location of the LCD.

As you input '+' or '–' from the keypad, the cursor moves to the next position. The four zeros (0000) allow you to input the 4-digit number from the keypad. The digits are taken one by one. Then '.00' is displayed on the LCD, allowing you to input digits in decimal places as well. Pressing the 'AC' key while executing the code clears the LCD and the program starts executing from the beginning.

Fig. 9 shows the user manual of the calculator. Fig. 10 shows the author's working prototype.

Download source code: http://www.efymag.com/admin/issuepdf/Microcontroller%20Based%20Scientific%20Calculator.rar

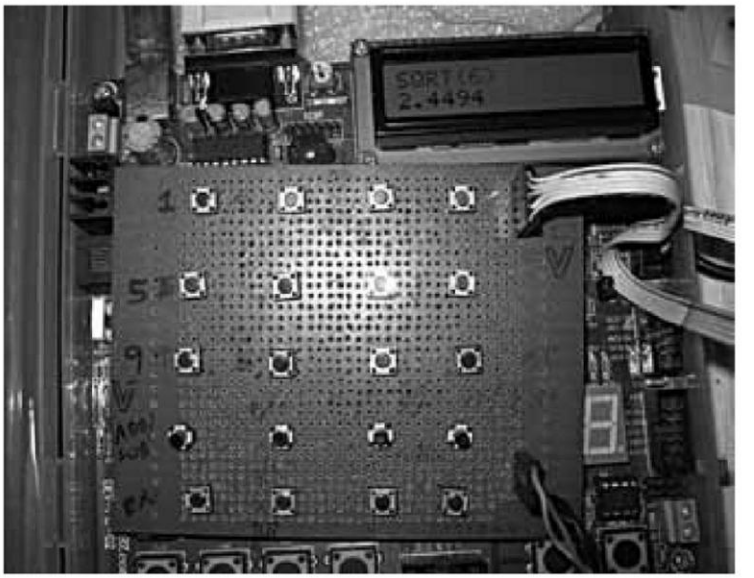


Fig. 10: Working prototype of scientific calculator

ARDUINO BASED VEHICLE PARKING COUNTER

■ VINAY CHADDHA

This vehicle counter counts the number of cars and the vacant space available in a parking lot and shows the values on a three-digit dual-colour display. The number of cars is shown in red colour and the space available in green colour.

The circuit design is the same as for the 'Traffic Light Count-Down Timer with Dual-Colour Display' project described under Measurement Section. So if you already have this project, you don't have to buy or assemble a new Arduino board. Just by burning the parking_counter code given in this article into the ATmega328 microcontroller, the functioning of the board can be changed to a vehicle parking counter. For parts-list, circuit description and AVR programming using Arduino IDE, refer to Page No. 264.

Fig. 1 shows the block diagram of the Arduino based vehicle parking counter.

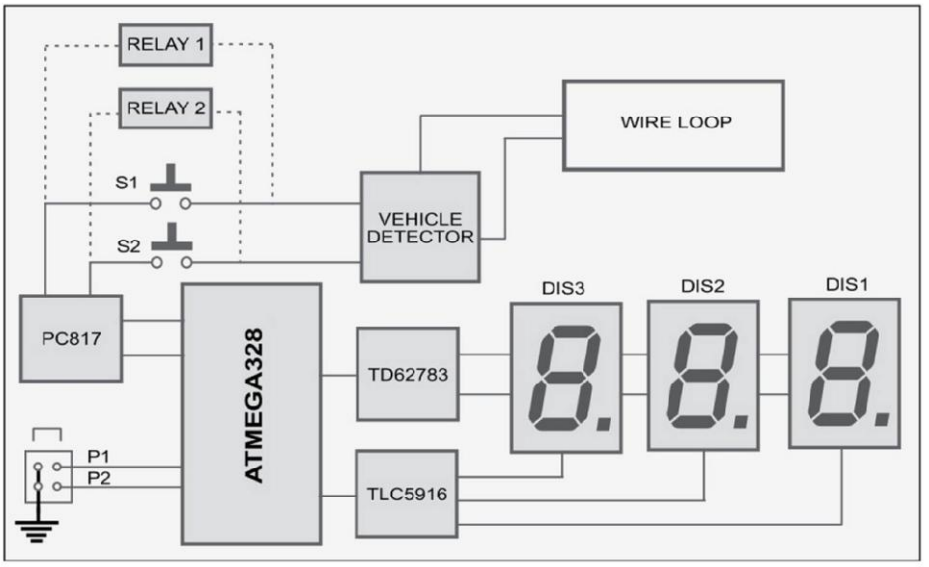


Fig. 1: Block diagram of Arduino based vehicle parking counter

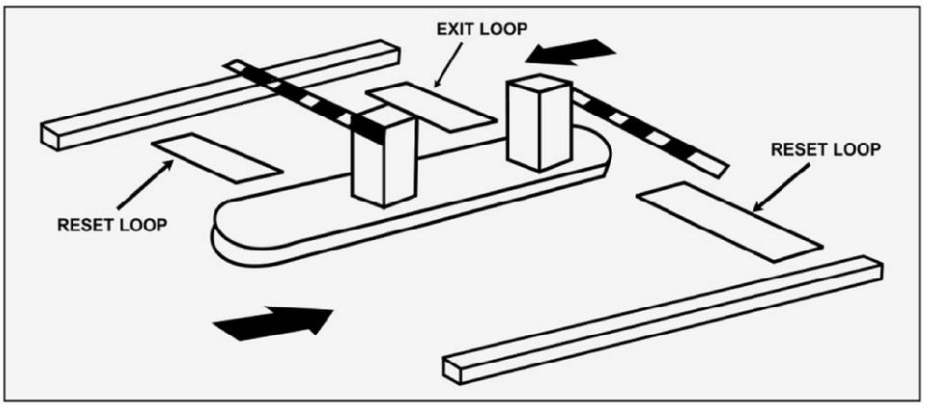


Fig. 2: A typical two-way gate system with boom barriers

Vehicle loop detector

Almost all parking lots in buildings or malls use loop detectors buried inside the road to detect the movement of vehicles crossing the gates. They detect the vehicles but ignore the gate crossing by humans, bicycles, etc. Usually, these detectors are part of electronic oscillator circuits. The loop detector circuitry and its description are not included here. However, its basic concepts are described here to give you some idea about how the loop detector works along with the vehicle parking counter.

A typical two-way gate system with boom barriers is shown in Fig. 2. When you approach a vehicle parking boom gate, you may notice a rectangular scar where the road surface has been cut with a saw and then re-sealed. This is the sensor loop. The loop consists of one or more turns of wire buried in the roadway and connected to an electronic circuit which can detect a vehicle passing over it.

Jumper Settings for Different Modes of Operation

Mode	Display	Jumper P1	Jumper P2	Display functions
Setting	**Red**	Open	Close	Setting the parking lot capacity
Vehicle-In	**Red**	Open	Open	Showing the number of vehicles inside the parking
Vacancy	**Green**	Close	Open	Showing the space available in the parking lot

The boom barrier is raised by pressing a switch to allow vehicles cross the gate. The boom barrier automatically lowers down after the vehicle crosses the gate.

Fig. 3: Typical preformed loop with extension cable

Vehicle loop detector connections to parking counter

Fig. 3 shows a typical preformed loop with extension cable. The two ends of the loop wire are connected to the loop extension cable, which, in turn, connects to the vehicle detector consisting of an electronic circuitry enclosed in a box.

The detector powers the loop causing a magnetic field in the loop area. A base frequency is established when there is no vehicle over the loop. When a vehicle crosses the loop, the resonant frequency of the loop increases. This increase in frequency is sensed and, depending on the design of the detector, causes a normally-open relay to close. The relay will remain closed until the vehicle leaves the loop and the frequency returns to its base level. The relay activates the control devices such as an audio intercom system, gate motor and vehicle counter.

Fig. 4: A typical dual-channel loop detector

Fig. 4 shows a typical dual-channel loop detector. It has two relays. One relay energises when the vehicle enters the parking area and the other energises when the vehicle exits the parking lot. The normally-open (N/O) terminals of these relays are connected to the parking counter.

Circuit operation

In case you do not have access to loop detectors, the circuit can be tested using switches S1 and S2 in place of the relays (see Fig. 1).

There are three modes of operation, which can be selected using jumpers P1 and P2 as shown in the table.

When jumper P1 is kept open and P2 is closed to ground, the mode of operation is 'setting.' Set the parking lot capacity (say, 999) using switches S1 and S2, wait for ten seconds and switch off the circuit. The parking space capacity is now stored in the internal EEPROM of ATmega328. Setting mode is required only once during installation in a new parking lot.

When jumpers P1 and P2 are kept open, the circuit operates in vehicle counting mode. Here the display shows the number of vehicles inside the parking lot. When the vehicle count reaches 90 per cent of the maximum capacity of the parking lot, the display will start blinking. So if the maximum count is set at 60, as soon as the vehicle count is 54 or more the display starts blinking. This feature can be used to put 'parking full' sign and stop entry of vehicles except the priority vehicles.

When jumper P1 is closed and jumper P2 is opened, the circuit operates in vacancy counting mode. The display turns green to shows the total number of vacant spaces available in the parking lot.

Installation of the parking counter

One vehicle detector is installed at the exit gate and the other at the entry gate. Connections are simple. Switch S1 is connected to N/O contact of the relay at the exit gate and S2 to N/O contact of the relay at the entry gate.

Every time a vehicle crosses the entry gate, the count increments by 1, and when a vehicle crosses the exit gate the count decrements by 1.

During normal operation, jumpers P1 and P2 should be kept open. These may be replaced with suitable push-to-on switches during installation. The operator can use jumpers P1 and P2 as per the modes listed in the table to check the capacity and space available in the parking lot.

Download source code: http://www.efymag.com/admin/issuepdf/Arduino%20Based%20Parking%20Counter.rar

EIGHT-CHANNEL DATA ACQUISITION & LOGGING SYSTEM

■ DEVESH SAMAIYA

In environments like factories, power plants and transformers in electricity substations, controlling temperature to a safe value is important. Supervisory and control systems are used to monitor the temperature and other physical parameters on a centralised machine whereby one can monitor and control the remote devices. The AVR microcontroller-based system described here does the same job of acquiring the analogue data and sending it to a remote terminal for monitoring.

Fig. 1 shows the block diagram of the eight-channel data acquisition and logging system using AVR microcontroller and Fig. 2 shows the author's prototype. The key features of this system are:

1. The software is user-friendly and written in VB 6.0.
2. Data is acquired through serial port of the PC and displayed on the screen of the PC monitor.
3. Precise analogue signal conversion using AVR analogue-to-digital converter with 10-bit resolution.
4. All data acquired by the system is logged into a database for future reference with date and time of sampling.
5. The internal analogue-to-digital conversion (ADC) channels of the AVR are used to acquire real-time data in the form of analogue signal. The data is sent to the PC via UART channel.

Circuit description

Fig. 3 shows the circuit of the eight-channel data acquisition and logging system using AVR. At the heart of the circuit is ATMega32 AVR microcontroller from Atmel.

The ATMega32 microcontroller has 32 kB of flash program memory, 2 kB of SRAM, internal analogue-to-digital converter (ADC) with 10-bit resolution, internal EEPROM and full-duplex UART channel. This data logger uses ADC channels of the AVR to acquire real-time data in the form of analogue signal and sends this data to the PC via UART channel.

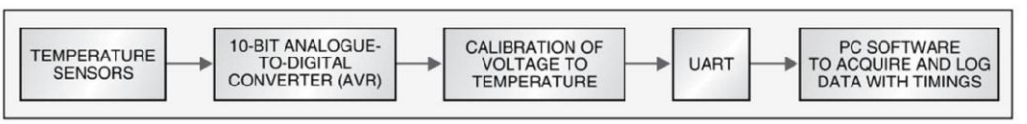


Fig. 1: Block diagram of eight-channel data acquisition and logging system

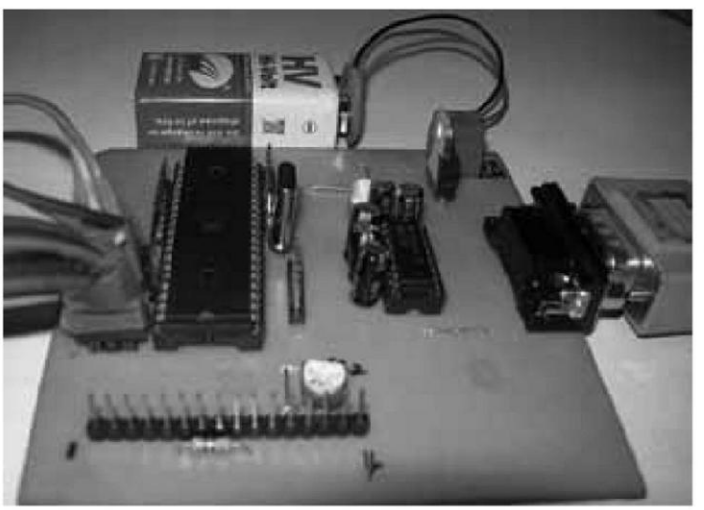


Fig. 2: Author's prototype

Vcc (pin 10) and AVcc (pin 30) of the AVR are connected to +5V for operation. By default, this AVR works with the internal RC oscillator at 1 MHz. Here, fuse bits of the AVR are set to operate an external oscillator. We have used an external stable crystal oscillator to run at a frequency of 16 MHz.

The AVR has internal power-on reset facility. Resistor R2 (10-kilo-ohm), capacitor C5 (10µF) and switch S1 make up the external reset circuitry. Switch S1 allows you to reset the system at run time.

Analogue reference voltage pin V_{REF} (pin 32) is connected to the variable terminal of the 10-kilo-ohm preset. Using this preset, you can adjust the ADC reference voltage.

We have used all the eight channels of the 10-bit ADC for acquiring the analogue voltage propor-

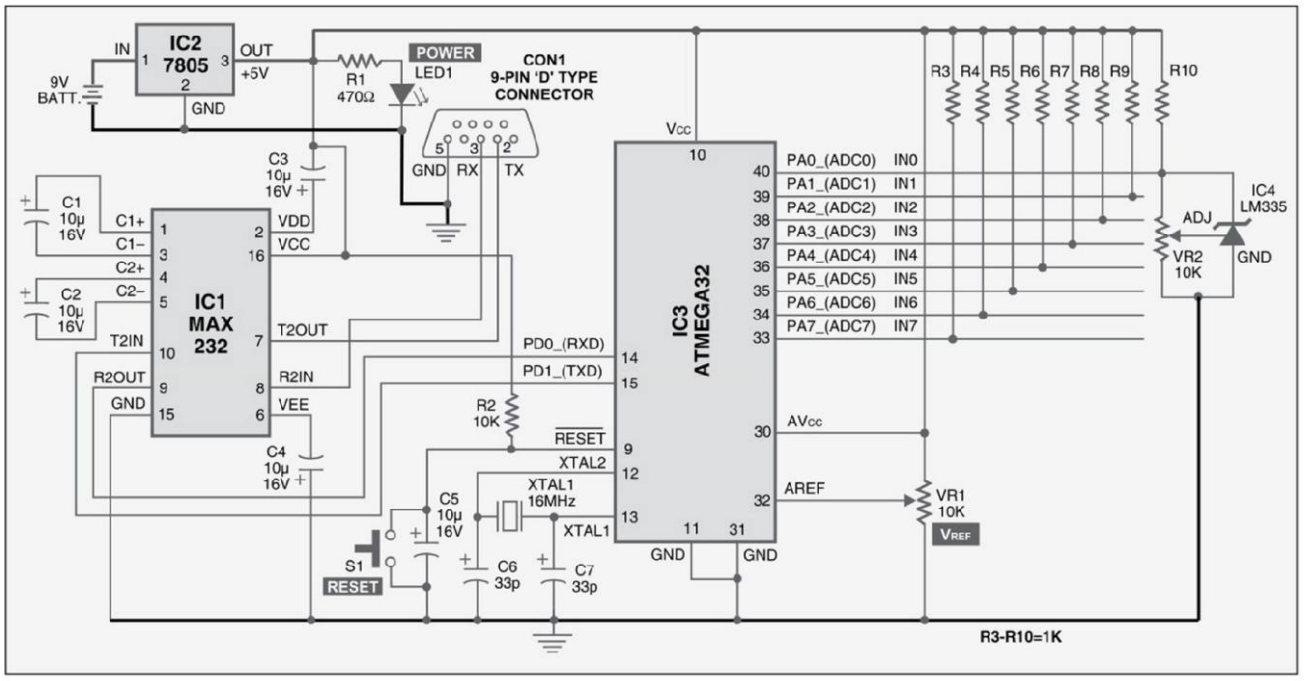


Fig. 3: Circuit for eight-channel data acquisition and logging

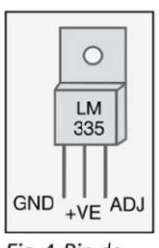


Fig. 4: Pin details of LM335

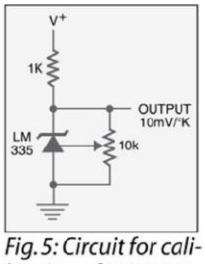


Fig. 5: Circuit for calibration of LM335 to 2.982V at 25°C

tional to the environmental temperature of temperature sensors.

The in-built UART channel of ATMega32 is used to send the current data to the host PC. UART works on 9600 bauds per second. The length of RS-232 serial cable is tested for operation up to 10 metres but it should work up to 15 metres.

Data acquisition and logging

Temperature sensor. Temperature sensor LM335 from National Semiconductors has been used in this project. Its pin details are shown in Fig. 4.

LM335 has a breakdown voltage directly proportional to absolute temperature at 10 mV/°K with less than 1-ohm dynamic impedance. The device operates over a current range of 400 µA to 5 mA with virtually no change in performance. LM335 can be used in any kind of temperature sensing application over the temperature range of –55°C to 150°C. Low impedance and linear output make it easier to interface with the readout and control circuitry. It is not internally calibrated for degree Celsius (°C), so you need some external circuitry in the form of a 10-kilo-ohm preset and

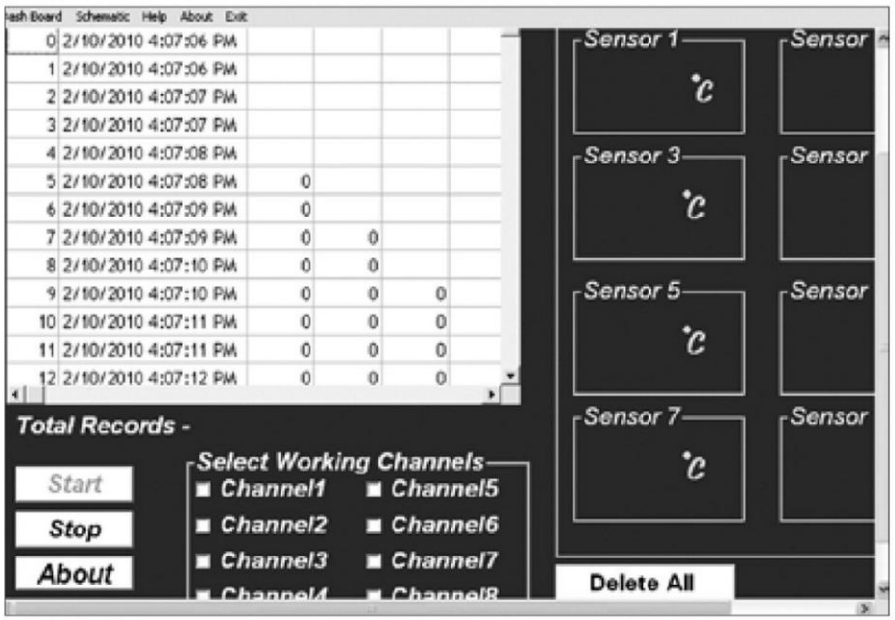


Fig. 6: GUI program output on the dashboard

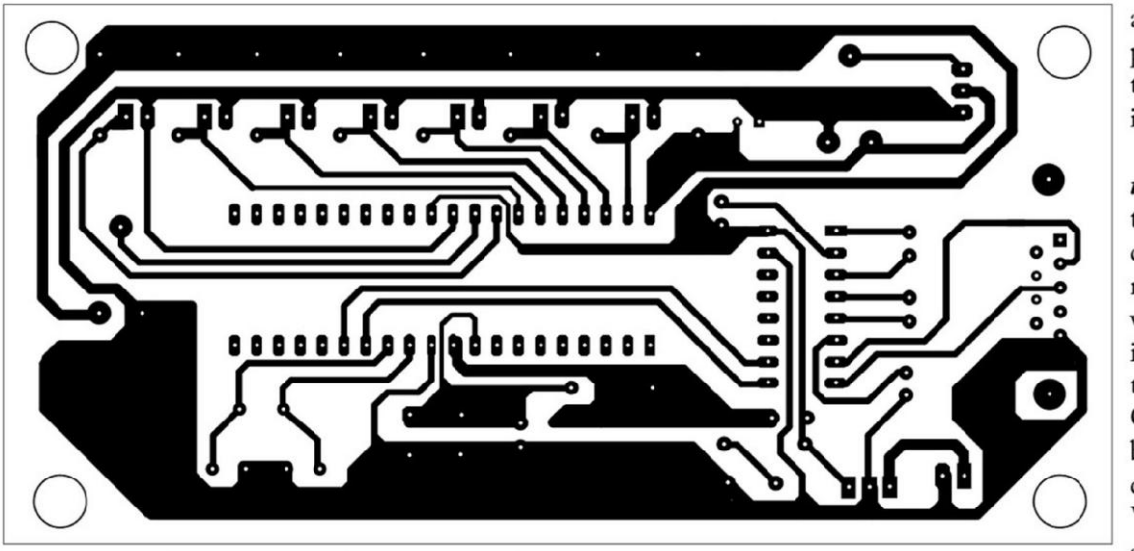


Fig. 7: A single-side, actual-size PCB layout for eight-channel data acquisition and logging system

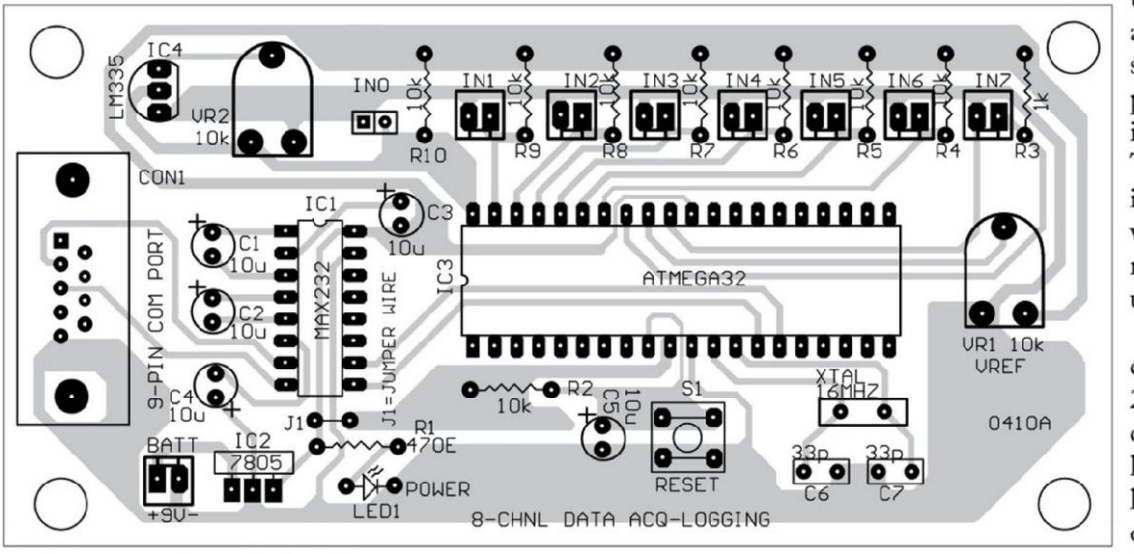


Fig. 8: Component layout for the PCB

a 1-kilo-ohm pull-up resistor as shown in Fig. 5.

Calibration. Calibration is done carefully to map voltage values exactly into temperature in degree Celsius. Calibration procedure is simple. Voltage values are measured for different temperatures and a constant multiplying factor is obtained. This constant is multiplied with the current ADC value every time.

When calibrated at 25°C, typically, LM335 has an error of less than 1°C over a range of 100°C. Most of all, it has a linear output. The voltage across the output terminal of LM335 is 2.982V at 25°C.

This microcontroller works with TTL digital logic, while the RS-232 standard specifies different voltage levels of the digital logic. So you need a signal-level converter for communication between the microcontroller and the PC over RS-232 port.

Signal-level conversion. MAX232 is used as the signal-level converter. For voltage-level conversion, four electrolytic capacitors (10μF, 16V) are used with MAX232.

There are eight input lines (IN0 through IN7) through which analogue inputs are fed into the circuit. The analogue input is converted into digital level by the AVR and transmitted to the PC through the 9-pin, D-type serial com port connector. Here, we have used only three pins of the connector (Rx, Tx and Gnd) for communication with the PC.

PC GUI software

The graphic user interface (GUI) displays on the dashboard the stored data with date and time of logging. This

PARTS LIST

Semiconductors:

IC1	- MAX232 RS-232 converter
IC2	- 7805, 5V regulator
IC3	- Atmega32 AVR microcontroller
IC4	- LM335 temperature sensor
LED1	- 5mm light-emitting diode

Resistors (all ¼-watt, ±5% carbon):

R1	- 470-ohm
R2	- 10-kilo-ohm
R3-R10	- 1-kilo-ohm
VR1, VR2	- 10-kilo-ohm preset

Capacitors:

C1-C5	- 10µF, 16V electrolytic
C6, C7	- 33pF ceramic

Miscellaneous:

X_{TAL1}	- 16MHz crystal oscillator
CON1	- 9-pin D-type female connector
BATT.	- 9V PP3 battery
S1	- Push-to-on switch

can be useful to analyse the trend of change in temperature. The software dashboard has eight blocks for displaying data of eight different analogue channels.

The GUI software is written in Visual Basic and has MSComm ActiveX controls for communication with the serial port of the PC. It is programmed to scan real-time incoming data from the external hardware. The entire working logic is asynchronous; it doesn't matter which channel has what data. The software can capture the data from a particular channel and put it into an appropriate location in the database. A special protocol is used to synchronise the software with the hardware in order to make the program identify the data and channel number currently active on the serial port. The microcontroller first sends the channel number followed by the current captured data on the channel.

The software is calibrated such that it shows the temperature directly in degree Celsius by multiplying a calibration constant with the incoming analogue voltage.

The software is configured to work with fixed values such as 'com1' for the serial port and '9600' for the baud rate by default. But you can easily configure it to work with different serial ports (like com2, com3 or com4) and baud rates.

The software can save data of the different input channels into a 'daq.mdb' database with time and date of each channel input data. Start/stop buttons are provided to start or stop the logging activity any time by the user. The GUI program output is shown in Fig. 6.

Construction

A single-side, solder-side PCB layout of the circuit for eight-channel data acquisition and logging is shown in Fig. 7 and its component layout in Fig. 8. A 9V PP3 battery is used to power the circuit. 9V is converted into 5V using a 7805 regulator. The glowing of LED1 indicates the presence of 5V supply in the circuit. The circuit acquires analogue data from the eight channels through IN0 through IN7 inputs. The analogue temperature data at IN0 channel is acquired from LM335 temperature sensor (IC4). Temperature calibration for IC4 is done using a 10-kilo-ohm preset (VR2). The remaining inputs can be fed from external temperature sources.

Assemble the temperature sensor along with preset separately on a small general-purpose PCB for each channel. Extend two wires from each of the general-purpose PCBs to the respective input points (IN1 through IN7) in the main PCB. Two-pin SIL male and female pair connector may be used for connecting the PCB to the general-purpose PCB for each channel input. As shown in Fig. 5, a 10-kilo-ohm preset is used for calibration of each temperature sensor.

Calibrate each temperature sensor (LM335) before connecting the circuit to the PC. After calibration is done, install the sensors at appropriate locations or on the device whose temperature is to be monitored. (All the relevant files of the GUI software are stored in 'EFYproject/VB Project' folder of 'D' drive.) Now, run the datalogger8chnl.exe GUI software and click 'start' button to start the data acquisition and logging process. If data display on the dashboard is not proper, press reset switch S1 momentarily, or switch off the power supply and then switch it on. Using preset VR1, adjust ADC reference voltage such that it is exactly 5V across pin 32 of IC3.

Microcontroller firmware

The main.c source code for ATMega32 (given at the end of this article) is written in 'C.' It is compiled using avr-gcc cross-compiler to generate hex code. Avrdude is used to burn hex code into the ATMega32 microcontroller.

WinAVR is a free software development tool for AVR series microcontrollers hosted on the Windows plat-

form. It has avr-gec, avr-liboc, avr-binutils and avr-dude within one package. Linux users have to install these components separately. For details of AVR programming hardware and software, visit 'www.electroons.com'

Download source code: http://www.efymag.com/admin/issuepdf/Datalogger%208-channel.zip

MAIN.C

```
#include<avr/io.h>
#include<util/delay.h>

void UART_transmit(unsigned char data)
{
  while(!(UCSRA & (1<<UDRE)));
  UDR=data;
}

void adc_getdata(void)
{
unsigned int temp;
unsigned char ch=0;
 while(1)
 {
  for(ch=0;ch<8;ch++)
  {
   ADMUX=0x00+ch;  // Selecting Channels
   ADCSRA=0xE7;     // ADC free running mode, 128
prescaling division factor
   temp=(ADC)*(48);
   temp=temp/10;
   temp=temp-2280;
   temp=temp/10;
   _delay_us(1000);
   UART_transmit(ch+'0'); // Sending channel number
for synchronization
   _delay_ms(10);
   UART_transmit(temp);   // Sending temperature
value
   _delay_ms(1000);
   }
  }
}

int main(void)
{

DDRA=0x00;
PORTA=0x00;

UCSRA=0;
UCSRB=1<<TXEN; // UART transmit enable
UCSRC=1<<URSEL | 1<<UCSZ1 | 1<<UCSZ0;  // 8 data
bit, a stop, none parity
UBRRH=0;
UBRRL=103; // for 9600 baud at 16MHz

adc_getdata();

return 0;

}
```

PROGRAMMABLE INDUSTRIAL ON-OFF TIMER WITH RF REMOTE

■ A.M. BHATT

In most of the modern manufacturing and processing industries, there is complete industrial automation through sophisticated hardware and software like programmable logic controller (PLC), distributed control system (DCS), and supervisory control and data acquisition (SCADA). Microcontroller-based embedded systems play major role in industrial automation. One such widely used system is the programmable timer. The major applications of programmable timer are as follows:

1. Initiating the process after desired time
2. Switching on/off the process after predetermined time
3. Providing delay in between processes
4. Applying input to on/off type open-loop control system

Depending upon the requirement of process, one can set the time of programmable timer. As the time period expires, the timer will either trigger or shut the process. Earlier there were mechanical timers that used gear assembly (same as wall clock) and mechanical contacts. But the problem with these was that due to mechanical parts and movements, they were not durable. Electronic timers have become very popular as these have more functionalities and long operating life.

A simple electronic timer can be made using a single IC 555 in monostable mode that can switch on/off the process after desired time. Also, in chain process (where the end of first process starts second process and so on), one can use a number of such monostable blocks to make a sequential timer. But these circuits do not include additional features like digital display, system failure indication, remote operation and alarms. Since the precision and accuracy of these timer circuits depend upon the value of the resistor-capacitor components that may deviate, we might not get the exact and precise timing. To enhance the programmable timer for generating precise timing and additional features, microcontrollers (embedded controllers) are used with peripheral devices.

Some of the features of programmable industrial on/off timer presented here include:

1. Time set from 1 to 60 seconds (can be extended)
2. 'On' time and 'off' time can be programmed (from 1 to 60 seconds)
3. Repeat (continuous) and single operation
4. Fully remote-controlled within 100-metre range
5. User-friendly front-panel controls and display panel with LCD
6. Emergency stop buttons (on control panel as well as on remote)
7. Provision of potential-free relay contacts for connecting any 230VAC at 10A or 28V DC at 10A device/application

PARTS LIST

Semiconductors:

IC1	- AT89C51 microcontroller
IC2	- HT12D decoder
IC3	- 74LS21 four input AND gate
IC4	- 7805, 5V regulator
IC5	- HT12E encoder
T1	- BC337 npn transistor
T2	- BC547 npn transistor
D1-D5	- 1N4007 rectifier diode
D6-D13	- 1N4148 switching diode
LED1-LED3	- 5mm LED
LCD	- 2-line and 16-character LCD
	- 433MHz ASK transmitter-receiver module

Resistors (all ¼-watt, ±5% carbon):

R1	- 10-kilo-ohm
R2, R3, R7	- 1-kilo-ohm
R4	- 3.9-kilo-ohm
R5	- 47-kilo-ohm
R6, R8	- 330-ohm
VR1	- 10-kilo-ohm preset

Capacitors:

C1, C2	- 33pF ceramic disk
C3	- 1µF, 16V electrolytic
C4	- 1000µF, 25V electrolytic
C5	- 0.1µF ceramic disk

Miscellaneous:

X1	- 230V AC primary to 12V, 500mA secondary transformer
X_{TAL}	- 12MHz crystal
S1-S5, S10-S14	- Push-to-on tactile switch
S6, S7	- SPDT switch
S8	- 4PDT toggle Switch
S9	- SPST toggle switch
RL1	- 12V, 1 C/O relay

Circuit description

The complete hardware circuit is divided into two sections—program-

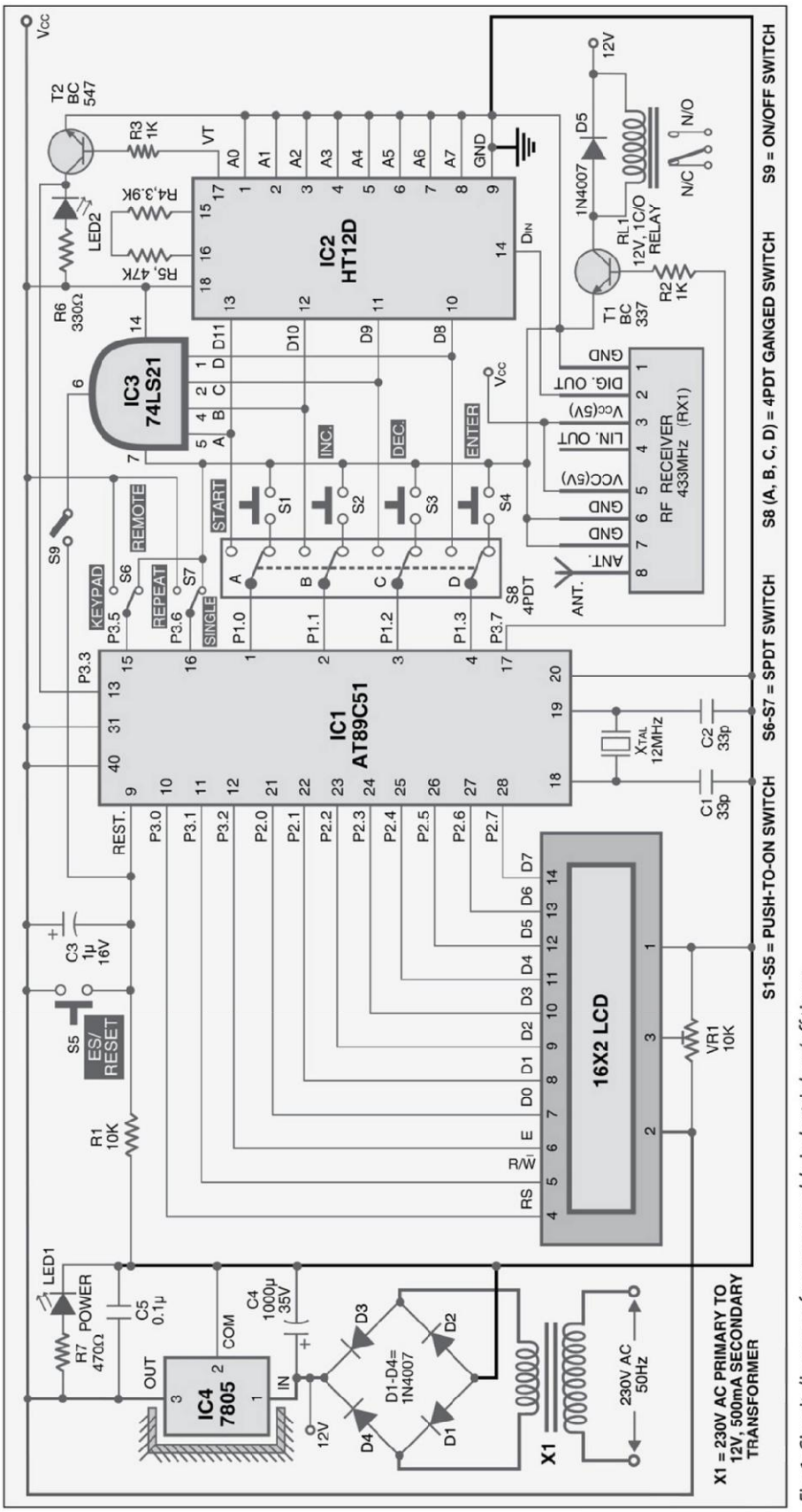


Fig. 1: Circuit diagram of programmable industrial on/off timer

mable timer main circuit and remote control transmitter and receiver units. The main circuit is a fully-functional standalone unit with front-panel control switch and LCD display. Remote control transmitter and receiver are add-on units. The receiver outputs are connected with respective control input pins of main circuit. Receiver output facilitates user to program and control timer operation using remote within 100-metre range.

Main circuit. Fig. 1 shows the circuit diagram of industrial on/off timer. The circuit comprises microcontroller AT89C51 (IC1), LCD, transistors and a few discret components. The data pins D0 through D7 of LCD are connected with port pins P2.0 through P2.7 of microcontroller AT89C51. The control pins register select (RS), read/write (R/W) and enable (E) are connected with port P3 pins P3.0, P3.1 and P3.2, respectively. Preset VR1 is connected with pin 3 of LCD for contrast control. Switches S1 through S4 are connected to port pins P1.0 through P1.3 by 4PDT toggle switch S8. Two single-pole double-throw (SPDT) switches S6 and S7 are connected at P3.5 and P3.6. Switch S6 selects for either remote or keypad control, and switch S7 selects control for either repeat or single mode operation. Port pin P3.7 drives relay through transistors T1. Diode D5 acts as free-wheeling diode for relay RL1.

Power-on reset is provided by the combination of resistor R1 and capacitor C3. Switch S5 is used for manual reset or

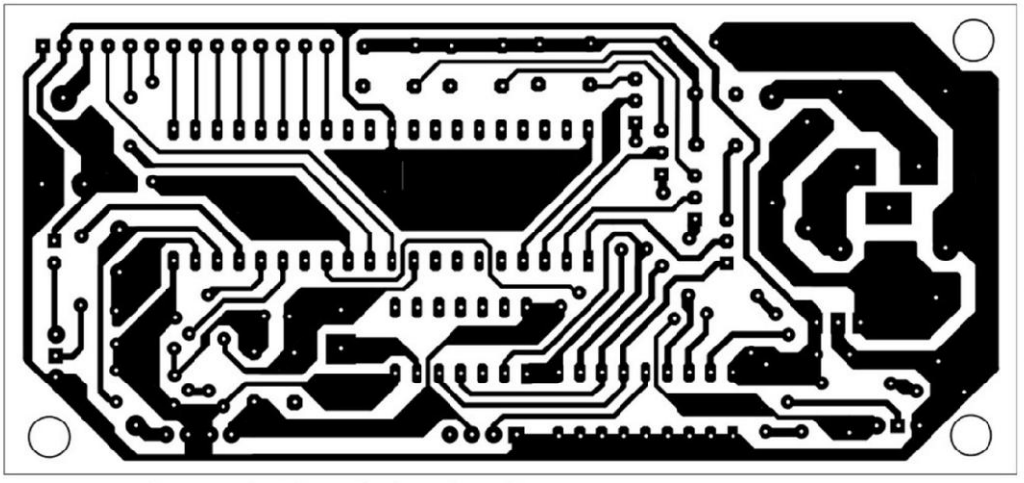


Fig. 2: An actual-size, single-side PCB for the industrial timer circuit

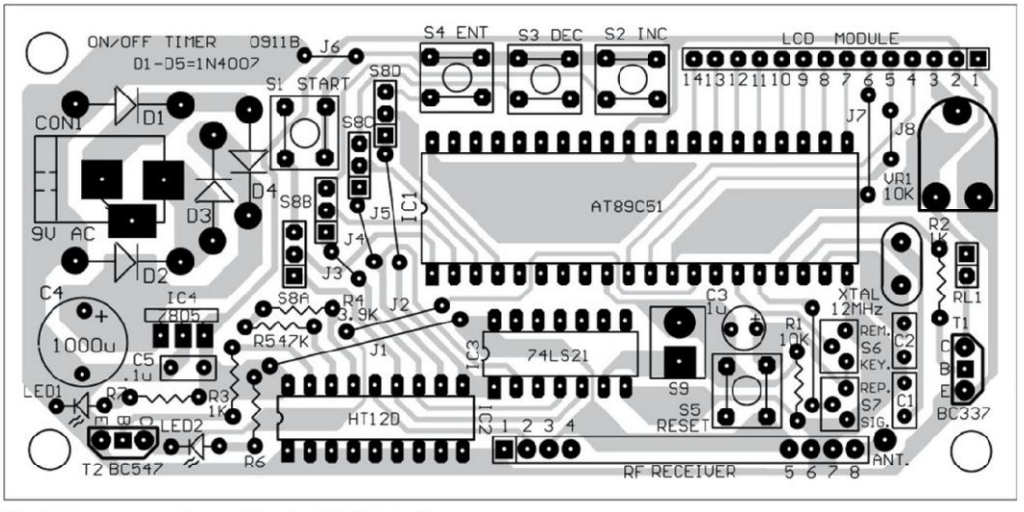


Fig. 3: Component layout for the PCB in Fig. 2

emergency stop function. A 12MHz quartz crystal along with two 33pF capacitors provides clock pulse to the microcontroller AT89C51.

The 230V, 50Hz AC mains is stepped down by transformer X1 to deliver a secondary output of 12V, 500mA. The transformer output is rectified by a full-wave rectifier comprising diodes D1 through D4, filtered by capacitor C4 and regulated by IC 7805 (IC4). Capacitor C5 bypasses the ripples present in the regulated supply. LED1 acts as the power indicator and R7 limits the current through LED1.

An actual-size, single-side PCB for the programmable industrial on/off timer is shown in Fig. 2 and its component layout in Fig. 3. Assemble the circuit on a PCB as it minimises assembly time and errors. Carefully assemble the components and double-check for any overlooked error.

Operation. The functions of all keys are given in Table I. The mode selection switch S7 selects either repeat or single mode. Single mode allows user to run the timer operation in 'on' time and 'off' time sequence once. In repeat mode of operation, the timer repeats 'on' time and 'off' time sequence continuously. During this cycle if this switch is changed to single-mode, the timer stops as the cycle completes. Also, if the emergency stop (ES) button is pressed during any mode of operation the timer operation will stop.

The step-by-step operation when the main circuit is powered is as follows:

1. The 'enter on time' message is displayed on LCD
2. User has to enter the desired time by incrementing/decrementing time using 'Inc.' (S2)/'Dec.' (S3) keys
3. After pressing the 'Enter' (S4) key the user will be prompted to enter 'off' time
4. Using the same S2 and S3 keys, the 'off' time may be entered and the 'Enter' key pressed
5. 'Press Start' message is displayed as the user enters the time
6. After pressing 'Start' (S1) key the operation starts
7. The relay is energised and the device remains on till the 'on' time counts down to 0. After that the relay is de-energised and the device turns off. It remains in this state till 'off' time counts down to 0
8. If the timer is operating in the repeat mode the cycle will repeat continuously and device will be switched

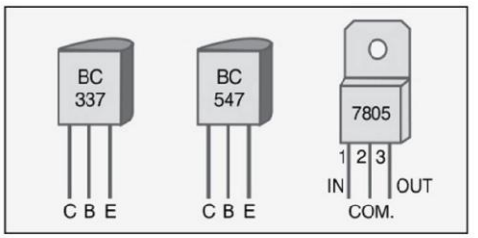


Fig. 4: Pin configurations of BC337, BC547 and 7805

on and off after required time intervals. In this mode if the operation has to be stopped then either switch S7 has to be toggled or ES button has to be pressed

9. If the timer is operating in single mode then as one on-off time cycle completes, the timer stops working. One has to enter 'on' time and 'off' time again to re-start operation

10. For remote control operation, port pins P1.0 through P1.3 of microcontroller AT89C51 are changed to D11 through D8 of HT12D, respectively, using four-pole double-throw (4PDT) switch S8. Also, switch S6 connected to port pin P3.5 is changed to remote position to set remote operation mode.

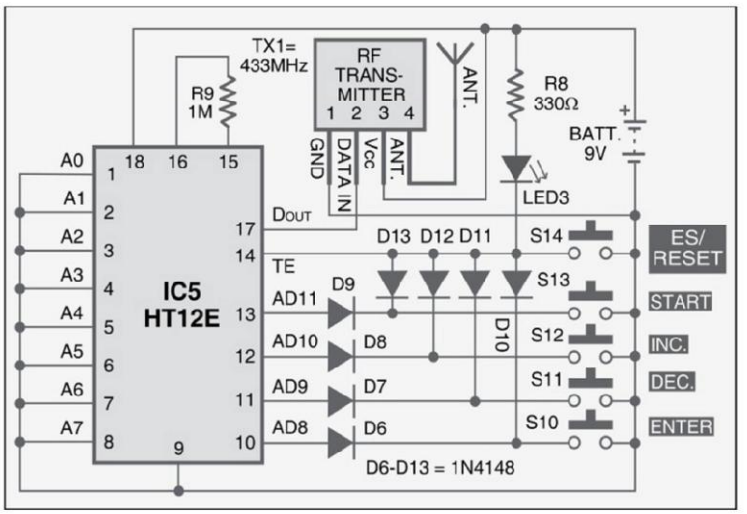


Fig. 5: Circuit diagram of RF transmitter

Remote control transmitter and receiver circuit. Remote control transmitter and receiver are made using readily available encoder (HT12E) and decoder (HT12D) chips.

Transmitter circuit. Fig. 5 shows the circuit diagram of transmitter consisting of two main components—encoder HT12E (IC5) and radio frequency transmitter module 433MHz (TX1). All the address pins A0 through A7 are tied to ground to set address '00h.' A one mega-ohm resistor is connected between oscillator pins 15 and 16 of HT12E. D_{OUT} pin17 of HT12E is connected with input of ASK transmitter (TX1). Switches S10 through S13 are connected with data input pins AD8 through AD11 of HT12E in such a way that when you press any key, TE pin is automatically grounded. Key S14 is directly connected between TE pin and ground as shown Fig. 5. LED3 is connected for indication of pressing the switch. An actual-size, single-side PCB for RF transmitter is shown in Fig. 6 and its component layout in Fig. 7.

TABLE I

Functions of Push Buttons

Switch number	Switch name	Function
S1	Start	Starts timer operation
S2	Inc. time	Increments time set by 1 sec. max limit is 60
S3	Dec. time	decrements time set by 1 sec. min limit is 1
S4	Enter	Used to enter time set value
S5	ES (RST)	Emergency stop or system reset.
S6	Control selection	Selection of either remote or keypad control
S7	Mode selection	Selection of either repeat or single control

TABLE II

Status of Data Input Pins and the Codes Transmitted

Switch	D3	D2	D1	D0	Code
start	0	1	1	1	7
Inc. time	1	0	1	1	B
Dec. time	1	1	0	1	D
enter	1	1	1	0	E
ES	1	1	1	1	F

Operation. When you set the address and data inputs on HT12E and pull down TE pin low, that address and data are serially transmitted through D_{OUT} pin 17 of HT12E. Here the address is already set to '00h.' Now as any key is pressed, respective data pin goes low and TE pin is grounded. Status of data along with 8-bit address is serially transmitted through ASK transmitter. ASK transmitter modulates these data signal with 433MHz carrier and transmits it through antenna. Table II shows the status of data input pins and the codes transmitted for a particular switch.

Receiver circuit. Receiver circuit is included in Fig. 1. All the address pins of decoder

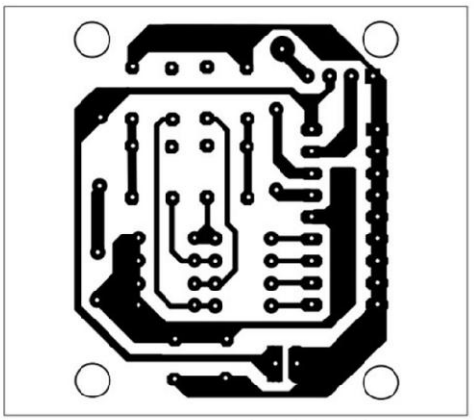


Fig. 6: An actual-size, single-side PCB for RF transmitter

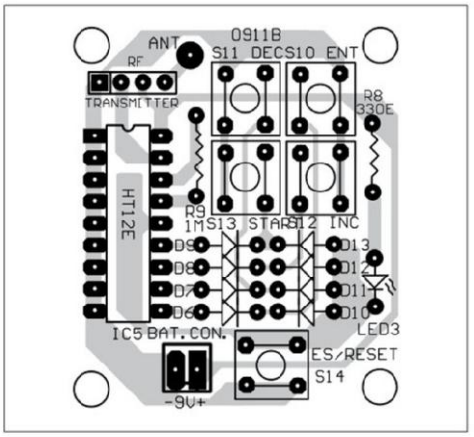


Fig. 7: Component layout for the PCB in Fig. 6

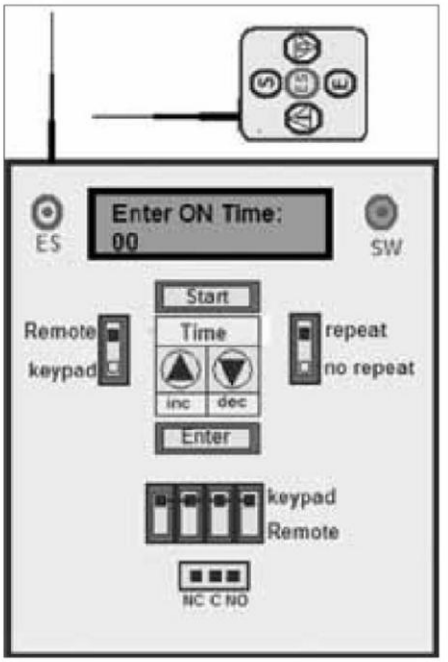


Fig. 8: Proposed arrangement for remote control and front panel for industrial on/off timer

HT12D (IC2) are connected to ground to set same address '00h' as on the transmitter. Data output pins D8 through D11 are connected to microcontroller (IC1) through contacts of 4PDT switch S8. Also, all four outputs are given as input to 4-input AND gate. The output of AND gate is connected to reset input pin 9 of microcontroller AT89C51 of main circuit. VT pin of HT12D drives LED2 through transistor T1 for indication of data reception. A 51-kilo-ohm resistor is connected between oscillator pins 15 and 16 of decoder HT12D. Output signal of ASK receiver (RX1) is connected with D_{IN} pin 14 of HT12D.

Operation. ASK receiver receives 433MHz carrier signal and demodulates the address and data before giving it to decoder HT12D. HT12D receives these bits serially through D_{IN} pin 14 and compares the address. If received address and set address match, the data will be latched on pins D8 through D11 of decoder HT12D. The VT pin 17 becomes high momentarily as data is latched. This is indicated by LED2. If AD11 pin goes low by pressing start switch (S1) of the transmitter side, only D11 pin 13 at the receiver side becomes low and all other pins become high. Since D11 pin is connected to port pin P1.0 of microcontroller AT89C51, it gives the start signal. Data is received similarly for D8, D9 and D10. When ES key is pressed all data pins D8 through D11 are high. So at the receiver side all the pins become high. That makes the output of AND gate high, which is given to the reset pin 9 of microcontroller.

Because there is a latch action at D8 through D11 outputs, all A through D inputs to AND gate remain high till any other key is pressed at the transmitter. This will continuously apply reset signal to microcontroller AT89C51. To remove reset condition, one SPST (S9) is used in between to make and break the direct connection. Open switch S9 will break the connection and reset pin will not get high logic from the AND gate. Now, after starting the operation, to activate ES from remote, press the switch again to make the connection. Fig. 8 shows the proposed arrangement for front panel and remote control for industrial on/off timer.

Software

The program is written in 'C' language and compiled through Keil µVision4 compiler. The program is well-commented and easy to understand. Complete program is divided in twelve different functions. Out of these twelve functions, four functions are for LCD handling, four for timer operation, two delay functions, one interrupt function and the last one is the main function.

LCD handling functions. The various functions are as follows:

'writecmd' function. It sends command byte to LCD by taking one argument byte that is sent to port P0

'writedata' function. It sends data byte to be displayed on LCD. It also takes one argument byte and sends it to P0

'writestr' function. It writes the whole string (message) on LCD taking pointer as an argument that points address of first character of string. Through the pointer it sends all the characters one by one to port P0

'busy' function. It checks the status of busy flag of LCD. If the flag is

set that means LCD is not ready and program remains within loop. When flag is reset that means LCD is ready and program comes out of loop

Delay functions.

'keydly' function. It generates key debounce delay of around 0.1 second using 'for loop' statement twice

'dly1sec' function. It generates approximately one second delay using 'for loop'

Interrupt function.

'takeinput' function. This interrupt function handles external interrupt1. It just saves the status of port P1 whenever external interrupt1 occurs

Timer operation functions.

'Start' function. It starts the timer operation by sending high logic to port pin P3.7 and energising the relay. The countdown time is displayed on LCD as 'Device ON.' Then it sends low logic to port pin P3.7 and de-energises the relay. Again countdown time is displayed as 'Device OFF.' It then checks the status of port pin P3.6. If it is high the cycle repeats continuously and if it is low the operation stops

'Inctime' function. It increments 'ontime' or 'offtime' variables till maximum limit (60 second) is reached. To display the variable on LCD the value must be in ASCII format. So first two digits are separated and then converted into equivalent ASCII values using array named 'ascii.' This array includes ASCII values of all 0 to 9 digits

'Dectime' function. It is similar to 'Inctime' function. The difference is that it will decrement 'ontime' or 'offtime' variable till minimum limit (1 second) is reached

'Enter' function. It changes the messages on LCD as 'Enter OFF time:,' 'Press Start,' etc. It is also used to enter 'ontime' and 'offtime' variables, alternatively.

Main function. It first initialises LCD and then displays message 'Enter ON time:.' It then checks the status of port pin P3.5 of microcontroller. If port pin P3.5 is high, it means timer operation is controlled through keypad and if it is low the timer operation is controlled through remote. It waits in continuous loop till any key is pressed from keypad or remote. As the key is pressed, the function is detected and it calls the particular function corresponding to that key like 'Start,' 'Inctime,' 'Dectime' or 'Enter.'

Download source code: http://www.efymag.com/admin/issuepdf/Programmable%20Industrial%20on-off%20Timer%20With%20RF%20Remote.zip

TIMER.C

```
#include <reg51.h>
#include <string.h>
sbit rs = P3^0;                         // declare
P3.0 as rs pin
sbit en = P3^2;             // declare p3.2 as en-
able pin
sbit rw = P3^1;             // declare p2.6 as
read/write pin
sbit b = P2^7;                          // busy
flag
sbit op=P3^7;                           // output pin
sbit pin=P3^6;                          // input pin
for repeat no repeat mode
sbit rmt=P3^5;                          // input pin
for remote or keypad operation
unsigned char ascii[10] = {0x30,0x31,0x32,0x33,0x34
,0x35,0x36,0x37,0x38,0x39};
unsigned char data byte;
unsigned int i=0;j=0,k=0,ontime=0,offtime=0,t1=0,t2
=0,tmp1,tmp2,flag,sflag;
void writecmd(unsigned char a);       // function to
send command to LCD
void writedat(unsigned char b);       // function to
send data to LCD
void busy();                                    //
function to check LCD is busy or not
void writestr(unsigned char *s);      // function to
write string on LCD
void keydly()                                   //
function for key debounce delay
 {
          int y,z;
          for(y=0;y<50;y++)
           for(z=0;z<1000;z++);
  }
void writecmd(unsigned char a)
{
          busy();                               //
check for LCD is busy or not
          rs = 0;                               //
clear rs pin for command
          rw = 0;                               //
clear rw pin to write
          P2 = a;                               //
send command character
          en = 1;                               //
strob LCD
          en = 0;
}
void writedat(unsigned char b)
{
          busy();                               //
check for LCD is busy or not
          rs = 1;                               //
set rs pin for data
          rw = 0;                               //
clear rw pin to write
          P2 = b;                               //
```

TIMER.C

```
send data character
        en = 1;                                 //
strob LCD
        en = 0;
}
void busy()
{
        en = 0;                                 //
disable display
        P2 = 0xFF;                              //
configure P0 as input
        rs = 0;                                 //
clear rs pin for command
        rw = 1;                                 //
set rw pin to read
        while(b==1)
        {
         en=0;                                  //
strobe LCD till P0.7 is 1
         en=1;
        }
        en=0;
}
void writestr(unsigned char *s)
{
        unsigned char l,i;
        l = strlen(s);                          //
get the length of string
        for(i=0;i<l;i++)
        {
                writedat(*s);                   //
write every char one by one
                s++;
        }
}
void inctime()
  {
         t2=0;                                  //
minimum limit flag cleared
          if(k==1)                  // check to
increase ontime or offtime
            {
                if(offtime<60) offtime++;  //
increase offtime till 60
                 else
                   {

writecmd(0xC0);
                                    writestr("60-
max limit  ");     // if it is 60 display message
                                    t1=1;
                // and set max limit falg

                        }
                  i=offtime/10;             //
separate upper and
                   j=offtime%10;    // lower dig-
its
                }
           else                                 // if
ontime-offtime flag is not 1
                {
                 if(ontime<60) ontime++;    //
increase ontime till 60
                  else
                              {

writecmd(0xC0);    // same as above
                                    writestr("60-
max limit  ");
                                    t1=1;

                              }
                    i=ontime/10;
                    j=ontime%10;
                    }
          if((t1==0)&& (t2==0))     // if max
limit is not reached
             {
                          writecmd(0xC0);    //
print ontime/offtime sec
                          writestr("press en-
ter:");
                          writedat(ascii[i]);
// as one by one digit

writedat(ascii[j]);
                          writestr("  ");

                 }
  }
void dectime()
// similar to above function
  {
         t1=0;
         if(k==1)
           {
                  if(offtime>1) offtime--;
                  if(offtime<=1)
                    {

writecmd(0xC0);
                                    writestr("01-
min limit  ");
                                    t2=1;

                          }
                    i=offtime/10;
                    j=offtime%10;
                }
           else
                {          if(ontime>1) ontime--;
                 if(ontime<=1)
                   {

writecmd(0xC0);
                                    writestr("01-
min limit  ");
                                    t2=1;

                          }
                    i=ontime/10;
                    j=ontime%10;
                    }
          if((t1==0)&& (t2==0))
             {
                          writecmd(0xC0);
                          writestr("press en-
ter:");
                          writedat(ascii[i]);

writedat(ascii[j]);
```

```
                        writestr("  ");

                }
  }
void enter()                                    //
enter ontime & offtime and
  {                                             //
display message
        k++;                                    //
increase count to enter off time
        if(k==1)                    // first time
          {
                writecmd(0x80);
                writestr("Set OFF time and"); //
display message and
                writecmd(0xC0);                 //
show value 00
                writestr("press enter:");
                writedat(0x30);
                writedat(0x30);
                i=j=0;
          }
        else if(k==2)
// second time
          {
                writecmd(0x01);                 //
clear LCD screen and
                writestr("Press start");
// display message
                sflag=1;                        //
set start flag
                k=0;
// reset counter
          }
  }
void delay1sec()                                //
generate delay of exact 1 sec
  {
    int x;
        TL1 = 0xBF;
        TH1 = 0x3C;
        TR1 = 1;
        for(x=0;x<=20;x++)
          {
                while(TF1==0);
                TF1 = 0;
                TL1 = 0xBF;
                TH1 = 0x3C;
          }
        TR1 = 0;
  }
void start()
  {
          int a,b;
          tmp1=ontime;                          //
save ontime and offtime values
          tmp2=offtime;
 back:writecmd(0x80);
          writestr("Device ON: ");              //
display message
          op=1;
// switch ON relay
          for(a=tmp1;a>0;a--)
// start countdown
            {
                        i=a/10;
                        j=a%10;
                        writecmd(0x8B);
                        writedat(ascii[i]);
// change the display digits
                        writedat(ascii[j]);
// on LCD after
                        delay1sec();
// every sec
                }
          op=0;                                 //
switch OFF relay
          writecmd(0x80);
          writestr("Device OFF:");
          for(b=tmp2;b>0;b--)
// start countdown again
            {
                        i=b/10;
                        j=b%10;
                        writecmd(0x8B);
                        writedat(ascii[i]);
// display digits after

writedat(ascii[j]);
                        delay1sec();
// 1 sec delay
                }
          if(pin==1) goto back;                 //
repeat continuously for repeat mode
          else
// otherwise
            {
                        flag=1;
// set flag
                        ontime=0;               //
clear ontime and offtime
                        offtime=0;
// and stop operation
                        }
        }
void takeinput() interrupt 2          // for remote
operation mode
  {
        byte=P1;                                //
take input from P1 when interrupt arrives
  }

main()
{
        TMOD=0x10;
        P3=0x68;                                //
set keypad operation and repeat mode
        P2=0x00;                                // P2
as output
        writecmd(0x3C);                         //
Initialize LCD
        writecmd(0x0E);
bgin:   writecmd(0x01);                         //
clear display
        writestr("Set ON time and");
// display message and
        writecmd(0xC0);
        writestr("press Enter:");
        writedat(0x30);                         //
value as 00
        writedat(0x30);
        IT1=1;
// enable interrupt on falling edge
        IE=0x84;                                //
enable ext interrupt 1
        flag=0;
// clear flag for entering values
        sflag=0;                                //
```

```
start flag cleard
agin:P1=0xFF;                                   // P1
as input
         byte=0xFF;
// initialize byte variable
         if(rmt==1) while(P1==0xFF); // for keypad
mode check status of P1
         else while(byte==0xFF);              //
for remote mode check byte content
         if(rmt==1) byte=P1;
// if keypad mode save input in byte
         switch(byte)
// compare the content of byte
           {
                  case 0xFE:
         // for 1st key

                                        if(sflag==1)
start();// if start flag is set start
                                        else
 // operation else
                                          {


writecmd(0xC0);     // display message

writestr("time is not set");
```

```
flag=1;

                                                  break;
                    case 0xFD:
                                                  inctime();
// for 2nd key increase time

                                                  break;
                    case 0xFB:
                                                  dectime();
// for 3rd key decrease time

                                                  break;
                    case 0xF7:
                                                  enter();
// for 4th key enter values
                                                  break;
                    }
          keydly();
// key debounce delay
          if(flag==0) goto agin;
// till start button is not pressed change and
          else goto bgin;                          //
enter values otherwise start operation again
 }
```

Measurement

DIGITAL THERMOMETER-CUM-CONTROLLER

■ K.S. SANKAR

This standalone digital thermom- eter controls the temperature of a device according to its requirement. It also displays the temperature on four 7-segment displays in the range of –55°C to +125°C. At the heart of the circuit is the microcontroller AT89S8252, which controls all its functions. IC DS1821 is used as temperature sensor.

IC DS1821

Dallas Semiconductor's IC DS1821 is one-degree precision temperature sensor in a 3-pin pack like a transistor with single-wire communication protocol. It can operate as a standalone thermostat with user-programmable trip-points (set-points) or as an 8-bit temperature sensor with a single-wire digital interface. The open-drain DQ pin functions as the output for thermostat operation and as the data input/output (I/O) pin for single-wire communication. The single-wire interface lets user access the non-volatile memory (EEPROM) thermostat trip-point registers (TH and TL), status/configuration register and temperature register.

When configured as standalone thermostat, temperature conversions start immediately when power is switched on. In this mode, the DQ pin becomes active when the temperature of IC DS1821 exceeds the limit already programmed in the TH register, and remains active until the temperature drops below the limit programmed in the TL register.

The DS1821 uses Dallas' exclusive single-wire bus protocol that implements bus communication with one control signal.

Temperature sensor functionality

The core functionality of IC DS1821 is its proprietary direct-to-digital temperature sensing, which provides 8-bit (1°C increment) centigrade temperature readings over the range of –55°C to +125°C.

This circuit measures temperature by counting the number of clock cycles generated by an oscillator with a low temperature coefficient during a gate time-period determined by a high temperature-coefficient oscillator.

The low-temperature-coefficient counter is preset with a base count that corresponds to –55°C. If the counter reaches '0' before the gate period is over, the temperature register, which is preset at –55°C, is incremented by one degree, and the counter is again preset with a starting value determined by the internal slope accumulator circuitry of DS1821. The preset counter value is unique for every temperature increment and compensates for the non-linear behaviour of the oscillators over temperature.

At this time, the counter is clocked again until it reaches '0.' If the gate period is not over when the counter reaches '0,' the temperature register is incremented again. This process of presetting

Configuration Register

DONE	1	NVB	THF*	THL*	T/R–*	POL*	1SHOT*

*Store in EEPROM

PARTS LIST

Semiconductors:

IC1	- AT89S8252 microcontroller
IC2, IC3	- CD4511 7-segment driver
IC4	- CD4050 non-inverting buffer
IC5	- DS1821 temperature sensor
IC6	- 7805, 5V regulator
T1	- 2N2222 npn transistor
D1-D5	- 1N4007 rectifier diode
DIS1-DIS4	- Common-cathode, 7-segment display
LED1, LED2	- 5mm light-emitting diode

Resistors (all ¼-watt, ±5% carbon):

R1-R4	- 10-kilo-ohm
R5-R23, R25-R28	- 330-ohm
R24	- 5-kilo-ohm
R29	- 1-kilo-ohm

Capacitors:

C1	- 1000µF, 25V electrolytic
C2	- 0.1µF ceramic disk
C3	- 10µF, 16V electrolytic
C4, C5	- 22pF ceramic disk

Miscellaneous:

X1	- 230V primary to 7.5V, 300mA secondary transformer
S1-S4	- Push-to-on switch
S5	- On/off switch
RL1	- 6V, 150-ohm, 1C/O relay

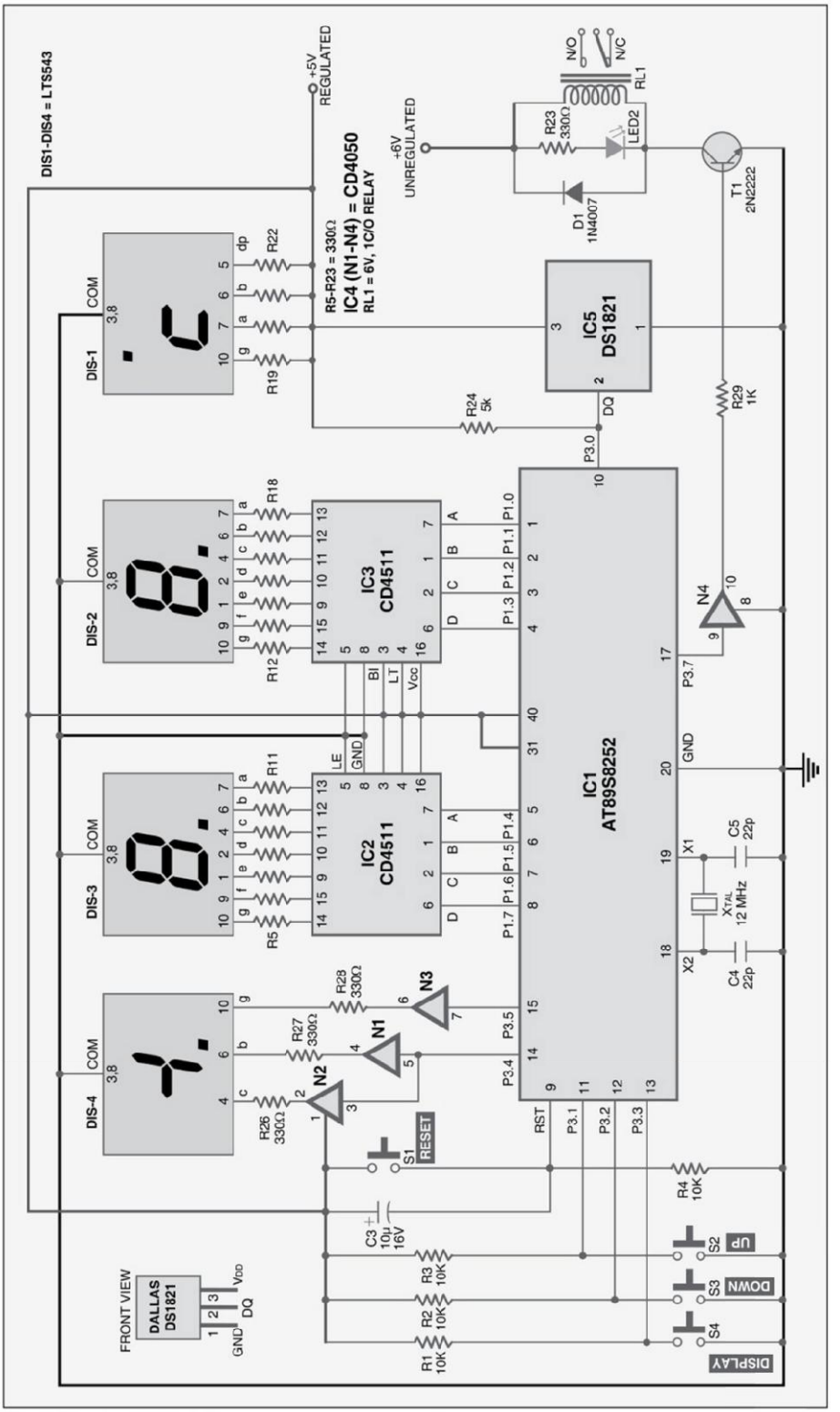


Fig. 1: Circuit of digital thermometer-cum-controller

the counter, counting down to '0,' and incrementing the temperature register is repeated until the counter takes less time to reach '0' than the duration of the gate period of the high-temperature-coefficient oscillator. When this iterative process is complete, the value in the temperature register will indicate the centigrade temperature of the device.

Operating modes

The DS1821 has two operating modes: single-wire mode and thermostat mode. The power-'on' operating mode is determined by the user-programmable T/$\overline{R}$ bit in the status/configuration register: if T/$\overline{R}$ = 0 the device works in single-wire mode, and if T/$\overline{R}$ = 1 the device works in thermostat mode. The T/$\overline{R}$ bit is stored in the non-volatile memory (EEPROM), so it will retain its value when the device is powered down.

Single-wire mode. The DS1821 is supplied by the manufacturer in single-wire mode (T/$\overline{R}$=0). In this mode, the DQ pin of the DS1821 is configured as a single-wire port for communication with a control unit (microcontroller) using the protocols described in the single-wire bus system section of the datasheet. These communications can include reading and writing the high and low thermostat trip-point registers (TH and TL) and the configuration register, and reading the temperature, counter and slope accumulator registers. Also in this mode, the microcontroller can initiate and stop temperature measurements as described in the operation-measuring temperature section of the datasheet.

The TH and TL registers and certain bits (THF, TLF, T/$\overline{R}$, POL and 1SHOT) in the status/configuration

register are stored in the non-volatile EEPROM memory, so these will retain data when the device is powered down. This allows the registers to be preprogrammed when the DS1821 is to be used as standalone thermostat.

Writing to these non-volatile registers can take up to 10 ms. To avoid data corruption, no write action to the non-volatile memory should be initiated while a write to the non-volatile memory is in progress. Non-volatile write status can be monitored by reading the NVB bit in the status/configuration register:

If NVB=1, a write to EEPROM memory is in progress.

If NVB=0, the non-volatile memory is in idle state.

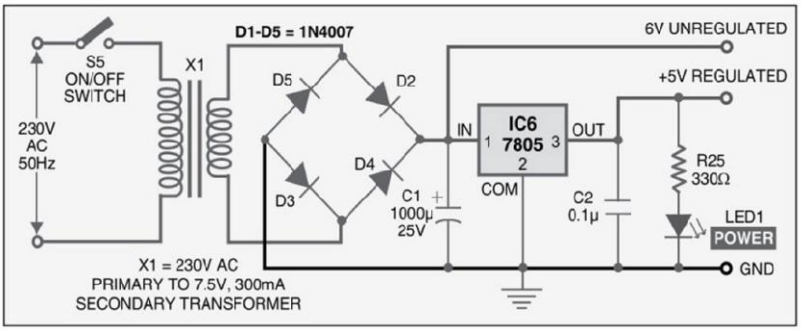


Fig. 2: Power supply circuit

Circuit description

Fig. 1 shows the circuit of the temperature controller using Dallas DS1821 temperature sensor. Microcontroller AT89S8252 is interfaced to DS1821 temperature sensor, three 7-segment displays and relay RL1. Port P1 of IC1 is used to output the data on the segment display. Port pins P1.0 through P1.3 and port pins P1.4 through P1.7 are connected to IC3 and IC2, respectively. ICs CD4511 (IC3 and IC2) receive the BCD data and provide the compatible code for 7-segment displays DIS2 and DIS3.

Port pins P3.4 and P3.5 are used for 'b,' 'c' and 'g' segments of DIS4 through buffers N1, N2 and N3, respectively. Segments 'b' and 'c' become active when temperature exceeds 99°C. Segment 'g' becomes active when temperature goes below 0°C. This indicates '–' sign for negative temperature. DIS1 is used in reverse direction for indication of °C. Segments 'a,' 'b,' 'g' and 'dp' (decimal point) are made permanently high with resistors R19 through R22 to indicate °C.

Port pins P3.1 through P3.3 of IC1 are connected to S2, S3 and S4 switches for 'up,' 'down' and 'display' respectively. These pins are pulled high through a 10-kilo-ohm resistor. Switches S1 through S3 are used for setting/changing the temperature. When the set temperature is exceeded, the relay connected to port pin 3.7 through a transistor is latched on. Switch S1 is used as a reset switch. Power-'on' reset is achieved by capacitor C3 and resistor R4.

Port pin P3.0 of IC1 receives the data

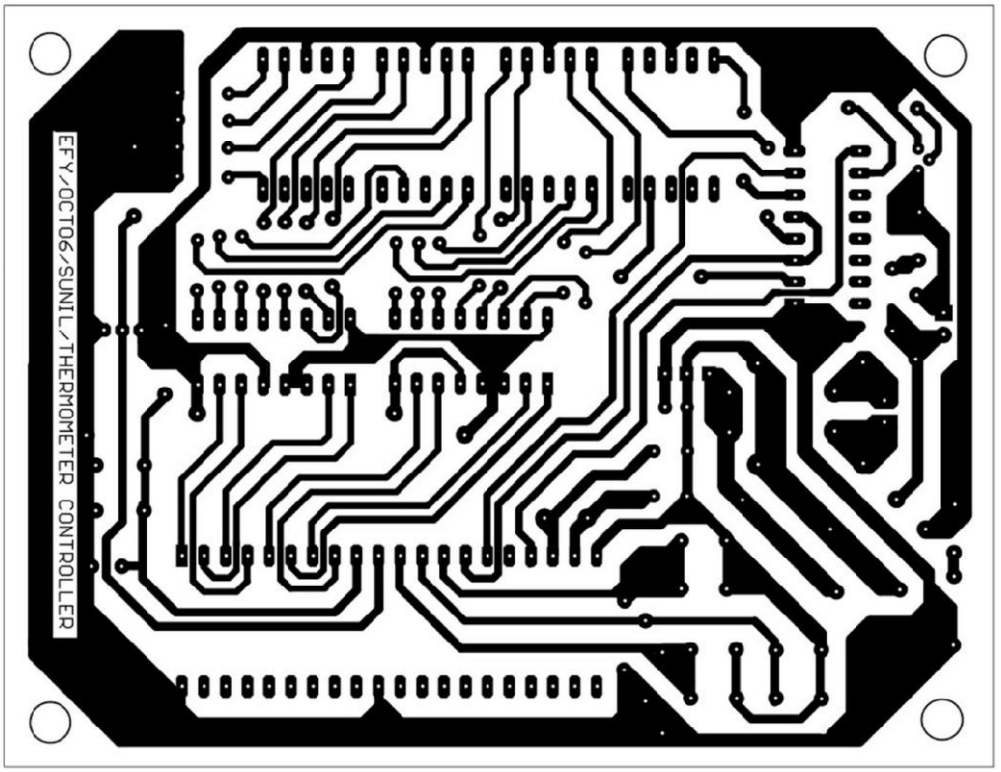


Fig. 3: Actual-size, single-side PCB layout for digital thermometer-cum-controller

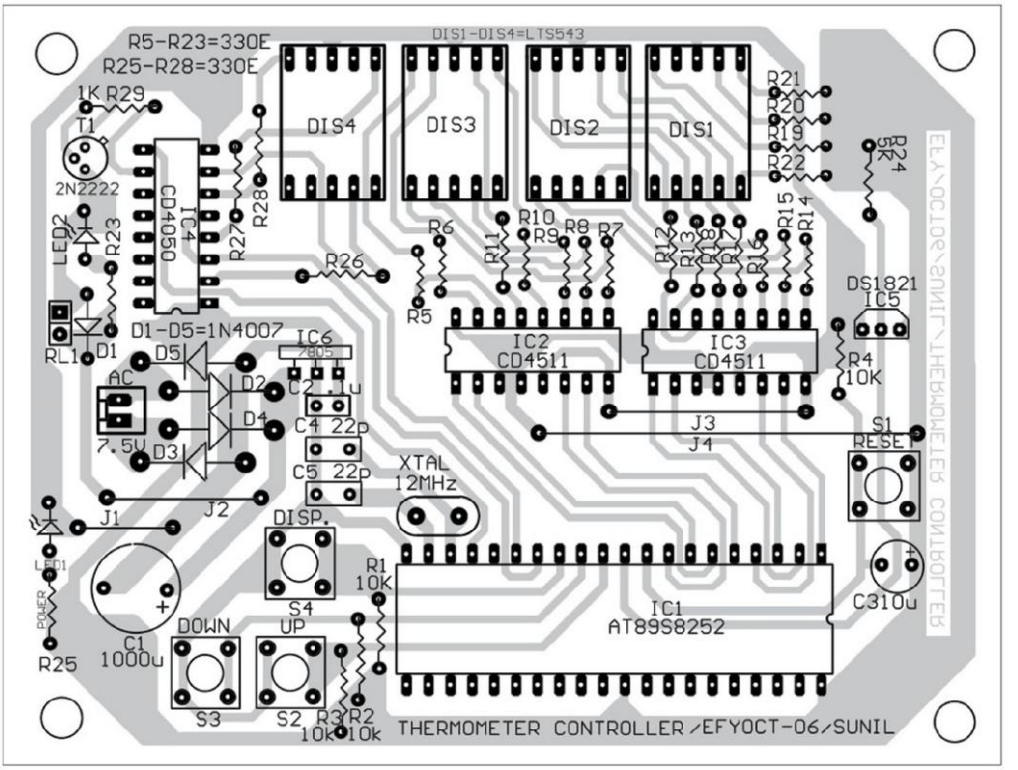


Fig. 4: Component layout for the PCB in Fig. 3

from temperature sensor DS1821. Pin 17 (P3.7) of IC1 is connected to the base of transistor T1 through buffer N4. The signal from port pin P3.7 drives relay RL1. Diode D1 is used as a free-wheeling diode and LED2 is used for relay-'on' indication. The device is connected through contacts of RL1. Resistors R5 through R22 and R26 through R28 limit the current through the 7-segment display. A 12MHz crystal is used for microcontroller clock.

Fig. 2 shows the circuit of power supply. The AC mains is stepped down by transformer X1 to deliver a secondary output of 7.5V at 300 mA. The transformer output is rectified by a full-wave bridge rectifier comprising diodes D2 through D5, filtered by capacitor C1 and regulated by IC6. Capacitor C2 bypasses any ripple present in the regulated output. Regulated 5V is used for circuit operation and unregulated 6V is used for the relay.

An actual-size, single-side PCB for temperature controller (Fig. 1) including its power supply (Fig. 2) is shown in Fig. 3 and its component layout in Fig. 4.

The software

The software for the temperature controller is compiled using Bascom51 version. The demo version of Bascom-8051 is available on website 'www.mcselec.com/download_8051. htm.'

First, define the crystal speed and include the header file for microcontroller. Initialise all ports to '0.' Timer-0 is used as an internal counter and increments by a second. This is used here for timing delays. Pin P3.0 of the microcontroller is configured for single-wire communication. Normally, DQ pin (single-wire bus) of DS1821 is high. Through DQ, the device gets its power to perform tasks.

When the microcontroller waits for the event through single-wire bus, it issues a reset command. Then DQ goes low for some time. This resets the device and it acknowledges a pulse and then follows the microcontroller. The communication on the single-wire bus is initiated by the microcontroller, and issued by low time slots on a normally-high DQ line, issued by the device.

Declare the variables as bits, bytes and words. Define the various port pins as per the connections. Set the maximum temperature to '40' as default. Subroutines 'dispset' and 'disptemp' are used for display preset and real temperature, respectively. The source program is well commented for easy understanding.

Download source code: http://www.efymag.com/admin/issuepdf/Temp-contr%20code.zip

TEMPR.BAS

```
' tempr.bas 22-8-06                    | ' by K.S.Sankar www.mostek.biz for EFY Magazine
' ds-1821 chip with 12 mhz xtal        | ' written in embedded basis- Bascom-51
```

```
' Language downloaded from http://www.mcselec.com hol-
land

' p3.0 =data wire

' 1 wire communication with Dallas temperature sen-
sor DS1821
 ' small PR35 Package, 3 pin , see data sheet for
more details
 ' Temperatures are in degres  when >-1 , <125
 ' for temperature <0 , 256 - tempdegree will give
 ' from -55 to -1
 ' count range
 ' - - - - - - - - - - -0++++++++++++
 ' -55 -54 -3   -2 -1   0 1 2 3 ...125
 '201 202  253 254 255  0 1 2 3 ...125

' port p1-yellow
' = two 7seg display thru two 4511 bcd-7decoder ics

' port-p3 blue
' p3.0= 1 wire interface
' p3.1= increase set value
' p3.2= decrease set value
' p3.3= display  set value

' p3.4= '1' for hundreds
' p3.5= '-' minus segment for negative

' p3.7= relay out

'define xtal speed
$crystal = 12000000
$regfile = "89s8252.dat"
'select chip used

P1 = 0
P0 = 0
P2 = 0
P3 = 0
'all ports off

P3 = &B01001110

' input port high for switches

' declare function used
Declare Sub Fn7seg(_i As Byte)
Dim _i As Byte


Config Timer0 = Timer , Gate = Internal , Mode = 2
'Timer0             use timer 0
'Gate = Internal    no external interrupt
'Mode = 2           8 bit auto reload

' set t0 internal interrupt  4000 times a sec with
12mhz xtal
On Timer0 Timer_0_overflow_int
Load Timer0 , 250
Priority Set Timer0
Enable Interrupts
Enable Timer0
' do not start timer0 here

Config 1wire = P3.0
'use P3.0 for 1 wire communication

Dim Sec_count As Byte
Dim Clock_word As Word
Dim I As Byte , J As Byte
Dim Tempdegree As Byte , Stat_buf As Byte , Disp_temp
As Byte
Dim Settemp As Byte , Set_disp_temp As Byte
Dim Set_mode As Bit
Dim Ans As Byte
Dim Comp_temp As Byte , Comp_set As Byte

Relay_out Alias P3.7
Sw_set_up Alias P3.1
Sw_set_down Alias P3.2
Sw_set_disp Alias P3.3
Sw_in_port Alias P3

Display_port Alias P1

' set default max temperature for relay to activate
Settemp = 40
Set_mode = 0

Begin:

1wreset
1wwrite &H0C
' write status
1wwrite &B01000010
' continue convertion
1wreset
1wwrite &HEE
 ' start conversion
1wreset
1wwrite &HAA
' get temperature
Tempdegree = 1wread()
1wreset

Gosub Disptemp

'-----------------
Rem check if set keys   pressed
' if pressed stay in set loop for 3 seconds
' after inactivity and display will be in flicker mode

Ans = &B01001110 And Sw_in_port
If Ans <> &B01001110 Then
' some input key pressed
Start Timer0
Sec_count = 0
Set_mode = 1

Begin2:

While 1 = 1
If Sec_count >= 3 Then
Set_mode = 0
Exit While
End If

If Sw_set_up = 0 Then
Sec_count = 0
Settemp = Settemp + 1

' - - - - - - - - - - -0++++++++++++
' -55 -54 -3   -2 -1   0 1 2 3 ...125
'201 202  253 254 255  0 1 2 3 ...125

' plus range
 If Settemp <= 200 Then
 If Settemp >= 125 Then
 Settemp = 125
 ' limit + reached
 End If
 End If
```

```
End If
'-----------------
If Sw_set_down = 0 Then
Sec_count = 0
Settemp = Settemp - 1

' - - - - - - - - - - -0++++++++++++
' -55 -54 -3   -2 -1   0 1 2 3 ...125
'201 202  253 254 255  0 1 2 3 ...125

 If Settemp < 201 Then
 If Settemp >= 200 Then

 Settemp = 201
 ' (-55 degrees)
 ' limt exceeded
 End If
 End If
'-----------------
End If

If Sw_set_disp = 0 Then
Sec_count = 0
End If
'-----------------
Gosub Dispset

Wend

Stop Timer0
End If

' - - - - - - - - - - -0++++++++++++
' -55 -54 -3   -2 -1   0 1 2 3 ...125
'201 202  253 254 255  0 1 2 3 ...125

' check if real temperature is higher than set value
' add 55 to both sides to avoid negative errors for
comparison
' byte variable does not support (-) values
Comp_temp = Tempdegree + 55
Comp_set = Settemp + 55

''''If Tempdegree >= Settemp Then
If Comp_temp >= Comp_set Then
Relay_out = 1
'relay on
Else
Relay_out = 0
'relay off
End If

Goto Begin

' -==-=-=-=-=-=-=- subroutines below-----
Disptemp:
Disp_temp = Tempdegree

Rem display real temperature
' display on 7 seg
P3.4 = 0
P3.5 = 0
' - - - - - - - - - - -0++++++++++++
' -55 -54 -3   -2 -1   0 1 2 3 ...125
'201 202  253 254 255  0 1 2 3 ...125

If Tempdegree >= 100 Then
 If Tempdegree <= 125 Then
 Disp_temp = Tempdegree - 100
 P3.4 = 1
 ' switch on hundred segment b/c
 End If
End If

If Tempdegree >= 201 Then
Disp_temp = 256 - Tempdegree
P3.5 = 1
' switch on minus [-] segment g
End If

Call Fn7seg(disp_temp)

Return
'--------------------------------
Dispset:
Rem display preset  temperature
Set_disp_temp = Settemp
P3.4 = 0
P3.5 = 0
' - - - - - - - - - - -0++++++++++++
' -55 -54 -3   -2 -1   0 1 2 3 ...125
'201 202  253 254 255  0 1 2 3 ...125
If Settemp >= 100 Then
 If Settemp <= 125 Then
 Set_disp_temp = Settemp - 100
 P3.4 = 1
 ' switch on hundred segment b/c
 End If
End If

If Settemp >= 200 Then
Set_disp_temp = 256 - Settemp
P3.5 = 1
' switch on minus [-] segment g
End If

Set_mode = 1
Call Fn7seg(set_disp_temp)
Waitms 50
Return

'=-=-=-=-=-= function below---- - -- -
Sub Fn7seg(_i As Byte)
Dim _ans As Byte
' display on two 7 seg
_ans = Makebcd(_i)
Display_port = _ans
If Set_mode = 1 Then
' if in set mode make display flicker
Display_port = 255
' blankout the display
Waitms 10
' turn it on again
Display_port = _ans
Waitms 10
End If

End Sub

' interrupt subroutine -----------------
Timer_0_overflow_int:
' program comes here 4000 times a sec
' with a 12mhz xtal
Incr Clock_word

If Clock_word > 4000 Then
Clock_word = 0
Incr Sec_count
End If
Return
End
' prog size = 914 bytes
' end of program
```

AT89S52-BASED INDUSTRIAL TIMER

■ JAYARAMAN KIRUTHI VASAN

Industrial timers can be con-structed using discrete compo-nents including up/down counters and timers. However, to incorporate various facilities like setting the count, start, stop, reset and display, these circuits would require too many ICs and discrete components.

A microcontroller-based industrial timer can be programmed and used as a timer, counter and time totaliser. Here is a simple design based on 40-pin Atmel AT89S52 microcontroller that performs count-down operation up to 9999 minutes/second with four 7-segment displays showing the actual time left. The relay energises as you press the start switch and remains on till the countdown reaches '0000.' Four tactile, push-to-on switches are used to start/stop, select either minutes or seconds, and set the initial value for countdown operation (using 'up' and 'down' keys).

Circuit description

Fig. 1 shows the circuit of the microcontroller-based industrial timer. The microcontroller is Atmel AT89S52 (IC1), which is a 40-pin device with 8 kB of program flash memory, 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. The power-down mode saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next interrupt or hardware reset is activated.

Port P0 of microcontroller AT89S52 is configured for segments of the 7-segment display. Port 0 is an 8-bit open-drain bidirectional I/O port. Port 0 is pulled up with 10-kilo-ohm resistor network RNW1. Port pins P0.0 through P0.6 are connected to pins of segments 'a' through 'g' via resistors R2 through R8, respectively. Port P0.7 is connected to decimal via resistor R9. Resistors R2 through R9 are used as current limiter for various seg-

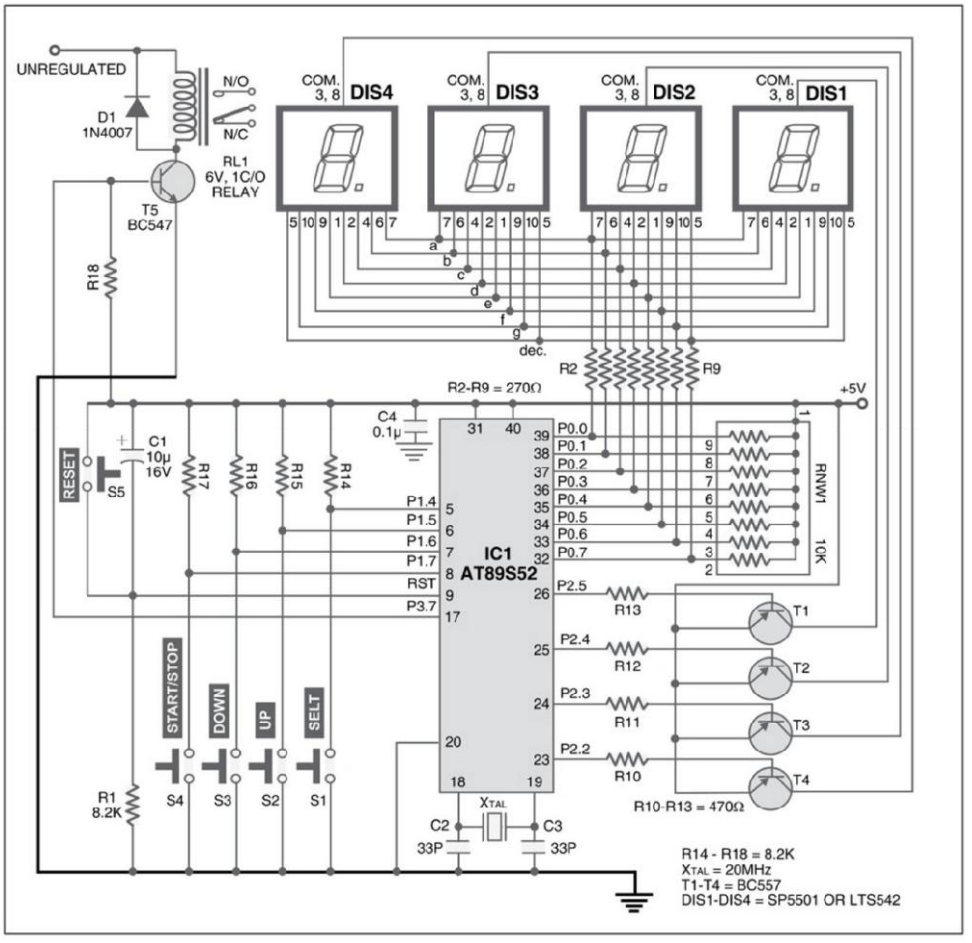


Fig. 1: Microcontroller-based industrial timer

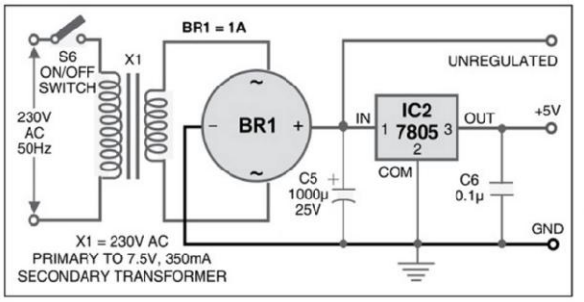


Fig. 2: Power supply circuit

ments of displays, respectively.

Port 2 is used to control DIS1 through DIS4. Port 2 is an 8-bit bidirectional I/O port with internal pull-ups. When port-2 pin is low, the transistor conducts and provides supply to the common pin of 7-segment display. Port pins P2.5 through P2.2 control DIS1 through DIS4 with the help of transistors T1 through T4, respectively.

The microcontroller drives the 7-segment displays in multiplex mode. This helps in reducing current consumption while maintaining the brightness of the display. For driving the displays, timer 2 inside the microcontroller is used. It enables display of each digit every two milliseconds.

For driving the displays, the microcontroller uses port-0 to send the segment outputs. It selects the corresponding unit's, ten's, hundred's and thousand's displays through P2.5, P2.4, P2.3 and P2.2, respectively.

Four pins of port 1 are used for various switches like select, up, down and start/stop. Port 1 is an 8-bit bidirectional I/O port with internal pull-ups. Switches S1 through S4 are connected to pins 5 through 8 of the microcontroller and used for select, up, down and start/stop functions, respectively.

Pin P3.7 controls relay RL1. When pin P3.7 goes high, transistor T5 is driven into saturation and relay RL1 energises. Diode D1 serves as a free-wheeling diode. Any appliance can be connected with contacts of relay RL1.

Power-on-reset is achieved by connecting resistor R1 and capacitor C1 to pin 9 of the microcontroller. Other ends of the capacitor and resistor are connected to Vcc and ground, respectively. Switch S5 is used for manual reset. The microcontroller is operated with the clock derived from a 20MHz crystal oscillator.

Power supply. Fig. 2 shows the circuit of the power supply. The AC mains is stepped down by transformer X1 to deliver a secondary output of 7.5V at 350 mA. The transformer output is rectified by a full-wave bridge rectifier BR1, filtered by capacitor C5 and regulated by IC2. Capacitor C6 bypasses any ripple present in the regulated

PARTS LIST

Semiconductor:

IC1	- AT89S52 micrcontroller
IC2	- 7805 5V regulator
T1-T4	- BC557 pnp transistor
T5	- BC547 npn transistor
BR1	- 1A bridge rectifier
D1	- 1N4007 rectifier diode
DIS1-DIS4	- LTS542 common-anode display

Resistors (all ¼-watt, ±5% carbon):

R1, R14-R18	- 8.2-kilo-ohm
R2-R9	- 270-ohm
R10-R13	- 470-ohm
RNW1	- 10-kilo-ohm resistor network

Capacitors:

C1	- 10µF,16V electrolytic
C2, C3	- 33pF ceramic disk
C4, C6	- 0.1µF ceramic disk
C5	- 1000µF, 25V electrolytic

Miscellaneous:

X1	- 230V AC primary to 6V, 350mA secondary transformer
X_{TAL}	- 20MHz crystal
RL1	- 6V, 1C/O relay
S1-S5	- Push-to-on switch
S6	- On/off switch

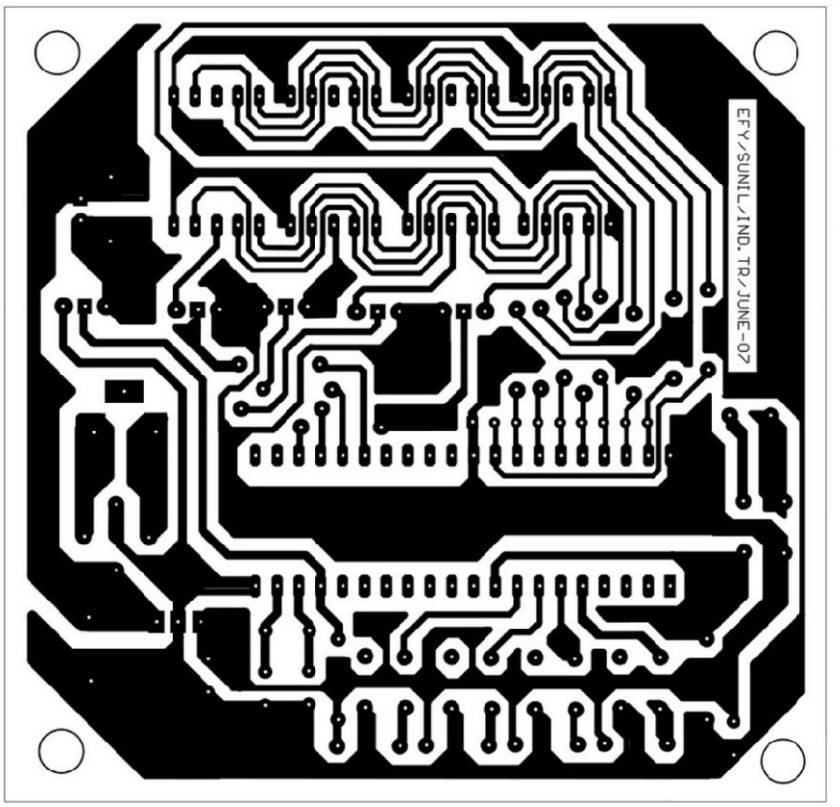


Fig. 3: An actual-size, single-side PCB layout for the microcontroller-based industrial timer including power supply

output. Unregulated power supply is used for relay RL1.

An actual-size, single-side PCB layout for the microcontroller-based industrial timer (Fig. 1) including power supply (Fig. 2) is shown in Fig. 3 and its components layout in Fig.4.

Operation

Switch on the circuit using using ON/OFF switch S6. The microcontroller is reset by power-on-reset and then timer is in seconds mode. The 'select' key selects the mode between 'seconds' and 'minutes.' This is displayed as '0' for seconds and '1' for minutes on the hundred's digit display (DIS3), respectively.

'Up' key increments the time setting in seconds and minutes.

'Down' key decrements the time setting in seconds and minutes.

After setting the desired time with the help of 'up' and 'down' keys, press 'start' key. This energises the relay. The timer counts down for the set time and once the display becomes zero, the relay de-energises. The timer will stop before preset time by pressing 'start' key again.

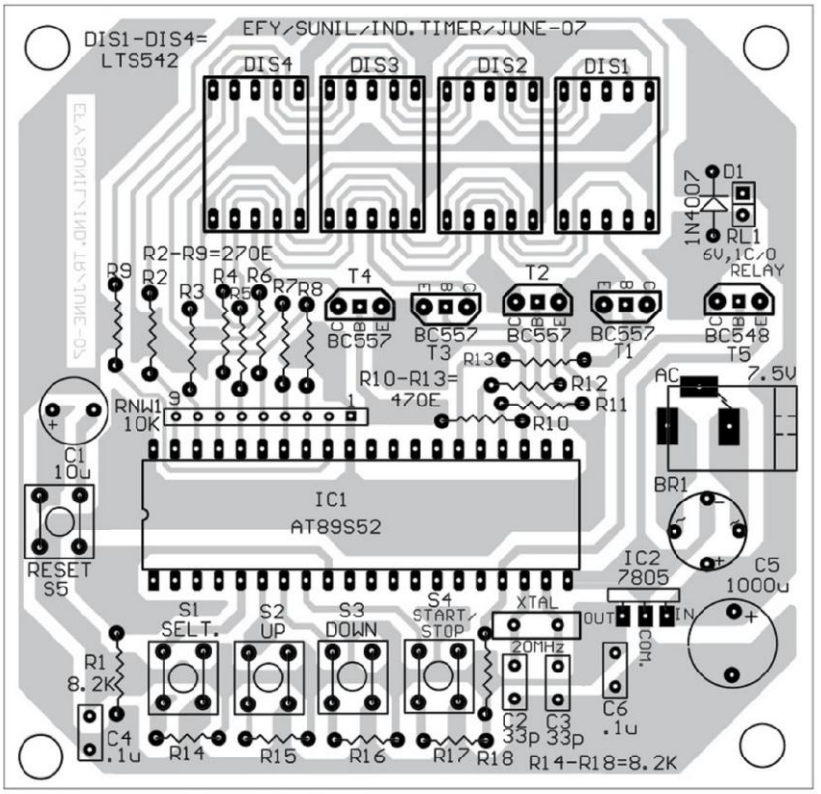


Fig. 4: Components layout for the PCB

Software

The source program is written in 'C' language and compiled with Keil Microvision 3 IDE. It is well commented and easy to understand. Download C51V808A.EXE from 'www.keil.com/demo/eval/c51.htm.' This file is a freely available and self-extracting setup program for Keil Microvision 3 IDE.

Normally, when there is no interrupt, the microcontroller executes 'while' loop in the main function. Here it scans the keys and acts according to the key pressed.

Two interrupts are enabled in the software, namely, timer 0 and timer 2. Timer 0 counts milliseconds, which are then accumulated to seconds or minutes according to the user selection. Timer 2 drives the displays in multiplex mode.

For time counting, timer 0 is initialised by the void Timer0_init(unsigned char Timer0h,unsigned char Timer0l) function. Timer 0 interrupts the microcontroller every millisecond.

When interrupted by timer 0, the microcontroller executes the void isr_t0(void) function wherein it increments two counter variables, namely, Timer0Counter and LedCounter. Timer0Counter is responsible for counting the number of milliseconds elapsed and increments the minutes/seconds counter according to the mode selected ('seconds' or 'minutes' count). Once the set value is reached, the timer-0 interrupt is disabled and time counting stops.

The LED counter makes the dot LED of the unit's digit flash every second once.

Display-driving process is taken care of by the built-in timer 2. Timer 2 is initialised by the void Timer2_init(unsigned char Timer2h,unsigned char Timer2l) function.

Timer 2 gives an interrupt to the microcontroller to switch on the common pin of each 7-segment display for every two milliseconds. When an interrupt occurs, the void isr_t2(void) function is executed and the microcontroller returns to 'while' loop in the main function.

Download Source Code: http://www.efymag.com/admin/issuepdf/Industrial%20Timer.zip

SOURCE PROGRAM

```
/*** Include Files ***/
#include <At89x52.h>

/*** RENAMING OF PORTS ****/
#define SegPort P0
#define DigPort P2

/* CODE FOR LIGHTING EACH SEGMENT OF THE SEVEN SEG-
MENT LED DISPLAY */
#define seg_a 0xfe
#define seg_b 0xfd
#define seg_c 0xfb
#define seg_d 0xf7
#define seg_e 0xef
#define seg_f 0xdf
#define seg_g 0xbf
#define seg_dot 0x7f
/* SEVEN SEGMENT CODE FOR EACH NUMBER FROM 0 TO 9
,DOT AND SPACE */
#define NUM_0 (seg_a & seg_b & seg_c & seg_d & seg_e
& seg_f)
#define NUM_1 (seg_b & seg_c)
#define NUM_2 (seg_a & seg_b & seg_d & seg_e & seg_g)
#define NUM_3 (seg_a & seg_b & seg_c & seg_d & seg_g)
#define NUM_4 (seg_b & seg_c & seg_f & seg_g)
#define NUM_5 (seg_a & seg_c & seg_d & seg_f & seg_g)
#define NUM_6 (seg_a & seg_c & seg_d & seg_e & seg_f
& seg_g)
#define NUM_7 (seg_a & seg_b & seg_c)
#define NUM_8 (seg_a & seg_b & seg_c & seg_d & seg_e
& seg_f & seg_g)
#define NUM_9 (seg_a & seg_b & seg_c & seg_d & seg_f
& seg_g)
#define NUM_DOT (seg_dot)
#define NUM_SPACE 0Xff;

const unsigned char hex_table[] =
{
NUM_0,NUM_1,NUM_2,NUM_3,NUM_4, NUM_5,NUM_6,NUM_7,
 NUM_8,NUM_9,NUM_DOT
};
/* ADDRESS FOR SELECTING THE COMMON PIN OF THE DIS-
PLAY FOR EACH DIGIT */
#define UNITS 0xdf
#define TENS 0xef
#define HUNDS 0xf7
#define THS 0xfb
/* RELOAD VALUE FOR TIMER2 FOR INTERRUPT DURATION OF
2 MILLISECONDS */
#define TIMER2H_2MS 0xf2
#define TIMER2L_2MS 0xfb
/* RELOAD VALUE FOR TIMER0 FOR INTERRUPT DURATION OF
1 MILLISECOND */
#define TIMER0H_1MS 0xf9
#define TIMER0L_1MS 0x7e
/* VECTOR VALUE FOR TIMER INTERRUPT */
#define TIMER0VECTOR 1
#define TIMER2VECTOR 5
/* MINUTES AND SECONDS CONSTANTS */
#define SEC 999
#define MIN (60 * SEC)
/* VARIABLES DEFINITION */
unsigned char Units;
unsigned char Tens;
unsigned char Hunds;
unsigned char Ths;
unsigned int Timer0Counter=0;
unsigned char DisplayCounter=0;
unsigned int LedCounter=0;
unsigned char TimeDig;
unsigned int OneSecCount=0;
unsigned int SetSec=0;
unsigned char OneMinCount;
unsigned char key;
unsigned char KeyCount;
unsigned char Mode;
/* FUNCTION PROTOTYPES */
void Timer0_init(unsigned char Timer0h,unsigned char
Timer0l);
void Timer2_init(unsigned char Timer2h,unsigned char
Timer2l);
void Display(unsigned char Digit);
void IntToSevSeg(unsigned int TimeTemp);
void KeyDebounce(unsigned char dly);
void Keyscan(void);
/* RENAMING PORT PINS FOR EASY USAGE */
sbit RELAY = P3^7;
sbit SEL = P1^4;
sbit UP = P1^5;
sbit DN = P1^6;
sbit STRT = P1^7;
/* MAIN FUNCTION */
void main(void)
{
RELAY = 0;
Timer2_init(TIMER2H_2MS,TIMER2L_2MS);
Timer0_init(TIMER0H_1MS,TIMER0L_1MS);
TR0=0;
EA=1;
while(1)
    {
    if(TR0==0)IntToSevSeg(SetSec);
    else  if(  (TR0==1)&&  (Timer0Counter==0)  )
IntToSevSeg(OneSecCount);
    Keyscan();
                switch(key)
                {
                case 4:

if(SetSec>0)

{

if(TR0==0)

    {

    OneSecCount = SetSec;
```

```
    RELAY = 1;

    TR0=1;

    }

else

    {

    TR0=0;

    }

}
                                        key=0;
                                        break;
                case 3:
                                        Set-
Sec--;

if(SetSec>9999)

SetSec=9999;
                                        key=0;
                                        break;
                case 2:
                                        Set-
Sec++;

if(SetSec>9999)

SetSec=0;
                                        key=0;
                                        break;
                case 1:
                                        Mode++;

if(Mode>1)Mode=0;

Hunds=hex_table[Mode];
                                        Tens =
NUM_SPACE;
                                        Units
= NUM_SPACE;

key=0;
                                        KeyDe-
bounce(10);
                                        break;
                }
    }
}

/* TIMER INITIALISATION FUNCTIONS */
void Timer0_init(unsigned char Timer0h,unsigned char
Timer0l)
{
TMOD &= 0xf0;
TMOD |= 0x01;
TH0 = Timer0h;
TL0 = Timer0l;
ET0=1;
TR0=1;
}

void Timer2_init(unsigned char Timer2h,unsigned char
Timer2l)
{
T2CON = 0x04;
T2MOD = 0x00;
TH2 = Timer2h;
RCAP2H=Timer2h;
TL2 = Timer2l;
RCAP2L=Timer2l;
ET2=1;
TR2=1;
}

/* KEYSCAN FUNCTIONS */
void Keyscan(void)
{
while( (SEL==0) || (STRT==0) )
                {
                KeyDebounce(1);
                if(SEL==0)key=1;
                else if(STRT==0)key=4;

                }
if ((UP==0) || (DN==0) )
    {
    KeyCount--;
    if(KeyCount<1)KeyCount=1;
    KeyDebounce(KeyCount);
    if(UP==0)key=2;
    else if(DN--0)key=3;
    }
else
    KeyCount=10;

}

void KeyDebounce(unsigned char dly)
    {
    unsigned int z;
    while(dly>0)
                {
                dly--;
                for(z=0;z<8000;z++);
                }
    }

/* DISPLAY FUNCTIONS */
void Display(unsigned char DigCount)
{
switch(DigCount)
    {
    case 0:
                DigPort = UNITS;
                SegPort = Units;
                break;
    case 1:
                DigPort = TENS;
```

```
                SegPort = Tens;
                break;
    case 2:
                DigPort = HUNDS;
                SegPort = Hunds;
                break;
    case 3:
                DigPort = THS;
                SegPort = Ths;
                break;
    }
}

/* INTEGER VALUE TO SEVEN SEGMENT CONVERSION FUNC-
TION */
void IntToSevSeg(unsigned int TimeTemp)
{
TimeDig = TimeTemp/1000;
TimeTemp -=(TimeTemp/1000)*1000;
if(TimeDig==0)
    {
    Ths=NUM_SPACE;
    }
else
    {
    Ths = hex_table[TimeDig];
    }
TimeDig = TimeTemp/100;
TimeTemp -=(TimeTemp/100)*100;
if((Ths==0xff)&&(TimeDig==0))
    {
    Hunds=NUM_SPACE;
    }
else
    {
    Hunds = hex_table[TimeDig];
    }
TimeDig = TimeTemp/10;
TimeTemp -=(TimeTemp/10)*10;
if((Hunds==0xff)&&(TimeDig==0))
    {
    Tens=NUM_SPACE;
    }
else
    {
    Tens = hex_table[TimeDig];
    }
TimeDig = TimeTemp%10;
Units = hex_table[TimeDig];
}

/* Interrupt Routines */

/* Drives LED Displays */
void isr_t2(void) interrupt TIMER2VECTOR
{
    DisplayCounter++;//Increments every 50 ms.
    if(DisplayCounter>3)DisplayCounter=0;
    Display(DisplayCounter);
    TF2 = 0;
}

/* Counts Seconds */
void isr_t0(void) interrupt TIMER0VECTOR
{
TH0 = TIMER0H_1MS;
TL0 = TIMER0L_1MS;
TF0=0;
Timer0Counter++;//Counts every 1msec.
LedCounter++;
    if(LedCounter<200)
                Units= Units & NUM_DOT;
    else if( (LedCounter>=200) && (LedCounter <999)
)
                Units = Units | 0x80;
    else
                LedCounter=0;

if(Mode==0)
    {
    if(Timer0Counter>SEC)
                {
                Timer0Counter=0;
                OneSecCount--;
                if(OneSecCount==0)
                                        {
                                        //OneSec-
Count=0;
                                        TR0=0;
                                        RELAY = 0;
                                        }
                }
    }
else if(Mode==1)
    {

    if(Timer0Counter>MIN)
                {
                Timer0Counter=0;
                OneSecCount--;
                if(OneSecCount>MIN)
                                        {
                                        OneSecCount=0;
                                        TR0=0;
                                        RELAY = 0;
                                        }
                }
    }
}

/***************** END ****************/
```

LOW-COST LCD FREQUENCY METER

■ K.S. SANKAR

Frequency meters have always been expensive tools for the average hobbyists. Now, with microcontrollers and liquid-crystal displays (LCDs) having become very economical and popular, it is possible to build a compact and low-cost LCD-based frequency meter that can measure up to 250 kHz.

A sample photo of the LCD module is shown in Fig. 1. These modules are available in 14- and 16-pin configurations. The 16-pin module has a backlight option. Popular

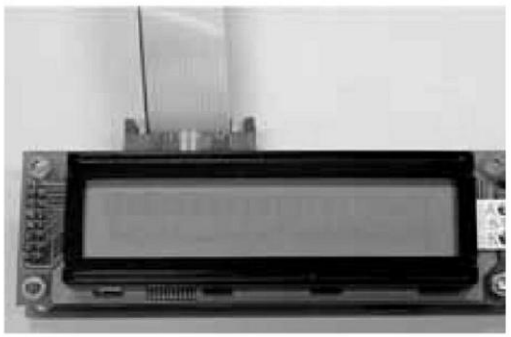


Fig. 1: A Typical LCD module

TABLE I
Pin Details of LM16200

Pin No.	Description	Pin no.	Description
1	Gnd	5	R/W
2	Vcc	6	E
3	Vo	7 to 14	DB0 to DB7
4	RS	15 & 16	LED BL A & K

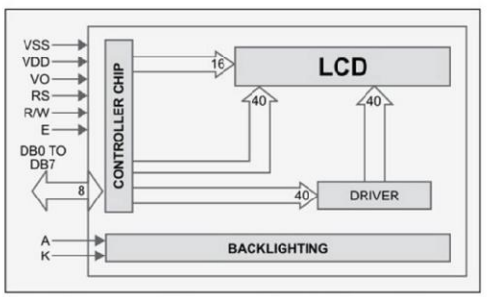


Fig. 2: Functonal diagram of the LCD module

Syntax of Important Instructions Used in the Program with Examples

1. CONFIG LCDPIN. This instruction stores the pin usage in your program:
Syntax:
CONFIG LCDPIN = PIN,
Example:
CONFIG LCDPIN = PIN, DB4= P1.1, DB5=P1.2, DB6=P1.3, DB7=P1.4, E=P1.5, RS=P1.6
Note. LCD-module pin names are as used in Fig. 3.
2. CONFIG LCD. This instruction is used for configuring the LCD display type:
Syntax:
CONFIG LCD = LCD type
LCD type can be one of the following:
40x4, 40x2, 16x1, 16x2, 16x4, 20x2 or 20x4
or 16x1a or 40x4a.
Note. Default 16x2 is assumed. The 16x1a LCD display is a special one. It is intended for the display that has the memory organised as two lines of eight characters. The 40x4a LCD display is also a special one. It has two ENABLE lines.
Example:
CONFIG LCD = 40x4
LCD "Hello" instruction is used for displaying 'Hello' on the LCD screen.
FOURTHLINE instruction selects line No. 4 of the screen for subsequent instruction.
LCD "4" displays '4' on the screen
END
3. CONFIG TIMER0, TIMER1. This instruction is used for configuring timer-0 or timer-1.
Syntax:
CONFIG TIMERx = COUNTER/TIMER, GATE=INTERNAL/EXTERNAL, MODE=0/3
Remarks:

TIMERxTIMER0 or TIMER1.
COUNTER will configure TIMERx as a COUNTER and TIMER will configure
TIMERx as a TIMER.
A TIMER has built-in clock input and a COUNTER has external clockinput.

GATE INTERNAL or EXTERNAL. Specify EXTERNAL to enable gate controlwith the INT input.

MODE Specify timer/counter mode 0-3. See the datasheet for more details.

Example:
CONFIG TIMER0=COUNTER, MODE=1, GATE=INTERNAL
COUNTER0 = 0 instruction resets counter 0
START COUNTER0 enables the counter to start counting
DELAY 'wait a while
PRINT COUNTER0 instruction prints the counter0 count.
END

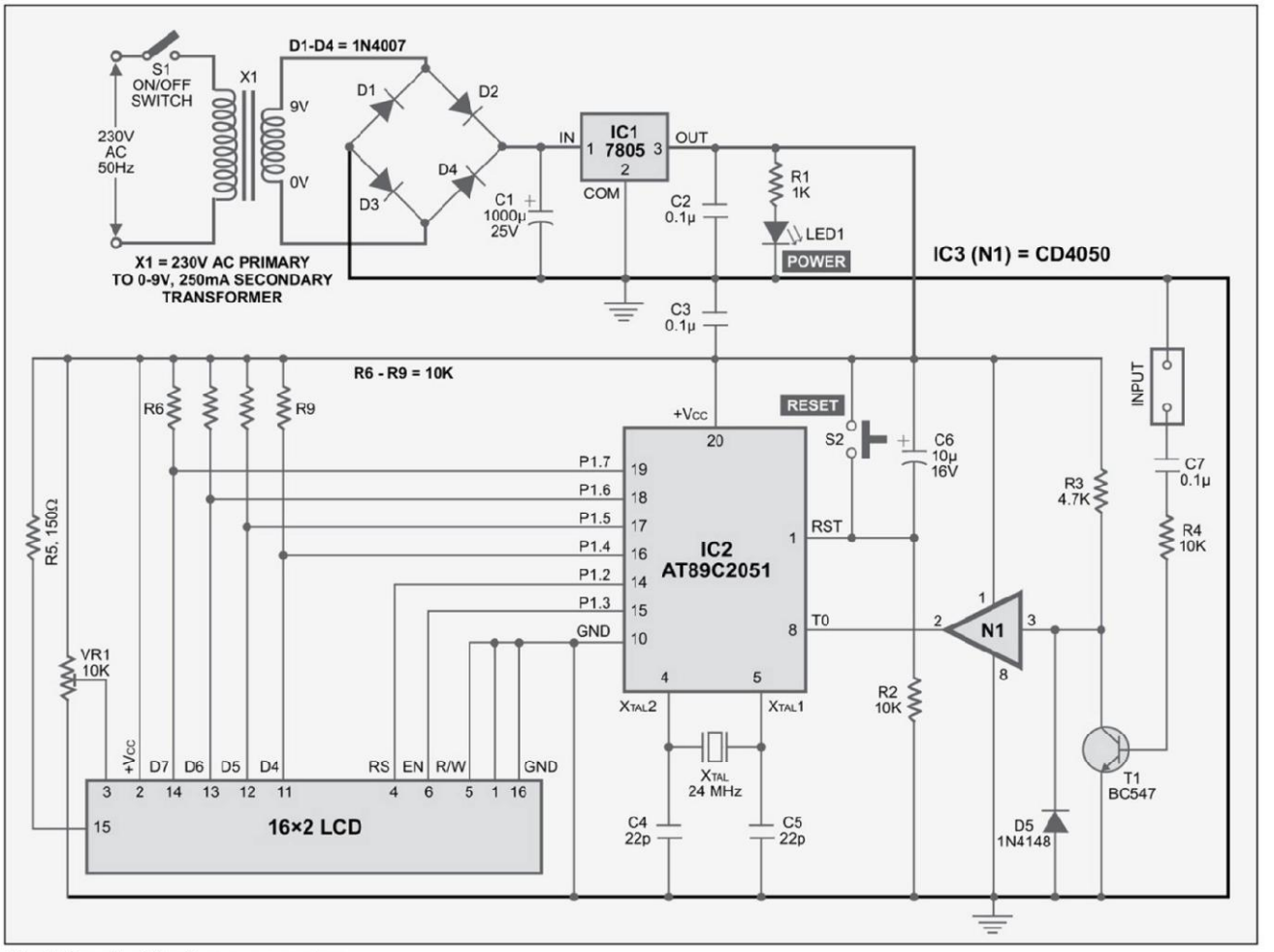


Fig. 3: Circuit of the frequency meter

brands are Lampex and Hantronix. Note the pin numbers before soldering to the circuit.

In this project, the LCD module used is Lampex LM16200 with 16 alphanumeric characters and two lines with backlight option. Pin details of this module are given in Table I. A functional diagram of the module is shown in Fig. 2.

However, you may use any branded or unbranded 2-line, 16-character LCD module for this project. The 10k potentiometer, which controls the contrast of the LCD module, works best when its wiper contact is nearer to ground potential.

TABLE II

LCD Connections Used for 4-Bit Data Mode

LCD display	Port	Pin
DB7	P1.7	14
DB6	P1.6	13
DB5	P1.5	12
DB4	P1.4	11
E	P1.3	6
RS	P1.2	4
RW	Ground	5
Vss	Ground	1
Vdd	+5 volt	2
Vo	0-5 volt	3

Circuit description

Fig. 3 shows the circuit of the frequency counter

PARTS LIST

Semiconductors:
IC1 - 7805 5V regulator
IC2 - AT89C2051 microcontroller
IC3 - CD4050 hex buffer
T1 - BC547 npn transistor
D1-D4 - 1N4007 rectifier diode
D5 - 1N4148 switching diode
LED1 - 5mm LED

Resistors (all ¼-watt, ±5% carbon):
R1 - 1-kilo-ohm
R2, R4, R6-R9 - 10-kilo-ohm
R3 - 4.7-kilo-ohm
R5 - 150-ohm

Capacitors:
C1 - 1000µF, 25V electrolytic
C2, C3, C7 - 0.1µF ceramic disk
C4, C5 - 22pF ceramic disk
C6 - 10µF, 16V electrolytic

Miscellaneous:
X1 - 230V AC primary to 9V, 250mA secondary transformer
S1 - On/Off switch
S2 - Push-to-on switch
X_{TAL} - 24 MHz
16×2 LCD

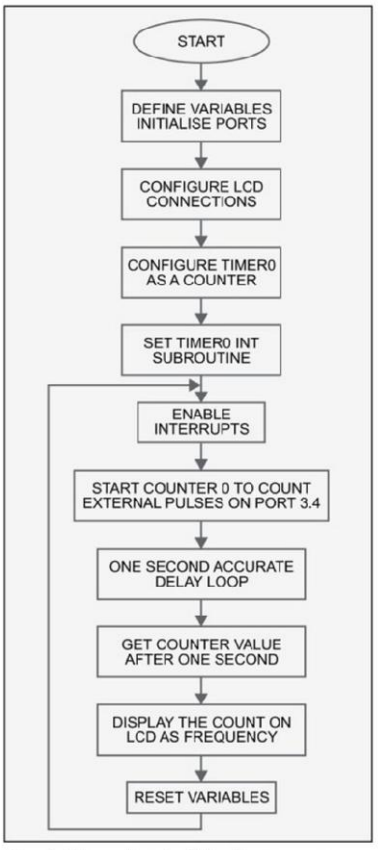


Fig. 4: Flow-chart of the frequency-counting program

including the power supply. The microcontroller used is AT89C2051, which features 2 kB of Flash, 128 bytes of RAM, 15 input/output (I/O) lines, two 16-bit timers/counters, a five-vector two-level interrupt architecture, a full-duplex serial port, a precision analogue comparator, an on-chip oscillator and clock circuitry.

Port-1 is used to drive the LCD in 4-bit mode with 10-kilo-ohm pull-up resistors. The 24MHz crystal used gives a processing speed of 2 million instructions per second (MIPS).

Timer-0 is used as an external counter to count the input pulses. Transistor T1 amplifies the input signal, while non-inverting gate N1 (1/6 CD4050) serves as a buffer for coupling the amplified pulses to input pin 8 (P2.4) of timer-0.

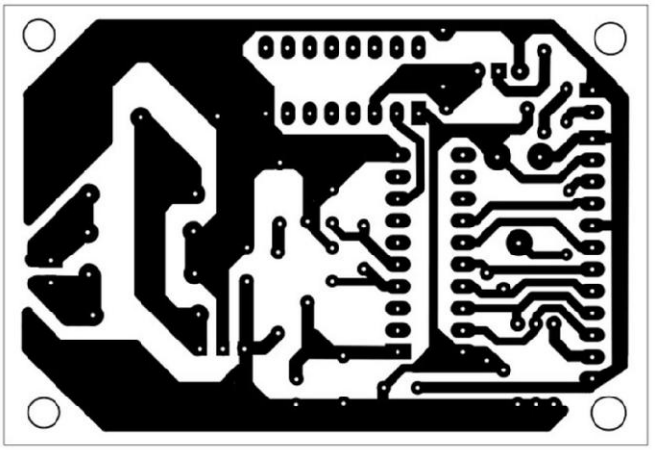


Fig. 5: Actual-size, single-side PCB layout for frequency meter

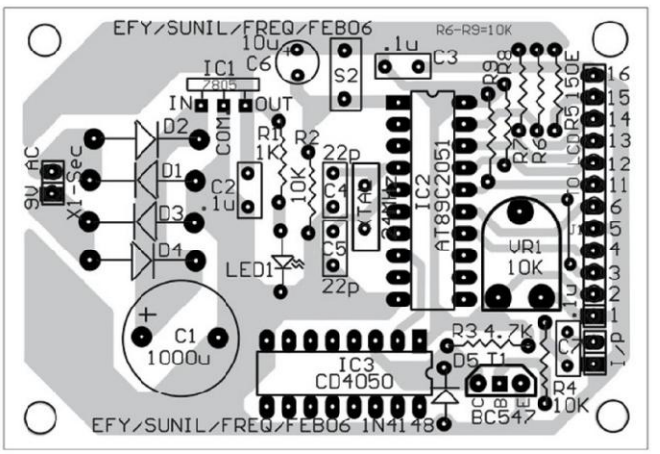


Fig. 6: Component layout for the PCB

A software gate of one-second duration is used to count the number of pulses corresponding to the frequency of the input signal source. The count value is read and displayed on the 2-line, 16-character LCD module. The flow-chart of the frequency-counting routine is shown in Fig. 4.

A conventional power supply circuit comprising a step-down transformer followed by a bridge rectifier, smoothing capacitor and 5V regulator is used to power the circuit. Capacitor C2 (0.1µF) filters ripples in the output of the regulator and LED1 shows the supply status. To test the circuit, connect any pulse generator output to the probe and check the frequency displayed on the LCD screen.

The LCD module is used in the 4-bit data interface mode, wherein only data pins DB4 through DB7 are used for data transfer. The configuration used is shown in Table II.

An actual-size, single-side PCB for the LCD frequency meter (Fig. 3) is shown in Fig. 5 and its component layout in Fig. 6.

The software

The software is compiled using the demo version of BASCOM-8051, which can be downloaded from website 'www.mcselec. com.'

Syntax of some of the important instructions used in the program is shown in the box along with examples. The BASCOM compiler provides special instructions for use and display of data on the LCD module.

For use of BASCOM, you may refer to the article 'Real-Time Clock' published in Jan. 2005 issue of EFY.

The source code file EFY80FM24.BAS for this LCD frequency meter in BASCOM-51 is given at the end of this article. The same may be modified to meet your specific requirements.

Download Source Code: http://www.efymag.com/admin/issuepdf/Frequency%20Meter.zip

EFY80FM24

```
'-----------------------------------------------
------------
' file: efy80fm24.BAS  3-12-05
' ok with word variable 45440
'  Frequency  Meter  Program  using  AT89c2051  micro
controller
' written using bascom-51
' from www.mcselec.com holland
' an embedded visual basic compiler for 8051 micro-
controllers
' by K.S.Sankar Web: www.mostek.biz
'-----------------------------------------------
------------
' Connect the timer0 input P3.4 to a frequency gen-
erator
' freq meter
' 24  mhz xtal ok upto 300khz

' define crystal speed and include file
$regfile = "89C2051.dat"
$crystal = 24000000

' define variables used
Dim A As Byte
Dim C As Long , D As Long
Dim Count As Word
Dim Onceasec As Bit
Dim T0ic As Long
Dim Green As Byte
Dim Delayword As Word

' Initialize variables
Onceasec = 0
Count = 0
T0ic = 0
D = 0
Green = 0

' initialize ports
P1 = 0
P3 = 255

' configure lcd display
Config Lcd = 16 * 2
Config Lcdpin = Pin , Db4 = P1.4 , Db5 = P1.5 , Db6 =
P1.6 , Db7 = P1.7 , E = P1.3 , Rs = P1.2
Cls
 'clear the LCD display
Lcd "Frequency Meter"

' define timer0
Config Timer0 = Counter , Gate = Internal , Mode = 1
'Timer0 = counter : timer0 operates as a counter
'Gate = Internal  : no external gate control
' exte/internal makes no difference
'Mode = 1         : 16-bit counter
' set t0 internal interrupt

On Timer0 Timer_0_overflow_int
' interrupt will be generated on every 65536 count
Priority Set Timer0
Enable Interrupts
Enable Timer0

Counter0 = 0
'clear counter
Start Counter0
'enable the counter to count

Do
'set up a 1 sec accurate DO NOTHING loop
Enable Interrupts
'wait 1 as per BASCOM-51 is not accurate

For Delayword = 1 To 45440
Next Delayword

Disable Interrupts
C = Counter0
'get counter value
D = T0ic * 65536

Lowerline
C = C + D
T0ic = 0
Lcd "                "
Lowerline
' show the frequency
Lcd "f=" ; C ; " Hz"
Waitms 255
Waitms 255

C = 0

Counter0 = 0
Start Counter0
're-start it because it was stopped by accessing the
COUNTER Loop

' timer0 int subroutine
Timer_0_overflow_int:
Rem timer0 overflow ( 65535 ) interrupt comes here
' increment the variable
Incr T0ic
Return
End

' end of program
' uses 1114 bytes of program memory
```

SOPHISTICATED BUT ECONOMICAL SCHOOL TIMER

■ U.B. MUJUMDAR

The basic requirements of a realtime programmable timer generally used in schools and colleges for sounding the bell on time are:

- Precise time base for time keeping.
- Read/write memory for storing the bell timings.
- LCD or LED display for displaying real time as well as other data to make the instrument user-friendly.
- Keys for data entry.
- Electromechanical relay to operate the bell.

We are describing here a sophisticated, yet economical, school timer based on Motorola's 20-pin MC68HC705J1A microcontroller.

Description

The pin assignments and main features of the microcontroller are shown in Fig. 1 and the Box, respectively. The complete system is divided into four sections, namely, the time keeping section, the input section (keyboard), the output (display, indicators, and relay driving) section, and power supply and battery backup.

The time-keeping section. Accurate time-keeping depends on the accuracy of time base used for driving the microcontroller. In this project, the microcontroller is driven by

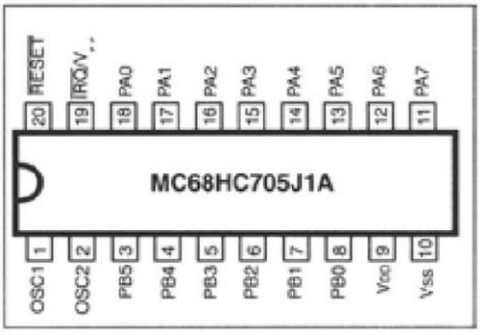


Fig. 1: MC68HC705J1A pin assignment

Main features of MC68H705J1A

- 14 bidirectional input/output (I/O) lines.
  (All the bi-directional port pins are programmable as inputs or outputs.)
- 10 mA sink capability on four I/O pins (PA0-PA3).
- 1,240 bytes of OTPROM, including eight bytes for user vectors.
- 64 bytes of user RAM.
- Memory-mapped I/O registers.
- Fully static operation with no minimum clock speed.
- Power-saving stop, halt, wait, and data-retention modes.
- Illegal address reset.
- A wide supply voltage range from-0.3 to 7 volts.
- Up to 4.0 MHz internal operating frequency at 5 volts.
- 15-stage multifunction timer, consisting of an 8-bit timer with 7-bit pre-scaler.
- On-chip oscillator connections for crystal, ceramic resonator, and external clock.

PARTS LIST

Semiconductors:

IC1	- 68HC705JIACP Microcontroller
IC2	- CD4532 8-bit priority Encoder
IC3	- 74LS138 3-line to 8-line decoder
IC4	- 74LS47 BCD-to-7-segment decoder/driver
T1-T3	- BC547/BC147 npn transistor
T4-T7	- 2N2907 pnp transistor
D1-D7	- 1N4007 diode
ZD1	- 5.6V, 0.5 watt zener

Resistors (1/4-watt, ±5% carbon, unless stated otherwise)

R1	- 210-ohm, 0.5 watt
R2	- 27-ohm
R3, R12-R14, R24-R-27	- 1-kilo-ohm
R4-R8	- 100-kilo-ohm
R9-R11, R23, R29	- 10-kilo-ohm
R15-R22	- 47–ohm
R28	- 10-mega-ohm

Capacitors:

C1	- 350µF, 25V electrolytic
C2, C3	- 1µF, 16V electrolytic
C4, C5	- 27µF ceramic disk
C6	- 0.1µF ceramic disk

Miscellaneous:

S1-S5	- Push-to-on switch (key)
S6	- On/off switch
PZ1	- Piezo buzzer
RL1	- Relay 12V, 300-ohm, 1C/O
XTAL	- 3.2768MHz crystal
X1	- 230V AC primary to 12V-0-12V, 500mA secondary transforer
DIS.1-DIS.4	- LTS542 common-anode display
	- 4 x 1.2V Ni-Cd cells

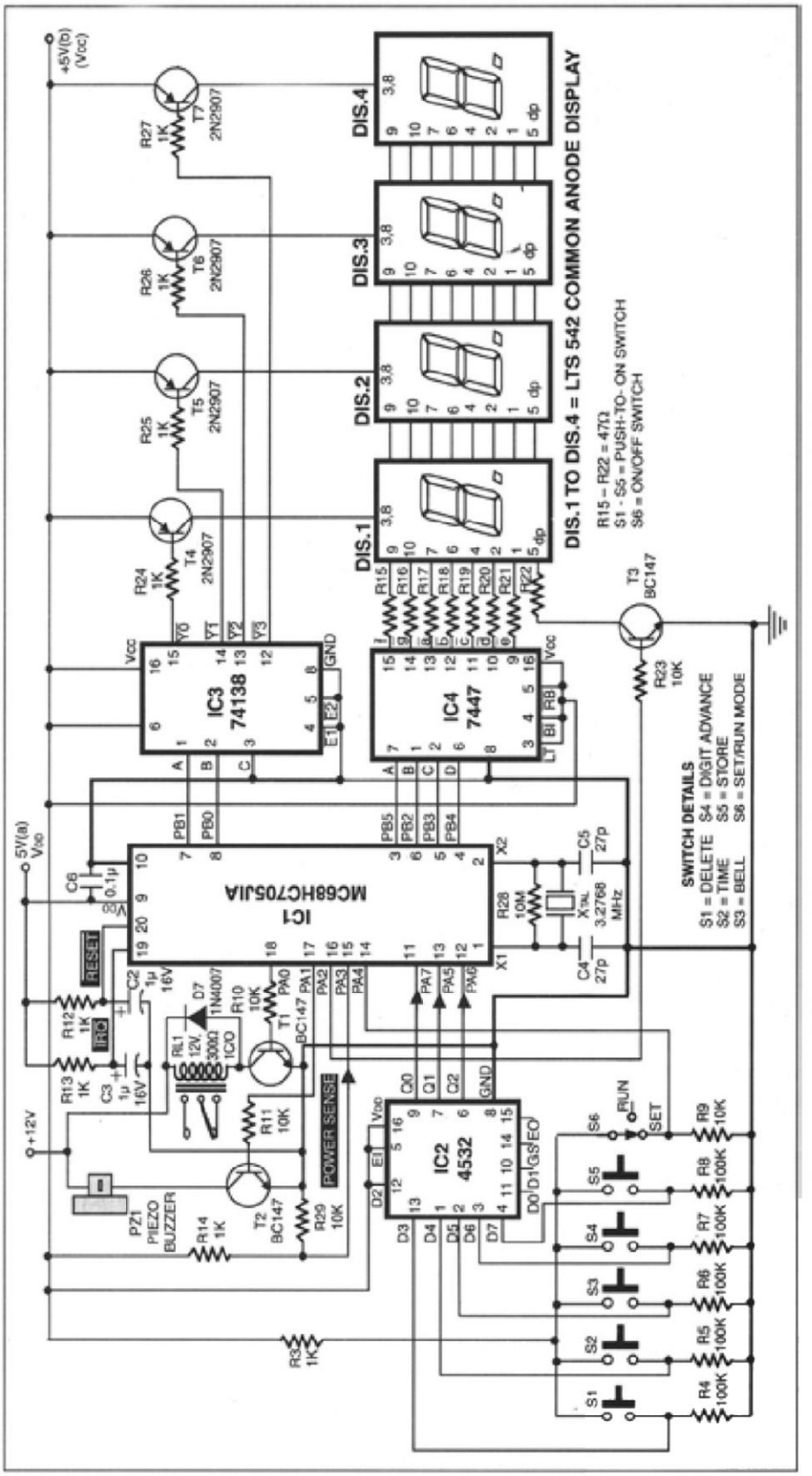


Fig. 2: Schematic diagram of the microcontroller-based school timer

AT-cut parallel resonant crystal oscillator that is expected to provide a very stable clock. A 3.2768MHz crystal provides a time base to the controller. The frequency (fosc) of the oscillator is internally divided by 2 to get the operating frequency (fop). This high-frequency clock source is used to control the sequencing of CPU instructions.

Timer. The basic function of a timer is the measurement or generation of time-dependant events. Timers usually measure time relative to the internal clock of the microcontroller. The MC68HC705J1A has a 15-stage ripple counter preceeded by a pre-scaler that divides the internal clock signal by 4. This provides the timing references for timer functions.

The programmable timer status and control register (TSCR) is used for deciding the interrupt rate. It can be programmed to give interrupts after every 16,384, 3,2768, 65,536, or 131,072 clock cycles. In Table I, the control word is set to provide the interrupts after every 16,384 cycles. For a 32,768MHz crystal, the interrupt period will be 10ms. Thus, timer interrupts will be generated after every 10 ms (100 Hz). That is, 100 interrupts will make 1 second.

Now time-keeping becomes very simple. As we are having a precise 1-second time count, a real-time clock can be easily built.

The MC68HC705J1A has a 64 byte RAM that is used for data storage, Real time (in terms of seconds, minutes, hours, days of a month, and months) is stored in this RAM. Thus an accurate real-time clock is generated.

The input section. For setting the real-time clock and storing operating times, the timer requires to be programmed externally. Data is fed using the keyboard.

Press-to-on type keys are interfaced to the microcontroller using an 8-bit priority encoder CD 4532. This encoder detects the key-press operation and generates the equivalent 3-bit binary data. Its truth table is shown in Table II. The priority encoder is interfaced to port A of the microcontroller.

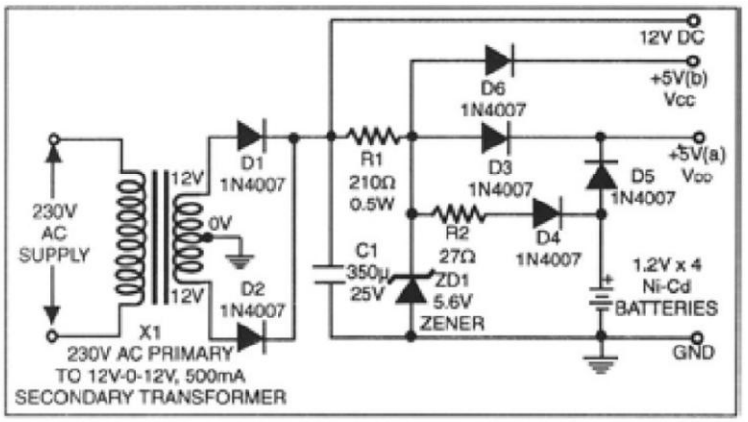


Fig. 3: Power supply circuit for the school timer

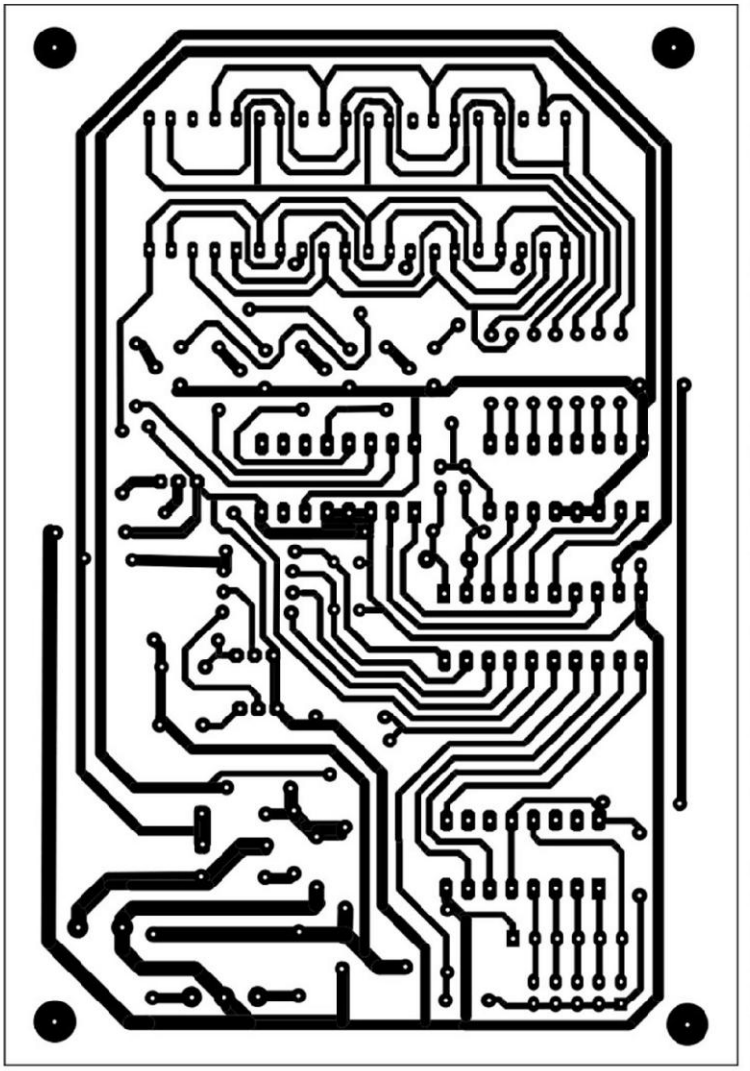


Fig. 4: Actual-size single-sided PCB for the circuits in Figs 1 and 2

Various keys used in the timer, along with their functions, are described below:

Time (4): For setting real time in minutes and hours.

Bell (5): For setting the bell's operating timings.

Digit Advance (6): Data setting is done digitwise (hour's digit followed by minute's digit). The Digit Advance key shifts the decimal point to the right.

Store (7): For storing the data (real time or bell time).

Delete (3): For deleting a particular bell timing.

Here, the figures within parentheses indicate the decimal equivalents of 3-bit binary data from the keyboard.

Set and run modes. Data setting is possible only in set mode. Set mode or run mode can be selected by toggle switch S6. By using a lock switch for S6, the timer can be protected from unauthorized data entry/storage.

In run mode if you press 'Bell' key once, the display shows the bell's various operating timings one after the other, in the same order in which these had been previously stored. In case you want to discontinue seeing all the bell timings, you may press 'Time' key at any stage to revert back to the display of real time.

The output section. Seven-segment displays are used for data display. As LEDs are brighter, these have been used in the system. There are two techniques for driving the displays: (i) driving each display using a separate driver (like 74LS47 or CD4511) and (ii) using multiplexed displays.

The first technique works well, but practically it has two problems: it uses a large number of IC packages and consumes a fairly large amount of current. By using multiplexed display both the problems can be solved. In multiplexing only one input is displayed at any given instant. But if you chop or alter inputs fast enough, your eyes see the result as a continuous display. With LEDs, only one digit is lighted

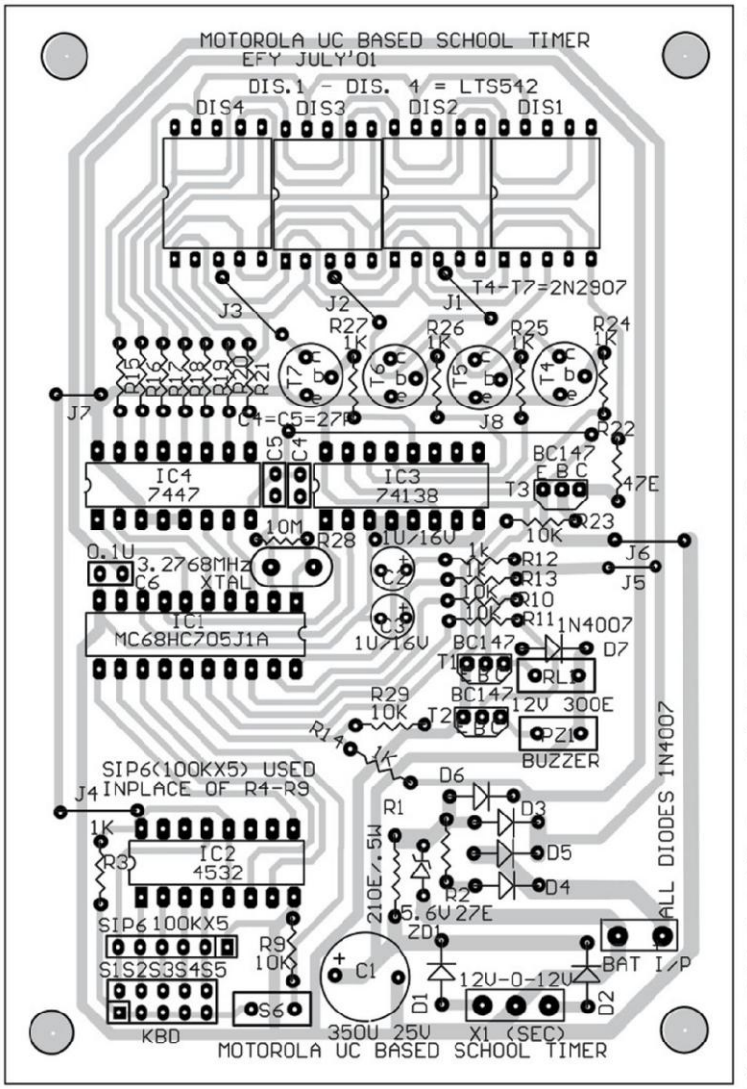


Fig. 5: Component layout for the PCB

up at a time. This saves a lot of power and also components, making the system economical.

Generally, displays are refreshed at a frequency of 50 to 150 Hz. Here, displays are refreshed at a frequency of 100 Hz (after every 10 ms). The display-refreshing program is an interrupt service routine program. BCD-to-7-segment decoder/driver 74LS47, along with transistor 2N2907, and 3-line-to-8-line decoder 74LS138 are used for driving common-anode displays.

In multiplexed display, the current through the segments is doubled to increase the display's brightness. 74LS47 is rated for sinking a current of up to 24 mA. As the current persists for a very small time in multiplexed display, it is peaky and can be as high as 40 mA per segment.

The decimal point is controlled individually by transistor BC547, as 74LS47 does not support the decimal point. PA0 and PA1 bits of port A are used for controlling the electro-mechanical relay and buzzer, respectively.

Power supply and battery backup. The microcontroller and the associated IC packages require a 5V DC supply, while the relay and the buzzer require 12V DC supply. A simple rectifier along with zener diode-regulated power supply is used. The microcontroller is fed through a battery-backed power supply, so that in the case of power supply, so that in the case of power failure the functioning of the controller's timer section is not affected. During power failure the timer is taken to 'low power' mode (called 'wait' mode). In this mode the controller draws a very small current. So small Ni-Cd batteries can provide a good backup.

A simple diode-resistance (27-ohm, ¼-watt) charger maintains the charge of the battery at proper charging rate.

Software

Freescale offers Integrated Development Environment (IDE) software for programming its microcontroller and complete development of the system. The development board comes with Editor, Assembler, and Programmer software to support Freescale's device programmer and software simulator. The ICS05 in-circuit simulator and non-real-time I/O emulator for simulating, programming, and debugging code for a MC68HC705J1A/KJ1 family device.

When you connect the pod to your host computer and target hardware, you can use the actual inputs and outputs of the target system during simulation of the code. You can also use the ISC05 software to edit and assemble the code in standalone mode, without input/output to/from pod. The pod (MC68HC705J1CS) can be interfaced to any Windows 3.x-or Windows 95-based IBM computer using serial port.

The software for the timer has been so developed that the system becomes as user-friendly as possible. The main constraint is read/write memory (RAM) space. As mentioned earlier, the microcontroller has only 64 byte

RAM. About twenty bell operating timings are required to the stored. So the efficient use of RAM becomes essential.

Basically, the following functions are performed by the software program:

1. Initialisation of ports and the timer.
2. Reading of key-pressed data.
3. Storing of real time and bell timings.
4. Comparison of real time and bell time. If the two match, the bell rings.
5. Display of data.
6. Time-keeping.

For a user-friendly system, the associated software is required to perform many data manipulation tricks and internal branching. The operation and logic can be understood from the Assembly language listings. The software is mainly divided into the following modules:

TABLE I
Timer Status and Control Register (TSCR)

Bit	7	6	5	4	3	2	1	0
Signal	TOF		RTIF	TOIF	RTIE	TOFR	RTIFR	RTI	RTO
Reset	0	0	0	0	0	0	1	1	

TOF: Timer overflow flag
RTIF: Real-time interrupt flag
RTIE: Real-time interrupt enable
RTI and RTO: Real-time interrupt select bit

RTI	RTO	Interrupt period
0	0	fop ÷ 2^{14}
0	1	fop ÷ 2^{15}
1	0	fop ÷ 2^{16}
1	1	fop ÷ 2^{17}

For 3.2768 MHz crystal
Frequency of operation (fop)
= 3.7268x106/2 = 1.638x106MHz
For RTI=RTO=0
Interrupt period = 10ms (100Hz)

TABLE II
Truth Table for Priority Encoder CD4532

Keys	E1	D7	D6	D5	D4	D3	D2	D1	D0	Q2	Q1	Q0
Store	1	1	X	X	X	X	X	X	1	1	1	1
Digit Adv.	1	0	1	X	X	X	X	X	1	1	0	0
Bell	1	0	0	1	X	X	X	X	1	0	1	1
Time	1	0	0	0	1	X	X	X	1	0	0	0
Delete	1	0	0	0	0	1	X	X	0	1	1	1

Keyboard. When a key is pressed, CD4532 sends the corresponding data. After reading the data, the controller decides on the action. 'Set/ Run' key (S6) is connected to port PA4.

Bell. This part of the program is used for displaying the bell operating timings stored in the RAM. The operating timings are displayed one by one with a delay of 5 seconds between tow consecutive timings.

Set. The real time and bell timings are stored using this part of the software. Data is entered digitwise; for example, 08:30 a.m. will be stored a 0, followed by 8, followed by 3, and finally 0. Data is stored in 24-hour format.

Data fed from the keyboard is converted into equivalent hex and stored in RAM. Any particular operating timing can be deleted from the memory using 'Delete' key, provided the timing is already stored in the memory.

Run. Here the real time is compared with bell operating time. If the two match, the relay is operated.

DataCon. This part of the software is used for finding out the decimal equivalent of hex data. The microcontroller manipulates the hex data and converts it into BCD format for display.

Timer. The timer of the microcontroller is initialized to give an interrupt after every 10 ms. A real-time clock is generated using the interrupt. Also the display is refreshed during the interrupt service routine.

For real-time systems battery backup is very essential, because power failure affects the time keeping. In interrupt service routine, the availability of power supply is checked. If the power is available, displays are refreshed and the timer operates normally. However, during the power-failure period, displays are off and system is taken to 'low power' mode. In this mode only the timer part of the microcontroller remains activated while operations of all other peripherals are suspended. This considerably reduces the power consumption. When the supply gets restored, the controller starts operating in normal fashion.

Operating procedure

When the power is switched on, the display shows 12.00. Two settings are required in the timer: (a) setting of real time and (b) setting of bell operating timings. For setting real-time clock 'Time' key is used.

Storing of real time. To store real time, say, 05:35 p.m., flip 'Run'/'Set' key (S6) to set mode. The display will

show '0.000'. Press 'Time' key. Further pressing of 'Time' key will increment the data, like 0.000, 1.000, 2.000, and thereafter it will repeat 0.000, etc. To select the digit, press 'Digit Advance'. This stores the present digit and the next digit is selected as indicated by the decimal pointer. Data is stored in 24-hour format. The time to be stored is 17.35, of which the first digit will be 1.000. The second, third, and fourth digits can be stored in similar fashion. After the fourthdigit, press 'Digit Advance' key once more. The display will show 1735 (with no decimal). Now press 'Store' to store the data.

Storing of bell timings. The procedure to store bell operating timings is similar to that of setting real time. The only difference is that here data is changed by 'Bell' key in place of 'Time' key. Any number of bell timings (<20) can be stored in the same fashion. If the number of bell operating timings exceeds 20, the timer will not accept any new bell timings until one of the previously stored timings is deleted.

Deletion of bell operating timings. For deleting a particular timing, first store this timing using the steps given above. Then press 'Delete' key to delete the specific data from the memory.

Display of real time. If 'Run'/'Set' key is taken to run mode, real time will be displayed.

Checking of bell operating times. For checking the bell operating times, press bell key in 'Run' mode only. The stored bell operating timings will be displayed one by one with a delay of 5 seconds between two consecutive timings.

Programming

There are two ways to program the EPROM/OTPROM (one-time programmable ROM):

1. Manipulate the control bits in the EPROM programming register to program the EPROM/OTPROM on a byte-by-byte basis.
2. Program the EPROM/OTPROM with Motorola's MC68HC705J in-circuit simulator.

The author has used the second method for programming the OTPROM.

An actual-size, single-sided PCB for the circuits in Figs 2 and 3 is shown in Fig. 4, with its component layout shown in Fig. 5.

Download source code: http://www.efymag.com/admin/issuepdf/Schooltimer.zip

RPM COUNTER USING MICROCONTROLLER AT89C4051

■ A.M. BHATT

Counting the revolutions per minute (RPM) of motors—determining the motor speed—is essential in the field of industrial automation. It is useful especially for closed-loop control systems where proper action can be taken in case the actual RPM deviates from the set RPM.

Here is a project based on microcontroller AT89C4051 that measures and shows on an LCD the RPM of a running motor. Using a proper transducer, first the rotations of the motor are converted into pulses. The generated pulses are counted by the microcontroller for a fixed time (say, one second). The count is multiplied by a factor to get the exact RPM and then displayed; if time is one second, the factor is 60.

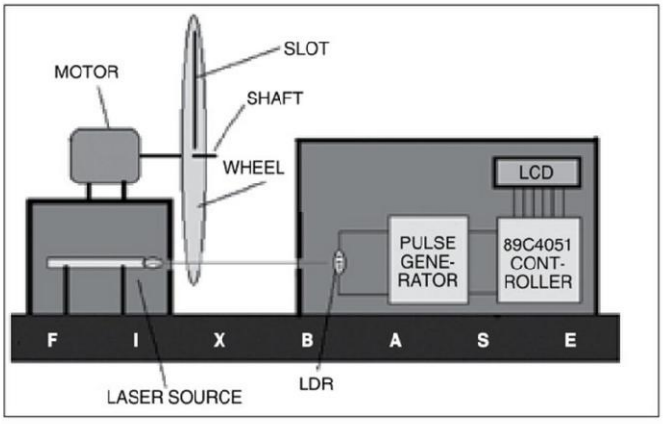


Fig. 1: Block diagram of the RPM counter based on microcontroller AT89C4051

Circuit description

Fig. 1 shows the block diagram of the RPM counter based on AT89C4051 microcontroller that generates pulses for every rotation of the motor, counts them and shows on the LCD.

On a fixed base, there is a laser source on one side and the combination of a light-dependent resistor (LDR), pulse generator, microcontroller and LCD on the other side. Both the arrangements are housed in separate wooden cabinets such that the laser beam falls directly on the LDR. The motor is placed on top of the laser source. A slotted wheel is attached to the motor shaft. The wheel is so big that it can interrupt the laser beam falling on the LDR.

As the motor rotates, the slotted wheel also rotates.

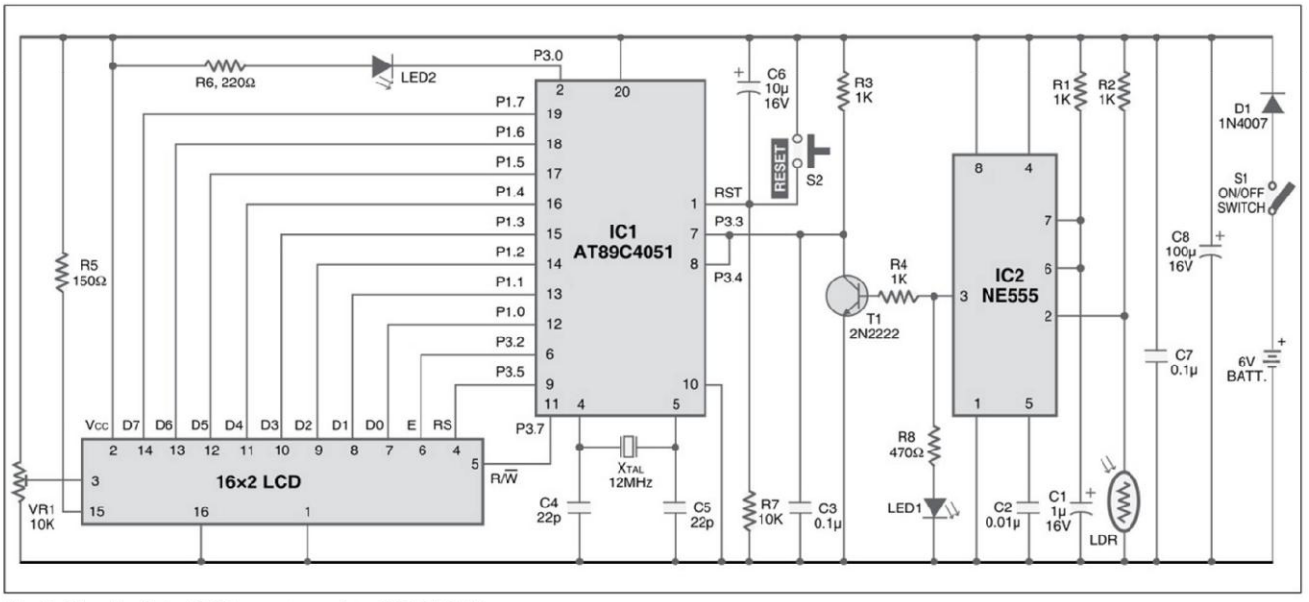


Fig. 2: Circuit of the RPM counter using AT89C4051

The laser beam falls on the LDR whenever the slot aligns with the laser beam and LDR, producing one pulse. Thus the rotations of the motor are converted into pulses that can be counted by the program in the microcontroller. Pulses are counted for one second. The pulse count is multiplied by 60 (because 1 RPM = 60 RPS) and finally shown on the LCD.

Fig. 2 shows the circuit of the RPM meter. It comprises microcontroller AT89C4051, timer NE555, LCD module (16×2 line) and a few discrete components. Timer NE555 is configured as a monostable multivibrator whose time period depends upon the combination of resistor R1 and capacitor C1. Trigger pin 2 of NE555 is pulled high via resistor R2. The LDR is connected along with resistor R2 to pin 2 of NE555 such that when the laser light falls on the LDR, pin 2 goes low to trigger NE555.

The output from pin 3 of NE555 is inverted by transistor T1 and fed to port pins P3.3 and P3.4 of the microcontroller. LED2 is connected to port pin P3.0 (pin 2) of the microcontroller. Data pins D0 through D7 of the LCD are connected to port pins P1.0 through P1.7 of the microcontroller, respectively. Control pins E, RS and R/$\overline{W}$ of the LCD are connected to port pins P3.2, P3.5 and P3.7, respectively. A 12MHz crystal connected between pins 4 and 5 of the microcontroller, along with two 22pF capacitors C4 and C5, generates the basic clock frequency. Power-on reset is derived with the combination of resistor R7 and capacitor C6. Switch S2 is used for manual reset.

An actual-size, single-side PCB for the RPM counter is shown in fig. 3 and its component layout in fig. 4.

PARTS LIST

Semiconductors:

IC1	- AT89C4051 microcontroller
IC2	- NE555 timer
T1	- 2N2222 npn transistor
D1	- 1N4007 rectifier diode
LED1, LED2	- 5mm LED
LCD	- 16×2 line

Resistors (all ¼-watt, ±5% carbon):

R1-R4	- 1-kilo-ohm
R5	- 150-ohm
R6	- 220-ohm
R7	- 10-kilo-ohm
R8	- 470-ohm
VR1	- 10-kilo-ohm preset

Capacitors:

C1	- 1µF, 16V electrolytic
C2	- 0.01µF ceramic disk
C3, C7	- 0.1µF ceramic disk
C4, C5	- 22pF ceramic disk
C6	- 10µF, 16V electrolytic
C8	- 100µF, 16V electrolytic

Miscellaneous:

S1	- On/off switch
S2	- Push-to-on switch
X_{TAL}	- 12MHz crystal
BATT.	- 6V battery
	- LDR

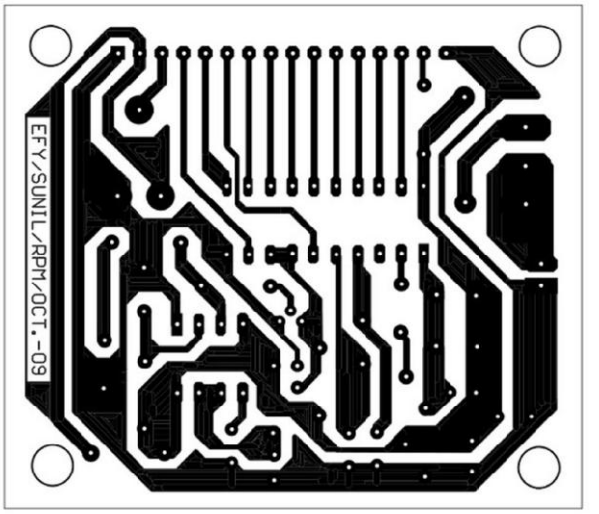


Fig. 3: Actual-size, single-side PCB for the RPM counter using microcontroller AT89C4051

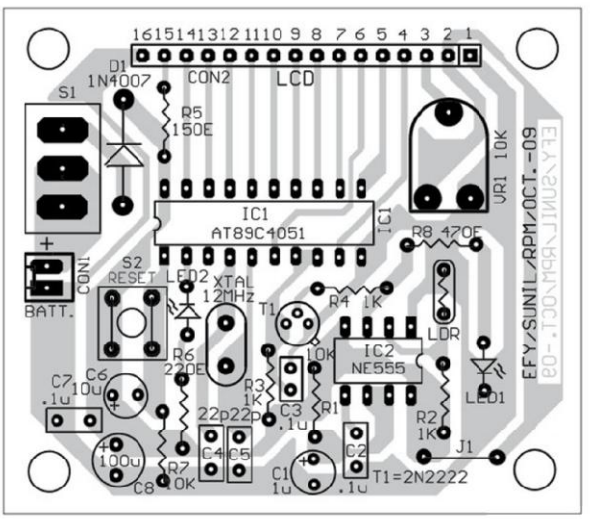


Fig. 4: Component layout for the PCB

Operation

1. As the motor starts rotating, the laser light falls on the LDR when the slot aligns with the laser beam and LDR

2. Every time the motor completes one rotation, the monostable (NE555) triggers to generate one pulse, which is indicated by LED1. So LED1 blinks at the rate of motor speed

3. As the first pulse arrives, it generates an interrupt for the microcontroller and immediately the microcontroller starts counting the pulses. This is indicated by LED2. The LCD shows the message "Counting RPM..."

4. The microcontroller counts the pulses for a period of one second. Thereafter, LED2 shows the message "Counting finished..." and goes off. The microcontroller stores the count and multiplies it by 60 to give

the final RPM count

5. The count is in hex format, so you have to convert it into decimal first. As the LCD accepts only ASCII values, the decimal values are converted into ASCII and shown on the LCD one by one

6. Now if you press RST switch, the process repeats

Software

The software for the RPM counter is written in 'C' language and compiled using Keil μVision3 compiler. Burn the generated '.hex' file into the microcontroller using a suitable programmer. The source program is well commented and easy to understand.

The various functions are detailed below:

1. Main function initialises the timer, LCD, ports, etc, displays the message "RPM counter," clears the timer content and enables the external interrupt. It remains in the loop till 'rpm flag' (flag) is not set. When the RPM counting completes, it displays the RPM value if it is within the range.

2. Int1 is an interrupt-based function that is called automatically when there is a negative edge on external interrupt 1 pin (P3.3). As soon as the first negative edge arrives, the counter starts counting the external pulses for one second. After one second, the content of TL0 is compared with A6h (166d). For any number greater than 166, multiplication with 60 will yield a number that is longer than four digits, which is out of the display range. This means RPM of max. 9960 (166x60) can be displayed. Otherwise, the LCD will show the message "RPM Out of Range".

3. Display function performs three tasks one by one:

(i) Converts hex number in TL0 into decimal and multiplies it by 60

(ii) Converts the final decimal number into equivalent ASCII characters one by one

(iii) Displays all the ASCII characters one by one on the LCD

4. writecmd function sends command byte to the LCD. It takes one argument byte and sends it to P1

5. writedata function sends the data byte to be displayed on the LCD. It also takes one argument byte and sends it to port P1

6. writestr function writes the entire string (message) on the LCD. It takes the pointer as an argument that points the address of the first character of the string. Then through the pointer, it sends all the characters one by one to P1

7. busy function checks the status of the busy flag of the LCD. If the flag is set, that means the LCD is not ready and the program remains within the loop. When the flag is reset, the LCD is ready and the program comes out of the loop

8. delay function generates a fixed delay of one second using timer 0. The basic delay is of 50 milliseconds, which is rotated in the loop for 20 times to generate total delay of 20×50 ms=1000 ms =1 second

Download Source Code: http://www.efymag.com/admin/issuepdf/RPM%20counter.zip

RPM.C

```
Here is the program code with necessary comments
#include<reg51.h>
#include<string.h>

sbit rs = P3^5;    // rs pin of LCD
sbit en = P3^2;    // en pin of LCD
sbit rw = P3^7;    // rw pin of LCD
sbit b = P1^7;     // busy flag
sbit led = P3^0;   // led indication
unsigned int flag=0; // rpm flag

void delay()
// 1 sec delay
  {
         int k;
         TL0 = 0xAF; // use timer 0
         TH0 = 0x3C; // to generate 50 ms delay
         TR0 = 1; // start timer
         for(k=0;k<20;k++)                    /   /
rotate loop for 20 times
           {
             while(TF0==0); // wait till timer over-
flow
             TF0 = 0; // reset the flag
             TL0 = 0xAF; // reload it
             TH0 = 0x3C;
           }
         TR0 = 0; // stop timer
  }
void writecmd(unsigned char a)               /   /
send command to LCD
  {
```

```
        busy();  // check busy flag
        rs = 0;  // select command register
        rw = 0;  // write enable
        P1 = a;  // send byte to LCD
        en = 1;  // apply strobe pulse
        en = 0;
  }
void writedata(unsigned char b)                // send
data to LCD
  {
        busy();  // check busy flag
        rs = 1;  // select data register
        rw = 0;  // write enable
        P1 = b;  // write enable
        en = 1;  // send byte to LCD
        en = 0;  // apply strobe pulse

  }
void busy()
// check busy flag of LCD
  {
        en = 0;  // disable display
        P1 = 0xFF; // P0 as input
        rs = 0;  // select command register
        rw = 1;  // read enable
        while(b==1)
// if busy bit is 1
        {
         en=0; // remain within loop
         en=1;
        }
        en=0;

  }
void writestr(unsigned char *s)                // send
string message to LCD
 {
        unsigned char l,i;
        l = strlen(s);
// get length of string
        for(i=0;i<l;i++)

        {
                writedata(*s);
// till the length of string
                s++;
// send characters one by one
        }
 }
void int1(void) interrupt 2          // external in-
terrupt 1 function
{
        EA=0;                                  // first
disable interrupts
        led=0;                                   //
give indication
        writecmd(0x01);
        writestr("counting RPM....");   // display
message
        TR0=1;   // start timer 0
        delay(); // give 1 sec delay
        TR0=0;   // stop timer
        writecmd(0x80);
        writestr("counting finish ");    // display
message
        led=1;
        if(TL0>0xA6)
          {                                    // if
value more then 166
                writecmd(0xC0);
                writestr("RPM  out  of  range");//
send message
          }
        else flag=1;                           // if
not then set the flag
 }

 void display() // convert hex to desimal and
  {                                              //
decimal to ascii
  unsigned int tmp1,tmp2,t,t1,i,j;
        unsigned char asci[4];
        tmp1 = (TL0 & 0x0F);
 // get lower nibble of TL0
        tmp2 = TL0 >> 4;
 // get upper nibble of TL0
        tmp2 = tmp2*16;
 // multiply upper nibble with 16
        t = tmp1+tmp2;                         //
get decimal number
        t=t*60;                                //
multiply it with 60
        i=3;
        if(t>=1000)
 // if more then 4 digits
          {
                while(t>10)
                  {
                        t1=t%10;
                        asci[i]=t1+0x30;      //
convert them one by one
                        t=t/10;
 // into ASCII
                        i--;
                  }
                asci[0]=t+0x30;
          }
        else
 // otherwise convert
          {
                while(t>10)
                  {
                        t1=t%10;
                        asci[i]=t1+0x30;      //
last three digits
                        t=t/10;
                        i--;
                  }
                asci[1]=t+0x30;
                asci[0]=0x30;
   // and put first digit as 0
          }
        writecmd(0xC0);
        writestr("currentRPM:");        //  display
```

```
current RPM and
         for(j=0;j<4;j++)
                  writedata(asci[j]);            //
all four digits one by one
  }
  void main()
    {
                  TMOD=0x15;             // timer 0
in 16 bit counter and timer 1 in 16 bit counter
                  P1 = 0x00;
  // P1 as output port
                  rs=0;
  // clear all LCD control pins
                  en=0;
                  rw=0;
                  writecmd(0x3C);                //
initialize LCD
                  writecmd(0x0E);
                  writecmd(0x82);
                  writestr("RPM Counter");  // ini-
tially display message
                  TH0=0x00;            // clear T0
                  TL0=0x00;
                  IE=0x84;                       //
enable external interrupt 1
                  while(flag==0);                //
remain within loop till rpm flag is clear
                  display();                     //
when flag is set display current RPM value
                  while(1);
  // continuous loop
         }
```

SPEEDOMETER-CUM-ODOMETER FOR MOTORBIKE

■ **ARUN KUMAR VADLA**

Normally, digital speedometers are found only in luxury cars and high-end motorbikes. Even if your motorbike has a mechanical speedometer, what will you do when it gets damaged? First, you need to replace the mechanical worm gear and then the cable.

Anyway, we describe here how to build a digital speedometer-cum-odometer for your motorbike. The circuit uses a microcontroller, an LCD display and some commonly available components. It is a better alternative to the mechanical speedometer and even a beginner with minimal skill level can assemble it.

The features of the digital speedo-meter-cum-odometer are:

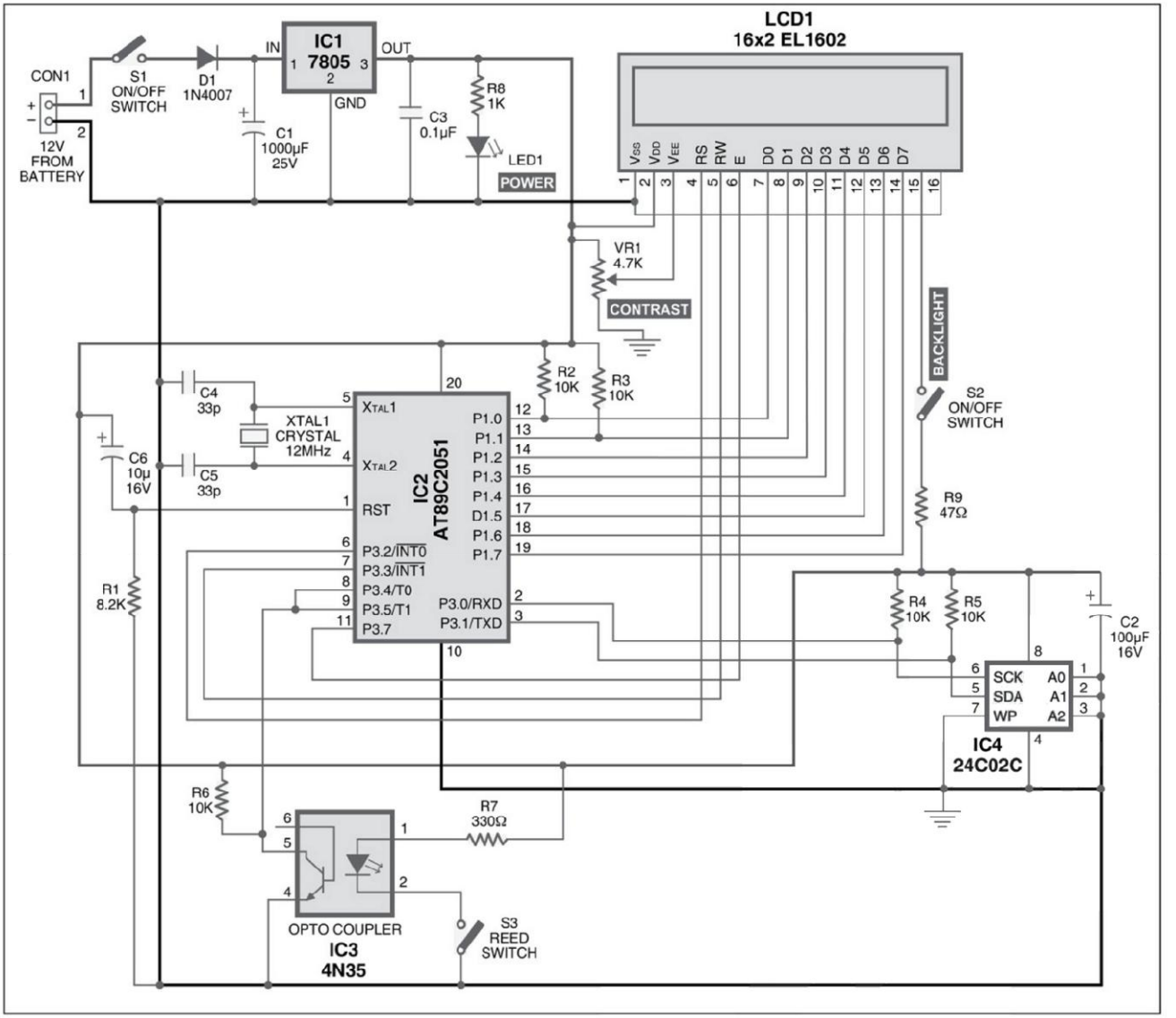


Fig. 1: Circuit of microcontroller-based speedometer-cum-odometer

1. Digital readout
2. Speed displayed in km/hour
3. Distance traveled displayed in kilometres
4. Readings saved in non-volatile memory (EEPROM)
5. Reliability due to use of the microcontroller
6. No mechanical wear and tear
7. Home-brewed speed transducer/sensor
8. Self reset to zero after completion of 99,999.9 km
9. Easy to build and fix onto the bike

Calculations

You first need to know the radius of the bike's front wheel. The calculations here are based on Hero Honda's Splendor model. The radius of the front wheel is 30 cm. (This can vary with the brand or model.)

Circumference of the wheel= $2\pi r$ (where 'r' is in cm)
= 2×3.14×30
= 188.4 cm or 1.884 metres

Speed. Let's assume that in 1 second the wheel completes one revolution. In other words, in one second, the bike has covered 1.88 metres. Therefore the speed in km/hour:

N×1.88×3600/1000
= N×6.784 or N×6.8

where 'N' is the number of revolutions per second. '6.8' is a constant and only 'N' varies; for example, if 'N' is 5, the speed equals 5x6.8= 34 km/hour.

Distance. The odometer is updated every 100 metres. To cover 100 metres, the wheel is required to make approximately 53 revolutions (100/1.88). The microcontroller takes care of the tasks of revolutions counting, speed calculation, conversion and display of results.

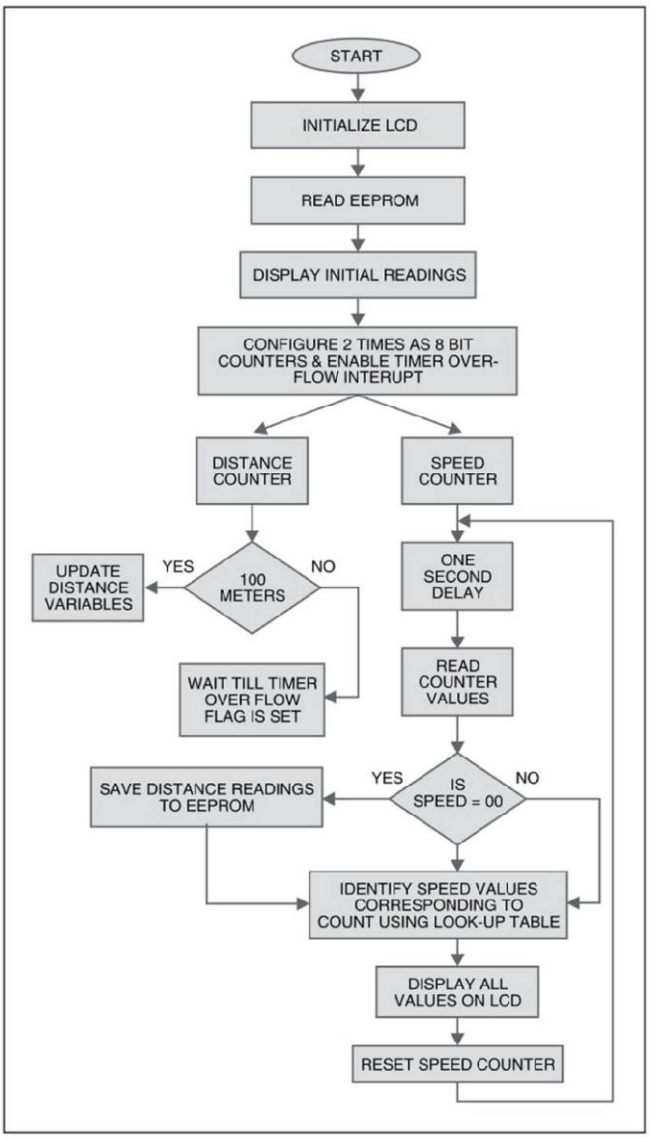


Fig. 2: Flow-chart of the program

Circuit description

The circuit of the microcontroller-based digital speedometer-cum-odometer is shown in Fig. 1. The functions of various components used in the circuit are described below.

Microcontroller. A 20-pin AT89C2051 microcontroller from Atmel is used here because of its low pin count, affordability and compatibility with CISC-based 8051 family. All the available pins of the microcontroller are utilised in the project. This microcontroller features 2 kB of Flash, 128 bytes of RAM, 15 input/output (I/O) lines, two 16-bit timers/counters, a five-vector two-level interrupt architecture, a full-duplex serial port, a precision analogue comparator, on-chip oscillator and clock circuitry.

LCD module. To display the speed and distance

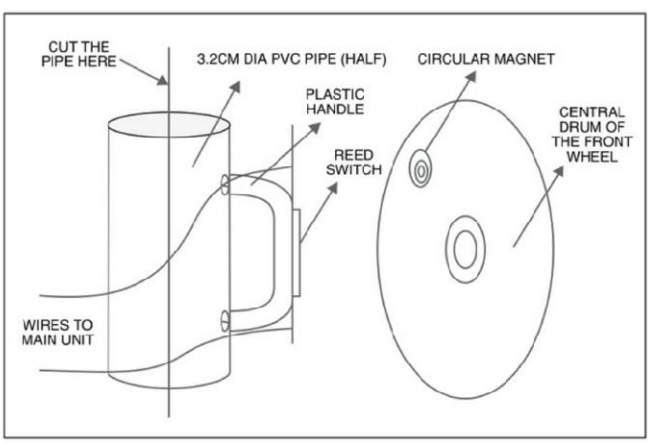


Fig. 3: Arrangement of reed switch and magnet on the front wheel of motor bike

PARTS LIST

Semiconductors:

IC1	- 7805 5V regulator
IC2	- AT89C2051 microcontroller
IC3	- 4N35 optocoupler
IC4	- 24C02 EEPROM
D1	- 1N4007 rectifier diode
LED1	- 5mm light-emitting diode

Resistors (all ¼-watt, ±5% carbon):

R1	- 8.2-kilo-ohm
R2-R6	- 10-kilo-ohm
R7	- 330-ohm
R8	- 1-kilo-ohm
R9	- 47-ohm
VR1	- 4.7-kilo-ohm preset

Capacitors:

C1	- 1000µF, 25V electrolytic
C2	- 100µF, 16V electrolytic
C3	- 0.1µF ceramic
C4, C5	- 33pF ceramic
C6	- 10µF, 16V electrolytic

Miscellaneous:

CON1	- 2-pin SIP male connector
S1, S2	- SPST 'on'/'off' switch
S3	- Reed switch
LCD1	- 16x2 EL1602 LCD module or equivalent
X_{TAL1}	- 12MHz crystal

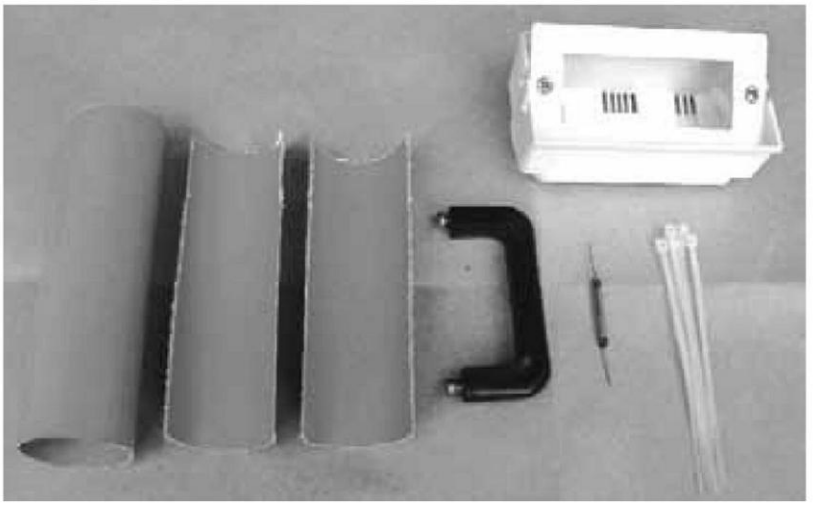


Fig. 4: The materials required to build a PVC contraption

Fig. 5: Reed switch and magnet fixed on the front wheel of motor bike

traveled, we have used a 16×2 alphanumeric LCD based on HD44780 controller. The backlight feature of the LCD allows data to be visible even at night. The pin configuration and features of this LCD have earlier been published in several issues of EFY.

Serial EEPROM. The readings of the distance traveled are saved in an external serial EEPROM. Here, a 24C02 serial EEPROM based on Philips I2C protocol is used.

I²C bus protocol. The I²C bus consists of two active wires and a ground connection. The active wires, serial data line (SDA) and serial clock line (SCL) are bidirectional.

Every device hooked up to the bus has its own unique address, no matter whether it is an MCU, LCD driver, memory or ASIC. Each of these chips can act as a receiver and/or transmitter, depending on the functionality. Obviously, an LCD driver is only a receiver, while a memory or I/O chip can be both transmitter and receiver.

The I²C bus is a multi-master bus. This means that more than one IC capable of initiating a data transfer can be connected to it. The I²C protocol specification states that the IC that initiates a data transfer on the bus is considered the bus master. Bus masters are generally microcontrollers. Consequently, all the other ICs are regarded as bus slaves at that instant.

Let's assume that the MCU wants to send data to one of its slaves. First, the MCU will issue a START condition. This acts as an 'attention' signal to all of the connected devices. All ICs on the bus will listen to the bus for incoming data. Then the MCU sends the address of the device it wants to access, along with an indication whether the access is a 'read' or 'write' operation. Having received the address, all ICs will compare it with their own ad-

Fig. 6: Speedometer-cum-odometer unit on a bike's handle bar

dress. If it doesn't match, they simply wait until the bus is released by the stop condition. If the address matches, the chip will produce a response called 'acknowledge' signal. We have used write operation in this project.

Once the MCU receives the acknowledge signal, it can start transmitting or receiving data. In our case, the MCU will transmit data. When all is done, the MCU will issue the stop condition. This signals that the bus has been released and that the connected ICs may expect another transmission to start any moment.

We have several states on the bus: start, address, acknowledge, data and stop. These are all unique conditions on the bus. In our project, the microcontroller is the master and the serial EEPROM is the slave.

The readings are periodically stored in the EEPROM and the previous reading is retrieved from the EEPROM each time the bike is started.

Speed sensor. For this project, we make use of a simple home-made speed transducer. The rotation of the wheel is sensed by the combined action of a reed switch and a magnet fixed on the wheel. The sensor sends a pulse to the microcontroller each time a revolution is made.

Optocoupler. An optocoupler is used to counter the effects of bouncing when the contact of reed switch is closed.

Power supply. The power supply for various parts of the circuit is drawn from the vehicle's 12V battery after reducing it to 5V using a three-terminal voltage.

Software

The 'Init_EEPROM' and 'Speedo' source codes of this project are written in Assembly language. These are compiled using an open-source ASEM-51 assembler to generate the Init_EEPROM.hex and Speedo.hex files. The hex files are burnt into the microcontroller chip.

Two internal timers of the microcontroller are configured as 8-bit counters to count the number of pulses generated by the speed sensor. One timer is used to measure the distance and the other for speed calculation.

A software delay of one second is generated after the speed counter is triggered. The speed count value is obtained from the counter registers. To speed up the process, a look-up data table is stored in the ROM that helps the microcontroller to convert the number of pulses into the corresponding speed values. The program flow-chart is shown in Fig. 2.

The 'distance' counter is incremented every 100 metres. The wheel has to make 53 revolutions to achieve this. The distance counter is loaded with an initial value of 203 (255-53+1) and is incremented on each revolution. After 53 counts, the timer overflows and generates an interrupt to notify the microcontroller that 100 metres are covered.

In the interrupt service routine, the microcontroller updates the corresponding 'DS1' distance variable. Instead of saving distance variables after each cycle, the microcontroller saves these readings when the vehicle is at

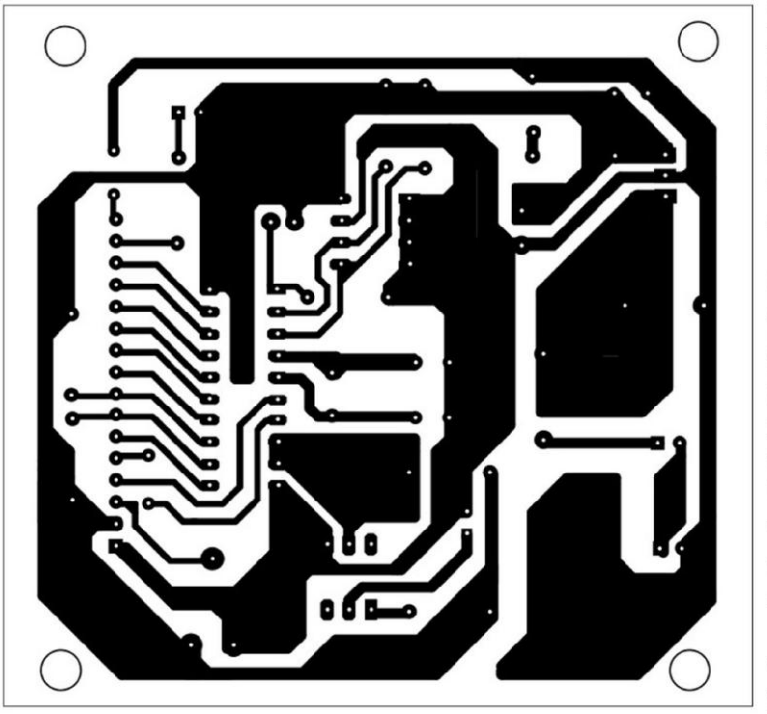


Fig. 7: Actual-size, single-side PCB for the microcontroller-based digital speedometer-cum-odometer

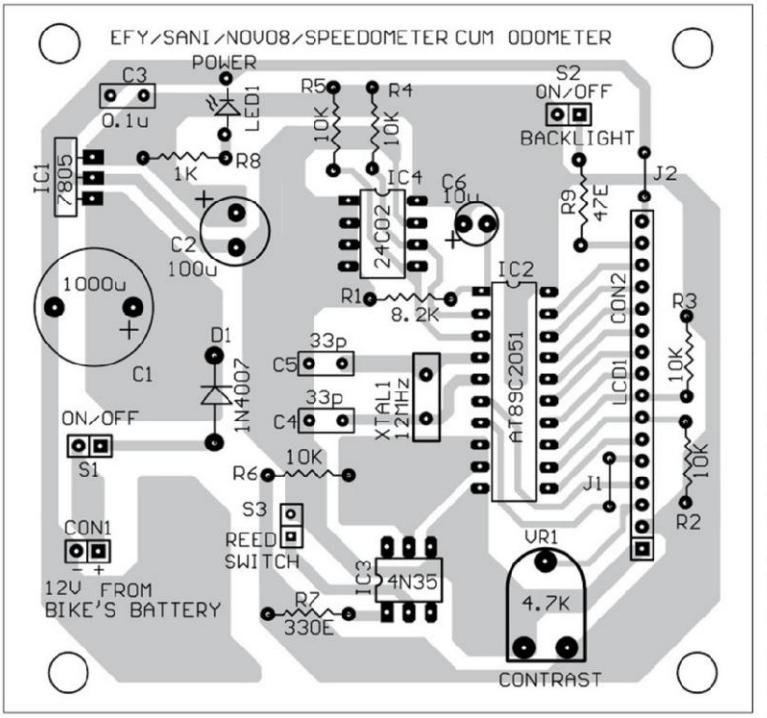


Fig. 8: Component layout for the PCB

halt (speed is 00.0 km/hour). In other words, when the vehicle is stopped at traffic signals or before the ignition key is turned off, the last reading is saved to the EEPROM. The same reading is again retrieved from the EEPROM when the bike is turned on next time and the readings are updated for each trip.

Construction

The reed switch and a magnet need to be fixed on the front wheel of the motor bike (Hero Honda's Splendor). A small circular magnet (about 2 cm in diametre), normally used in speakers of small toys, can be used. Fix the magnet to the central drum of the wheel just below the spokes connected to the drum. Secure the magnet using hot glue or Araldite.

For fixing the reed switch, a PVC pipe contraption needs to be made so that the magnet and reed switch are aligned as shown in Fig. 3. The materials required to build the contraption are shown in Fig. 4. Cut a 3.2cm diameter PVC pipe measuring 15.2 cm in length perpendicularly into two halves. Use only one half of the PVC pipe. Mount and secure the reed switch using Araldite and cable ties on the plastic handle (normally used in emergency lights). Once it dries up, solder two wires to the two opposite end leads of the reed switch. Fix the plastic handle on the half cut PVC pipe using screws. Now, place the pipe on the front shock-absorber fork such that reed switch faces towards the magnet.

Connect a multimeter, set in continuity mode, to the two wires coming from the reed switch. Rotate the wheel slowly and see whether the reed switch closes when the magnet passes across it. If it does, the multimeter will give a continuity beep. When the magnet moves away from the reed switch, the beep will stop, indicating that the reed switch is open. Make a few trials to find the optimal position for mounting and fixing the PVC pipe such that the reed switch works smoothly. Mark the location on the front shock-absorber fork.

Now you can fix the PVC pipe contraption to the shock-absorber fork using hot glue as shown in Fig. 5. Use liberal amount of hot glue to secure it to the pipe. Carefully route the two wires up to the bike's handle bar using cable ties to secure the wire. This completes the sensor mounting part.

The main circuit and the LCD module can be housed in suitable plastic enclosures, which are readily available in electronic projects shops. These enclosures should have precut slot for easy mounting of the LCD panel. If such boxes are not available, you can use the plastic boxes of electronic chokes by suitably removing some portions for the LCD panel.

Power supply can be taken either directly from the bike's 12V battery or tapped from the console which houses horn, headlight and indicator light switches. For this, you need to remove the switch console and identify positive wire and ground wire using a multimeter. When carrying out this step, remember to turn the ignition key to 'on' position. Solder a 60cm two-core wire to the positive and negative terminals inside the switch console. The advantage of taking supply from the switch console is that the ignition key controls the power supply to the main unit without having a separate on-off switch.

An actual-size, single-side PCB layout of the microcontroller-based speedometer-cum-odometer is shown in Fig. 7 and its component layout in Fig. 8.

Testing

After all the components are soldered on the PCB, program the microcontroller with Init_EEPROM.hex file and place the microcontroller in a 20-pin IC base and switch on the circuit.

In the first line of the LCD, 'INIT_EEPROM' appears. After five seconds, '00000.0' is displayed in the second line. This process erases any previous data and sets the initial readings in the EEPROM to zero. Now switch off the supply and program the microcontroller with 'speedo.hex' main file. After programming, place the microcontroller back in the circuit and switch on the supply. The LCD will show 'Kms: 00000.0' in the first line and 'Speed-Kms/Hr: 00.0' in the second line. Now, the unit is ready to mount on your bike.

Connect the two wires coming from the reed switch and the power supply wires to the main unit. Mount the unit at the centre of the bike's handle bar on top of the 'U' clamps that secure the handle bar to the chassis. You can use cable ties to accomplish this. Mounting arrangement of the unit is shown in Fig. 6.

Now start the bike, take a test ride and if connections are correct, the speed and the distance will be displayed on the LCD. A protective cover like polythene can be used for the main unit on rainy days.

Download source code: http://www.efymag.com/admin/issuepdf/SPEEDO%20CUM%20ODOMETER.zip

TRAFFIC LIGHT COUNT-DOWN TIMER WITH DUAL-COLOUR DISPLAY

■ VINAY CHADDHA

Normally, count-down timers of traffic lights display the count-down time only in red colour. The drawback here is that the drivers may get confused when the signal light is green but the timer display is in red.

Here we describe a dual-colour traffic count-down timer that displays count-down time in either red or green colour matching with the signal light. It is a 3-digit timer based on AVR Atmega328 microcontroller. The timer board can retrofit into existing signal lights.

Other key features of this timer are constant display intensity despite voltage variation, low power consumption (which makes it suitable for battery-powered systems), and an optional feature for intensity variation with time of the day (high intensity in daytime, medium intensity during dusk and dawn, and low intensity at night).

How does count-down timer works?

All retrofit timers used in traffic light systems work by learning the traffic light 'on' and 'off' time first. Learning

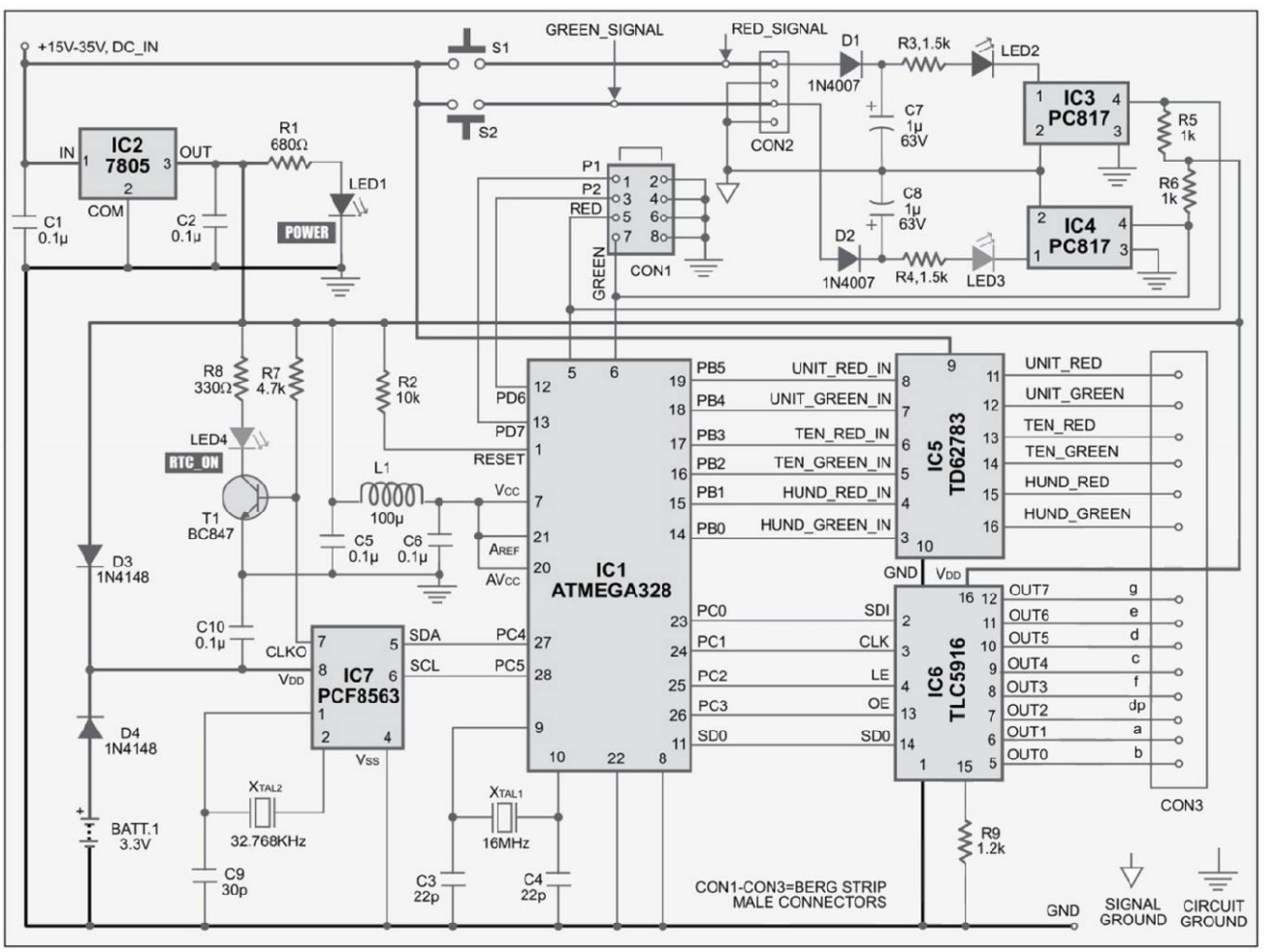


Fig. 1: Circuit of Avr-based traffic light count-down timer

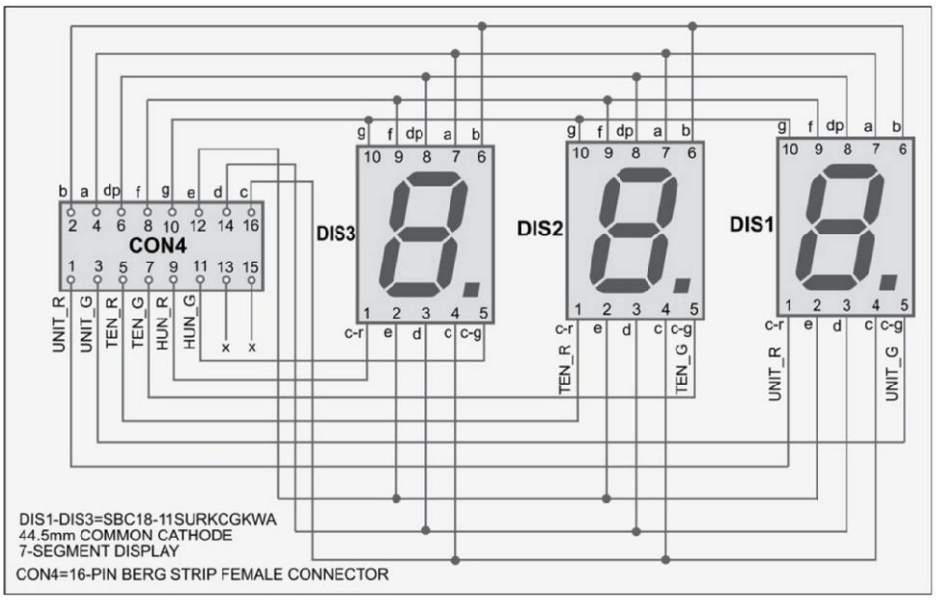


Fig. 2: Display section

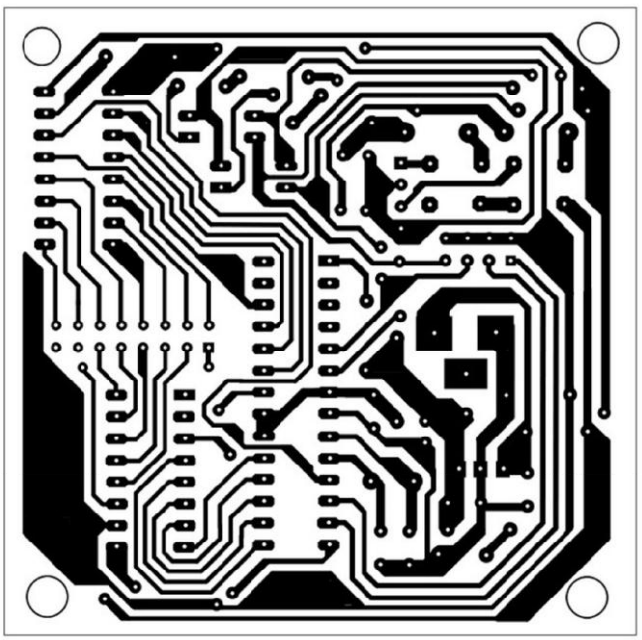


Fig. 3: An actual-size, single-side PCB for the AVR-based traffic light count-down timer

process of the count-down timer is explained below. At power-'on', the software program starts looking for the red signal. As soon as the red signal is 'on', it starts counting the time in seconds until the signal goes off. At that moment, the count in seconds is saved as 'learn time.'

The software program again keeps waiting for the red signal. As soon as the red signal is sensed second time, it starts counting time again until the red light goes off. At that moment, the count in seconds is saved as 'verify time.' Now if 'learn time' and 'verify time' are same, this is treated as 'on time.'

Next time, as soon as the red signal is sensed, the software program knows that the red signal will remain 'on' for 'on time' based on the experience of past two readings. The count-down starts during this 'on' time.

The same process is applicable to the green traffic signal as well.

The working of the count-down timer is detailed in the next section.

Circuit description

Fig. 1 shows the circuit of the AVR-based traffic light count-down timer with dual-colour display. It consists of the microcontroller, voltage regulator, optocoupler (PC817), red and green light input section, real-time clock and display driver section.

Microcontroller section. This section is built around ATMEL ATmega328 microcontroller, which is a 28-pin DIP IC. The microcontroller IC can be programmed with embedded software using a standard programmer. A 16MHz oscillator is used to supply a timing reference. Out of 28 pins, only 20 are available for input/output (I/O).

We have used following I/O lines for different purposes:

1. Grids—six lines
2. Red/green input—two lines
3. Interface to the segment driver—five lines
4. I^2C interface—two lines

PD7 is connected to ground through a shorting jumper at CON1.

Voltage regulator section. This is a standard circuit which converts the 15V-35V DC input into 5V. Minimum and maximum DC inputs to the circuit are 15V and 35V, respectively. The regulated 5V DC provided by IC

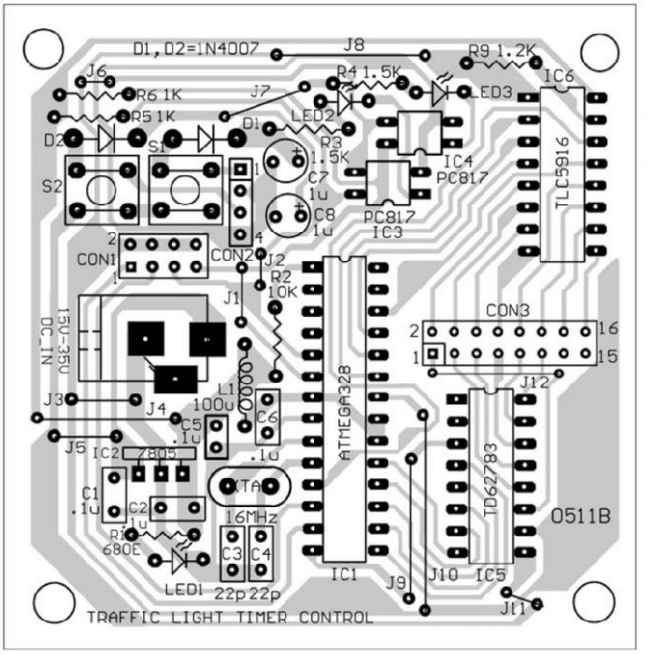


Fig. 4: Component layout for the PCB in Fig. 3

7805 is used to operate the microcontroller and other digital ICs.

Optocoupler section. Two PC817 optocouplers are used to isolate the input and output circuits. Red and green signal inputs are external signals coming from the traffic light control circuitry.

Red and green light inputs section. The input signals for the red and green lights varying from 15V to 30V are fed to the timer control circuit through diode-optocoupler combinations D1-IC3 and D2-IC4, respectively. When the red or green traffic light is 'on', the optocoupler conducts and the corresponding pin of the MCU goes low. When the red or green light is 'off', the corresponding pin of the MCU is high.

RTC section. IC PCF8563 used in the RTC section is an I^2C interface that uses SCL and SDA pins. Both these pins are available on the microcontroller too. RTC needs a Li-ion battery backup so that time is maintained even after power failure. The battery may need replacement once in three years.

CLKOUT signal of IC PCF8563 drives BC847. As a result, LED4, connected to the collector of transistor BC847, flashes at a frequency of 1 Hz to indicate proper working of RTC IC PCF8563. Note that the RTC section was not used when testing this project at EFY Lab.

Display driver. We have used three dual-colour, common-anode, seven-segment LED displays. Each display has eight segment pins and two grid points (C-r and C-g). For three-digit display, we have used eight segments and six grid outputs. As the 15V and up to 100mA current required to drive the display cannot be supplied by the microcontroller, we have used a segment driver and a grid driver (TD62783). The six outputs of TD62783 are used to drive the six grids (three for red colour and three for green colour).

TLC5916 is used as the seven-segment display driver. It can drive up to eight segments. The unique feature of this IC is that it sinks constant current and is not dependent on the supply voltage. So in traffic signals, where the battery voltage varies, the intensity of the display still remains the same.

Software program

The program is written in 'C' language and compiled using Open Source ARDUINO 0022 IDE. It runs smoothly on Linux (Ubuntu), Windows XP as well as MAC OSX, but you may face problems on Fedora and Windows 7.

The source code can be divided into the following parts:

1. Hardware definitions. To make the programming easy, we have defined each pin by its actual function in the circuit along with arbitrary number. For example, the green light signal received at pin 6 of the MCU is named as GREEN_SENSE as follows:

```
#define GREEN_SENSE 4
```

2. ***Subroutines for display driver TLC5916.*** The segment data is transferred to TLC5916 between subroutines like Active_output_5916, Inactive_output_5916, Send_byte_5916, Switch2special_5916, Pulse_le, Pulse_clock and Delay_small. Write these subroutines after going through the 5916 datasheet.

3. Timer. We have used timer_ms2 library from Arduino website. It can be set to call a subroutine after a preset time. Here this library calls Set_time every 5 ms.

The following commands are used to implement this function:

```
MsTimer2::set(5,Set_time);
MsTimer2::start();
```

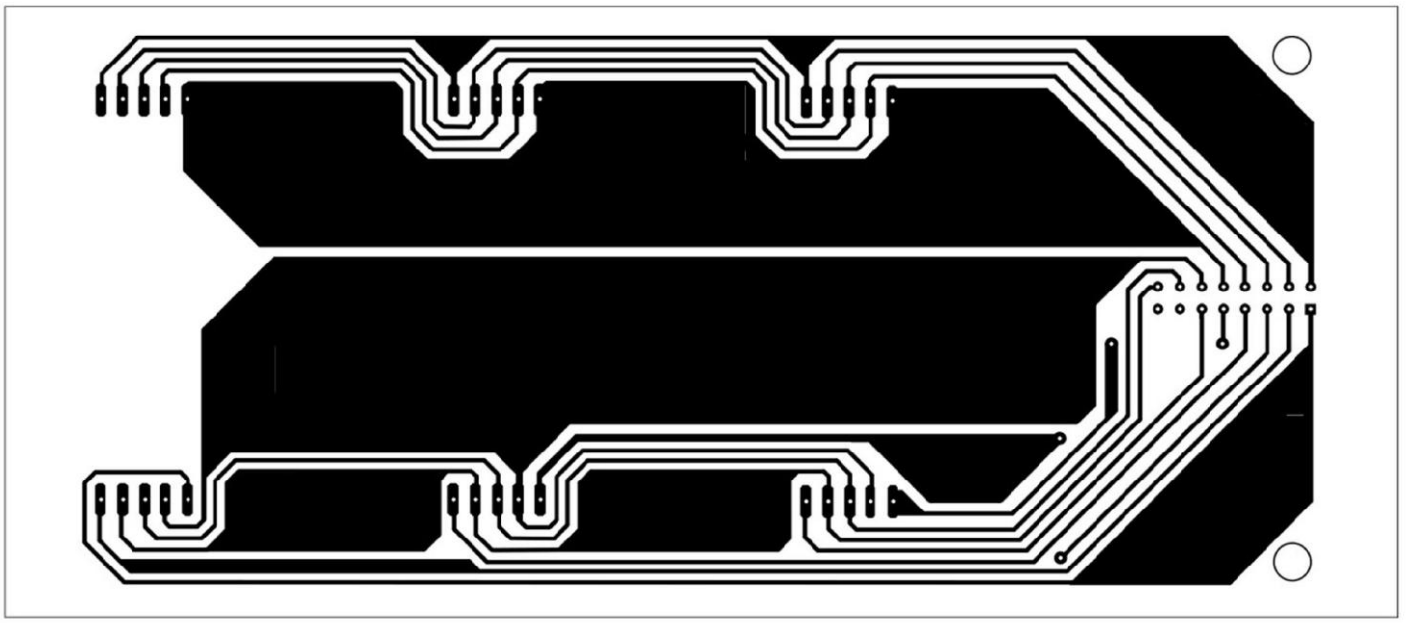


Fig. 5: An actual-size, single-side PCB layout for the display section

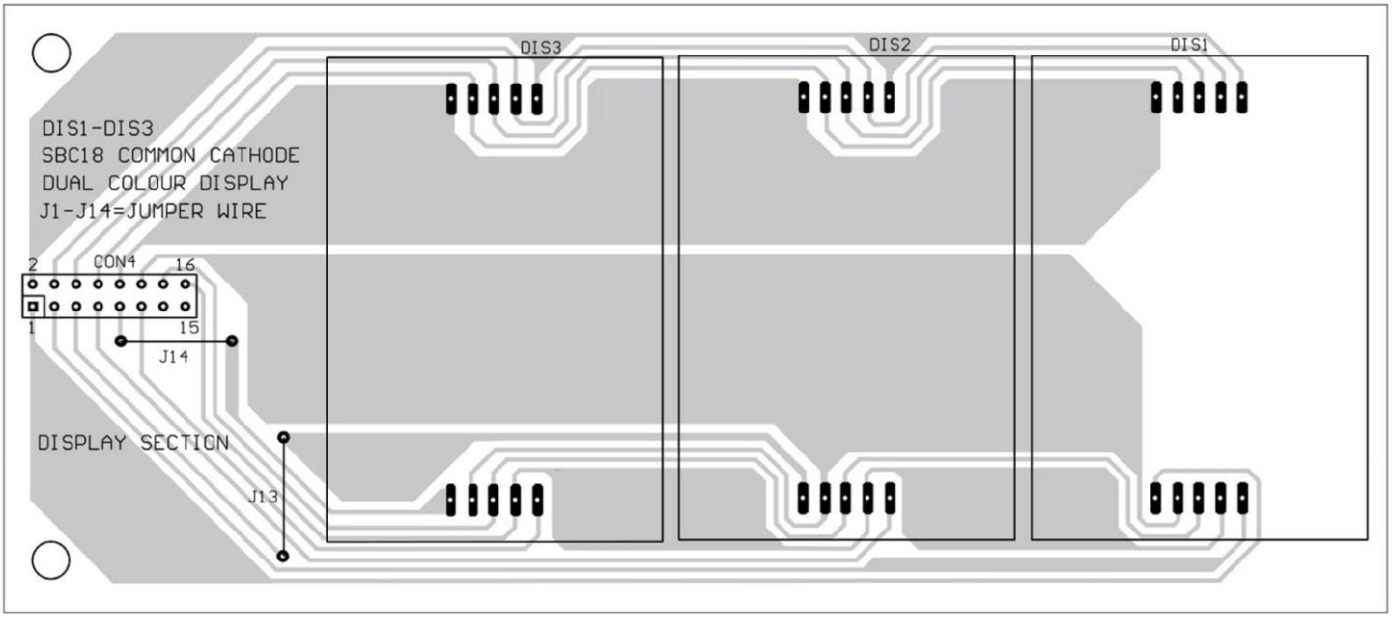


Fig. 6: Component layout for the PCB in Fig. 5

Of these two commands, the first sets the time period to 5 ms and declares the subroutine name as 'Set_time'. The second command starts the timer function.

4. Set_time. Within the Set_time that repeats every 5 ms, the display is refreshed automatically to show the next digit. Set the Half_sec_over flag when half second is over, i.e., when Set_time is called 100th time. Also, set the One_sec_over flag when one second is over, i.e., when Set_time is called 200th time. At this time, reset the count to zero so as to set Half_sec_over and One_sec_over flags again.

5. Refresh_grid. This function refreshes the display automatically to show only one digit at a time.

6. Setup. This function is called only once at power-on. It sets all the port pins as I/O pins, initialises the timer and sets the mode as read/scan.

PARTS LIST

Semiconductors:

IC1	- ATmega328 microcontroller
IC2	- 7805, 5V regulator
IC3, IC4	- PC817 optocoupler
IC5	- TD62783 grid display driver
IC6	- TLC5916 segment display driver
LED1-LED4	- 5mm light-emitting diode
D1, D2	- 1N4007 rectifier diode
D3	- 1N4148 signal diode

Resistors (all ¼-watt, ±5% carbon):

R1	- 680-ohm
R2, R10	- 10-kilo-ohm
R3, R4	- 1.5-kilo-ohm
R5, R6	- 1-kilo-ohm
R7	- 4.7-kilo-ohm
R8	- 330-ohm

Capacitors:

C1, C2, C5, C6, C10	- 0.1µF ceramic
C3, C4	- 22pF ceramic
C7, C8	- 1µF, 63V electrolytic

Miscellaneous:

CON1	- 8-pin dual-line berg strip male connector
CON2	- 4-pin SIL connector
CON3	- 16-pin berg strip male connector
CON4	- 16-pin DIL berg strip female connector
X_{TAL1}	- 16MHz crystal
DIS1-DIS3	- 44.5mm dual-colour common-anode seven-segment display

Fig. 7: Author's prototype of traffic light count-down timer with dual-colour display

Compile the program

Download Arduino 0022 or the latest version from http://arduino.cc/ and install like any other program. Now download Mstimer2.zip library file from http://arduino.cc/en/Reference/Libraries, unzip and save it in sketchbook/libraries folder.

Start Arduino 0022, load the source file and click Sketch⊠Verify/ Compile or press CNTR-R to compile the program. There should be no error in the program. However, a wrongly placed Mstimer2 library file may cause error.

Burn the program

You can burn the hex code generated by Arduino software in the microcontroller by:

1. Using a third-party universal programmer, or
2. Using an Arduino hardware with bootloader program and burning using Arduino IDE itself. At EFY lab, we used this option to burn our program into the MCU.

The download link given at the end of this article contains the hex code file for the traffic timer that will work in ATmega8, ATmega168 and ATmega328. This hex file, along with a standard universal programmer, can be used to program the MCU.

Construction and testing

An actual-size, single-side PCB for the AVR-based traffic light count-down timer is shown in Fig. 3 and its component layout in Fig. 4. An actual-size, single-side PCB layout for the display section is shown in Fig. 5 and its component layout in Fig. 6. Connect switches S1 and S2 to the red and green signal input lines for manual testing of the circuit.

Grid and segment control signals are fed to the display section through a 16-pin, dual-line, 2.54mm berg strip type female connector provided on the PCB of the display section. Note that the RTC section is not included in

the PCB.

Make sure that all the components are mounted and wiring done as per the schematic diagrams. At power-'on', the timer unit reads the red and green traffic signals. It learns the time after two cycles of red-green signaling and the next cycle onwards, the count-down time is displayed in the same colour as the traffic signal light. That is, when the traffic light is green the count-down time too is displayed in green colour, and when the traffic light is red the timer display is in red.

Before connecting the traffic light signals into the circuit, manually test the circuit for proper working as follows: Set the red-colour timer display by keeping switch S1 pressed until the desired time, say, 10 counts, is reached. Release S1 and then press it again until the display shows 10 counts. Similarly, for the green-colour timer display, press switch S2 until the desired time, say, 7 counts. Release it and then press again until the display shows 7 counts. Now if you press S1 again and hold it in this position, you will see the count-down in red colour from 10 to 0. Next, if you press S2 and hold it in this position, you will see the count-down in green colour from 7 to 0. This means that the system is working well. Now you can connect the red and green signal lines to respective points of CON2 as shown in Fig. 1.

The steps to connect your Arduino board to the computer and upload the code in the MCU follow:

1. Get a USB cable. Arduino is a simple board to start working with electronics and microcontroller programming. You need a standard USB cable (A plug to B plug)—like the one you connect to a USB printer—to connect the board to the PC.

2. Download the Arduino environment. To program the Arduino board, you need the Arduino environment. Download the latest version from the Internet. When the download completes, there would be a few files and sub-folders inside a folder named Arduino. Unzip the downloaded files retaining the folder structure. Note that you should have Java JDK pre-installed in your system.

3. Locate the USB drivers. If you are using a USB-based Arduino, you need to install drivers for the FTDI chip on the board. These can be found in the drivers/FTDI USB drivers directory of the Arduino distribution. The latest version of the drivers can be found on the FTDI website.

4. Connect the board. When you connect the board to the PC using the USB cable, 'Add New Hardware' wizard pops up on the screen. Select 'Install from a list or specified location (Advanced)' option and click 'Next.' Browse to the location of USB drivers in your PC. The 'New Hardware' wizard pops up.

Repeat step 4 to install 'USB Serial Port.' Open the Windows Device Manager from 'Control Panel.' If you find 'USB Serial Port' listed under its 'Ports' section, the Arduino board is installed in your PC. After successful installation, the green power LED (PWR) on the Arduino/Freeduino board should glow.

5. Run the Arduino environment. Open the Arduino folder and double-click the Arduino application.

6. Upload a program. Open the LED blink sample file as follows:

File→Sketchbook→Examples→Digital→Blink

Select the serial port of the Arduino board from the Tools→Serial Port menu. On Windows, this should be COM1 or COM2 for a serial board, or COM3, COM4 or COM5 for a USB board.

Now select the entry from the Tools→Board menu that corresponds to your Arduino board.

We have used Freeduino 1.15 board for programming the microcontroller. Select the board as 'Arduino BT w/Atmega328' in Arduino environment to upload the code in the ATmega328 MCU.

Click 'Upload' button in the Arduino environment. After a few seconds, you will see the RX and TX LEDs on the board flashing. If the upload is successful, the message 'Done uploading' appears in the status bar.

7. Look for the blinking LED. A few seconds after the upload, the LED on pin 13 of the MCU on the board should start blinking. If it does, congratulations! You've got Arduino up-and-running.

8. Upload the traffic light dual colour timer program. Run the Arduino IDE, click File→Open (browse the location where you have saved the source code). Compile the code from Sketch→Very/Compile option. Connect Freeduino board with ATmega328 MCU and upload the program. Remove the programmed MCU from the Freeduino board and put it in the PCB of the dual-colour timer. Perform manual testing of the timer as explained earlier.

Download source code: http://www.efymag.com/admin/issuepdf/Traffic_timer_source-code.zip

TRAFFIC_TIMER.CPP

```
/* Traffic_timer.cpp */
#include "WProgram.h"
void Small_delay (void);
void Pulse_clock(void);
void Pulse_le(void);
void Switch2special_5916 (void);
void Switch2normal_5916 (void);
void Send_byte_5916 (unsigned char Display_byte);
void Active_output_5916(void);
void Inactive_output_5916(void);
void Refresh_grid (void);
void Set_pins_dir(void);
void init_timer(void);
void Sense_input (void);
void Set_time(void);
void Binary2segment(unsigned int val);
void setup();
void loop();
void Reset_serial_packet(void);
void Act_one_tick_over ();
void Act_one_sec_over ();
void Act_half_sec_over (void);
void Send_packet(unsigned int x);
void Blankout_display();
void Act_setup();
unsigned char Grid_no,Seg_unit,Seg_ten,Seg_hun;
unsigned char I,J,K;
unsigned char Bit_5916,Small_tick_over, One_tick_
over, Half_sec_over, One_sec_over, One_min_over;
unsigned char Red_input,Green_input,Current_
red,Current_green,New_red_found,New_green_found;
unsigned char Ticks, Secs, Mins,New_red_count,New_
green_count,Color ;
unsigned char Numeric_table[10] = {0x7b, 0x11,
0xe3, 0xb3, 0x99, 0xba, 0xf8, 0x13, 0xfb, 0x9b};
/* Bit7    6    5    4    3    2    1    0
   G       E    D    C    F    dp   A    B */
#define FND_BLANK 0x00
#define DP_ON  0x04
//unsigned char Numeric_table[10] = {0xc0, 0xf9,
0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x98};
/*Bit7    6    5    4    3    2    1    0
  dp      g    f    e    d    c    b    a */
 #define UNIT_G 12
 #define UNIT_R 13
 #define TEN_G 10
 #define TEN_R 11
 #define HUN_G 8
 #define HUN_R 9
 #define SCL   19
 #define SDA   18
 #define OE    17
 #define LE    16
 #define CLK   15
 #define SDI   14
 #define P1    7
 #define P2    6
 #define SDO   5
 #define GREEN_SENSE  4
 #define RED_SENSE    3
 #define RE_DE        2
 #define TXD          1
 #define RXD          0
 #define INACTIVE_GRID 0
 #define ACTIVE_GRID 1
 #define ACTIVE_INPUT 0
 #define INACTIVE_INPUT 1
 #define COLOR_RED  1
 #define COLOR_GREEN 2
 #define COLOR_ORANGE 3
 #define COLOR_OFF 0
 #define MAX_GRID_NO 2
/*This file is sepcific to TLC5916
All routines specific to TLC5916 will be here*/
void Small_delay (void)
{
  for (J=0;J<2;J++)
  {
    J++;
    J--;
  }
}
void Pulse_clock(void)
{
  digitalWrite(CLK,HIGH);
  Small_delay();
  digitalWrite(CLK,LOW);
  Small_delay();
}
void Pulse_le(void)
{
  digitalWrite(LE,LOW);
  Small_delay();
  digitalWrite(LE,HIGH);
  Small_delay();
  digitalWrite(LE,LOW);
  Small_delay();
}

void Switch2special_5916 (void)
{
  digitalWrite(OE,HIGH);
  Small_delay();
  digitalWrite(LE,LOW);
  Small_delay();
  Pulse_clock();
  Small_delay();
  digitalWrite(OE,LOW);
  Small_delay();
  Pulse_clock();
  digitalWrite(OE,HIGH);
  Small_delay();
  Pulse_clock();
  digitalWrite(LE,HIGH);
  Small_delay();
  Pulse_clock();
  digitalWrite(LE,LOW);
  Small_delay();
  Pulse_clock();
}
void Switch2normal_5916 (void)
{
  digitalWrite(OE,HIGH);
  Small_delay();
  digitalWrite(LE,LOW);
  Small_delay();
  Pulse_clock();
  digitalWrite(OE,LOW);
  Small_delay();
  Pulse_clock();
  digitalWrite(OE,HIGH);
  Small_delay();
```

```
  Pulse_clock();
  Small_delay();
  Pulse_clock();
  Small_delay();
  Pulse_clock();
  Small_delay();
}
void Send_byte_5916 (unsigned char Display_byte)
{
  digitalWrite(OE,HIGH);
  Small_delay();
  digitalWrite(LE,LOW);
  Small_delay();
  for (I=0;I<8;I++)
  {
    if (Display_byte & 0x80)
      digitalWrite(SDI,HIGH);
    else
      digitalWrite(SDI,LOW);
    Display_byte = Display_byte<<1;
    Small_delay();
    Pulse_clock();
    digitalWrite(SDI,LOW);
  }
  Small_delay();
  Pulse_le();
}
void Active_output_5916(void)
{
  Small_delay();
  digitalWrite(OE,LOW);
  Small_delay();
}
void Inactive_output_5916(void)
{
  Small_delay();
  digitalWrite(OE,HIGH);
  Small_delay();
}

void Refresh_grid (void)
{
  digitalWrite(UNIT_G,INACTIVE_GRID);
  digitalWrite(UNIT_R,INACTIVE_GRID);
  digitalWrite(TEN_G,INACTIVE_GRID);
  digitalWrite(TEN_R,INACTIVE_GRID);
  digitalWrite(HUN_G,INACTIVE_GRID);
  digitalWrite(HUN_R,INACTIVE_GRID);

  Grid_no++;
  if (Grid_no > MAX_GRID_NO)
    Grid_no = 0;
  if (Grid_no == 0)
    Send_byte_5916(Seg_unit);
  if (Grid_no == 1)
    Send_byte_5916(Seg_ten);
  if (Grid_no == 2)
    Send_byte_5916(Seg_hun);
  Active_output_5916();
  if (Grid_no==0)
  {
    if ((Color==COLOR_RED) || (Color == COLOR_OR-
ANGE))
      digitalWrite(UNIT_R,ACTIVE_GRID);
    if ((Color==COLOR_GREEN) || (Color == COLOR_OR-
ANGE))
      digitalWrite(UNIT_G,ACTIVE_GRID);
  }
  if (Grid_no==1)
  {
    if ((Color==COLOR_RED) || (Color == COLOR_OR-
ANGE))
      digitalWrite(TEN_R,ACTIVE_GRID);
    if ((Color==COLOR_GREEN) || (Color == COLOR_OR-
ANGE))
      digitalWrite(TEN_G,ACTIVE_GRID);
  }
    if (Grid_no==2)
  {
    if ((Color==COLOR_RED) || (Color == COLOR_OR-
ANGE))
      digitalWrite(HUN_R,ACTIVE_GRID);
    if ((Color==COLOR_GREEN) || (Color == COLOR_OR-
ANGE))
      digitalWrite(HUN_G,ACTIVE_GRID);
  }

}

#include <MsTimer2.h>
 void Set_pins_dir(void)
 {
   //All Output Pins set here
     pinMode(UNIT_G, OUTPUT);
     pinMode(UNIT_R, OUTPUT);
     pinMode(TEN_G, OUTPUT);
     pinMode(TEN_R, OUTPUT);
     pinMode(HUN_G, OUTPUT);
     pinMode(HUN_R, OUTPUT);
     pinMode(SCL, OUTPUT);
     pinMode(SDA, OUTPUT);
     pinMode(OE, OUTPUT);
     pinMode(LE, OUTPUT);
     pinMode(CLK, OUTPUT);
     pinMode(SDI, OUTPUT);
     pinMode(RE_DE, OUTPUT);
     pinMode(TXD, OUTPUT);

   //All Input Pins set here
     pinMode(SDO,INPUT);
     pinMode(P1, INPUT);
     pinMode(P2, INPUT);
     pinMode(GREEN_SENSE, INPUT);
     pinMode(RED_SENSE, INPUT);
     pinMode(RXD, INPUT);

 }
void init_timer(void)
{
    // set timer int at 5 msec
  MsTimer2::set(5,Set_time); // 5ms period
  MsTimer2::start();
}

void Sense_input (void)
{
  Current_red = digitalRead(RED_SENSE);
  Current_green = digitalRead(GREEN_SENSE);

  if (Current_red != Red_input)
  {
    if (New_red_count<10)
    {
      New_red_count++;
      if (New_red_count == 10)
      {
         Red_input=Current_red;
         New_red_found = 1;
```

```
      }
    }
  }
  else
    New_red_count=0;

  if (Current_green != Green_input)
  {
    if (New_green_count<10)
    {
      New_green_count++;
      if (New_green_count == 10)
      {
         Green_input=Current_green;
         New_green_found = 1;
      }
    }
  }
  else
    New_green_count=0;
}

void Set_time(void)
{
    //Program comes here at every 5 ms interrupt

   One_tick_over = 1;
   Ticks++;
   Refresh_grid();
   Sense_input();
   if     (Ticks == 100)              //500 ms
   {
         Half_sec_over = 1;
   }

   if (Ticks == 200)                  //1000 ms
   {
         Ticks = 0;
         Secs++;
         One_sec_over = 1;

   }

   if (Secs == 60)
   {
         Mins++;
         One_min_over = 1;
         Secs = 0;
   }

}
/*general purpose routines are here.do not use any
hardware specific functions*/

void Binary2segment(unsigned int val)
{
  Seg_hun = Numeric_table[val/100];
  val = val% 100;
  Seg_ten = Numeric_table[val/10];
  Seg_unit = Numeric_table[val%10];
}

 void setup()
{
  //On power on call this routine for set all pins
directions for In/Out
    Set_pins_dir();
  //initialize Timer
    init_timer();
    Act_setup();
    K=0x01;
    digitalWrite(RE_DE,LOW);     // step 2 done
    Serial.begin(9600);
}

void loop()
{
  if (One_tick_over)
  {
     One_tick_over=0;
     Act_one_tick_over();
  }

  if (Half_sec_over)
  {
    Half_sec_over=0;
    Act_half_sec_over();
  }


  if (One_sec_over==1)
  {
    One_sec_over=0;
    Act_one_sec_over();
  }

}
unsigned char Mode, Learn_red, Verify_
red,Time4input_active,Phase[2],z=0,Input_
color[2],j;
unsigned int Remaining_time[2],Learn_
time[2],Verify_time[2],Display_time[2];
unsigned char Serial_timeout, Serial_count, Serial_
data, Slave_phase, Master_display, Packet_count;
unsigned char  Master_learn_time;
unsigned int Slave_time=0;
unsigned char Count_learn_time,Wait4_
inactive[2],Wait4_active[2],TEST_MODE,Display_
on,SELF_TEST_MODE,MS_MODE;  // flags
#define MASTER_MODE 1
#define SLAVE_MODE 0
#define START_BYTE 85
#define DISPLAY_BLANK 1000
#define DP_BLINK  1001
#define ERROR_CODE 1002
#define POWER_ON 0
#define LEARN   1
#define VERIFY  2
#define DISPLAY_REMAINING_TIME 3
#define TOLERANCE    2
void Reset_serial_packet(void)
{
  Serial_count = 0;

}

void Act_one_tick_over ()
{
  if (MS_MODE == MASTER_MODE)
  {
    if (SELF_TEST_MODE)
      return;

    if((Red_input == ACTIVE_INPUT) && (Green_input
== ACTIVE_INPUT))
```

```
{
  Time4input_active=0;
  Count_learn_time = 0 ;
  for(j=0;j<2;++j)
  {
    Wait4_inactive[j]=1;
    Wait4_active[j]=0;
    Phase[j]=LEARN;
  }

}

Input_color[0]=Red_input;
Input_color[1]=Green_input;

  for(j=0;j<2;++j)
{
  if (Wait4_inactive[j]==1)
  {
    if (Input_color[j] == INACTIVE_INPUT)
    {
      Wait4_active[j]=1;
      Wait4_inactive[j]=0;
      Phase[j] = LEARN;
    }
  }
} //if wait4red_inactive

for(j=0;j<2;++j)
{
  if (Wait4_active[j]==1)
  {
    if (Input_color[j] == ACTIVE_INPUT)
    {
      z=j;
      Wait4_active[j]=0;
      if(j==0)
        Color=COLOR_RED;
      else
        Color=COLOR_GREEN;
      Time4input_active=0;
      Count_learn_time=1;
      if (Phase[j] == DISPLAY_REMAINING_TIME)
      {
        Remaining_time[j]=Learn_time[j];
        Display_on=1;
      }
    }
  }// if wait4red_active
}

if (Display_on == 1)
{
  if (Input_color[z] == INACTIVE_INPUT)
  {
    Wait4_active[z]=1;
    Display_on=0;
    Blankout_display();
    Send_packet(DISPLAY_BLANK);
    if (Remaining_time[z]>TOLERANCE)
      Phase[z]=LEARN;
    Wait4_active[z]=1;
  }
}

if(Phase[z] ==DISPLAY_REMAINING_TIME )
{
  if(Input_color[z]==ACTIVE_INPUT)
  {
      if(Display_on==0)
      {
        if((Time4input_active-Learn_
time[z])>TOLERANCE)
          Phase[z]=LEARN;
      }
    }
  }


  if (Count_learn_time==1)
  {
    if (Input_color[z] == INACTIVE_INPUT)
    {
      Count_learn_time = 0 ;
      Blankout_display();
      Send_packet(DISPLAY_BLANK);
      Wait4_active[z]=1;
      if (Phase[z] == LEARN)
      {
        Learn_time[z]=Time4input_active;
        Phase[z] = VERIFY;
      }// learn phase

      else if (Phase[z] == VERIFY)
      {
        Blankout_display();
        Send_packet(DISPLAY_BLANK);
        Verify_time[z]=Time4input_active;
        if (Verify_time[z] > Learn_time[z])
        {
          if((Verify_time[z]-Learn_
time[z])<TOLERANCE)
          {
            Learn_time[z]=Verify_time[z];
            Phase[z]=DISPLAY_REMAINING_TIME;
          }
          else
          {
            Phase[z]=VERIFY;
            Learn_time[z]=Verify_time[z];
          }
        }
        else if (Verify_time[z] < Learn_time[z])
        {
          if((Learn_time[z]-Verify_
time[z])<TOLERANCE)
          {
            Learn_time[z]=Verify_time[z];
            Phase[z]=DISPLAY_REMAINING_TIME;
          }
          else
          {
            Phase[z]=VERIFY;
            Learn_time[z]=Verify_time[z];
          }
        }
        else
          Phase[z] = DISPLAY_REMAINING_TIME;
      }//verify phase

      else if (Phase[z] == DISPLAY_REMAINING_
TIME)
      {
        Display_time[z] == Time4input_active;
        if (Display_time[z] > Learn_time[z])
        {
          if((Display_time[z]-Learn_
time[z])<TOLERANCE)
```

```
            {

              Learn_time[z]=Display_time[z];
              Phase[z] = VERIFY;
            }
          }
          else if (Display_time[z] < Learn_time[z])
          {
            if((Learn_time[z]-Display_
time[z])<TOLERANCE)
            {
              Learn_time[z]=Display_time[z];
              Phase[z]=VERIFY;
            }
          }
        } // display_remaining_time phase
      }// inactive input
    }// count learn time
  }
  else if (MS_MODE==SLAVE_MODE)
  {
    if (Serial_timeout > 0)
    {
      Serial_timeout--;
      if (Serial_timeout == 0)
        Reset_serial_packet();
    }
    if (Serial.available() > 0)
    {
      Serial_timeout = 10;
      Serial_data = Serial.read();
      if (Serial_count == 0)
      {
        if (Serial_data == START_BYTE)
          Serial_count++;
        else
          Reset_serial_packet();
      }
      else if (Serial_count == 1)
      {
        if (Serial_data == START_BYTE)
          Serial_count++;
        else
          Reset_serial_packet();
      }
      else if (Serial_count == 2)  // time for ac-
tive input
      {
        Slave_time=Serial_data * 256;
        Serial_count++;
      }
      else if (Serial_count == 3)  // remaining
time
      {
        Slave_time=Slave_time + Serial_data;
        Serial_count++;
      }
      else if (Serial_count == 4)  // phase(z)
      {
        Slave_phase=Serial_data;
        Serial_count++;
      }
      else if (Serial_count == 5)  // color
      {
        Color=Serial_data;
        Reset_serial_packet();
      }
    }
    if( Slave_time == DISPLAY_BLANK)
      Blankout_display();
    else if (Slave_time == DP_BLINK)
    {
      if (Slave_phase == LEARN)
        Seg_hun |= DP_ON;
      if (Slave_phase == VERIFY)
        Seg_ten |= DP_ON;
      if (Slave_phase == DISPLAY_REMAINING_TIME)
        Seg_unit |= DP_ON;
    }
    else if (Slave_time == ERROR_CODE)
    {
      Seg_hun |= DP_ON;
      Seg_ten |= DP_ON;
      Seg_unit |= DP_ON;
    }
    else if(Slave_time!=768)
      Binary2segment(Slave_time);
  }
}//Act_one_tick_over()

void Act_one_sec_over ()
{
  if(MS_MODE == MASTER_MODE)
  {
    if (SELF_TEST_MODE)
    {
      Color=COLOR_RED;
      if (Remaining_time[z] > 1000)
      {
        SELF_TEST_MODE=0;
        Blankout_display();
        Send_packet(DISPLAY_BLANK);
        for(j=0;j<2;j++)
        {
          if (Input_color[j] == ACTIVE_INPUT)
          {
            Wait4_inactive[j]=1;
            Wait4_active[j]=0;
          }
          else
          {
            Wait4_active[j]=1;
            Wait4_inactive[j]=0;
            Phase[j] = LEARN;
          }
        }
      }
      else
      {
        Binary2segment(Remaining_time[z]);
        Send_packet(Remaining_time[z]);
        Remaining_time[z]=Remaining_time[z]+111;
      }
    }
    else
    {
      if((Input_color[0] == ACTIVE_INPUT) && (In-
put_color[1] == ACTIVE_INPUT))
      {
        Color=COLOR_RED + COLOR_GREEN;
        Seg_hun |= DP_ON;
        Seg_ten |= DP_ON;
        Seg_unit |= DP_ON;
        Send_packet(ERROR_CODE);
        return;
      }
```

```
      if (Count_learn_time==1)
      {
        Time4input_active ++;
        if (Display_on==0)
        {
          if (TEST_MODE==1)
          {
            //Send_packet(Time4input_active);
            Binary2segment(Time4input_active);
          }
          else
          {
            Blankout_display();
          }
          Send_packet(Time4input_active);
        }
      }
      else
      {
        Blankout_display();
        Send_packet(DISPLAY_BLANK);

      }
      if (Display_on==1)
      {
        if (Remaining_time[z]==0)
        {
          Display_on=0;
          Blankout_display();
          Send_packet(DISPLAY_BLANK);
        }
        else
        {
          Binary2segment(Remaining_time[z]);
          Send_packet(Remaining_time[z]);
          Remaining_time[z]--;
        }
      }

    }
  }
}

void Act_half_sec_over (void)
{
  if (MS_MODE == MASTER_MODE)
  {
    if (SELF_TEST_MODE==1)
      return;

    if((Red_input == ACTIVE_INPUT) && (Green_input
== ACTIVE_INPUT))
    {
      Blankout_display();
      Send_packet(DISPLAY_BLANK);
      return;
    }

    if ((Count_learn_time == 0) && (TEST_MODE==1))
    {
      if(z==0)
        Color=COLOR_RED;
      else if(z==1)
        Color=COLOR_GREEN;

      if (Phase[z] == LEARN)
        Seg_hun |= DP_ON;
      if (Phase[z] == VERIFY)
        Seg_ten |= DP_ON;
      if (Phase[z] == DISPLAY_REMAINING_TIME)
        Seg_unit |= DP_ON;
    }
    if (Count_learn_time == 0 )
      Send_packet(DP_BLINK);
  }
}
void Send_packet(unsigned int x)  //step 4 done
{
  digitalWrite(RE_DE,HIGH);
  delayMicroseconds(100);
  Serial.print (START_BYTE,BYTE);
  delayMicroseconds(100);
  Serial.print (START_BYTE,BYTE);
  delayMicroseconds(100);
  Serial.print(x/256,BYTE);
  delayMicroseconds(100);
  Serial.print(x%256,BYTE);
  delayMicroseconds(100);
  Serial.print(Phase[z],BYTE); // 1 learn, 2 veri-
fy, 3 display remaining time
  delayMicroseconds(100);
  Serial.print(Color,BYTE);  // z= 0 for red, 1 for
green
  delay(2);  // addd delay of 1 character before
switching off re_de
  digitalWrite(RE_DE,LOW);
}
void Blankout_display()
{
  Seg_hun=FND_BLANK;
  Seg_ten=FND_BLANK;
  Seg_unit=FND_BLANK;
}
void Act_setup()
{
  Remaining_time[0] = 000;
  Grid_no=0;
  Phase[0] = LEARN;  //At power on bydefault goes
application in learn mode
  Phase[1] = LEARN;
  Blankout_display();
  digitalWrite(P1,HIGH);
  digitalWrite(P2,HIGH);
  TEST_MODE = !digitalRead(P1);
  MS_MODE = digitalRead(P2);
  SELF_TEST_MODE=1;
}

int main(void)
{
        init();
        setup();
        for (;;)
        loop();
        return 0;
}
```

Display Systems

RANK DISPLAY SYSTEM FOR RACE AND QUIZ COMPETITIONS

■ **PANKAJ KISHOR VARMA**

There are so many games where the winner is the one who takes the least time in successfully completing the task given. Take for instance the 'fastest finger first' in Kaun Banega Crorepati quiz show on TV or the 'fastest crossing first' in any type of race. In these competitions, sometimes there may be two or more players who appear to complete the task in equal time. In such cases, it becomes difficult for the judge to announce the winner, though there may be time difference of a few milliseconds between the individuals in accomplishing the task.

Here is a circuit based on Atmel microcontroller AT89C51 that can resolve the time-difference ambiguity and indicate correct ranking of all the participants on a liquid-crystal display (LCD) module. It has been designed for a maximum of eight participants playing at a time, denoted by alphabets 'A' through 'H,' and can be used both for a 'fastest finger first' quiz and 'fastest crossing first' race.

Circuit description

Fig. 1 shows the circuit of the microcontroller-based rank display system. It comprises microcontroller AT89C51, transistor array IC ULN2803, LCD module (16x2), regulator IC 7805 and a few discrete components.

IC AT89C51 is a low-power, high-performance, 8-bit microcomputer with 4 kB of Flash programmable and erasable read-only memory (PEROM), 128 bytes of RAM, 32 input/output (I/O) lines, two 16-bit timers/counters, a five-vector two-level interrupt architecture, a full-duplex serial port, on-chip oscillator and clock circuitry. Port-0 pins of the microcontroller (IC1) are pulled high via 4.7-kilo-ohm resistor network RNW1 and interfaced with switches S1 through S8 (corresponding to players 'A' through 'H'), respectively. Port-0 pins P0.0 through P0.7 are also connected to pins 11 through 18 of ULN2803. Port-2 pins P2.0 through P2.7 are connected to inputs 1 through 8 of IC2.

Each time a switch is pressed, a feedback is sent from port 2 through port 0 via ULN2803 (IC2). The purpose of this feedback is to make a given pin of port-0 'low,' independent of the condition of the corresponding switch (S1 through S8) just after it has been pressed once.

One more input switch S11 is provided to reset the circuit for starting the next round. Switch S11 is connected to P3.0 pin 10 of the microcontroller. Pressing of switch S11 starts the next round of questions. Thus, it acts as a latch. R1 through R8 are current-limiting resistors.

Port-1 pins P1.0 through P1.7 and port-3 pins P3.1 through P3.3 of the microcontroller are interfaced with data line pins 7 through 14 and pins 4 through 6 of the LCD module for displaying the various rankings. Contrast pin 3 of the LCD module can be controlled using preset VR1. Pin 3.4 of the microcontroller is

PARTS LIST

Semiconductor:

IC1	- AT89C51 microcontroller
IC2	- ULN2803 Darlington array
IC3	- 7805, 5V regulator
D1-D4	- 1N4001 rectifier diode
LED1	- 5mm LED

Resistors (all ¼-watt, ±5% carbon):

R1-R8, R12	- 4.7-kilo-ohm
R9	- 100-ohm
R10	- 10-kilo-ohm
R11	- 330-ohm
VR1	- 10-kilo-ohm preset
RNW1	- 4.7-kilo-ohm resistor network

Capacitors:

C1, C2	- 33pF ceramic disk
C3	- 1000µF, 25V electrolytic
C4	- 10µF, 16V electrolytic
C5	- 22µF, 16V electrolytic
C6-C9	- 0.1µF ceramic disk

Miscellaneous:

X1	- 230V primary to 12V, 500mA secondary transformer
S1-S8, S10, S11	- Push-to-on switch
S9	- Slide switch
S12	- On/Off switch
X_{TAL}	- 11.0592MHz crystal
PZ1	- Piezobuzzer
	- LCD module (16×2) line

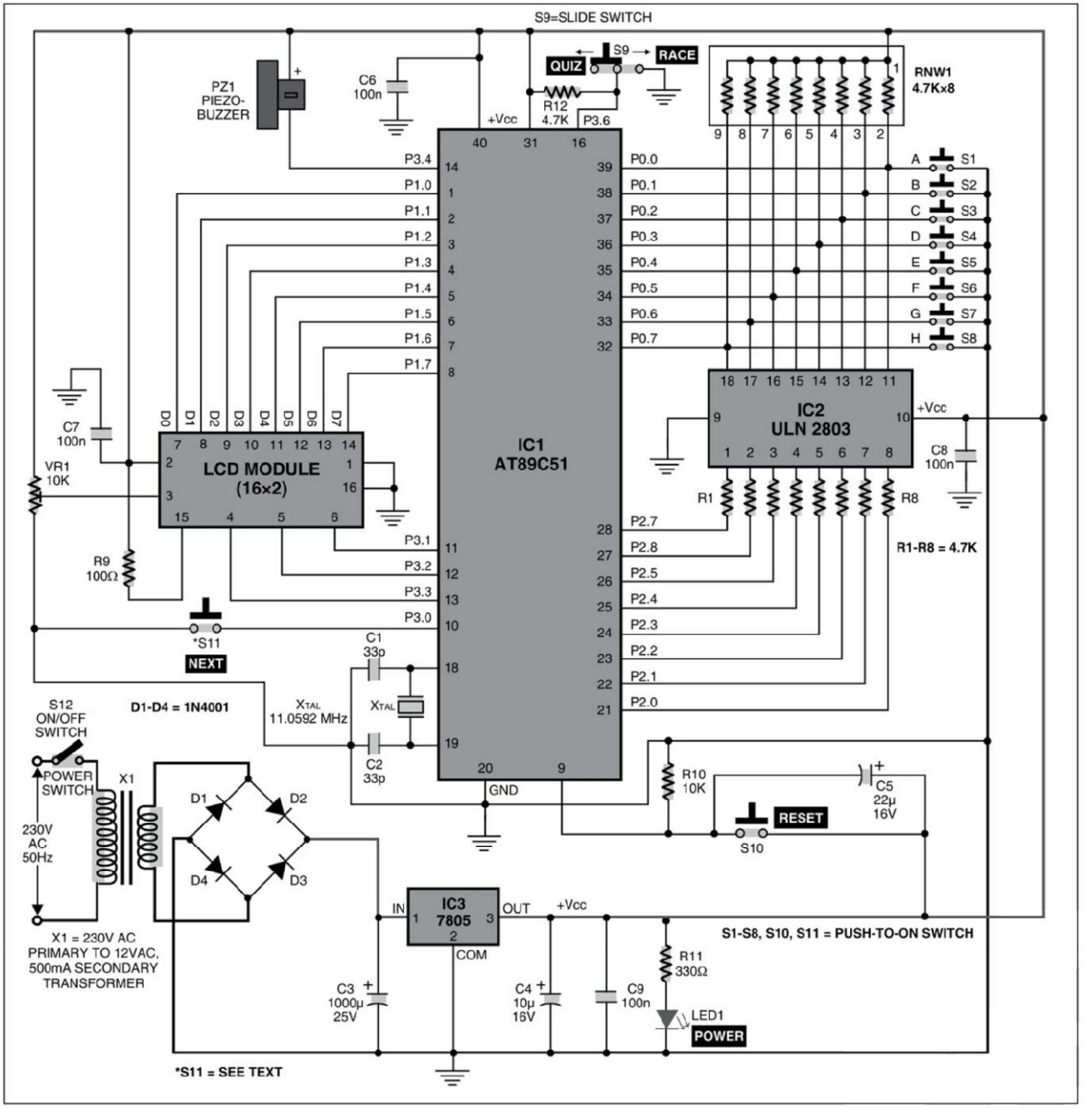


Fig. 1: Circuit of the microcontroller-based rank display system for race and quiz competitions

used to control piezobuzzer PZ1. When P3.4 goes low, piezobuzzer PZ1 sounds. Pin 3.6 of IC1 is connected to slide switch S9 for selection between race and quiz.

Switch S10 is used to manually reset the microcontroller, while the power-on reset signal for the microcontroller is derived from the combination of capacitor C5 and resistor R10. A 11.0592MHz crystal generates the basic clock frequency for the microcontroller.

230V AC mains is stepped down by transformer X1 to deliver the secondary output of 12V, 500mA. The transformer output is rectified by a full-wave bridge rectifier comprising diodes D1 through D4, filtered by capacitor C3 and then regulated by IC 7805 (IC3). Capacitors C4 and C9 bypass any ripple present in the regulated power supply.

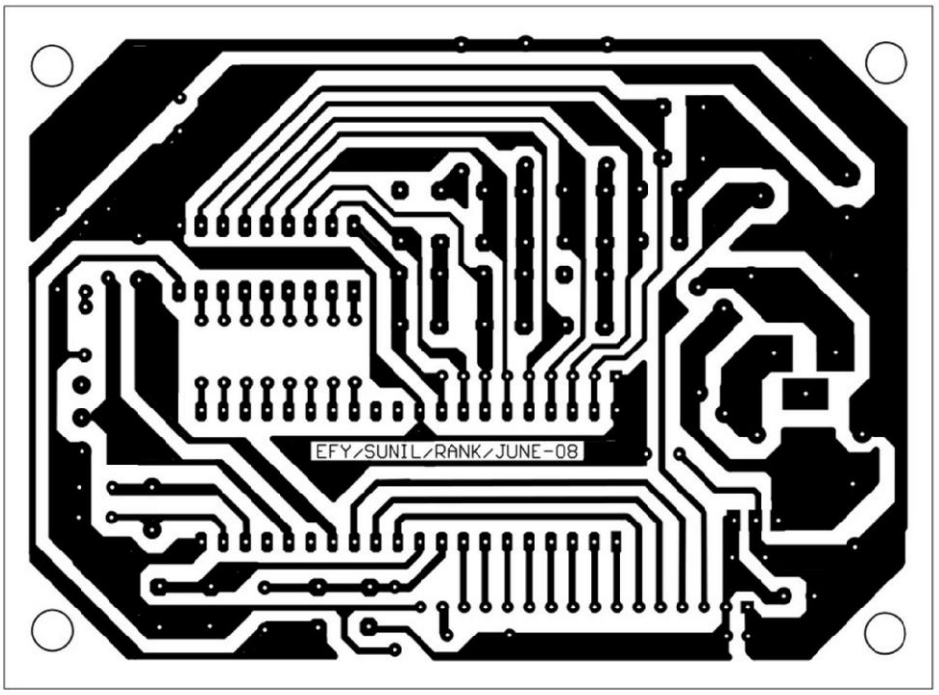


Fig. 2: Actual-size, single-side PCB for the microcontroller-based rank display system

How it works?

First of all, select the mode by using switch S9. For example, when you slide switch S9 towards 'quiz' option, the given circuit can be used as 'fastest finger first' quiz system. At the resumption of power, a message "Welcome to all the players" is displayed on the LCD. After some time, it displays players' names 'A' through 'H' in the first line and the message 'First Question' in the second line. Now the quizmaster can ask the first question. The monitor program continuously checks the status of port 0. If any input switch S1 through S8 is found low (by the controller program), it means that the corresponding switch has been pressed.

As soon as any of competitors 'A' through 'H' presses his switch first, his rank is stored as '1' immediately in the rank table at the corresponding RAM location. The buzzer sounds a beep and the timer starts recording. The status of the rank table is updated repeatedly at intervals into the rank table and this status is maintained (due to latching at port 0, offered by feedback from IC2) until the next key is pressed.

Now the second case arises:

1. If the remaining seven players press their switches within 30 seconds, the LCD displays the message "Done in Time" and the buzzer sounds a beep. Thereafter, the LCD displays the ranks ('1' through '8') of all players below their names.

2. If the remaining seven players don't press their switches within 30 seconds, at the end of this period the LCD displays the message "Time Out" and the buzzer sounds a beep. After that, the ranks of those who have pressed their switches within time are displayed below their names, while '-' is displayed below the names of the remaining players.

The display stays in this position and the monitoring program checks the status of pin P3.0. If P3.0 (pin 10) of IC1 goes low on pressing switch S11 for about 2 ms, all the things are initialised for the next question and the LCD shows the message 'Next Question' ('Next Round' for race).

For race competitions, the circuit works in a similar manner but you need to replace switch S11 with a foot switch or LDR-laser light combination. The message "Welcome to all the players" is replaced with "Welcome to all the racers." 'Next Question' is replaced with 'Next Round.' "Done in Time" is replaced with "Covered in Time."

An actual-size, single-side PCB for the microcontroller-based rank display system is shown in Fig. 2 and its component layout in Fig. 3.

This PCB is designed for demo purpose only. In actual application, where contestants are seated in different locations, you can extend input switches S1 through S8 to the players. For this, connect a pair of wires to the connecting terminals of each of these switches on the PCB to extend these to players. In this case, tactile switches S1 through S8 are not required on the PCB.

Software

The software is written in Assembly language and assembled using ASM51 cross-assembler. Intel hex code is

generated and burnt into the microcontroller chip using a suitable programmer. The software is well commented and easy to understand. Timer 0 is used to introduce time-out after 30 seconds from the instant when any key is pressed first. RAM locations '31H' through '38H' store the ranks of players 'A' through 'H,' respectively.

Download Source Code: http://www.efymag.com/admin/issuepdf/Rank%20Display%20System%20for%20Race%20and%20Quiz%20Competitions.zip

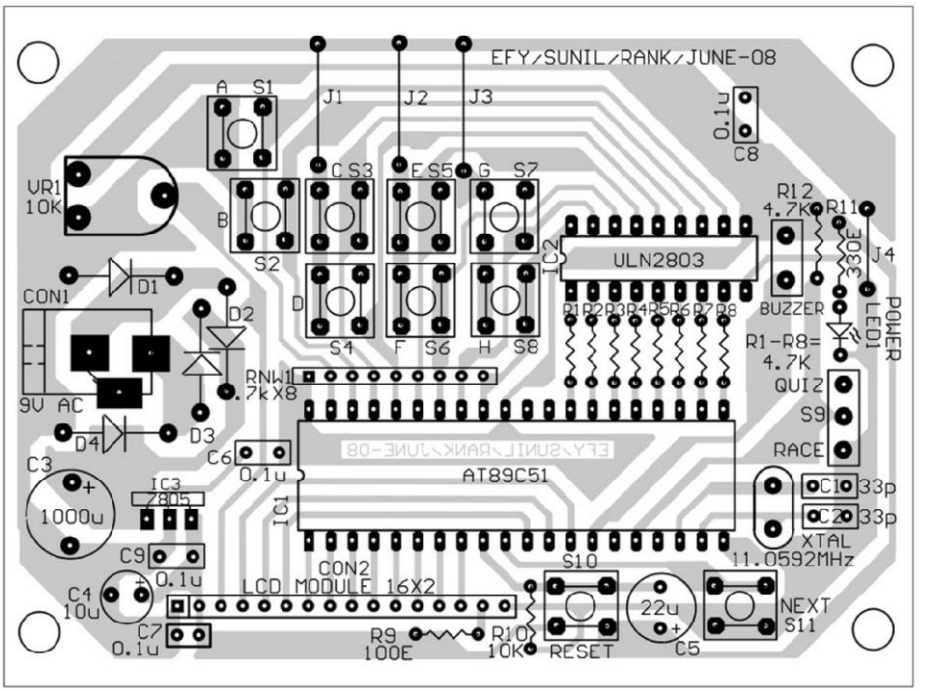


Fig. 3: Component layout for the PCB

RANK.ASM

```
   $mod51
   latch   equ p2          ;port for providing feedback
   keys    equ p0          ;input stage port
   dline   equ p1          ;8-bit databus for LCD
   restart equ p3.0        ;flag for next question/round
   rs      equ p3.3        ;register select for LCD
   rw      equ p3.2        ;read/write for LCD
   en      equ p3.1        ;LCD enable pin
   buzzer  equ p3.4        ;pin for sounding buzzer
    mode     equ p3.6          ;'fff' or 'race' selec-
tion pin
   flag    equ p3.5        ;flag to check 'done in time'
or 'time out'

   org 0000h
   sjmp main

   org 000bh               ;timer0 interrupt vector ad-
dress
      clr tr0              ;clear timer0 run bit
      mov tl0,#00h
       mov th0,#00h        ;reload timer0 with initial
count
      djnz r5,a1           ;left buzzer on unless r5=0
      setb buzzer          ;stop buzzer
   a1: djnz r7,return
      mov r7,#136
      djnz r6,return
      mov r6,#3
      mov r5,#20
      clr buzzer           ;make buzzer on
      mov a,#0c0h
      acall command
      mov dptr,#timeout
      acall read           ;display 'timeout'
        acall dispdly          ;keep on displaying for
sometime
      setb buzzer          ;stop buzzer
      setb flag
   return: reti            ;return from interrupt ser-
vice routine

   main: mov ie,#00h
         setb ea
         setb et0               ;enable timer0 interrupt
          mov tmod,#01h           ;timer0 configured in
mode 1
         mov tcon,#00h
         mov tl0,#00h
          mov th0,#00h           ;set initial count to
0000H
         mov r7,#136
         mov r6,#3
         mov r5,#20
         mov latch,#0           ;initialize input stage
         mov keys,#0ffh         ;initialize input stage
         acall lcdly
         acall lcd_int          ;initialize LCD
         mov dptr,#welcome
         acall read             ;display 'welcome to all'
          mov a,#0c0h             ;starting address for
2nd line
         acall command
           jnb mode,race1           ;jump to race1 when
'race' mode selected
          mov dptr,#welcome1
          sjmp fff1
   race1: mov dptr,#welcome2
   fff1: acall read
         acall dispdly
         mov a,#80h
         acall command
         mov dptr,#plrname
         acall read             ;display player names
         mov a,#0c0h
         acall command
         jnb mode,race2
         mov dptr,#firstqtn
         sjmp fff2
   race2: mov dptr,#firstrnd
```

```
 fff2: acall read

  here:  clr flag
         mov latch,#0        ;initialize input stage
         mov keys,#0ffh      ;initialize input stage
         acall lcdly
          mov 04h,#48          ;ASCII code of '0' is
loaded into r4
  blankrank: mov r0,#31h     ;point the first location
of rank table
       again: mov @r0,#'-'    ;initialize rank table
by '-'
              inc r0
              cjne r0,#39h,again

  chk: mov a,keys            ;mov status of input into
accumulator
       cjne a,#0ffh,first    ;check if any switch is
pressed
       sjmp chk

  first: clr buzzer          ;sound buzzer

  next_rank: inc 04h         ;load r4 with ASCII code
of next rank
             mov r0,#31h     ;point the first location
of rank table
       chk1: jb flag,result1
             setb tr0        ;make timer0 run
             mov a, keys     ;mov status of input into
accumulator
             cpl a
             xrl a,latch
             cjne a,#0,scan ;check for any change in
previous status
             sjmp chk1

   scan: jb acc.0,store ;now start scanning for the
pressed switch from here
        inc r0          ;point to next address in rank
table (32H)
         jb acc.1,store
         inc r0         ;point to next address in rank
table (33H)
         jb acc.2,store
         inc r0         ;point to next address in rank
table (34H)
         jb acc.3,store
         inc r0         ;point to next address in rank
table (35H)
         jb acc.4,store
         inc r0         ;point to next address in rank
table (36H)
         jb acc.5,store
         inc r0         ;point to next address in rank
table (37H)
         jb acc.6,store
         inc r0         ;point to next address in rank
table (38H)
         jb acc.7,store

  store: mov @r0,04h    ;move ASCII code of current
rank to the pointed location
         orl a,latch
         mov latch,a
           cjne r4,#56,next_rank ; 56 is ASCII code
for '8'
         sjmp result

  result: clr tr0
          mov tl0,#00h
           mov th0,#00h          ;reload timer0 with
initial count
          mov r7,#136
          mov r6,#3
          mov r5,#20
          clr buzzer           ;stop buzzer
          mov a,#0c0h
          acall command
          jnb mode,race4
          mov dptr,#intime
          sjmp fff4
  race4: mov dptr,#intime1
  fff4: acall read
        acall dispdly
        setb buzzer
  result1: clr tr0
           mov r0,#31h          ;reinitialize rank
table pointer
          mov a,#0c0h
          acall command
     show: mov a,@r0
            acall display        ;display the rank
below name
          mov a,#' '
           acall display       ;provide a space be-
tween ranks
          inc r0
            cjne r0,#39h,show    ;keep on displying
upto the rank of 'H'

  stay: jb restart,stay        ;now stay here unless
a remote-key is pressed
        mov r1,#4
    s1: mov r2,#200
        djnz r2,$
        djnz r1,s1
         jb restart,stay         ;recheck if p3.0 is
high

  nxt_Ques: mov a,#0c0h
            acall command
            jnb mode,race3
            mov dptr,#nxtqtn
            sjmp fff3
  race3: mov dptr,#nxtrnd
  fff3: acall read           ;display 'next question
(or round)'
         jmp here               ;get ready for next
question/round

  read:  clr a
         movc a,@a+dptr
         jz down
         acall display
          inc dptr
          sjmp read
    down: ret

   lcd_int: mov a,#38h          ;2 line disp, 8bits/
ch, 5x7 matrix
            acall command     ;write into the LCD com-
mand register
            mov a,#0ch        ;display on, cursor off
            acall command
            mov a,#01h        ;clr screen & bring cur-
sor home (80H)
            acall command
            mov a,#06h        ;right shifting for cur-
sor movement
            acall command
            ret

  command: mov dline,a
            clr rs            ;select command regis-
ter of LCD
           nop
           nop
           clr rw             ;select write mode for
LCD
           nop
           nop
           setb en
           acall lcdly1
```

```
           clr en              ;now write into com-
mand register
          acall lcdly1
          ret

  display: mov dline,a
           setb rs            ;select data register
of LCD
          nop
          nop
          clr rw             ;select write mode for
LCD
          nop
          nop
          setb en
          acall lcdly
           clr en             ;now write into data
register
          acall lcdly
          ret

   lcdly: mov r1,#0ah
     lp1: mov r2,#0ffh
          djnz  r2,$
          djnz r1,lp1
          ret
  lcdly1: mov r1,#3fh
     lp2: mov r2,#0ffh
          djnz  r2,$
          djnz r1,lp2
          ret

 dispdly: mov r1,#255
      d1: mov r2,#255
      d2: mov r3,#30
          djnz r3,$
          djnz r2,d2
          djnz r1,d1
          ret

 ;Look up table starts from here

 plrname:  db 'A',' ','B',' ','C',' ','D',' ','E','
','F',' ','G',' ','H',0
 welcome:  db ' WELCOME TO ALL ',0
 welcome1: db '  THE  PLAYERS  ',0
 welcome2: db '  THE  RACERS   ',0
 firstqtn: db ' FIRST QUESTION ',0
 firstrnd: db '  FIRST ROUND   ',0
 nxtqtn:   db ' NEXT QUESTION  ',0
 nxtrnd:   db '  NEXT ROUND    ',0
 intime:   db ' DONE IN TIME   ',0
 intime1:  db 'COVERED IN TIME ',0
 timeout:  db '    TIME OUT    ',0
           end
```

AT89C51-DRIVEN DATA DISPLAY

■ A.R. KARKARE

This project shows as to how you can use the Atmel microcontroller AT89C51 to drive an LCD display module and in turn use it as a handheld device to set the parameters of a control unit through RS-232 serial link.

The circuit

Figs 1 and 2 show the circuits of a microcontroller-driven control unit and microcontroller-driven remote handheld device comprising LCD module, respectively. The circuit around IC1 (IC AT89C51) is configured as a control unit, while the circuit around IC2 (another IC AT89C51) is configured as the LCD driver unit. The two units are connected via an RS-232 serial link. The combination of an 8.2k resistor (R) and a 10µF capacitor (C) provides hardware power-on-reset to IC1 and IC2 at their pin 9. A 11.059MHz crystal is connected between pins 18 and 19 of microcontrollers IC1 and IC2 each to generate the required clock and baud rate of 9600.

Eight LEDs are connected to pins 39 (P0.0) through 32 (P0.7) of IC1, so we can see the status of each pin of port 0. Txd (pin 11) and Rxd (pin 10) are used to transmit and receive serial data through IC MAX232. IC3 and IC4 (MAX232) serve the purpose of linking the microcontrollers. Pin 14 (T1 OUT) of IC3 is connected to pin 13 (R1 IN) of IC4 and vice versa. The control unit contains the program 'contr.asm' to send and receive data to the handheld device (LCD module).

IC2 contains the program 'module.asm' to drive the LCD. A 16-character x 4-row LCD display is used to display the day-month-year. The LCD module is interfaced through 8-bit data bus of IC2 on its port 2 (pins 21 through 28). These pins are pulled high through the 10k resistor network. Internal registers of the LCD module are selected by pin 1 (P1.0) of IC2. The Write and Chip-Enable signals of LCD module are connected to pins 2 (P1.1) and 3 (P1.2) of IC2, respectively.

Backlight current (intensity) is controlled through series resistor R12 at pin 16 of the LCD module. The contrast and viewing angle are controlled through preset VR1 at pin 3 of the LCD module.

Four pins of port 1 (pins 4 through 7) are used to sense which key has been pressed. The keys are Esc, Ok, Up, and Down. Usually, pins 4 through 7 are held high through 4.7k resistors, but any of the pins can be pulled

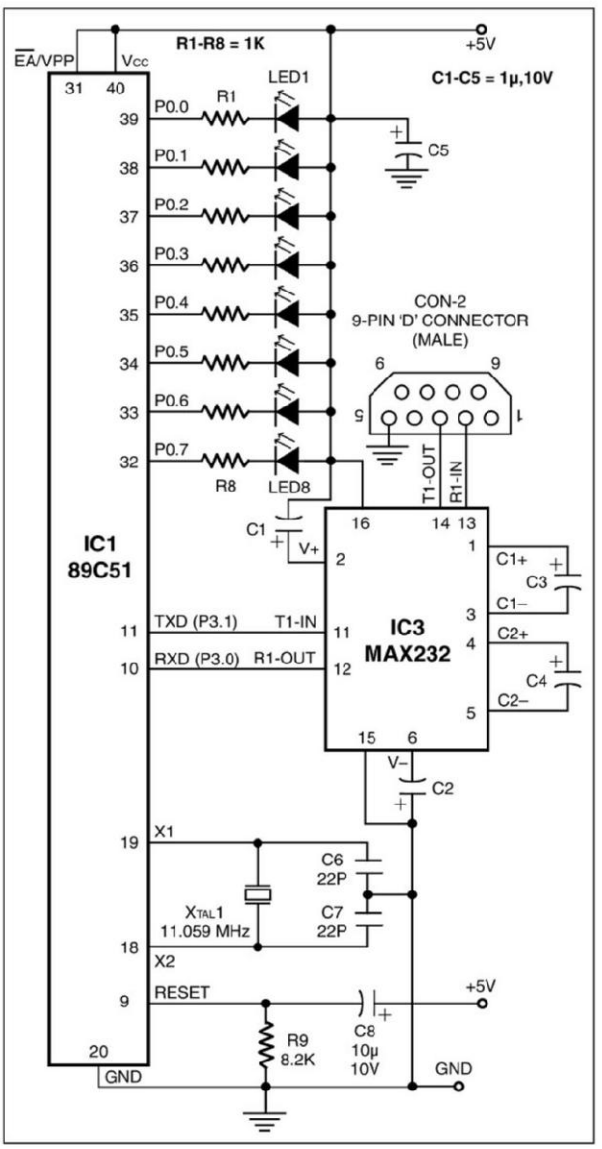


Fig. 1: Control unit

down using the corresponding switches S1 through S4.

RS-232 link between the two circuits serves the purpose of transferring serial data from one microcontroller to the other.

Functioning

It is assumed that the control unit has some basic data, say, someone's birthday, stored in it. The day, month, and the year data are stored at 30H, 31H, and 32H RAM locations, respectively.

When the remote handheld device (LCD module) is connected to the control unit through RS-232 link (IC MAX232), IC2 is reset to start functioning. The data stored in the control unit is displayed on the LCD screen. The user can then select the data (day, month or year). To change the data, increment or decrement it using Up or Down key, and then transfer the data back to the control unit.

Software

In the beginning section of the assembly file (refer Fig. 3 and the module.lst file), RAM locations are reserved for saving various variables such as the day's units and tens digits.

One location (45H) has been defined for sensing the flag to find whether serial port has been interrupted or not. Port pins connected to pins 4 through 6 of the LCD module are defined as 'rs', 'rw', and 'en'. Keys Esc, Ok, Up, and Down are defined as Port 1, which are connected to pins 4 through 7 of IC2, respectively.

The main program starts at location 0000H, while a jump instruction has been set at location 0023H for the serial port interrupt service routine (ISR). Whenever the serial port is interrupted, the program is automatically branched to location 0023H.

Start. The main program starts at location 0030H. Initially the stack pointer is initialised to some safe location where it will not get disturbed by normal routines of the program. Timer 1 is set as a NOT-gated timer for 8-bit auto-reload function mode. The reload value of timer 1 is set for generating a baud rate of 9600 bits per second. The SCON register is set for Mode 1 operation and is kept ready for reception.

Start timer 1 and set the required interrupt request bits as enabled. The interrupt flag is kept cleared to start. Now proceed as per the flowchart shown in Fig. 3.

A few steps after the 'clr intflg' instructions and before step1 are for initialising the LCD module.

Step 1. Screen 1, screen 2, etc to be displayed on the LCD module are predefined as scr1, scr2, etc at respective locations. As the program enters step 1, it first sets the data pointer to point at the first screen to be displayed. The setup subroutine displays the screen. The first screen displayed is a welcome massage. The program waits for the user to press Ok key to come out from the welcome screen display. When the user presses Ok key, the program control passes to Step 2.

Step 2. The program now displays the birthday screen, indicating day, month, and year. A small arrow pointer (>) indicator gets added at LCD location C0H, so the arrow points at 'day', indicating that the parameter 'day' is being selected.

The first character of each line on the LCD module has a unique address: The first character of first, second, third, and fourth lines has address as 80H, C0H, 90H, and D0H, respectively.

As the program executes the add_day, add_month, and add_year subroutines, the day, month, and year data is retrieved from the master IC 89C51 (IC1), converted into proper ASCII format, and saved at LCD locations.

PARTS LIST

Semiconductors:

IC1, IC2	- AT89C51 microcontroller
IC3, IC4	- MAX232, RS-232 level converter
LCD module	- 16-character×4-line type
LED1-LED8	- Red LED

Resistors (all ¼-watt, ±5% carbon, unless stated otherwise):

R1-R8	- 1-kilo-ohm
R9, R10	- 8.2-kilo-ohm
R11-R14	- 4.7-kilo-ohm
R15	- 5-kilo-ohm
R16	- 18-ohm
RNW1	- 10-kilo-ohm×8 *SIL9
resistor network	
RNW2	- 4.7-kilo-ohm×4 SIL5
resistornetwork	
VR1	- 5-kilo-ohm preset

(***Note.*** *Serial-in-line 9-pin resistor, wherepin 1 is a common pin.)

Capacitors:

C1-C5, C9-C13	- 1µF, 10V electrolytic
C6, C7, C14, C15	- 22pF ceramic disk
C8, C16	- 10µF, 10V electrolytic

Miscellaneous:

X_{TAL1}, X_{TAL2}	- 11.059MHz
S1-S4	- Push-to-on tactile switch
Con-1	- 9-pin 'D' female connector
Con-2	- 9-pin 'D' male connector

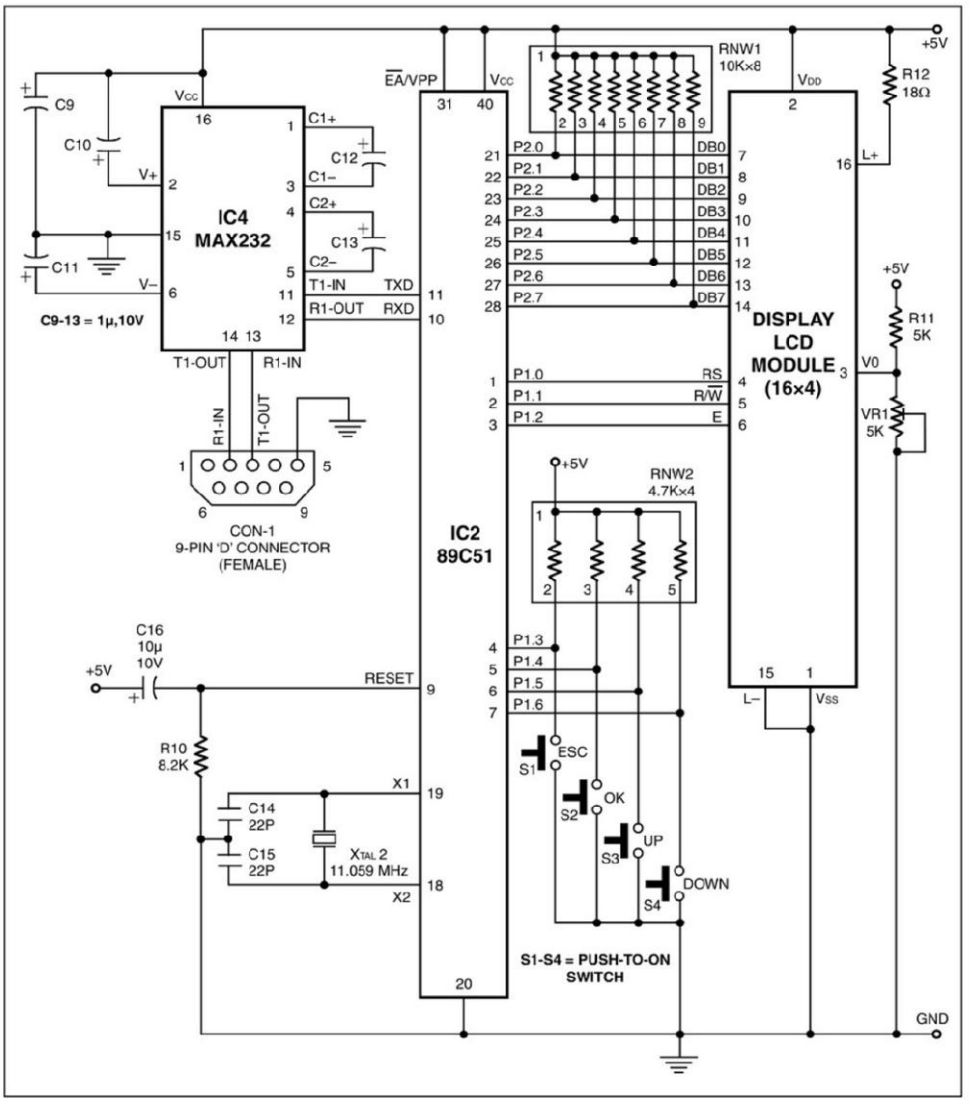


Fig. 2: Handheld unit comprising LCD module

The display now shows the day, month, and year also on the LCD screen.

If the user wishes to select month or year, he needs to press Down key and shift the arrow pointer to the required selection place. On pressing Down key, the arrow pointer shifts down.

Similarly, on pressing Up key, the arrow pointer shifts up. This way the user can select the parameter he wishes to change. In case no parameter is to be selected, by simply pressing Esc key, the user can go back to Step 1, which is the welcome screen. Once the user has selected the parameter, pressing Ok key takes the program to the next step.

Step 3. Here the screen displays all the birthday characters, except the arrow has been shifted to indicate month.

Step 4. Here also the screen displays all the birthday characters, except the arrow has been shifted to indicate year.

Step 5. Depending upon the user's selection of day or month or year, the program branches to Step 5 or Step 6 or Step 7, where the screen displays 'set day' or 'set month' or 'set year', respectively.

On screen 5, the LCD displays 'set day'. The day then gets added on the screen. At key5 label, the program checks, which key is pressed. As long as no key is pressed, the program keeps looping back to key5 label.

When the user presses Up key, the parameter increments, as the 'advance day' and'display day' subroutines are called in. Similarly, by pressing Down key, the parameter decrements.

During the 'advance day' subroutine, the program first checks whether the day is already 31. If so, it resets the day to 01, and doesn't allow it to increment to 32. Similarly, the month doesn't go beyond 12 and the year doesn't go beyond 99. However, if the user is decrementing the day parameter, the program first checks whether the day is already 01. If so, it resets the day to 31, the month to 12, and the year to 99.

Whenever the desired value of the day is seen on the screen, pressing Ok key takes the program to transfer the day data to the master IC 89C51 (IC1). The trfr_day subroutine transfers the value to the appropriate RAM location in the control unit and returns to the step2 screen.

Steps 6 and 7 are similar to Step 5.

As soon as the control unit of IC1 sends some data to the serial port, the serial interrupt at location 0023H gets activated and the program control is passed to the serial port by the spint ISR (serial port interrupt program).

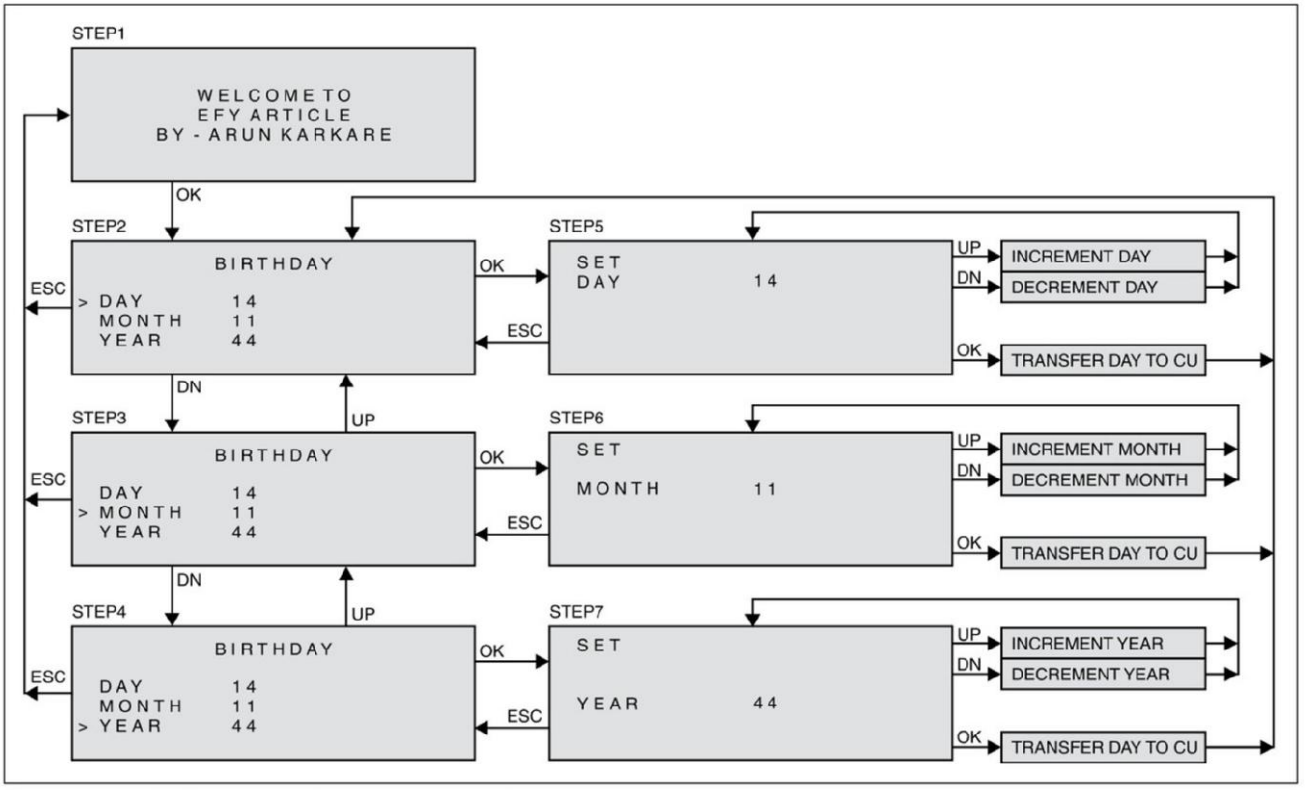


Fig. 3: Flowchart of microcontroller-driven data display

spint subroutine. First, all the interrupts are disabled, since we do not want any interrupt while serving this subroutine. Pushing the program status word (psw) on the stack saves any useful information on the psw and accumulator. The sbuf register is then read and the same is stored at register B. 'ri' bit is then cleared for receiving the next character; flag is set to indicate the interrupt had occurred, and finally the program returns from the subroutine.

send subroutine. The program first disables all the interrupts and clears the transmission completion flag. Then it loads the buffer register to start the transmission from IC2 to the control unit (IC1). As long as 'ti' bit remains low, we need to wait. When the transmission is over, 'ti' bit goes high. The program then enables the interrupt and returns to the main control.

setup subroutine. The program first sets the address pointer (register r2) to the first-line, first-column position (80H) of the LCD. It writes this address to the LCD using the wi subroutine. The program then gets the character from the screen data library and writes data to the LCD using the wd subroutine. The setup subroutine displays the character on the LCD screen.

Both the data pointer and the address pointer (register r2) are then incremented. The program checks whether the first line of LCD has been written. If so, it modifies the address pointer to the second line, which is C0H. Similarly, when the second line is over, the third-line, first-character address is set, and then fourth-line, first-character address is set as address pointer.

wi subroutine. This subroutine is used for transferring control instructions to the LCD. It first sets up the LCD for writing instructions (rw=0, en=0, rs=0) and then moves the data to Port 2 (P2.0 through P2.7) from the accumulator. It then reads the busy bit at the rdbusy subroutine and waits until the writing process is completed, and finally returns to the main program.

wd subroutine. This subroutine is used for transferring data to the LCD. It first sets the LCD for writing data (rw=0, en=0, rs=1) and moves data to Port 2 from the accumulator. It then reads the busy bit by the rdbusy subroutine and waits until the writing process is completed, and finally returns to the main program.

rdbusy subroutine. This subroutine is used for testing the busy bit during the writing operation to the LCD. It first selects the read set-up for the LCD (rw=1, en=0, rs=0). Then it sets Port 2-bit 7 (P2.7) and waits until this

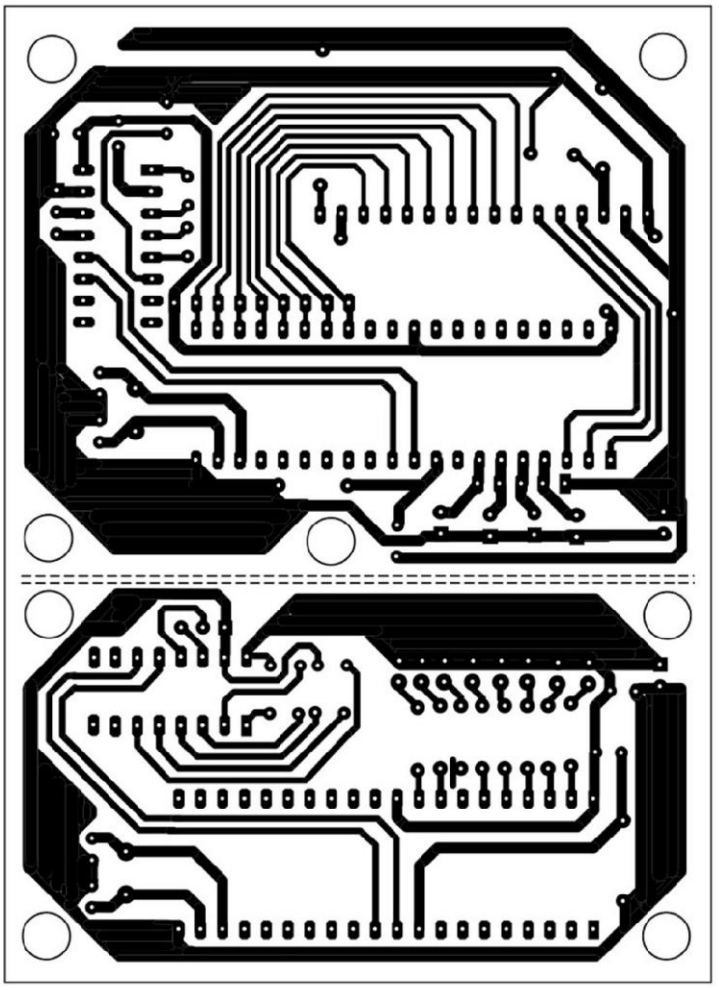


Fig. 4: Actual-size, single-side PCB layout for the handheld unit comprising LCD module (above) and the control unit (below)

bit becomes low after successfully writing to the LCD. Finally, it returns to the main program.

del1m to del100m subroutines. These are just time delay subroutines.

add_day subroutine. The accumulator is set as a pointer for the control unit where the day information is stored. The send_con subrou-tine gets the data from the address pointer of the control unit. This data is now directly available in the two-digit ASCII format for the tens and units digits of day. The tens and units digits of the day are stored and then displayed at LCD locations C7H and C8H, resp-ectively. The add_month and add_year subroutines are similar to the add_day subroutine.

keyprs? subroutine. This subroutine checks which key (Esc, Ok, Up, or Down) has been pressed. If no key is pressed, the subroutine returns with the accumulator containing FFH. Key switches are connected to Port 1 (P1.3 through P1.6). Pins P1.3 through P1.6 usually remain high until a key is pressed.

If any key is sensed low, the program jumps to confirm whether it was an unintentional low or it really happ-ened by keypress. For confirming so, the program waits for the bounce period of 10 milliseconds and then checks for the low again on the same key.

If the key is not sensed low now, it is assumed to be an accidental low and the subroutine returns as if no key was pressed.

But if the key is sensed low for the second time also, the program accepts the key and waits for the user to release the key in about 300 miliseconds. After 300 miliseconds, even if the user does not release the key, the program repeats the action as if the key is being pressed again and again. The program control returns with a code in the accumulator.

Codes for the keys are:

- '01' for pressing Esc key
- '02' for wpressing Ok key
- '03' for pressing Up key
- '04' for pressing Down key

trfr_day subroutine. This subroutine transfers the day data to the appropriate location in the control unit. When this subroutine is called, the data is available as two digits (tens and units) in the ASCII format. As the data needs to be stored at one RAM location in the hex format, the program has to convert the two ASCII digits into a single hex digit by the asci_hex subroutine. At the end of the asci_hex subroutine, an equivalent hex number is available as hex variable.

The program now starts sending the characters. First, start code 02H is sent to the control unit, signaling it to get ready as the data is coming. Second, the address 30H is sent, where the day data is to be stored. Finally, the hex variable is sent, which is the current day data. The trfr_month and trfr_year are similar subroutines.

Only the address where the data is to be stored is different in each case.

hex_asci subroutine. First the units and tens digits are reset to ASCII zero. Then check whether the hex number is already zero. If yes, simply return. Else, advance the units. If the units digit has crossed ASCII 9, we need to reset

the units digit to zero and advance the tens digit. Simultaneously, the hex number has to be decremented. The process keeps repeating until hex number becomes zero. The accumulated tens and units are equivalent to the hex number originally loaded.

asci_hex subroutine. Here the process is almost opposite to what we did while converting hex into ASCII. First, the hex number is reset to zero. Then we check whether both the units and tens digits are zero. If so, we simply return. Otherwise, we have to advance the hex number. Simultaneously, the units and tens digits are to be decremented. The process keeps repeating itself until units and tens digits become zero.

adv_day subroutine. This subroutine advances the day data, but ensures that it does not go beyond 31. The first part checks whether the day's units digit is 1 (decimal) and tens digit is 3 (decimal). If so, the program sets the units digit to 1 and the tens digit to 0 before returning.

The second part of the subroutine advances the day's units digit until it crosses 9 (ASCII 39). After 9, the day's units digit is reset to 0 and the tens digit is advanced. Similarly, if the tens digit crosses 9, the program sets it to 0.

dec_day subroutine. This subrou-tine decrements the day value. The first part checks whether the day's units digit is 1 (decimal) and the tens digit is 0 (decimal). If so, the program sets the tens digit to 3 before returning.

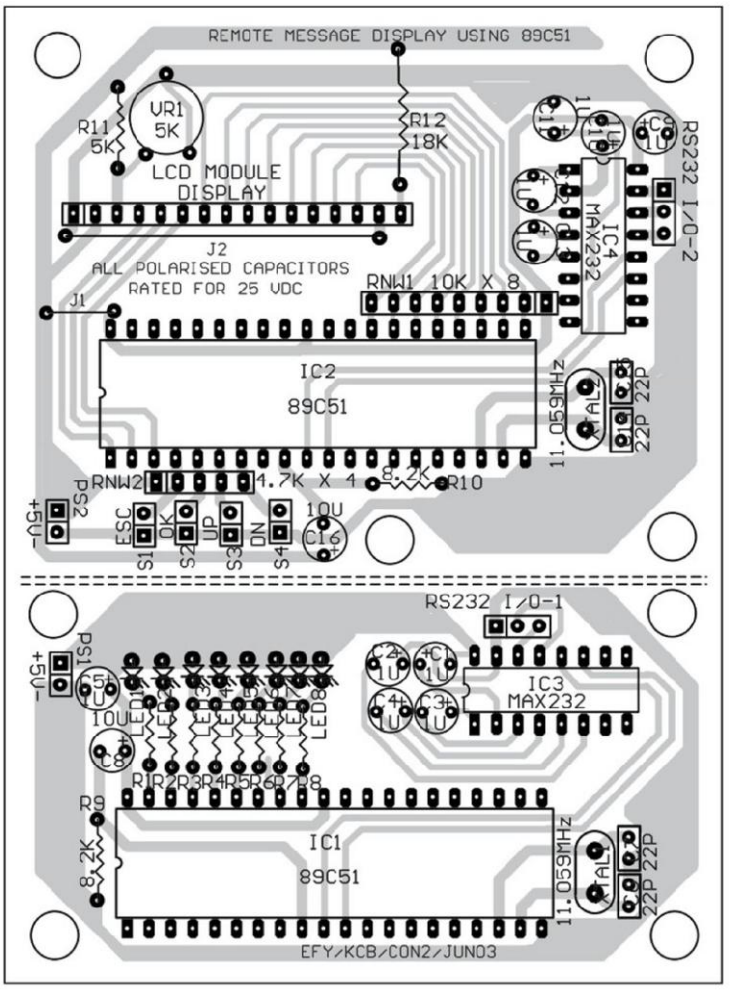


Fig. 5: Component layout for the PCB shown in Fig. 4

In the second part, as the day's units digit is decremented, the program tests whether it has gone below zero. (When ASCII 30h decrements, it will become ASCII 2fh.) The program sets the units digit to 9 (ASCII 39h) and decrements the tens digit. As the tens digit is decremented, the program tests whether it has gone below zero. If so, the program sets the tens digit to 9.

The adv_month and adv_year subroutines are similar to the adv_day subroutine, and the dec_month and dec_year subroutines are similar to the dec_day subroutine.

send_con subroutine. This subroutine first sends the address to the control unit and waits for the interrupt flag to go high. This means the data from the control unit is to be received at the specified address. After receiving the data, the interrupt flag gets cleared for the next instruction. The data is received from register B, saved as the hex variable, and converted into the ASCII code that is required for the LCD module.

The actual-size, single-side PCB layout for the handheld unit comprising LCD module and control unit is shown in Fig. 4 and its component layout in Fig. 5.

The combined PCB can be cut along the dotted lines to separate the control unit and handheld unit comprising LCD module.

Download Source Code: http://efymag.com/Microcontroller_Driven_Data_Display-July03.zip

LCD DRIVER UNIT (MODULE.LST)

```
PAGE 1
1       $mod51
2       ;program for 'LCD' module
3       ;article written for EFY
4       ;programmer-AR Karkare
5       ;Date last modified- 5th May 2003
6       ;uses 89c51 micro-controller with 11.059 mhz crystal
7       ;CU means Control Unit
8       ;modifications included.
9       ;1.incrementing/decrementing of day/month/year in continuous step
        (not in single step).
10      ;2.while advancing, limiting day/month/year to 30,12 and 99. w
11      ;3.while decrementing, resetting the day to 30,month to 12,year
        to 99 after 01.
12      ;file name - module.asm
13      ;RESERVED LOCATIONS
0030        14      day_t   equ     30h         ;day tens
0031        15      day_u   equ     31h         ;day units
0032        16      mon_t   equ     32h         ;month tens
0033        17      mon_u   equ     33h         ;month units
0034        18      yer_t   equ     34h         ;year tens
0035        19      yer_u   equ     35h         ;year units
0036        20      units   equ     36h         ;temp storage of units
0037        21      tens    equ     37h         ;temp storage of tens
0038        22      hex     equ     38h         ;temp storage of hex byte
            23                                  ;LIST OF I/O
0045        24      intflg  bit     45h         ;interrupt flag
            25                                  ;FOR LCD MODULE
0090        26      rs      equ     p1.0        ;0=instructions,1=data
0091        27      rw      equ     p1.1        ;0=write,1=read
0092        28      en      equ     p1.2        ;0=disabled,1=enabled
            29                                  ;p2 port for data from cpu to
                                                LCD module
            30                                  ;FOR KEYBOARD
0093        31      esc     equ     p1.3        ;Escape key
0094        32      ok      equ     p1.4        ;Ok key
0095        33      up      equ     p1.5        ;Plus key
0096        34      dn      equ     p1.6        ;Minus key
            35                                  ;FOR RS232 COMMUNICATION
0000        36      org     0000h               ;
0000 802E   37      sjmp    start               ;jump to main program
0023        38      org     0023h               ;
0023 21A0   39      ajmp    spint               ;jump to serial port int.program
0030        40      org     0030h               ;initialization of registers
0030 758160 41      start:  mov     sp, #60h    ;set stack pointer
            42                                  ;initialise SFRs
0033 758700 43      mov     pcon, #00h          ;smod=0
0036 758920 44      mov     tmod, #20h          ;timer1(gate=0,c/t=0,mode=8 bit
                                                auto reload)
0039 758BFD 45      mov     tl1, #0fdh          ;reload value for 9.6k baud rate
003C 758DFD 46      mov     th1, #0fdh          ;
003F 759850 47      mov     scon, #50h          ;mode 1,reception enabled
0042 D28E   48      setb    tr1                 ;start timer
0044 D2AF   49      setb    ea                  ;global int. on
0046 D2AC   50      setb    es                  ;serial int. on
0048 D2BC   51      setb    ip.4                ;high priority to serial port
int.
004A D2B0   52      setb    rxd                 ;float pin
004C D2B1   53      setb    txd                 ;float pin
004E C245   54      clr     intflg              ;clear interrupt flag
            55                                  ;initialise LCD module
0050 120245 56      lcall   del10m              ;power on delay
0053 120245 57      lcall   del10m              ;
0056 7430   58      mov     a, #30h             ;select in write mode-three times
0058 1201F5 59      lcall   wi                  ;write instruction
005B 120238 60      lcall   del4m               ;
005E 7430   61      mov     a, #30h             ;select in write mode-three times
0060 201F5  62      lcall   wi                  ;write instruction
0063 120238 63      lcall   del4m               ;
0066 7430   64      mov     a, #30h             ;select in write mode-three times
0068 1201F5 65      lcall   wi                  ;write instruction
006B 7438   66      mov     a, #38h             ;select 8-bit operation,
            67                                  ;number of lines 2,font 5x7 dots
006D 1201F5 68      lcall   wi                  ;
0070 740C   69      mov     a, #00001100b       ;display on,cursor off,blinking off
0072 1201F5 70      lcall   wi                  ;
0075 7401   71      mov     a, #01h             ;clear display,cursor to home
                                                position
0077 1201F5 72      lcall   wi                  ;
007A 7406   73      mov     a, #06h             ;set incrementing dd ram, no
                                                shift mode
007C 1201F5 74      lcall   wi                  ;
007F 9004C5 75      step1:  mov     dptr, #scr1 ;set pointer
0082 1201C3 76      lcall   setup               ;display "Welcome to EFY Article"
0085 1202C5 77      key1:   lcall   keyprs?     ;key pressed?
0088 B402FA 78      cjne    a, #02h,key1        ;keep waiting till 'ok' key pressed
008B 8000   79      sjmp    step2               ;if ok key-go to step 2
008D 900505 80      step2 : mov     dptr, #scr2 ;set pointer
0090 1201C3 81      lcall   setup               ;display "Birthday/day/month/y
                                                year"
0093 74C0   82      mov     a, #0c0h            ;set address line2/col1
0095 1201F5 83      lcall   wi                  ;write address on LCD module
0098 743E   84      mov     a, #3eh             ;set arrow pointer
009A 120208 85      lcall   wd                  ;write data on LCD module
009D 12025C 86      lcall   add_day             ;add day
00A0 12027F 87      lcall   add_month           ;add month
00A3 1202A2 88      lcall   add_year            ;add year
00A6 1202C5 89      key2:   lcall   keyprs?     ;key pressed?
00A9 B40102 90      cjne    a, #01h,_022?       ;
00AC 80D1   91      sjmp    step1               ;if escape key-go back to step 1
00AE B40202 92      _022?:  cjne    a, #02h,_042? ;
00B1 8060   93      sjmp    step5               ;if ok key-go to step 5
00B3 B404F0 94      _042?:  cjne    a, #04h,key2;if none-go back to key2
00B6 8000   95      sjmp    step3               ;if down key-go to step 3
00B8 900505 96      step3:  mov     dptr, #scr2 ;set pointer
00BB 1201C3 97      lcall   setup               ;display "Birthday/day/month/
                                                year"
00BE 7490   98      mov     a, #90h             ;set address line3/col1
00C0 1201F5 99      lcall   wi                  ;
00C3 743E   100     mov     a, #3eh             ;set arrow pointer
00C5 120208 101     lcall   wd                  ;
00C8 12025C 102     lcall   add_day             ;add day
00CB 12027F 103     lcall   add_month           ;add month
00CE 1202A2 104     lcall   add_year            ;add year
00D1 1202C5 105     key3:   lcall keyprs?       ;key pressed?
00D4 B40102 106     cjne    a, #01h,_023?       ;
00D7 80A6   107     sjmp    step1               ;if escape key-go back to step 1
00D9 B40202 108     _023?:  cjne    a, #02h,_033? ;
00DC 8064   109     sjmp    step6               ;if ok key-go to step6
00DE B40302 110     _033?:  cjne    a, #03h,_043?;
00E1 80AA   111     sjmp    step2               ;if up key-go to step2
00E3 B404EB 112     _043?:  cjne    a, #04h,key3 ;if none-go back to key3
00E6 8000   113     sjmp    step4               ;if down key-go back to step4
00E8 900505 114     step4:  mov     dptr, #scr2 ;set pointer
00EB 1201C3 115     lcall   setup               ;display "Birthday/day/month/
                                                year"
00EE 74D0   116     mov     a, #0d0h            ;add arrow pointer
00F0 1201F5 117     lcall   wi                  ;
00F3 743E   118     mov     a, #3eh             ;
00F5 120208 119     lcall   wd                  ;
00F8 12025C 120     lcall   add_day             ;add day
00FB 12027F 121     lcall   add_month           ;add month
00FE 1202A2 122     lcall   add_year            ;add year
0101 1202C5 123     key4:   lcall   keyprs?     ;key pressed?
0104 B40102 124     cjne    a, #01h,_024?       ;
0107 017F   125     ajmp    step1               ;if escape key-go back to step 1
0109 B40202 126     _024?:  cjne    a, #02h,_034? ;
010C 8063   127     sjmp    step7               ;if ok key-go to step7
010E B403F0 128     _034?:  cjne    a, #03h,key4 ;if none-go back to key4
0111 80A5   129     sjmp    step3               ;if up key-go back to step3
0113 900545 130     step5:  mov     dptr, #scr5 ;set pointer
0116 1201C3 131     lcall   setup               ;display "set/day"
0119 12025C 132     lcall   add_day   ;add day
011C 1202C5 133     key5:   lcall   keyprs?     ;key pressed?
011F B40102 134     cjne    a, #01h,_025?       ;
0122 018D   135     ajmp    step2               ;if escape key-go to step2
0124 B40205 136     _025?:  cjne    a, #02h,_035? ;
0127 12032C 137     lcall   trfr_day            ;if ok key-transfer day value
```

```
012A 018D   138    ajmp     step2                ;back to display
012C B40308 139    _035?:   cjne      a, #03h,_045? ;
012F 1203D7 140    lcall    adv_day              ;if up key-advance day value
0132 12026A 141    lcall    disp_day             ;display new day
0135 80E5   142    sjmp     key5                 ;back
0137 B404E2 143    _045?:   cjne      a, #04h,key5 ;none-back to key5
013A 1203FB 144    lcall    dec_day              ;if down key-decrement day value
013D 12026A 145    lcall    disp_day             ;display new day
0140 80DA   146    sjmp     key5                 ;back
0142 900585 147    step6:   mov       dptr, #scr6 ;set pointer
0145 1201C3 148    lcall    setup                ;display "set/month"
0148 12027F 149    lcall    add_month ;add month
014B 1202C5 150    key6:    lcall keyprs?        ;key pressed?
014E B40102 151    cjne     a, #01h,_026?        ;
0151 01B8   152    ajmp     step3                ;if escape key-go to step3
0153 B40205 153    _026?:   cjne      a, #02h,_036? ;
0156 120348 154    lcall    trfr_month           ;if ok key-transfer month value
0159 018D   155    ajmp     step2                ;back to display
015B B40308 156    _036?:   cjne      a, #03h,_046? ;
015E12041F  157    lcall    adv_month            ;if up key-advance month value
0161 12028D 158    lcall    disp_mnth            ;display new month
0164 80E5   159    sjmp     key6                 ;back
0166 B404E2 160    _046?:   cjne      a, #04h,key6 ;none-back to key6
0169 120449 161    lcall    dec_month            ;if down key-decrement month
                                                 value
016C 12028D 162    lcall    disp_mnth            ;display new month
016F 80DA   163    sjmp     key6                 ;back
0171 9005C5 164    step7:movdptr,#scr7           ;set pointer
0174 1201C3 165    lcall    setup                ;display "set/year"
0177 1202A2 166    lcall    add_year             ;add year
017A 1202C5 167    key7:    lcall     keyprs?    ;key pressed?
017D B40102 168    cjne     a, #01h,_027?        ;
0180 01E8   169    ajmp     step4                ;if escape key-go to step4
0182 B40205 170    _027?:   cjne      a, #02h,_037? ;
0185 120364 171    lcall    trfr_year            ;if ok key-transfer year value
0188 018D   172    ajmp     step2                ;back to display
018A B40308 173    _037?:   cjne      a, #03h,_047? ;
018D 12046D 174    lcall    adv_year             ;if up key-advance year value
0190 1202B0 175    lcall    disp_year            ;display new year
0193 80E5   176    sjmp     key7                 ;back
0195 B404E2 177    _047?:   cjne      a, #04h,key7 ;none-back to key7
0198 120491 178    lcall    dec_year             ;if down key-decrement year value
019B 1202B0 179    lcall    disp_year            ;display new year
019E 80DA   180    sjmp     key7                 ;back
            181                                  ;SUBROUTINES
01A0 C2AF   182    spint:   clr       ea         ;disable all interrupts
01A2 C0D0   183    push     psw                  ;
01A4 C0E0   184    push     acc                  ;
01A6 C3     185    clr      c                    ;
01A7 8599F0 186    mov      b, sbuf              ;save in reg. B
01AA C298   187    out:     clr       ri         ;
01AC D0E0   188    pop      acc                  ;
01AE D0D0   189    pop      psw                  ;
01B0 D2AF   190    setb     ea                   ;enable all interrupts
01B2 D245   191    setb     intflg               ;set interrupt flag
01B4 32     192    reti                          ;return from interrupt
01B5 C2AF   193    send:    clr       ea         ;disable all interrupts
01B7 C299   194    clr      ti                   ;pull ti flag low
01B9 F599   195    mov      sbuf, a              ;load sbuf
01BB 3099FD 196    waitt:   jnb       ti, waitt  ;wait till ti flag goes high
01BE C299   197    clr      ti                   ;pull ti flag low
01C0 D2AF   198    setb     ea                   ;enable all interrupts
01C2 22     199    ret                           ;return
01C3 7A80   200    setup:   mov       r2, #80h   ;line 1-column 0 position
01C5 EA     201    mov      a, r2                ;
01C6 1201F5 202    lcall    wi                   ;write instruction
01C9        203    display:
01C9 E4     204    clr      a                    ;
01CA 93     205    movc     a, @a+dptr;
01CB 120208 206    lcall    wd                   ;write data
01CE A3     207    inc      dptr                 ;
01CF 0A     208    inc      r2                   ;
01D0 BA9008 209    cjne     r2, #090h,_2ndln     ;
01D3 7AC0   210    mov      r2, #0c0h ;
01D5 EA     211    mov      a, r2                ;
01D6 1201F5 212    lcall    wi                   ;
01D9 80EE   213    sjmp     display   ;

01DB BAD008 214    _2ndln:  cjne      r2, #0d0h,_3rdln ;
01DE 7A90   215    mov      r2, #090h            ;
01E0 EA     216    mov      a, r2                ;
01E1 1201F5 217    lcall    wi                   ;
01E4 80E3   218    sjmp     display   ;
01E6 BAA008 219    _3rdln:  cjner2, #0a0h,_4thln ;
01E9 7AD0   220    mov      r2, #0d0h ;
01EB EA     221    mov      a, r2                ;
01EC 1201F5 222    lcall    wi                   ;
01EF 80D8   223    sjmp     display   ;
01F1 BAE0D5 224    _4thln:  cjne      r2, #0e0h,display ;
01F4 22     225    ret                           ;
01F5 C291   226    wi:      clr       rw         ;select write
01F7 C292   227    clr      en                   ;transfer disabled
01F9 C290   228    clr      rs                   ;select instruction
01FB F5A0   229    mov      p2, a                ;set data to port
01FD D292   230    setb     en                   ;transfer disabled
01FF 12022B 231    lcall    del1m                ;
0202 C292   232    clr      en                   ;
0204 12021B 233    lcall    rdbusy               ;test busy flag
0207 22     234    ret                           ;return
0208 C291   235    wd:      clr       rw         ;select write
020A C292   236    clr      en                   ;transfer disabled
020C D290   237    setb     rs                   ;select data
020E F5A0   238    mov      p2, a                ;set data to port
0210 D292   239    setb     en                   ;
0212 12022B 240    lcall    del1m                ;
0215 C292   241    clr      en                   ;transfer disabled
0217 12021B 242    lcall    rdbusy               ;
021A 22     243    ret                           ;return
021B C292   244    rdbusy:  clr       en;
021D C290   245    clr      rs                   ;
021F D291   246    setb     rw                   ;
0221 D2A7   247    setb     p2.7                 ;
0223 D292   248    setb     en                   ;
0225 20A7FD 249    wt:      jb        p2.7, wt   ;
0228 C292   250    clr      en                   ;
022A 22     251    ret                           ;
022B 755004 252    del1m:   mov       50h, #04h  ;delay of 1 milisec.
022E 755153 253    loopa:   mov       51h ,#53h  ;
0231 D551FD 254    loopb:   djnz      51h, loopb ;
0234 D550F7 255    djnz     50h, loopa;
0237 22     256    ret                           ;
0238 12022B 257    del4m:   lcall     del1m      ;delay of 4 milisec.
023B 12022B 258lcall        del1m                ;
023E 12022B 259    lcall    del1m                ;
0241 12022B 260    lcall    del1m                ;
0244 22     261    ret                           ;
0245 755028 262    del10m:  mov       50h, #28h  ;delay of 10 milisec.
0248 755153 263    loopc:   mov       51h, #53h  ;
024B D551FD 264    loopd:   djnz      51h, loopd ;
024E D550F7 265    djnz     50h, loopc           ;
0251 22     266    ret                           ;
0252        267    del100m:
0252 75560A 268    mov      56h, #0ah            ;delay of 100 milisec.
0255 120245 269    looph:   lcall     del10m     ;
0258 D556FA 270    djnz56h, looph     ;
025B 22     271    ret                           ;
025C        272    add_day:
025C 120245 273    lcall    del10m               ;wait for stabilisation
025F 7430   274    mov      a, #30h              ;Day value is stored at 30h in CU
0261 1204B5 275    lcall    send_con             ;send and convert
0264 853730 276    movday_t,tens                 ;save new tens
0267 853631 277    mov      day_u,units          ;save new units
026A        278    disp_day:
026A 74C7   279    mova, #0c7h                   ;set address in LCD module(line2/
                                                 col8)
026C 1201F5 280    lcall    wi                   ;
026F E537   281    mov      a, tens              ;print tens
0271 120208 282    lcall    wd                   ;
0274 74C8   283    mov      a, #0c8h             ;set address in LCD module
0276 1201F5 284    lcall    wi                   ;
0279 E536   285    mov      a, units             ;print units
027B 120208 286    lcall    wd                   ;
027E 22     287    ret                           ;
027F        288    add_month:
027F 120245 289    lcall    del10m               ;wait for stabilisation
```

```
0282 7431   290   mov     a, #31h             ;month value is stored at
                                              31h in CU
0284 1204B5 291   lcall   send_con            ;send and convert
0287 853732 292   mov     mon_t, tens         ;save new tens
028A 853633 293   mov     mon_u, units        ;save new units
028D        294   disp_mnth:
028D 7497   295   mov     a, #97h             ;set address in LCD module
                                              (line3/col8)
028F 1201F5 296   lcall   wi                  ;
0292 E537   297   mov     a, tens             ;print tens
0294 120208 298   lcall   wd                  ;
0297 7498   299   mov     a, #98h             ;set address in LCD module
0299 1201F5 300   lcall   wi                  ;
029C E536   301   mov     a, units            ;print units
029E 120208 302   lcall   wd                  ;
02A1 22     303   ret                         ;
02A2        304   add_year:
02A2 120245 305   lcall   del10m             ;wait for stabilisation
02A5 7432   306   mov     a, #32h             ;year value is stored at 32h in
CU
02A7 1204B5 307   lcall   send_con            ;send and convert
02AA 853734 308   mov     yer_t, tens         ;save new tens
02AD 853635 309   mov     yer_u, units        ;save new units
02B0        310   disp_year:
02B0 74D7   311   mov     a, #0d7h            ;set address in LCD module
                                              (line4/col8)
02B2 1201F5 312   lcall   wi                  ;
02B5 E537   313   mov     a, tens             ;print tens
02B7 120208 314   lcall   wd                  ;
02BA 74D8   315   mov     a, #0d8h            ;set address in LCD module
02BC 1201F5 316   lcall   wi                  ;
02BF E536   317   mov     a, units            ;print units
02C1 120208 318   lcall   wd                  ;
02C4 22     319   ret                         ;
02C5        320   keyprs?:
02C5 D293   321   setb    esc                 ;set escape pin high
02C7 D294   322   setb    ok                  ;set ok pin high
02C9 D295   323   setb    up                  ;set up pin high
02CB D296   324   setb    dn                  ;set down pin high
02CD 30930C 325   jnb     esc, con_esc        ;escape key low-confirm
02D0 30941D 326   jnb     ok, con_ok          ;ok key low-confirm
02D3 30952E 327   jnb     up, con_up          ;up key low-confirm
02D6 30963F 328   jnb     dn, con_dn          ;down key low-confirm
02D9 74FF   329   retmt:  mov       a, #0ffh  ;no key-make accu. all high
02DB 22     330   ret                         ;return empty
02DC        331   con_esc:
02DC 120245 332   lcall   del10m             ;debounce delay
02DF 309302 333   jnb     esc, accesc         ;escape key low-accept escape
02E2 80F5   334   sjmp    retmt               ;otherwise return empty
02E4 120252 335   accesc: lcall     del100m   ;wait for 300 milisec
02E7 120252 336   lcall   del100m            ;
02EA 120252 337   lcall   del100m            ;
02ED 7401   338   mov     a, #01h             ;set accu. to 01
02EF 22     339   ret                         ;return
02F0 120245 340   con_ok: lcall     del10m    ;debounce delay
02F3 309402 341   jnb     ok, accok           ;ok key low-accept ok
02F6 80E1   342   sjmp    retmt               ;otherwise return empty
02F8 120252 343   accok:  lcall     del100m   ;wait for 300 milisec
02FB 120252 344   lcall   del100m            ;
02FE 120252 345   lcall   del100m            ;
0301 7402   346   mov     a, #02h             ;set accu. to 02
0303 22     347   ret                         ;return
0304 120245 348   con_up: lcall     del10m    ;debounce delay
0307 309502 349   jnb     up, accup           ;up key low-accept up
030A 80CD   350   sjmp    retmt               ;otherwise return empty
030C 120252 351   accup:  lcall     del100m   ;wait for 300 milisec
030F 120252 352   lcall   del100m            ;
0312 120252 353   lcall   del100m            ;
0315 7403   354   mov     a, #03h             ;set accu. to 03
0317 22     355   ret                         ;return
0318 120245 356   con_dn: lcall     del10m    ;debounce delay
031B 309602 357   jnb     dn, accdn           ;down key low-accept down
031E 80B9   358   sjmp    retmt               ;otherwise return empty
0320 120252 359   accdn:  lcall     del100m   ;wait for 300 milisec
0323 120252 360   lcall   del100m            ;
0326 120252 361   lcall   del100m            ;
0329 7404   362   mov     a, #04h             ;set accu. to 04
032B 22     363   ret                         ;return
032C        364   trfr_day:
032C 1203A8 365   lcall   asci_hex           ;convert ascii characters to hex
032F 120245 366   lcall   del10m             ;wait for stabilisation
0332 7402   367   mov     a, #02h             ;load start character <STX>
0334 1201B5 368   lcall   send                ;send to CU
0337 120245 369   lcall   del10m             ;wait for stabilisation
033A 7430   370   mov     a, #30h             ;address in CU where to store
                                              the data
033C 1201B5 371   lcall   send                ;send address
033F 120245 372   lcall   del10m             ;wait for stabilisation
0342 E538   373   mov     a, hex              ;get day value-hex format
0344 1201B5 374   lcall   send                ;send day value
0347 22     375   ret                         ;return
0348        376   trfr_month:
0348 1203A8 377   lcall   asci_hex           ;convert ascii characters to hex
034B 120245 378   lcall   del10m             ;wait for stabilisation
034E 7402   379   mov     a, #02h             ;load start character <STX>
0350 1201B5 380   lcall   send                ;send to CU
0353 120245 381   lcall   del10m             ;wait for stabilisation
0356 7431   382   mov     a, #31h             ;address in CU where to store
                                              the d data
0358 1201B5 383   lcall   send                ;send address
035B 120245 384   lcall   del10m             ;wait for stabilisation
035E E538   385   mov     a, hex              ;get month value in hex format
0360 1201B5 386   lcall   send                ;send month value
0363 22     387   ret                         ;return
0364        388   trfr_year:
0364 1203A8 389   lcall   asci_hex           ;convert ascii characters to hex
0367 120245 390   lcall   del10m             ;wait for stabilisation
036A 7402   391   mov     a, #02h             ;load start character <STX>
036C 1201B5 392   lcall   send                ;send to CU
036F 120245 393   lcall   del10m             ;wait for stabilisation
0372 7432   394   mov     a, #32h             ;address in CU where to store
                                              the data
0374 1201B5 395   lcall   send                ;send address
0377 120245 396   lcall   del10m             ;wait for stabilisation
037A E538   397   mov     a, hex              ;get year value in hex format
037C 1201B5 398   lcall   send                ;send year value
037F 22     399   ret                         ;return
0380        400   hex_asci:
0380 C3     401   clr     c                   ;clear carry
0381 E4     402   clr     a                   ;clear accu.
0382 753630 403   mov     units, #30h         ;ascii zero in units
0385 753730 404   mov     tens, #30h          ;ascii zero in tens
0388 E538   405   zero?:  mov       a, hex    ;get hex value
038A B40001 406   cjne    a, #00h,adv_units   ;
038D 22     407   ret                         ;if hex value is zero-return
038E        408   adv_units:
038E 0536   409   inc     units               ;advance units
0390 E536   410   mov     a, units            ;get new units
0392 B43A0F 411   cjne    a, #3ah,dcr_hex     ;if units exceeding ascii 39
0395 753630 412   mov     units, #30h         ;set zero in units
0398 0537   413   inc     tens                ;advance tens
039A E537   414   mov     a, tens             ;get new tens
039C B43A05 415   cjne    a, #3ah,dcr_hex     ;if tens exceeding ascii 39
039F 753730 416   mov     tens, #30h          ;set zero in tens
03A2 80FE   417   sjmp    $                   ;this is illegal stage-so hang up
03A4        418   dcr_hex:
03A4 1538   419   dec     hex                 ;decrement hex value
03A6 80E0   420   sjmp    zero?               ;check is it zero yet?
            421                               ;this subroutine converts ascii
                                              characters
            422                               ;to equivalent hex format
03A8        423   asci_hex:
03A8 C3     424   clr     c                   ;clear carry for safety
03A9 E4     425   clr     a                   ;clear accu.
03AA 753800 426   mov     hex, #00h           ;clear destination hex register
            427                               ;remember ascii zero=30h
03AD E536   428   allzro:  mov       a, units  ;gets units
03AF B43006 429   cjne    a, #30h,adv_hex     ;is it zero?
03B2 E537   430   mov     a, tens             ;get tens
03B4 B43001 431   cjne    a, #30h,adv_hex     ;is it zero?
03B7 22     432   ret                         ;all zero-then return
03B8        433   adv_hex:
03B8 0538   434   inc     hex                 ;advance hex count
03BA E538   435   mov     a, hex              ;get hex count
03BC B40002 436   cjne    a, #00h,dcr_asc     ;crossed ff?
03BF 80FE   437   sjmp    $                   ;this is illegal-so hang up
```

```
03C1        438   dcr_asc:
03C1 1536   439   dec     units                ;decrement units
03C3 E536   440   mov     a, units  ;get units
03C5 B42FE5 441   cjne    a, #2fh,allzro       ;crossed zero?
03C8 753639 442   mov     units, #39h          ;if crossed-set units to 9
                                               (ascii39)
03CB 1537   443   dec     tens                 ;decrement tens
03CD E537   444   mov     a, tens              ;get tens
03CF B42FDB 445   cjne    a, #2fh,allzro       ;crossed zero?
03D2 753739 446   mov     tens, #39h           ;set tens to 9-illegal state
03D5 80FE   447   sjmp    $                    ;hang operation
03D7        448   adv_day:
03D7 E531   449   mov     a, day_u             ;get current day units
03D9 B4310C 450   cjne    a, #31h,cont_ad      ;test-is it ascii one?
03DC E530   451   mov     a, day_t             ;yes-get current day tens, no-
                                               continue advnacing day
03DE B43307 452   cjne    a, #33h,cont_ad      ;test-is it ascii three?
03E1 753631 453   mov     units, #31h          ;yes-set units-ascii one
03E4 753730 454   mov     tens, #30h           ;set tens-ascii zero
03E7 22     455   ret                          ;return
03E8        456   cont_ad:
03E8 E531   457   mov     a, day_u             ;continue advancing day
03EA 04     458   inc     a                    ;
03EB B43A55 459   cjne    a, #3ah,save_units   ;
03EE 753630 460   mov     units, #30h          ;
03F1 E530   461   mov     a, day_t             ;
03F3 04     462   inc     a                    ;
03F4 B43A4F 463   cjne    a, #3ah,save_tens    ;
03F7 753730 464   mov     tens, #30h           ;
03FA 22     465   ret                          ;
03FB        466   dec_day:
03FB E531   467   mov     a, day_u             ;get current day units
03FD B4310C 468   cjne    a, #31h,cont_dd      ;test-is it ascii one?
0400 E530   469   mov     a, day_t             ;yes-get current day tens,no-
                                               continue decrementing day
0402 B43007 470   cjne    a, #30h,cont_dd      ;test-is it ascii zero?
0405 753631 471   mov     units, #31h          ;yes-set units-ascii one,
                                               no-continue decrementing day;;;;
0408 753733 472   mov     tens, #33h           ;set tens-ascii three
040B 22     473   ret                          ;return
040C        474   cont_dd:
040C E531   475   mov     a, day_u             ;continue decrementing day
040E 14     476   dec     a                    ;
040F B42F31 477   cjne    a, #2fh,save_units   ;
0412 753639 478   mov     units, #39h          ;
0415 E530   479   mov     a, day_t  ;
0417 14     480   dec     a                    ;
0418 B42F2B 481   cjne    a, #2fh, save_tens   ;
041B 753739 482   mov     tens, #39h           ;
041E 22     483   ret                          ;
041F        484   adv_month:
041F E533   485   mov     a, mon_u             ;get current month units
0421 B4320C 486   cjne    a, #32h, cont_am     ;test-is it ascii two?
0424 E532   487   mov     a, mon_t             ;yes-get current month tens,no-
                                               continue advancing month
0426 B43107 488   cjne    a, #31h, cont_am     ;test-is it ascii one?
0429 753631 489   mov     units, #31h          ;yes-set units-ascii one
042C 753730 490   mov     tens, #30h           ;set tens-ascii zero
042F 22     491   ret                          ;return
0430        492   cont_am:
0430 E533   493   mov     a, mon_u             ;continue advancing month
0432 04     494   inc     a                    ;
0433 B43A0D 495   cjne    a, #3ah, save_units  ;
0436 753630 496   mov     units, #30h          ;
0439 E532   497   mov     a, mon_t             ;
043B 04     498   inc     a                    ;
043C B43A07 499   cjne    a, #3ah,save_tens    ;
043F 753730 500   mov     tens, #30h           ;
0442 22     501   ret                          ;
0443        502   save_units:
0443 F536   503   mov     units, a             ;
0445 22     504   ret                          ;
0446        505   save_tens:
0446 F537   506   mov     tens, a              ;
0448 22     507   ret                          ;
0449        508   dec_month:
0449 E533   509   mov     a, mon_u             ;get current month units
044B B4310C 510   cjne    a, #31h,cont_md      ;test-is it ascii one?
044E E532   511   mov     a, mon_t             ;yes-get current month tens,no-
                                               continue decrementing month
0450 B43007 512   cjne    a, #30h,cont_md      ;test-is it ascii zero?
0453 753632 513   mov     units, #32h          ;yes-set units-ascii two,no-
                                               continue decrementing month
0456 753731 514   mov     tens, #31h           ;set tens-ascii one
0459 22     515   ret                          ;return
045A        516   cont_md:
045A E533   517   mov     a, mon_u             ;continue decrementing month
045C 14     518   dec     a                    ;
045D B42FE3 519   cjne    a, #2fh,save_units   ;
0460 753639 520   mov     units, #39h          ;
0463 E532   521   mov     a, mon_t             ;
0465 14     522   dec     a                    ;
0466 B42FDD 523   cjne    a, #2fh,save_tens    ;
0469 753739 524   mov     tens, #39h           ;
046C 22     525   ret                          ;
046D        526   adv_year:
046D E535   527   mov     a, yer_u             ;get current year units
046F B4390C 528   cjne    a, #39h, cont_ay     ;test-is it ascii nine?
0472 E534   529   mov     a, yer_t             ;yes-get current year tens, no-
                                               continue advancing year
0474 B43907 530   cjne    a, #39h, cont_ay     ;test-is it ascii nine?
0477 753630 531   mov     units, #30h          ;yes-set units-ascii zero,no
                                               continue advancing year
047A 753730 532   mov     tens, #30h           ;set tens-ascii zero
047D 22     533   ret                          ;return
047E        534   cont_ay:
047E E535   535   mov     a, yer_u             ;continue advancing year
0480 04     536   inc     a                    ;
0481 B43ABF 537   cjne    a, #3ah,save_units   ;
0484 753630 538   mov     units, #30h          ;
0487 E534   539   mov     a,y er_t  ;
0489 04     540   inc     a                    ;
048A B43AB9 541   cjne    a, #3ah,save_tens    ;
048D 753730 542   mov     tens, #30h           ;
0490 22     543   ret                          ;
0491        544   dec_year:
0491 E535   545   mov     a, yer_u             ;get current year units
0493 B4300C 546   cjne    a, #30h, cont_yd     ;test-is it ascii one?
0496 E534   547   mov     a, yer_t             ;yes-get current year tens, no-
                                               continue decrementing year
0498 B43007 548   cjne    a, #30h,cont_yd      ;test-is it ascii zero?
049B 753639 549   mov     units, #39h          ;yes-set units-ascii nine, no-
                                               continue decrementing year
049E 753739 550   mov     tens, #39h           ;set tens-ascii nine
04A1 22     551   ret                          ;return
04A2        552   cont_yd:
04A2 E535   553   mov     a, yer_u             ;continue decrementing year
04A4 14     554   dec     a                    ;
04A5 B42F9B 555   cjne    a, #2fh,save_units   ;
04A8 753639 556   mov     units, #39h          ;
04AB E534   557   mov     a, yer_t  ;
04AD 14     558   dec     a                    ;
04AE B42F95 559   cjne    a, #2fh, save_tens   ;
04B1 753739 560   mov     tens, #39h           ;
04B4 22     561   ret                          ;
04B5        562   send_con:
04B5 1201B5 563   lcall   send                 ;send request to CU
04B8 3045FD 564   wait:   jnb       intflg, wait ;wait till serial data comes in
04BB C245   565   clr     intflg               ;clear indicator flag
            566                                ;on serial inturrupt,
            567                                ;the 'spint' subroutine works
            568                                ;data received is stored in
                                               register b
04BD E5F0   569   mov     a, b                 ;get what has been received
04BF F538   570   mov     hex, a               ;save hex value
            571                                ;LCD module needs acsii input
04C1 120380 572   lcall   hex_asci             ;so, convert received hex data
to ascii
04C4 22     573   ret                          ;return
04C5 57     574   scrl:   db        'W'        ;
04C6 45     575   db      'E'                  ;
04C7 4C     576   db      'L'                  ;
04C8 43     577   db      'C'                  ;
04C9 4F     578   db      'O'                  ;
```

```
04CA 4D    579   db      'M'             ;
04CB 45    580   db      'E'             ;
04CC 20    581   db      ' '             ;
04CD 54    582   db      'T'             ;
04CE 4F    583   db      'O'             ;
04CF 20    584   db      ' '             ;
04D0 20    585   db      ' '             ;
04D1 20    586   db      ' '             ;
04D2 20    587   db      ' '             ;
04D3 20    588   db      ' '             ;
04D4 20    589   db      ' '             ;
04D5 45    590   db      'E'             ;
04D6 46    591   db      'F'             ;
04D7 59    592   db      'Y'             ;
04D8 20    593   db      ' '             ;
04D9 20    594   db      ' '             ;
04DA 20    595   db      ' '             ;
04DB 20    596   db      ' '             ;
04DC 20    597   db      ' '             ;
04DD 20    598   db      ' '             ;
04DE 20    599   db      ' '             ;
04DF 20    600   db      ' '             ;
04E0 20    601   db      ' '             ;
04E1 20    602   db      ' '             ;
04E2 20    603   db      ' '             ;
04E3 20    604   db      ' '             ;
04E4 20    605   db      ' '             ;
04E5 41    606   db      'A'             ;
04E6 52    607   db      'R'             ;
04E7 54    608   db      'T'             ;
04E8 49    609   db      'I'             ;
04E9 43    610   db      'C'             ;
04EA 4C    611   db      'L'             ;
04EB 45    612   db      'E'             ;
04EC 20    613   db      ' '             ;
04ED 20    614   db      ' '             ;
04EE 20    615   db      ' '             ;
04EF 20    616   db      ' '             ;
04F0 20    617   db      ' '             ;
04F1 20    618   db      ' '             ;
04F2 20    619   db      ' '             ;
04F3 20    620   db      ' '             ;
04F4 20    621   db      ' '             ;
04F520     622   db      ' '             ;
04F6 20    623   db      ' '             ;
04F7 20    624   db      ' '             ;
04F8 20    625   db      ' '             ;
04F9 20    626   db      ' '             ;
04FA20     627   db      ' '             ;
04FB 20    628   db      ' '             ;
04FC 20    629   db      ' '             ;
04FD 20    630   db      ' '             ;
04FE 20    631   db      ' '             ;
04FF 20    632   db      ' '             ;
0500 20    633   db      ' '             ;
0501 20    634   db      ' '             ;
0502 20    635   db      ' '             ;
0503 20    636   db      ' '             ;
0504 20    637   db      ' '             ;
0505 42    638   scr2:   db      'B'     ;
0506 49    639   db      'I'             ;
0507 52    640   db      'R'             ;
0508 54    641   db      'T'             ;
0509 48    642   db      'H'             ;
050A 44    643   db      'D'             ;
050B 41    644   db      'A'             ;
050C 59    645   db      'Y'             ;
050D 20    646   db      ' '             ;
050E 20    647   db      ' '             ;
050F 20    648   db      ' '             ;
0510 20    649   db      ' '             ;
0511 20    650   db      ' '             ;
0512 20    651   db      ' '             ;
0513 20    652   db      ' '             ;
0514 20    653   db      ' '             ;
0515 20    654   db      ' '             ;
0516 44    655   db      'D'             ;
0517 41    656   db      'A'             ;
0518 59    657   db      'Y'             ;
0519 20    658   db      ' '             ;
051A 20    659   db      ' '             ;
051B 20    660   db      ' '             ;
051C 20    661   db      ' '             ;
051D 20    662   db      ' '             ;
051E 20    663   db      ' '             ;
051F 20    664   db      ' '             ;
0520 20    665   db      ' '             ;
0521 20    666   db      ' '             ;
0522 20    667   db      ' '             ;
0523 20    668   db      ' '             ;
0524 20    669   db      ' '             ;
0525 20    670   db      ' '             ;
0526 4D    671   db      'M'             ;
0527 4F    672   db      'O'             ;
0528 4E    673   db      'N'             ;
0529 54    674   db      'T'             ;
052A 48    675   db      'H'             ;
052B 20    676   db      ' '             ;
052C 20    677   db      ' '             ;
052D 20    678   db      ' '             ;
052E 20    679   db      ' '             ;
052F 20    680   db      ' '             ;
0530 20    681   db      ' '             ;
0531 20    682   db      ' '             ;
0532 20    683   db      ' '             ;
0533 20    684   db      ' '             ;
0534 20    685   db      ' '             ;
0535 20    686   db      ' '             ;
0536 59    687   db      'Y'             ;
0537 45    688   db      'E'             ;
0538 41    689   db      'A'             ;
0539 52    690   db      'R'             ;
053A 20    691   db      ' '             ;
053B 20    692   db      ' '             ;
053C 20    693   db      ' '             ;
053D 20    694   db      ' '             ;
053E 20    695   db      ' '             ;
053F 20    696   db      ' '             ;
0540 20    697   db      ' '             ;
0541 20    698   db      ' '             ;
0542 20    699   db      ' '             ;
0543 20    700   db      ' '             ;
0544 20    701   db      ' '             ;
0545 53    702   scr5:   db      'S'     ;
0546 45    703   db      'E'             ;
0547 54    704   db      'T'             ;
0548 20    705   db      ' '             ;
0549 20    706   db      ' '             ;
054A 20    707   db      ' '             ;
054B 20    708   db      ' '             ;
054C 20    709   db      ' '             ;
054D 20    710   db      ' '             ;
054E 20    711   db      ' '             ;
054F 20    712   db      ' '             ;
0550 20    713   db      ' '             ;
0551 20    714   db      ' '             ;
0552 20    715   db      ' '             ;
0553 20    716   db      ' '             ;
0554 20    717   db      ' '             ;
0555 20    718   db      ' '             ;
0556 44    719   db      'D'             ;
0557 41    720   db      'A'             ;
0558 59    721   db      'Y'             ;
0559 20    722   db      ' '             ;
055A 20    723   db      ' '             ;
055B 20    724   db      ' '             ;
055C 20    725   db      ' '             ;
055D 20    726   db      ' '             ;
055E 20    727   db      ' '             ;
055F 20    728   db      ' '             ;
0560 20    729   db      ' '             ;
0561 20    730   db      ' '             ;
0562 20    731   db      ' '             ;
0563 20    732   db      ' '             ;
0564 20    733   db      ' '             ;
0565 20    734   db      ' '             ;
0566 20    735   db      ' '             ;
0567 20    736   db      ' '             ;
```

```
0568 20     737   db      ' '             ;
0569 20     738   db      ' '             ;
056A 20     739   db      ' '             ;
056B 20     740   db      ' '             ;
056C 20     741   db      ' '             ;
056D 20     742   db      ' '             ;
056E 20     743   db      ' '             ;
056F 20     744   db      ' '             ;
0570 20     745   db      ' '             ;
0571 20     746   db      ' '             ;
0572 20     747   db      ' '             ;
0573 20     748   db      ' '             ;
0574 20     749   db      ' '             ;
0575 20     750   db      ' '             ;
0576 20     751   db      ' '             ;
0577 20     752   db      ' '             ;
0578 20     753   db      ' '             ;
0579 20     754   db      ' '             ;
057A 20     755   db      ' '             ;
057B 20     756   db      ' '             ;
057C 20     757   db      ' '             ;
057D 20     758   db      ' '             ;
057E 20     759   db      ' '             ;
057F 20     760   db      ' '             ;
0580 20     761   db      ' '             ;
0581 20     762   db      ' '             ;
0582 20     763   db      ' '             ;
0583 20     764   db      ' '             ;
0584 20     765   db      ' '             ;
0585 53     766   scr6:   db      'S'      ;
0586 45     767   db      'E'             ;
0587 54     768   db      'T'             ;
0588 20     769   db      ' '             ;
0589 20     770   db      ' '             ;
058A 20     771   db      ' '             ;
058B 20     772   db      ' '             ;
058C 20     773   db      ' '             ;
058D 20     774   db      ' '             ;
058E 20     775   db      ' '             ;
058F 20     776   db      ' '             ;
0590 20     777   db      ' '             ;
0591 20     778   db      ' '             ;
0592 20     779   db      ' '             ;
0593 20     780   db      ' '             ;
0594 20     781   db      ' '             ;
0595 20     782   db      ' '             ;
0596 20     783   db      ' '             ;
0597 20     784   db      ' '             ;
0598 20     785   db      ' '             ;
0599 20     786   db      ' '             ;
059A 20     787   db      ' '             ;
059B 20     788   db      ' '             ;
059C 20     789   db      ' '             ;
059D 20     790   db      ' '             ;
059E 20     791   db      ' '             ;
059F 20     792   db      ' '             ;
05A0 20     793   db      ' '             ;
05A1 20     794   db      ' '             ;
05A2 20     795   db      ' '             ;
05A3 20     796   db      ' '             ;
05A4 20     797   db      ' '             ;
05A5 20     798   db      ' '             ;
05A6 4D     799   db      'M'             ;
05A7 4F     800   db      'O'             ;
05A8 4E     801   db      'N'             ;
05A9 54     802   db      'T'             ;
05AA 48     803   db      'H'             ;
05AB 20     804   db      ' '             ;
05AC 20     805   db      ' '             ;
05AD 20     806   db      ' '             ;
05AE 20     807   db      ' '             ;
05AF 20     808   db      ' '             ;
05B0 20     809   db      ' '             ;
05B1 20     810   db      ' '             ;
05B2 20     811   db      ' '             ;
05B3 20     812   db      ' '             ;
05B4 20     813   db      ' '             ;
05B5 20     814   db      ' '             ;
05B6 20     815   db      ' '             ;
05B7 20     816   db      ' '             ;
05B8 20     817   db      ' '             ;
05B9 20     818   db      ' '             ;
05BA 20     819   db      ' '             ;
05BB 20     820   db      ' '             ;
05BC 20     821   db      ' '             ;
05BD 20     822   db      ' '             ;
05BE 20     823   db      ' '             ;
05BF 20     824   db      ' '             ;
05C0 20     825   db      ' '             ;
05C1 20     826   db      ' '             ;
05C2 20     827   db      ' '             ;
05C3 20     828   db      ' '             ;
05C4 20     829   db      ' '             ;
05C5 53     830   scr7:   db      'S'      ;
05C6 45     831   db      'E'             ;
05C7 54     832   db      'T'             ;
05C8 20     833   db      ' '             ;
05C9 20     834   db      ' '             ;
05CA 20     835   db      ' '             ;
05CB 20     836   db      ' '             ;
05CC 20     837   db      ' '             ;
05CD 20     838   db      ' '             ;
05CE 20     839   db      ' '             ;
05CF 20     840   db      ' '             ;
05D0 20     841   db      ' '             ;
05D1 20     842   db      ' '             ;
05D2 20     843   db      ' '             ;
05D3 20     844   db      ' '             ;
05D4 20     845   db      ' '             ;
05D5 20     846   db      ' '             ;
05D6 20     847   db      ' '             ;
05D7 20     848   db      ' '             ;
05D8 20     849   db      ' '             ;
05D9 20     850   db      ' '             ;
05DA 20     851   db      ' '             ;
05DB 20     852   db      ' '             ;
05DC 20     853   db      ' '             ;
05DD 20     854   db      ' '             ;
05DE 20     855   db      ' '             ;
05DF 20     856   db      ' '             ;
05E0 20     857   db      ' '             ;
05E1 20     858   db      ' '             ;
05E2 20     859   db      ' '             ;
05E3 20     860   db      ' '             ;
05E4 20     861   db      ' '             ;
05E5 20     862   db      ' '             ;
05E6 20     863   db      ' '             ;
05E7 20     864   db      ' '             ;
05E8 20     865   db      ' '             ;
05E9 20     866   db      ' '             ;
05EA 20     867   db      ' '             ;
05EB 20     868   db      ' '             ;
05EC 20     869   db      ' '             ;
05ED 20     870   db      ' '             ;
05EE 20     871   db      ' '             ;
05EF 20     872   db      ' '             ;
05F0 20     873   db      ' '             ;
05F1 20     874   db      ' '             ;
05F2 20     875   db      ' '             ;
05F3 20     876   db      ' '             ;
05F4 20     877   db      ' '             ;
05F5 20     878   db      ' '             ;
05F6 59     879   db      'Y'             ;
05F7 45     880   db      'E'             ;
05F8 41     881   db      'A'             ;
05F9 52     882   db      'R'             ;
05FA 20     883   db      ' '             ;
05FB 20     884   db      ' '             ;
05FC 20     885   db      ' '             ;
05FD 20     886   db      ' '             ;
05FE 20     887   db      ' '             ;
05FF 20     888   db      ' '             ;
0600 20     889   db      ' '             ;
0601 20     890   db      ' '             ;
0602 20     891   db      ' '             ;
0603 20     892   db      ' '             ;
0604 20     893   db      ' '             ;
            894                           end
VERSION 1.2k ASSEMBLY COMPLETE, 0 ERRORS FOUND
```

CONTROL UNIT (CONTR.LST)

```
PAGE 1

1    $mod51
2    ;this is a small test program for CONTROL UNIT (CU)
3    ;sending/receiving data to/from the "LCD" module
4    ;article written for EFY
5    ;programmer-AR Karkare
6    ;date written-14 April 2002
7    ;uses 89c51 micro-controller with 11.059 mhz crystal
8    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
9    ;RESERVED LOCATIONS
10
0045          11    intflg  bit       45h       ;interrupt indicator
              12                                  ;LIST OF I/O
              13
              14                                  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
0000          15    org     0000h               ;
0000 802E     16    sjmp    start               ;skip all interrupt vactors
0023          17    org     0023h               ;
0023 8075     18    sjmp    spint               ;serial port program
0030          19    org     0030h               ;initialization of registers
0030 758160   20    start:  mov sp, #60h        ;set stack pointer
              21                                  ;set SFRs
0033 758700   22    mov     pcon,#00h           ;smod=0
0036 758920   23    mov     tmod,#20h           ;timer1
              24                                  ;gate=0,c/t=0,mode=8
                                                  bit,auto relcad
0039 758BFD   25    mov     tl1, #0fdh          ;reload value for 9.6k baud
rate
003C 758DFD   26    mov     th1, #0fdh          ;
003F 759850   27    mov     scon, #50h          ;mode 1,transmission/reception
                                                  enabled
0042 D28E     28    setb    tr1                 ;start timer
0044 C2AF     29    clr     ea                  ;global int. off
0046 D2AC     30    setb    es                  ;serial int. on
0048 D2BC     31    setb    ip.4                ;high priority to serial port
int.
004A D2B0     32    setb    rxd                 ;float pin
004C D2B1     33    setb    txd                 ;float pin
004E C245     34    clr     intflg              ;keep interrupt flag low
0050 7580FF   35    mov     p0, #0ffh           ;float all pins of port p0
              36                                  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
              37                                  ;save some trial values
0053 75300E   38    mov     30h, #14            ;Day=14
0056 75310B   39    mov     31h, #11            ;Month=11
0059 75322C   40    mov     32h, #44            ;Year=1944
              41                                  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
              42                                  ;wait till interrupt flag goes
                                                  high
              43                                  ;clear interruot flag
              44                                  ;get what address received
              45                                  ;show on port p0
              46                                  ;if start character <STX>
                                                  received,
              47                                  ;get ready to receive address
                                                  and data
              48                                  ;otherwise treat character
                                                  received as address
              49                                  ;send data contents of the
                                                  address
              50                                  ;while sending
              51                                  ;wait for a while for stabi-
lising
              52                                  ;send the data for the asked
                                                  address
005C D2AF     53    setb    ea                  ;global interrupt on
005E 3045FD   54    wait1: jnb       intflg, wait1    ;wait for serial port
flag
0061 C245     55    clr     intflg              ;clear flag
0063 E5F0     56    mov     a, b                ;get  received character
0065 F580     57    mov     p0, a               ;show on port p0 LEDs
0067 B40202   58    cjne    a, #02h, transmit   ;if character is 02 <STX>
                                                  received,
CNTRNEW PAGE 2

006A 800A     59    sjmp    receive             ;jump to receiver program
              60                                  ;otherwise accu. contains RAM
                                                  address
006C          61    transmit:
006C 1200BC   62    lcall   dell0m              ;stabilisation delay
006F F8       63    mov     r0, a               ;set address
0070 E6       64    mov     a, @r0              ;retrieve data
0071 12008C   65    lcall   send                ;send to RS232 port
0074 80E8     66    sjmp    wait1               ;wait for the next call
0076          67    receive:
0076 3045FD   68    wait2: jnb       intflg, wait2    ;wait for serial port
flag
0079 C245     69    clr     intflg              ;clear flag
007B E5F0     70    mov     a, b                ;get received character
007D F580     71    mov     p0, a               ;show on port p0 LEDs
007F F8       72    mov     r0, a               ;save address at r0
0080 3045FD   73    wait3: jnb       intflg, wait3    ;wait for serial port
flag
0083 C245     74    clr     intflg              ;clear flag
0085 E5F0     75    mov     a, b                ;get received character
0087 F580     76    mov     p0, a               ;show on port p1 LEDs
0089 F6       77    mov     @r0, a              ;save whatever data received
008A 80D2     78    sjmp    wait1               ;back to wait for next
                                                  character
              79                                  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
008C C2AF     80    send:   clr      ea         ;disable all interrupts
008E C299     81    clr     ti                  ;pull ti flag low
0090 F599     82    mov     sbuf, a             ;load sbuf
0092 3099FD   83    waitt:  jnb      ti, waitt  ;wait till ti flag goes high
0095 C299     84    clr     ti                  ;pull ti flag low
0097 D2AF     85    setb    ea                  ;enable all interrupts
0099 22       86    ret                         ;return
              87                                  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
009A C2AF     88    spint:  clr      ea         ;disable all interrupts
009C C0D0     89    push    psw                 ;save status
009E C0E0     90    push    acc                 ;save accu.
00A0 C3       91    clr     c                   ;clear carry
00A1 8599F0   92    mov     b, sbuf             ;save received character in B
00A4 C298     93    clr     ri                  ;clear RI bit
00A6 D0E0     94    pop     acc                 ;get back accu.
00A8 D0D0     95    pop     psw                 ;get back status
00AA D2AF     96    setb    ea                  ;enable all interrupts
00AC D245     97    setb    intflg              ;set interruppt flag
00AE 32       98    reti                        ;return from interrupt
              99                                  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
00AF 755004   100   dell1m:  mov      50h, #04  ;delay of 1 milisec.
00B2 755153   101   loopa:   mov      51h, #83  ;
00B5 D551FD   102   loopb:   djnz     51h, loopb;
00B8 D550F7   103   djnz    50h, loopa          ;
00BB 22       104   ret                         ;
00BC 755028   105   dell0m: mov       50h, #40  ;delay of 10 milisec.
00BF 755153   106   loopc:  mov       51h, #83                    ;
00C2 D551FD   107   loopd:  djnz      51h, loopd;
00C5 D550F7   108   djnz    50h, loopc          ;
00C8 22       109   ret                         ;
00C9 75520A   110   dell00m: mov      52h, #10  ;delay of 100 milisec.
00CC 1200BC   111   loope:   lcall   dell0m     ;
00CF D552FA   112   djnz    52h, loope          ;
00D2 22       113   ret                         ;
00D3 75530A   114   dell1s: mov       53h, #10  ;delay of 1 sec.
00D6 1200C9   115   loopf:  lcall     dell00m   ;
00D9 D553FA   116   djnz    53h, loopf          ;
00DC 22       117   ret                         ;
              118                   end

VERSION 1.2k ASSEMBLY COMPLETE, 0 ERRORS FOUND
```

INTERFACING A GRAPHICS LCD WITH THE MICROCONTROLLER

■ A.R. KARKARE

Today, most of the electronics devices come with a liquid crystal display (LCD). Even new fridges have it. It is interesting as well as useful to know how to use LCDs with any device. Here is how to interface a graphics LCD module with a microcontroller.

Graphics LCD module

There are various types of graphics LCD modules available in the market depending on pixel size such as 120×64, 128×128, 240×64 and 240×128. The graphics LCD used here has a pixel size of 240×128 and is based on T6963 controller IC. The module type is SS24E12DLNW-E from UTC.

The 240×128 LCD panel screen has 240 horizontally and 128 vertically arranged pixels. It can be used for graphics or for text in any combination, like one area of 240×64 for graphics and another area of 240×64 for text. Also, the entire area of 240×128 can be used either for graphics or text.

The LCD module gets the data or command instructions from the interfaced microcontroller through data lines D0 through D7. Depending upon the command received, the T6963 finds out the data from the internal VRAM and displays it as graphics or text at appropriate place on the screen. The module has a 128-character internal character-generator ROM (CG-ROM) and can control up to 64 kB of external video RAM (VRAM). A number of character attribute functions are also available.

For communication from/to the connected CPU, the LCD module has eight data lines (D0 through D7), write and read lines, chip-enable line, and a command or data line.

Resetting the module. As soon as the power is switched on, the T6963 IC must first be hardware reset by pulling the reset pin down for at least one millisecond.

Font selection. Font size of the LCD module, like 6×8 (which occupies six horizontal pixels and eight vertical pixels) or 8×8 (which occupies eight horizontal pixels and eight vertical pixels), is selected by pulling the FS pin high or low, respectively. Thus if a 6x8 font is selected, you can get 240/6=40 columns and 128/8=16 text display rows. If an 8x8 font is selected, you can get 240/8=30 columns and 128/8=16 text display rows.

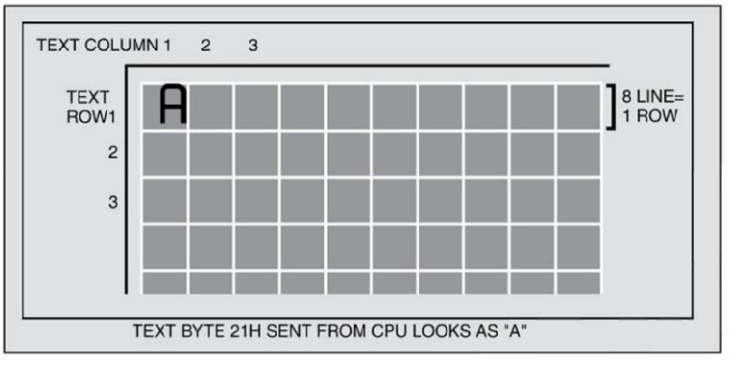


Fig. 1: Text Information Display

The text information is sent by the microcontroller byte by byte to the LCD module. The first byte is displayed in the left corner on the top as shown in Fig. 1. The second byte is displayed to the right of the first byte and so on. The last byte is the 30th byte in the row. The 31st byte is displayed in the second row.

The graphics image is also sent by the microcontroller byte by byte to the LCD module. If a bit is set as '1' it will be displayed in dark, and a '0' bit will be displayed as a clear pixel. The first byte is displayed in the left corner on the top as shown in Fig. 2. The second byte is displayed to the right of

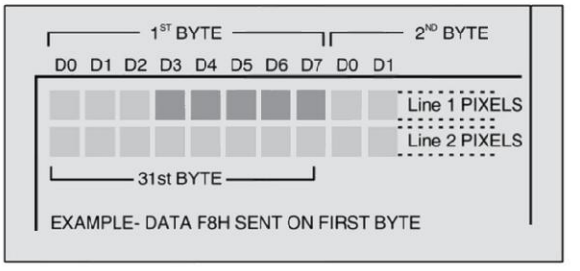


Fig. 2: Graphic Information Display

the first byte and so on. The last byte is the 30th byte in the line. Thus the 31st byte is displayed in the second line of pixels.

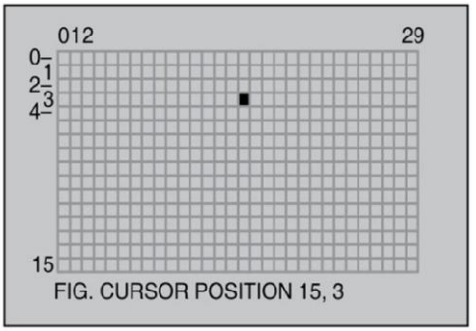


Fig. 3: Cursor Pointer

Initialisation of the LCD module

The initialisation process is always done at the beginning of the software program before we start displaying anything on the screen. It is done by sending a series of commands through data lines from the microcontroller to the LCD module. Some commands may need extra information and therefore should be sent with corresponding data bytes. The C/D pin of the LCD module must be pulled high when sending the command, and pulled low when sending the data.

Cursor pointer set. The 'cursor pointer set' command has two data bytes associated with it to specify the position for the cursor. This is the only command which will shift or move the cursor. The cursor cannot be shifted automatically by the data-write commands. Cursor position should be set within the actual display area.

First argument	Cursor column position
Second argument	Cursor row position
'Cursor pointer' command	(21H)

For example, sending data '0FH' and '03H' and command '21H' will set the cursor position to 15th column in the third line (refer Fig. 3). (Remember the computer starts counting columns from '0' to '29' and rows from '0' to '15.')

Offset register set. The lower five bits of the first data byte should be set to the upper five bits of the start address for the character-generator RAM (CG-RAM) area. The second data byte should be set to '0.'

First argument	CG-RAM Address
Second argument must be	(00H)
Offset register command	(22H)

Address pointer set. The 'address pointer set' command is used to specify the starting address for writing data

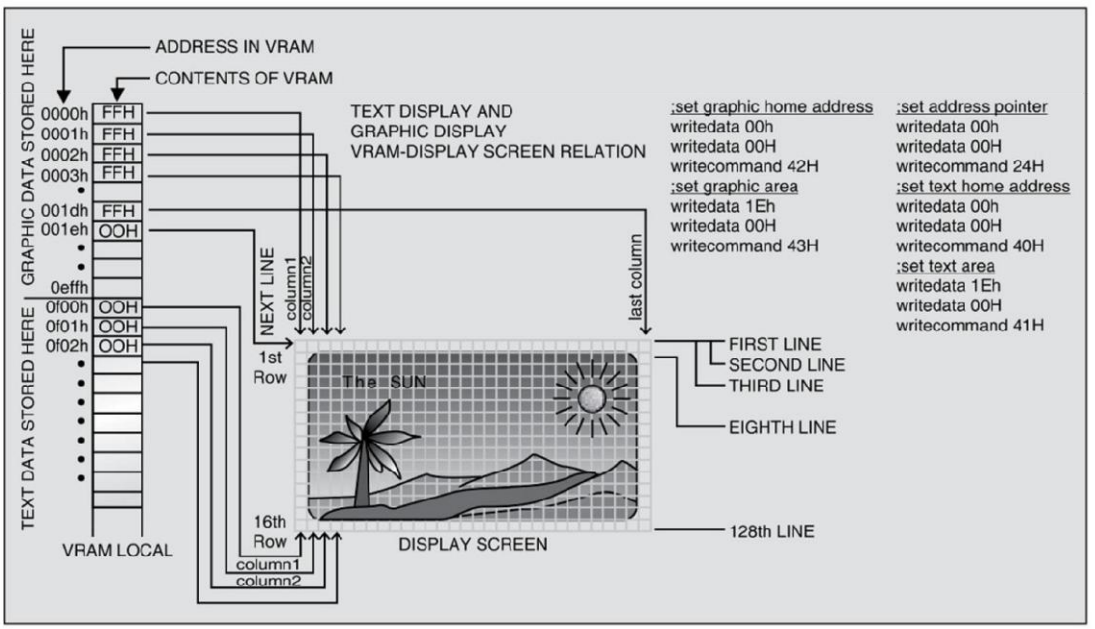


Fig. 4: Address pointer and VRAM storage

Internal CG-ROM

↓MSB lsb→	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	Blank	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
1	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
2	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
3	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
4	\	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
5	p	q	r	s	t	u	v	w	x	y	z	{	:	}	~	blank
6	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
7	É	–	–	ô	ö	ò	û	ù	ÿ	ö	ü	¢	£	¥	–	ƒ

to the VRAM or for reading data from the VRAM. The address should be set to a location in the actual display RAM area specified by the memory map.

First argument	Address pointer (lower)
Second argument	Address pointer (upper)
'Address pointer' command	(24H)

For example, sending data '00H' and '00H' and command '24H' sets the address pointer to start location '0000H' (shaded area in Fig. 4) in the VRAM.

Text home address set (TH). This command defines the starting address in the VRAM for text display data. The data is stored in the text home (TH). It is displayed at the top left-hand character position (the home position).

First argument	Text home address (TH lower)
Second argument	Text home address (TH upper)
'Text home address' command	(40H)

For example, sending data '00H' and '0FH' and command '40H' sets the text home address as '0F00H' (shaded-area address location) in the VRAM area. This is shown in Fig. 4.

Text area set (TA). This command defines the number of columns of text for the text area of VRAM. The text area (TA) may be set independently from the number of characters per line set by hardware settings on the T6963C controller chip. Connecting the FS pin to supply ground means you have selected 8x8 font size, which will allow up to 30 characters per line. However, TA can be programmed for even less than 30, but usually it is set to the same number of characters per line as the LCD module will display.

First argument	Number of columns of characters (TA)
Second argument	(00H)
'Text area' command	(41H)

For example, for 240x128 LCD with 8x8 font size selected, set TA=1EH.

Graphic home address set (GH). This command defines the starting address in the VRAM for graphics display area of VRAM (refer Fig. 4). The data stored in the graphics home (GH) address will be displayed in the first six or eight bits (without shaded area in Fig. 4) of the top line in the left-hand side of the LCD screen, depending on the font size selected. When using the attribute function, the GH address indicates the starting address for the attribute RAM area.

First argument	Graphic home address (GH lower)
Second argument	Graphic home address (GH upper)
'Text home address' command	(42H)

Graphic area set (GA). This command defines the number of columns of graphics data for the graphics area of VRAM. The graphics area (GA) may be set independently from the number of characters per line set by hardware settings on T6963C controller chip. It is usual to set the GA to the same number of characters per line as the LCD module will display.

First argument	Number of columns (GA)
Second argument	(00H)
'Text area' command	(43H)

Description of 'mode set' commands (command byte only)

This command does not require any data byte.

The higher four bits (D7 down through D4) of this command are always '1000,' while lower four bits (D3 down through D0) decide selection of CG-ROM or RAM and how the graphic display is merged with text display.

To use both the 128-character on-chip character-generator ROM (CG-ROM) and the 128-character external CG-RAM function, set D3 to '0.' To use only the 256-character CG-RAM user-defined character generator, set D3 to '1.'

D0 through D2 set the mode through which the text area is merged with the graphics area.

Text attributes can only be used in the text-only mode as the attribute data is stored in the graphics area.

D0 through D2 should be selected as per the need of merging graphics with text.

Logically 'OR' of text with graphics	0 0 0
Logically 'EXOR' of text with graphics	0 0 1
Logically 'AND' of text with graphics	0 1 1
Text only (with attribute data in graphics area)	1 0 0

Description of 'display mode set' commands (command byte only)

This command does not require any data byte.

The higher four bits (D7 through D4) of this command are always '1001,' while the lower four bits (D3 through D0) decide the cursor operation.

Display off	(90H)
D3-	'0' means graphic 'off,'
	'1' means graphics 'on'
D2-	'0' means text 'off,'
	'1' means text 'on'
D1-	'0' means cursor 'off,'
	'1' means cursor 'on'
D0-	'0' means cursor blinking 'off,'
	'1' means cursor blinking 'on'

After reset, D3 through D0 are reset to zero.

Description of 'cursor-pattern select' command (command byte only)

This command does not require any data byte.

The higher four bits (D7 through D4) of this command are always '1010,' while the lower four bits (D3 through D0) decide cursor pattern.

1 line cursor	(A0H)
2 line cursor	(A1H)
" " " " " " " " "	
" " " " " " " " "	
7 line cursor	(A6H)
8 line cursor	(A7H)

PARTS LIST	
Semiconductors:	
IC1	- 7805 5V regulator
IC2	- AT89S8252 microcontroller
D1-D4	- 1N4007 rectifier diode
D5	- 1N4148 switching diode
LED1, LED2	- 5mm light-emitting diode
LCD Module	- SS24E12DLNW-E graphic LCD module
Resistors (all ¼-watt, ±5% carbon):	
R1, R2	- 330-ohm
R3	- 10-kilo-ohm
VR1	- 20-kilo-ohm
RNW1	- 10-kilo-ohm resistor network
Capacitors:	
C1	- 1000µF, 25V electrolytic
C2, C3	- 0.1µF ceramic disk
C4	- 10µF, 16V electrolytic
C5,C6	- 22pF ceramic disk
Miscellaneous:	
X1	- 230V AC primary to 9V, 500mA secondary transformer
X_{TAL}	- 12MHz crystal
S1	- On/off switch

Description of 'data auto read/write' commands (command byte only)

This command does not require any data byte.

The higher six bits (D7 through D2) of this command are always '101100,' while the lower two bits (D1 and D0) decide auto advance mode.

These single-byte commands are very useful when transferring blocks of data to/from the VRAM.

After sending a 'data auto write' (B0H) command, it is not necessary to send the data-write instruction for each data byte being written to the VRAM. The 'data auto-write' command should be sent after the 'address pointer set' command. The address pointer will automatically increment by '1' for each data written. After sending all data, the 'auto mode-reset' command (B2H or B3H) should be sent to resume normal operation. When in the 'data auto-write' mode, no commands can be accepted, as all the following bytes are assumed to be display data bytes.

After sending a 'data auto-read' (B1H) command, it is not necessary to send the data-read instruction for each data byte being read from the VRAM. The 'data auto-read' command should be sent after the 'address pointer set' command. The address pointer will automatically increment by '1' for each data read. After reading all data, the 'auto mode reset' command (B2H or B3H) should be sent to resume normal operation. When in the 'data auto-read' mode, no commands can be accepted.

Data Auto Write Set	(B0H)
Data Auto Read Set	(B1H)
Auto Mode Reset	(B2H or B3H)

Description of 'data-read/data-write' commands (command byte only)

This command does not require any data byte.

The higher five bits (D7 down through D3) of this command are always '11000,' while the lower three bits (D2 down through D0) decide the pointer-advance mode.

These commands are used to write data to or read data from the VRAM. Data-read or data-write commands should be sent after setting an address by the 'pointer set' command. The address pointer may be automatically incremented, decremented or left unchanged depending on which data read/write command is being sent. A data-write or data-read command is required for each data byte written to or read from the VRAM.

This command should not be confused with B0H and B1H commands. In the case of B0H or B1H command,

you need not send a write or read command between each data-write or data-read. But in the case of C0H and C1H commands, after every data write or data read you need to send C0H or C1H for advancing the pointer.

Data Write	-	Address Pointer Auto Incremented (C0H)
Data Read	-	Address Pointer Auto Incremented (C1H)
Data Write	-	Address Pointer Auto Decremented (C2H)
Data Read	-	Address Pointer Auto Decremented (C3H)
Data Write	-	Address Pointer Auto Unchanged (C4H)
Data Read	-	Address Pointer Auto Unchanged (C5H)

Description of 'screen peeking' command (command byte only)

This command does not require any data byte.

This single-byte command is used to transfer one byte of displayed data to the data stack and may be read by the microcontroller using a 'data read' command. It is useful to read the logical combination of text and graphic data on the LCD screen. The status flag STA6 should be checked after each 'screen peeking' command. If the address pointer is not set to the graphics RAM area, the 'screen peeking' command is ignored and STA6 is set to '1.'

Screen Peeking	(E0H)

Description of 'screen copy' command (command byte only)

This command does not require any data byte.

This single-byte command is used to copy one row of data displayed on the LCD screen to the graphics RAM area specified by the 'address pointer set' command. This 'screen copy' command cannot be used if the row of displayed data contains 'text attribute' data as set by the 'mode set' command. The status flag STA6 should be checked after each 'screen copy' command. If the address pointer is not set to the graphics RAM area, the 'screen copy' command is ignored and STA6 is set to '1.'

Screen Copy	(E8H)

Description of 'bit-set/bit-reset' command (command byte only)

This command does not require any data byte.

The high-order four bits (D7 through D4) are always '1111.'

This single-byte command is used to set/reset individual bits in the RAM. The 'address pointer set' command decides which byte is to be operated on. Any one bit in the byte selected can be set or reset. Multiple bits in a byte cannot be set/reset at the same time. D2 through D0 specifies the location of the bit to be set/reset. '000' selects the least significant bit (LSB) and '111' the most significant bit (MSB). For resetting the bit, make D3 as '0' and for setting the bit make D3 as '1.'

Bit Reset	(F0H-F7H)
Bit Set	(F8H-FFH)

Circuit description

After having understood the basic operation of the LCD module, it is fairly simple to interface it with any micro-

controller. Fig. 5 shows the circuit for interfacing the graphics LCD with the microcontroller. Atmel's AT89S8252 microcontroller (IC2) is used to interface the LCD module. Port P0 (Pins 39 through 32) of the microcontroller is connected to data lines D0 through D7 of the LCD module. Port P0 pins are pulled high with the 10-kilo-ohm resistor network. This data port is used to send and receive the data between the microcontroller and the LCD module. The control lines $\overline{W}$, R, $\overline{CE}$, C/$\overline{D}$ and RES pins should be connected to port pins P3.3 through P3.7 of the microcontroller, respectively.

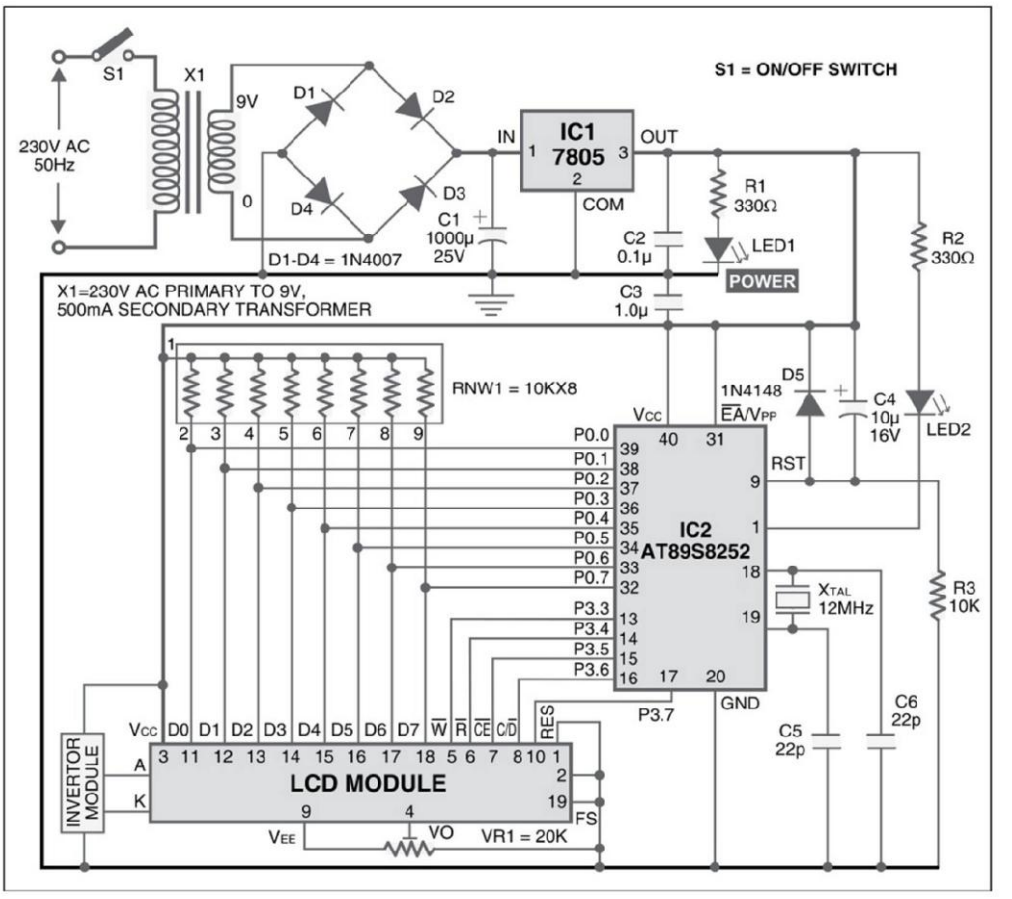


Fig. 5: LCD interface with microcontroller

$\overline{EA}/V_{PP}$ pin of the microcontroller is connected to 5 volts to use the internal flash program memory. The power-on-reset for microcontroller IC2 is achieved through capacitor C4 and resistor R3. Diode D5 connected across capacitor C4 allows short power supply failures. A 12MHz crystal is used for internal oscillator clocks. An LED (LED2) is connected at pin P1 of the microcontroller and acts as the program-running indicator.

Font-select pin 19 (FS) of the LCD module is grounded to generate 8x8-character font size.

Preset VR1 is used for contrast variation of the LCD. Normally –7.8V will give good contrast. V_{EE} is a negative supply voltage output from the LCD module that outputs around –14 volts.

The LCD modules have an inbuilt fluorescent tube as backlight source. This needs a special inverter that comes with the LCD module as optional. It works off 5V supply and generates 100-200V AC, which is connected to 'A' and 'K' terminals of the LCD module. If you opted for the LED backlight module, you may not need this inverter.

The 230V, 50Hz AC mains is stepped down by transformer X1 to deliver a secondary output of 9V, 500 mA. The transformer output is rectified by a full-wave bridge rectifier comprising diodes D1 through D4, filtered by capacitor C1 and regulated by IC 7805 (IC1). Provide adequate heat-sink for 7805 as the LCD backlight draws a significant current. Capacitor C2 bypasses the ripples present in the output. LED1 acts as the power indicator. Resistors R1 and R2 act as the current limiters for LED1 and LED2, respectively.

An actual-size, single-side PCB for interfacing the graphics LCD with the microcontroller is shown in Fig. 6 and its component layout in Fig. 7.

Software program

The software program presented here (EFY-CD) for the operation of LCD module is just an example and you have to re-develop it for your application. The software is written in ANSI C language and compiled using KEIL

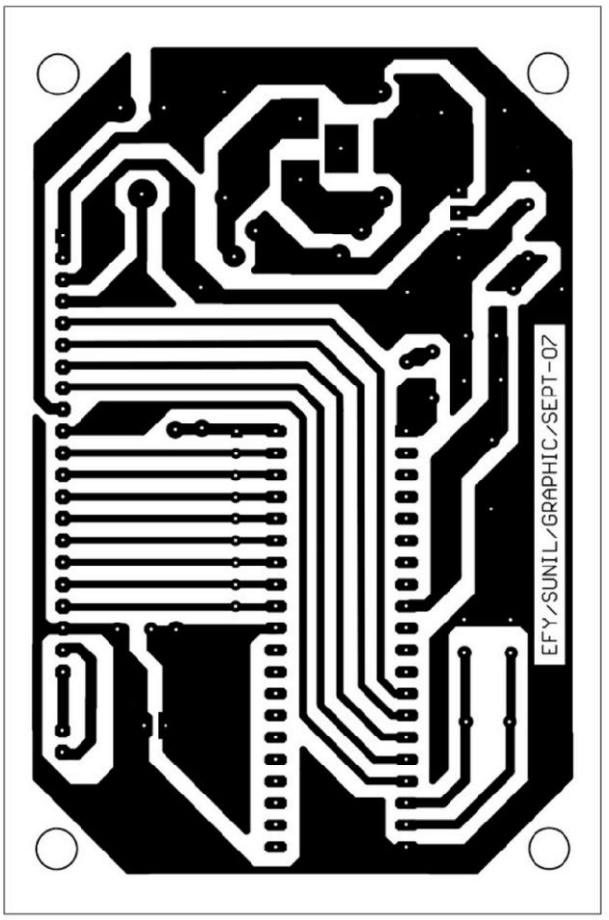


Fig. 6: A single-side, actual-size PCB layout for interfacing the graphics LCD with the microcontroller

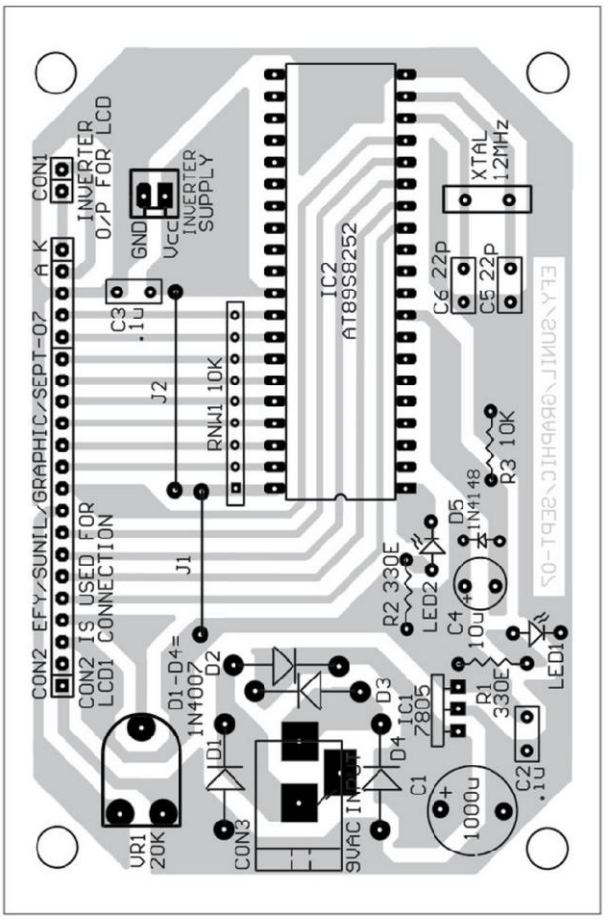


Fig. 7: Component layout for the PCB

compiler. The program has enough remarks inserted on the lines so you can easily understand what the program instructions are really doing.

Starting from the main line, first the ports are initialised, and then the LED blinks four times to indicate that the program and the hardware are working. Basically, the LCD module is first reset and initialised. After initialisation, the LCD module is loaded with graphics data and then with text data. The graphics data contains a scene of the Sun in the background of a tree and hills. The text inserted is 'The SUN.' A blinking cursor is also shown as an example. At the end, the 'graphic and text on' command is inserted before ending the program with an infinite wait loop.

Note that no data or command should be written to the LCD module if the module is busy doing its own internal operation. As such a subroutine is inserted before every write operation to check whether the LCD module is busy or free. This is done in the 'check status' subroutine where both D0 and D1 bits are checked before proceeding any further.

Also note that:

1. The LCD module has a built-in CG-ROM with 128 characters possible. You can use this address to insert any text character. For example, '21H' will mean 'A.'

2. You can make a graphics '.bmp' file of your choice in 'paint' software on a 240×128-pixel paper size and use FONTGEN software to convert the '.bmp' image into graphics data file as required by the LCD module. Copy the graphics data code (generated by FONTGEN software) and paste in the source program at the indicated place. For downloading the FONTGEN, visit '8052.com'.

Download source code: http://www.efymag.com/admin/issuepdf/Graphic%20Display.zip

VERSATILE PROGRAMMABLE STAR DISPLAY

■ **JUNUMON ABRAHAM**

Most of the display circuits avail-able in the market are not programmable. Here's a versatile star display that provides digital control of all the functions interactively and can be programmed for any desired display sequence. It is built around Atmel's Flash-based powerful microcontroller 89C2051.

The circuit

Fig. 1 shows the circuit of programmable star display.

Microcontroller 89C2051. This microcontroller is compatible with the MCS-51 family. Its pin configuration is shown in Fig. 2. The main features of the microcontroller are:

- 2 kB of reprogrammable Flash memory (endurance: 1000 write/erase cycles)
- 128 x 8-bit internal RAM
- Two 16-bit timers/counters
- Six interrupt sources

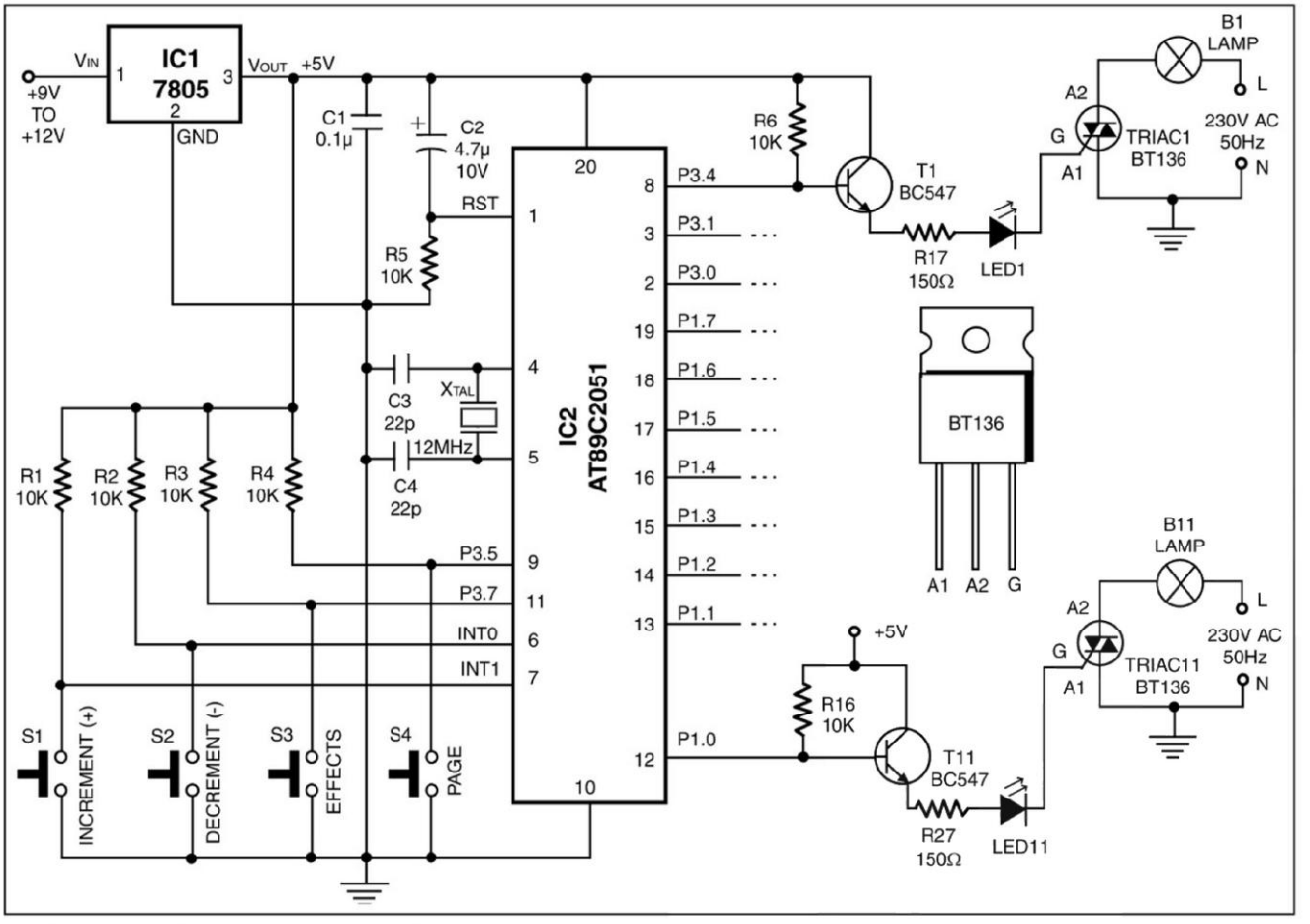


Fig. 1: The circuit diagram of programmable star display

- Programmable serial UART
- 15 programmable I/O lines

The microcontroller 89C2051 here uses clock frequency of 11.059 MHz. (You can also use a 12MHz or nearer-value crystal instead.) The power-on reset function is achieved by the combination of capacitor C2 and resistor R5.

Out of the 15 I/O lines, four lines (P3.2/ INT0, P3.3/ INT1, P3.5, and P3.7) are used as input lines. Hence, only eleven lines are available as outputs lines. A 10-kilo-ohm pull-up resistor is used at each input/output pin. Each output is connected to a transistor driver circuit for driving the corresponding triac, which, in turn, switches the mains supply to the light load (up to 400 watt load can be safely switched through triac BT136).

Four external controls have been provided via pushbutton switches S1 through S4. These switches are labeled as Increment (+), Decrement (–), Effects, and Page, respectively. INT1 and INT0 respond to Increment (+) and Decrement (–) switches respectively, while input pins P3.7 and P3.5 respond to Effects and Page switches, respectively.

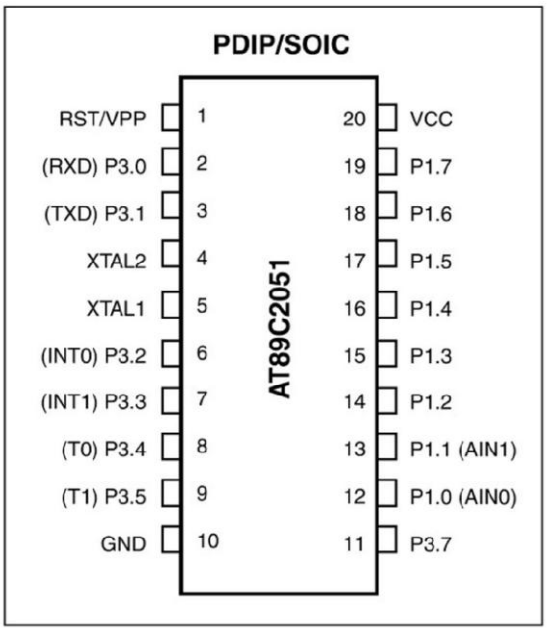


Fig. 2: Pin configuration of AT89C2051 microcontroller

Construction

An actual-size, single-side PCB for the circuit is shown in Fig. 3 with its component layout in Fig. 4. However, this programmable star display circuit can also be assembled on any general-purpose PCB.

The microcontroller should be fixed in an IC socket and the regulator IC should be provided with a heat-sink. For triacs a heat-sink is necessary when the connected load per triac exceeds 60 watt.

Caution. Since switching is accomplished by triacs, don't touch the circuit parts while AC supply is plugged in. Further ensure that Live and Neutral lines are not interchanged

Operating procedure

1. Pushswitches S1 (+) and S2 (–) are used for incrementing and decrementing, respectively, the speed of the display (while pushswitches S3 and S4 are held in open condition).

2. Different display sequences that are stored in different pages of the memory beforehand are accessed by holding Page button down and using (+) and (–) buttons for incrementing and decrementing the page numbers.

3. Different lighting effects are invoked by holding Effects button down and using (+) or (–) buttons for changing the effects. The available effects are blink and reverse. Reverse complements the current output states, i.e. 'on' state bulbs will be off, and vice versa.

4. For interrupting (stopping) the output sequence, hold both Effects and Page buttons down and use (+) or (–) buttons.

The firmware

The source code for the programmable star display circuit, along with suitable comments, is given at the end of this article. The output sequences are obtained from a look-up table (LUT). Each entry is output one by one with some specified delay in between the consecu-

PARTS LIST	
Semiconductors:	
IC1	- 7805 5V regulator
IC2	- AT89C2051 microcontroller
T1-T11	- BC547 npn transistors
LED1-LED11	- LED
TR1-TR11	- BT136 triac
Resistors (all ¼-watt, ±5% carbon, unless stated otherwise):	
R1-R16	- 10-kilo-ohm
R17-R27	- 150-ohm
Capacitors:	
C1	- 0.1μF ceramic disk
C2	- 4.7μF, 10V electrolytic
C3, C4	- 22pF ceramic disk
Miscellaneous:	
B1-B11	- 60W bulb/lamp
S1-S4	- Tactile switch
J1	- 20-pin ZIF socket
XTAL	- 11.059MHz crystal oscillator

tive outputs. This delay can be externally controlled by pushbuttons (+) and (–) or by the commands in the look-up table. Pushbuttons (+) and (–) invoke the interrupt service subroutines corresponding to INT1 and INT0, respectively.

The commands for display sequences are stored in different pages of the memory. Each page can be accessed by holding Page button down and using either (+) or (–) button.

The look-up table for setting up a versatile display is prepared as follows: A total of nine commands are used. All commands are 16-bit wide. Thus two memory locations are needed per command. Out of 16 bits, the three most significant bits (MSBs) are used for Opcode and the remaining 13 bits are used for storing the variables for that Opcode. The general format of commands is shown in Table I, while the bit mapping of each command, along with a brief explanation, is shown in Table II. For understanding their usage you may keep one of the four

TABLE I

General Arrangement of Command Bits

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
C3	C2	C1	C0	V11	V10	V9	V8	V7	V6	V5	V4	V3	V2	V1	V0

C3, C2, C1, C0=Command V11 – V0=Variables for the command

TABLE II

Output States

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	X	Q10	Q9	Q8	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0

Q10 – Q0=States at the outputs
X=Don't care

Blink

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	1	X	X	X	x	x	x	x	x	x	x	x	B

B=1 for blink and 0 for no blink
x=Don't care

Speed

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	0	X	X	X	x	S7	S6	S5	S4	S3	S2	S1	S0

S7 – S0=Speed data
x=Don't care

Delay

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	1	X	X	X	x	W7	W6	W5	W4	W3	W2	W1	W0

W7 – W0=Delay units

Label

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	0	X	X	X	x	L7	L6	L5	L4	L3	L2	L1	L0

L7 – L0=Label

TABLE III

Loop

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	1	N3	N2	N1	N0	L7	L6	L5	L4	L3	L2	L1	L0

N3 - N0=The number of times the loop should execute
L7 - L0=Label indicating the end of the block

Call

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	0	X	x	X	x	L7	L6	L5	L4	L3	L2	L1	L0

L7-L0=Label for the called subroutine

Return

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	1	X	x	x	x	x	x	x	x	x	x	x	x

Reset

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	1	X	x	X	x	x	x	x	x	x	x	x	x

pages of the look-up tables (towards the end of Assembly level program) in front of you.

Output states. This command specifies the states at the eleven outputs. Bits indicated in this command correspond to the outputs indicated in the schematic diagram of Fig. 2. For any output to be active, the corresponding bits in the output should be logic 1. Each output state is to be listed one by one.

Blink. Using this command the blinking effect can be enabled or disabled.

Speed. Using this command it is possible to change the speed in the sequence coding itself.

Delay. It inserts a specified delay between consecutive output states. (Here the delay per unit is 100 ms.)

Label. This command specifies a label for Loop command. It doesn't make any changes to the outputs, but acts as an associated command for Loop.

Loop. This command is used to execute a block of display (output) states for a specified number of times. The end of the loop is specified by the label (L7-L0). When Loop command is executed, the statements following this command are executed up to the label specified in the command, for N (N3-N0) times. This command reduces the burden of writing repeated commands.

Call. This command calls a subroutine.

Return. This command returns the control to the main program from where it was called.

Reset. This command resets the execution point to the beginning of the page.

The coding of commands to get a display sequence is shown as a look-up table in the Assembly listing itself. This Assembly program allows a maximum of five nestings for Loop and Call commands. Here, four pages of display are listed. One can go up to ten pages by specifying the total number of pages against NOOFPAGE in the constant declarations in the Assembly language listing (NOOFPAGE EQU; the total number of pages you have).

Program compilation and programming hints by EFY

The Assembly level program (source program) is written using instruction set mnemonics of Atmel's AT89C2051 microcontroller and assembler directives with the help of any text editor available in DOS or Windows operating system. By using 'save as' command from 'file' menu, the source program is saved under a proper file name with the extension .ASM. In this project, STAR.ASM is the source file.

The program listing starts with $MOD52, which is an assembler control that recognises predefined special function register symbols in the source program. This saves the user from having to define all the registers in the source program. $NOMOD disables the recognising function. This control accesses files of the same name that are included with the MetaLink 8051 cross-assembler distribution diskette. When a $MOD control is used in a source program, it is important that the specific $MOD file be available to the cross-assembler. (Note. This file, along with cross-assembler and its manual, will be included in the next month's EFY-CD.) The source program STAR.ASM has been compiled using ASM51 cross-assembler.

On running the ASM51.EXE, you are asked to enter the source drive where the .ASM file is located and also the name of the source file. After giving the names of the appropriate drive and the source file, press 'enter' to start compilation. The result of the compiled program is shown as 'Assembly Complete, 0 Errors Found' after all errors are successfully resolved. Now, close this screen and go to the directory of the source file. You will observe that two new files are created: STAR.HEX and STAR.LST. The file STAR.HEX is generated by the assembler program and contains the program code in Intel's hex format. Similarly, the file STAR.LST is generated by the assembler program for documentation purposes. It contains memory locations, hex code, mnemonics, and comments (descriptions).

(***Note.*** The program can be simulated by running a suitable software simulator program, for example, SIM31 by SPJ Systems, Pune.)

The programming of AT89C2051 was done at EFY using the Topview programmer from Frontline Electronics, Salem, Tamil Nadu. This device programmer is reasonably priced and can be used to program all the Atmel's 89CXXX family devices using the standard PC. The programming starts by loading the Intel hex/binary files into the device programmer interactively.

The programmer hardware comes along with installation software. Once the installation is done, select 'Atmel Flash programmer' and then click on 'ok'. Select 'load file/ Flash buffer' from 'file' menu. Load the STAR.HEX file into

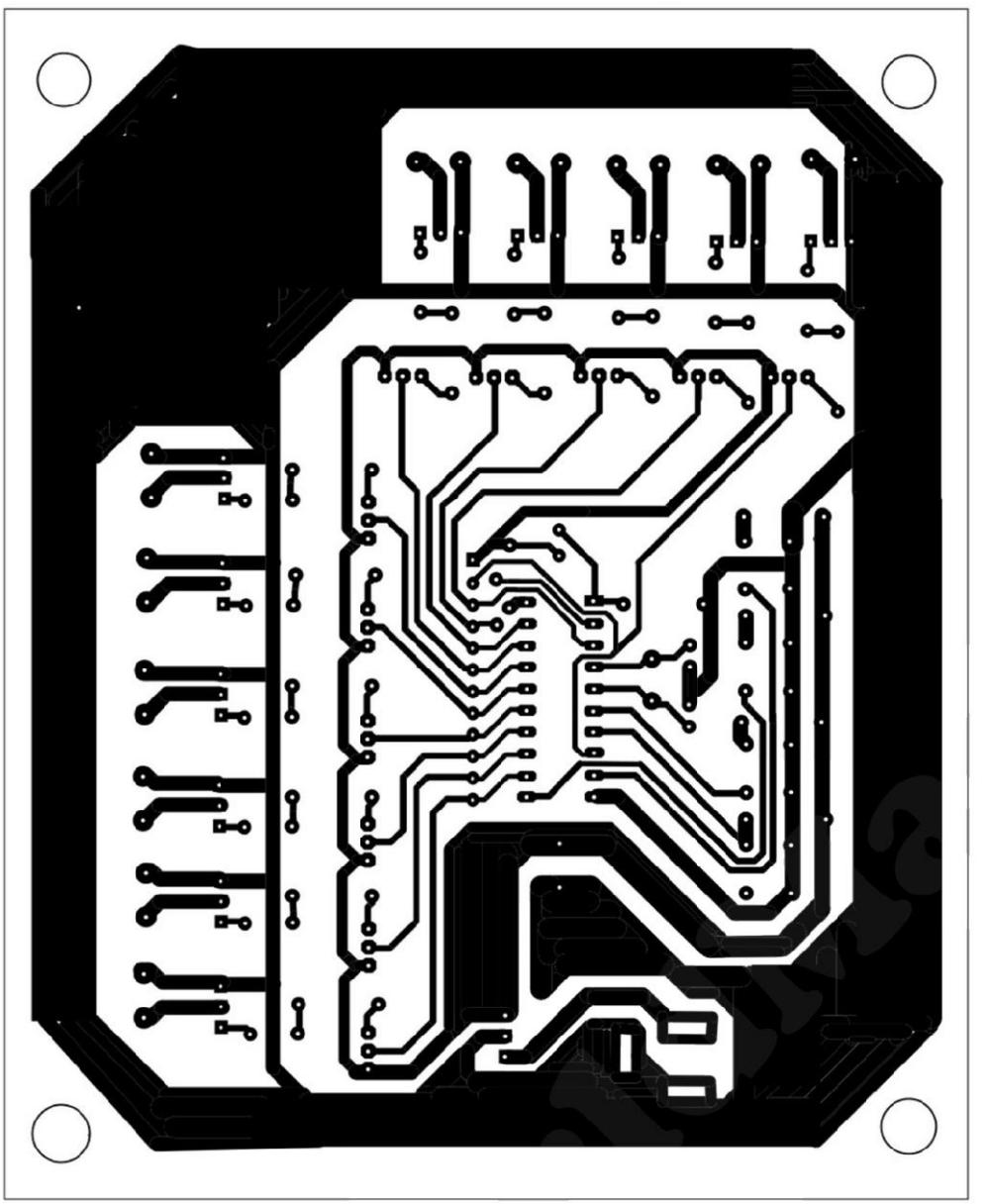


Fig. 3: Actual-size, single-side PCB for the programmable star display

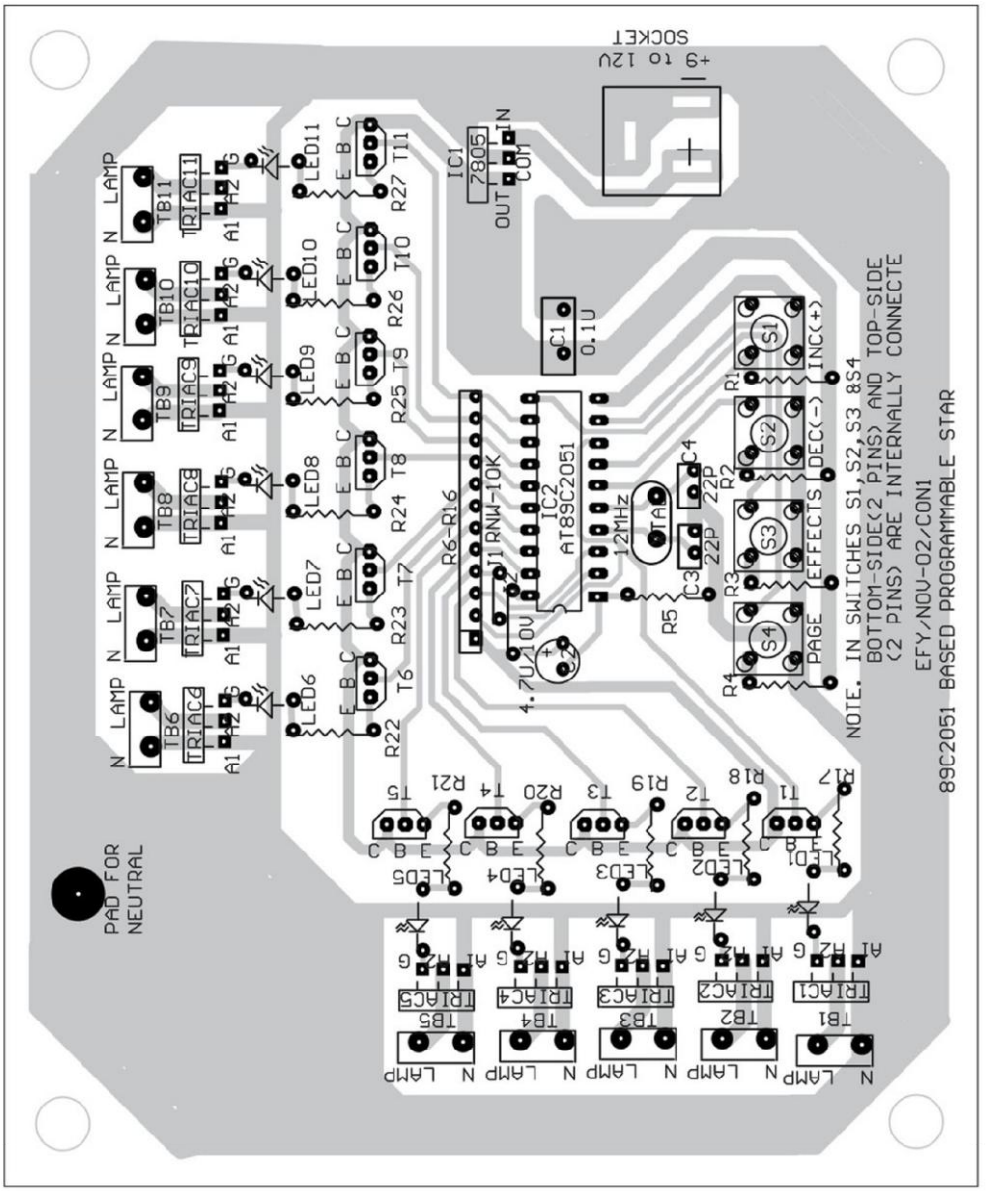


Fig. 4: Component layout for the PCB in Fig. 3

it. Once the file is loaded, follow the instructions given in the user's manual or 'help' menu. From 'device' menu, select Selection/89C2051/Parallel. From 'settings' menu, select the serial COM port, either COM1 or COM2, as appropriate. (The COM port is used to connect the programmer to the PC.)

Before programming, check and verify the loaded program with the help of STAR.LST file to ensure that the correct hex code has been loaded into the buffer. After verification, choose the auto-programming mode from 'settings' menu to program the microcontroller chip.

Download Source Code: http://efymag.com/Versatile_Star_Display-Nov02.zip

STAR.ASM

```
Star.Asm: Assembly Level Listing

$Mod52
;********CONSTANT DECLARATIONS********

COUNT              EQU      08h
COUNT1    EQU      09h
COUNT2    EQU      0Ah
SPEEDREG  EQU      0Bh
SPEED1    EQU      0Ch
LABELREG  EQU      0Dh
STA_DPH   EQU      0Eh
STA_DPL   EQU      0Fh
END_DPH   EQU      10h
END_DPL   EQU      11h
TIMES     EQU      12h
TEMP_SP   EQU      13h
PAGE      EQU      14h
TEMP_P1   EQU      15h
TEMP_P3   EQU      20h
STATUS    EQU      22h
TEMP3_0   BIT      00h
TEMP3_1   BIT      01h
TEMP3_2   BIT      02h
STATE     BIT      09h
ERROR     BIT      0Ah
GBLBLINK  BIT      0Bh
STOP      BIT      0Ch
REVERSE   BIT      0Dh
PLUS      BIT      0Eh
```

```
BLINKREG BIT      10h
NOOFPAGE EQU      04h
;number of pages using, this may change as your wish
SP_MIN   EQU      03h
SP_MAX   EQU      40h

;********PROGRAM STARTS HERE********

         LJMP             MAIN1
         ORG 0003h;INT0 VECTOR
         LJMP             PLUS1
         ORG 0013h;INT1 VECTOR
         LJMP             MINUS

;********ISS OF INT0********
;checking the valid key press of Decrement button

PLUS1:   LCALL    DEBOUNCE
         JNB      P3.2,FOLLOW41
                  RETI
FOLLOW41:SETB     PLUS
         JNB              P3.2,$
         LCALL            DEBOUNCE
         JNB      P3.2,FOLLOW41
         LJMP             KEY_READ

;********ISS OF INT1********
;checking the valid key press of Increment button

MINUS:   LCALL    DEBOUNCE
         JNB      P3.3,FOLLOW42
         RETI
FOLLOW42:CLR      PLUS
         JNB              P3.3,$
                  LCALL    DEBOUNCE
         JNB      P3.3,FOLLOW42
         LJMP             KEY_READ

;********SWITCH CONTACT DEBOUNCE DELAY********
;this delay allows the key debounce to settle

DEBOUNCE:
         MOV      R6,#15h
LOOPP2:  MOV      R7,#0FFh
         DJNZ     R7,$
         DJNZ     R6,LOOPP2
         RET

;********MAIN ROUTINE STARTS HERE********

MAIN1:   ;initialisations
         MOV      SPEEDREG,#11      ;default speed
         MOV      SPEED1,#11
         MOV      TEMP_SP,#00
         MOV      P3,#0FFh
         CLR      REVERSE           ;initially  no
Reverse action
         CLR      GBLBLINK ;initially no Blink
         MOV      PAGE,#01
         SETB     IT0               ;edge trigger-
ing for INT0
         SETB     IT1               ;edge trigger-
ing for INT1
         MOV      IE,#85h           ;enable INT0 &
INT1

MAIN:             CLR      BLINKREG
                  CLR      STOP
         MOV               SP,#24h           ;ini-
tialize stack pointer

;selecting the page address
         MOV      R0,PAGE
         CJNE     R0,#01,NEXTPG1
         MOV      DPTR,#PAGE1
         SJMP     NEXT1
NEXTPG1: CJNE     R0,#02,NEXTPG2
         MOV               DPTR,#PAGE2
         SJMP              NEXT1
NEXTPG2: CJNE     R0,#03,NEXTPG3
         MOV               DPTR,#PAGE3
         SJMP              NEXT1
NEXTPG3: CJNE     R0,#04,NEXTPG4
         MOV               DPTR,#PAGE4
         SJMP              NEXT1
NEXTPG4: CJNE     R0,#05,NEXTPG5
         MOV               DPTR,#PAGE5
         SJMP              NEXT1
NEXTPG5: CJNE     R0,#06,NEXTPG6
         MOV               DPTR,#PAGE6
         SJMP              NEXT1
NEXTPG6: CJNE     R0,#07,NEXTPG7
                  MOV      DPTR,#PAGE7
         SJMP              NEXT1
NEXTPG7: CJNE     R0,#08,NEXTPG8
         MOV               DPTR,#PAGE8
         SJMP              NEXT1
NEXTPG8: CJNE     R0,#09,NEXTPG9
         MOV               DPTR,#PAGE9
         SJMP              NEXT1
NEXTPG9: MOV      DPTR,#PAGE10
;execute the commands in the page

NEXT1:   LCALL             COMMON
         INC               DPTR
         LJMP              NEXT1

;********FOR LOOP1********
;LOOP command processing

FORLOOP1:
         PUSH              STA_DPH
         PUSH              STA_DPL
         PUSH              END_DPH
         PUSH              END_DPL
         PUSH              TIMES

         LCALL             FINDLABL
         JNB      ERROR,FOLLOW20
         SJMP              EXIT1

FOLLOW20:LCALL    LOOPING
EXIT1:            POP      TIMES
         POP               END_DPL
         POP               END_DPH
         POP               STA_DPL
         POP               STA_DPH
         RET

;********LOOPING********
;Execute the commands inside the LOOP command

LOOPING:
         MOV               DPH,STA_DPH
         MOV               DPL,STA_DPL
UP1:     LCALL             COMMON
```

```
         INC             DPTR
         MOV             A,DPH
         CJNE            A,END_DPH,UP1
         MOV             A,DPL
         CJNE            A,END_DPL,UP1
         DJNZ            TIMES,LOOPING
         INC             DPTR
         RET
;********FINDING THE LABEL********
;returns loop count (number of times the loop should
execute) in TIMES
;returns starting address of the loop in STA_DPH &
STA_DPL
;returns ending address of the loop in END_DPH &
END_DPL
;ERROR bit will set on error
;LOOP COUNT equals zero (obtained from the command)-
;and zero loop size are considered to be error

FINDLABL:DEC             DPL
         MOV             A,DPL
         CJNE            A,#0FFh,FOLLOW22
         DEC             DPH

FOLLOW22:CLR      A
         MOVC            A,@A+DPTR
         ANL             A,#0Fh
         CJNE            A,#00,DOWN3
         SJMP            ERROR1

DOWN3:           MOV             TIMES,A   ;get-
ting the loop count (number of
         ;times the loop should execute)
         INC             DPTR
         CLR             A
         MOVC            A,@A+DPTR
         MOV             LABELREG,A
         INC             DPTR
         MOV             STA_DPH,DPH       ;get-
ting the starting address of the loop
         MOV             STA_DPL,DPL
         SETB            ERROR
AGAIN2:
CALL_LB: CLR      A
         MOVC            A,@A+DPTR
         ANL             A,#0F0h
         CJNE            A,#40h,DOWN2
         MOV             END_DPH,DPH       ;get-
ting the ending address of the loop
         MOV             END_DPL,DPL
         INC      DPTR
                  CLR             A
         MOVC            A,@A+DPTR
         CJNE            A,LABELREG,DOWN2A
         RET

DOWN2:           INC             DPTR
DOWN2A:          INC             DPTR
         CLR             ERROR
         MOV             A,DPH
         CJNE            A,#08h,AGAIN2
ERROR1:          SETB            ERROR
         RET

;********CALL SUBROUTINE********
;execute the commands in the subroutine
;first it has to save the current informations that
are all
;needed when returns from the subroutine
;first it has to find the starting of suroutine block
;then execute the commands upto a return command

CALLSUB:
         PUSH            DPH
         PUSH            DPL
         PUSH            STA_DPH
         PUSH            STA_DPL
         PUSH            END_DPH
         PUSH            END_DPL
         PUSH            TIMES
         PUSH            TEMP_SP
         PUSH            STATUS
         CLR             A
         MOVC            A,@A+DPTR
         MOV             LABELREG,A

;finding the subroutine block
         MOV             DPTR,#PAGE1
         CLR             ERROR
         LCALL           CALL_LB
         JNB             ERROR,FOLLOW27
         SJMP            EXIT2

FOLLOW27:MOV      DPH,END_DPH
         MOV             DPL,END_DPL
         INC             DPTR
         INC             DPTR
UP1A:            CLR      A
         MOVC            A,@A+DPTR
         ANL             A,#0F0h
         CJNE            A,#70h,FOLLOW28
         SJMP            EXIT2

;execute the commands inside the subroutine

FOLLOW28:LCALL           COMMON
         INC             DPTR
         SJMP            UP1A

;reteiving the last states
EXIT2:           POP      STATUS
         POP             TEMP_SP
         POP             TIMES
         POP             END_DPL
         POP             END_DPH
         POP             STA_DPL
         POP             STA_DPH
         POP             DPL
         POP             DPH
         MOV             A,TEMP_SP
         ADD             A,SPEED1
         MOV             SPEEDREG,A
         RET

;********DELAY ROUTINE********
;this routine provides the delay between states ac-
cording to the speed
;set either by command or external control
;if Blink effect is active (either by commands or
external control) do blinking
;if stop action is done by external control wait in
this routine itself

DELAY1SE:
         CLR             STATE
         MOV             COUNT2,SPEEDREG
```

```
LOOP2:          MOV             COUNT,#32h
        MOV             C,GBLBLINK
        ORL             C,BLINKREG
        JNC             LOOP1
        MOV             COUNT,#30h
        CPL             STATE
        JNB             STATE,FOLLOW5
        MOV             P1,#00
        CLR             P3.0
        CLR             P3.1
        CLR             P3.4
        SJMP            LOOP1

FOLLOW5: MOV    P1,TEMP_P1
        MOV             C,TEMP3_0
        MOV             P3.0,C
        MOV             C,TEMP3_1
        MOV             P3.1,C
        MOV             C,TEMP3_2
        MOV             P3.4,C

LOOP1:          MOV
COUNT1,#0F7h
        DJNZ            COUNT1,$
        DJNZ            COUNT,LOOP1
        DJNZ            COUNT2,LOOP2
        MOV             P1,TEMP_P1
        MOV             C,TEMP3_0
        MOV             P3.0,C
        MOV             C,TEMP3_1
        MOV             P3.1,C
        MOV             C,TEMP3_2
        MOV             P3.4,C

        JB              STOP,DELAY1SE
        RET

;********COMMON ROUTINE********
;this routine execute the commands sequencially
;the commands are decoded by analizing the 4MSBs of
the command
;note the respective commands for the bit orienta-
tion

COMMON:
        CLR             A
        MOVC            A,@A+DPTR;GETTING  THE
EVEN POSITION LOOKUP DATA
        ANL             A,#0F0h
        CJNE            A,#00,DECIDE      ; I F
THE FIRST 4 BITS ARE NOT ZERO, THEN IT
        ;INDICATES A SPECIAL COMMAND

        CLR             A
        MOVC            A,@A+DPTR;OUTPUT  THE
BITS D10 TO D0 (GETTING FROM
                                ;2  LOCATIONS)
TO THE PORTS

;IF THE REVERSE IS ENABLED (BY THE EXTERNAL
                                ;CONTROL) COM-
PLEMENT THE RESPECTIVE

;DATA BEFORE OUTPUTTING
        JNB             REVERSE,FOLLOW26
        CPL             A

FOLLOW26: MOV   C,ACC.0
        MOV             P3.0,C
        MOV             C,ACC.1
        MOV             P3.1,C
        MOV             C,ACC.2
        MOV             P3.4,C
        MOV             TEMP_P3,A

        INC             DPTR
        CLR             A
        MOVC            A,@A+DPTR
        JNB             REVERSE,FOLLOW37
        CPL             A
FOLLOW37: MOV           P1,A
        MOV             TEMP_P1,A

        LCALL           DELAY1SE
        RET

DECIDE:         INC             DPTR
        CJNE            A,#20h,NEXT10
;CHECKING FOR 'SPEED' COMMAND
        CLR             A
        MOVC            A,@A+DPTR
        MOV             TEMP_SP,A
        ADD             A,SPEED1
        MOV             SPEEDREG,A        ;re-
load the SPEEDREG with new value
        RET

NEXT10:         CJNE            A,#10h,NEXT11
;CHECKING FOR 'BLINK' COMMAND
        CLR             A
        MOVC            A,@A+DPTR
        MOV             C,ACC.0
        MOV             BLINKREG,C
RET

NEXT11:         CJNE            A,#30h,NEXT12
;CHECKING FOR 'DELAY' COMMAND
        CLR             A
        MOVC            A,@A+DPTR
        CJNE            A,#00,LOOP12      ; I F
DELAY SPECIFIED IN THE INSTRUCTION IS = 0,

;NO NEED TO EXECUTE THE DELAY
                RET
LOOP12:         MOV             COUNT1,#190
;DELAY 100mSEC
LOOP12A: MOV            COUNT,#0FFh
                DJNZ            COUNT,$
                DJNZ            COUNT1,LOOP12A
                DJNZ            ACC,LOOP12
                RET
;now total time passed is=100mSec x value of ACC

NEXT12:         CJNE            A,#0F0h,NEXT13
;CHECKING FOR 'RESET' COMMAND
        LJMP            MAIN

NEXT13:         CJNE            A,#50h,NEXT14A
;CHECKING FOR 'LOOP' COMMAND
        LCALL           FORLOOP1
        RET

NEXT14A: CJNE           A,#60h,NEXT14
;CHECKING FOR 'CALL' COMMAND
        LCALL           CALLSUB
```

```
        RET

NEXT14:         RET

;********READING KEY DATA********
;this routine read the state of 'Effect' & 'Page'
button
;accordingly activate/deactivate different effects
;and change some parameters(such as speed, page
etc.)
;in response to the strocks of Increment and Decre-
ment button
;if both 'Effect' & 'Page'buttons are open ,then
speed will change in response to (+)&(-) button
;if only 'Page' button is pressed down then,page
changes in response to (+)&(-) buttons
;if only 'Effect' button is pressed down then, dif-
ferent effects(Blink&Reverse) can be activate/deac-
tivate by (+)&(-) buttons
;if both 'Effect'&'Page' buttons are pressed down
then,you can interrupt(stop) the display output se-
quence

KEY_READ:
        MOV             A,P3
        ANL             A,#0A0h
        CJNE            A,#0A0h,NEXT3
        JB              PLUS,NEXT2
        MOV             A,SPEED1
        CJNE            A,#SP_MIN,$+3
        JNC             FOLLOW1
        LJMP            RELEASE
FOLLOW1: DEC    SPEED1
TOP5:           MOV     A,TEMP_SP
        ADD             A,SPEED1
        MOV             SPEEDREG,A
        LJMP            RELEASE

NEXT2:  MOV     A,SPEED1
        CJNE            A,SP_MAX,$+3
        JC              FOLLOW1A
        LJMP            RELEASE
FOLLOW1A:INC    SPEED1
        SJMP            TOP5

NEXT3:  CJNE    A,#20h,NEXT4
        JB              PLUS,ALT1
        CPL             GBLBLINK
        SJMP            RELEASE

ALT1:           CPL     REVERSE
        SJMP            RELEASE

NEXT4:  CJNE    A,#80h,NEXT4A
        MOV             A,PAGE
        JB              PLUS,ALT2
        CJNE            A,#01,ALT3
        SJMP            RELEASE
ALT3:           DEC     PAGE
        SJMP            DOWN40

ALT2:           CJNE
A,#NOOFPAGE,ALT4
        SJMP            RELEASE
ALT4:           INC     PAGE
DOWN40: MOV     R0,SP
        MOV             DPTR,#MAIN
        MOV             @R0,DPH
        DEC             R0
        MOV             @R0,DPL
        CLR             IE0
        CLR             IE1
        RETI

NEXT4A: JB      PLUS,ALT5
        CPL             STOP
        SJMP            RELEASE
ALT5:           CPL     STOP

RELEASE: RETI
;********LOOKUP TABLE********
;While peparing this lookup table following points
should be noted
;you can have maximum of 10 pages (the size of the
pages are limited by the memory capacity(2KB) of
89C2051)
;you have to specify the total number of pages that
you are using
;in the initial constant declaration,ie. NOOFPAGE
EQU     ??(number of pages you are using)
;maximum of 5 nesting is allowed for CALL & LOOP
commands
;you can have any number of CALL& LOOP commands in
a page
;all commands are 16-bit long
;the labels for CALL & LOOP commands should not be
same
;don't use same labels multiple times(even in dif-
ferent pages also)
;output status (speed&blink) will be automatically
preserved when using a CALL command
;before proceeding you have to reffer the syntax of
all the commands
;

PAGE1:  ;THIS IS A NORMAL PAGE (WITHOUT ANY SPEED
COMMAND OR BLINK)
        DB      01010011B,00000001B         ;LOOP
3 TIMES THE BLOCK UP TO LABEL 1
        DB      00000100b,00000000b         ;DIS-
PLAY DATA
        DB      00000010B,00000000B
        DB      00000001B,00000000B
        DB      00000000B,10000000B
        DB      00000000b,01000000b
        DB      00000000b,00100000b
        DB      00000000b,00010000b
        DB      00000000b,00001000b
        DB      00000000b,00000100b
        DB      00000000b,00000010b
        DB      00000000b,00000001b
        DB      01000000B,00000001B         ;LABEL
1
        DB      01010100B,00000010B         ;LOOP
4 TIMES THE BLOCK UP TO LABEL 2
        DB      00000000b,00000001b
        DB      00000000b,00000010b
        DB      00000000B,00000100B
        DB      00000000B,00001000B
        DB      00000000B,00010000B
        DB      00000000b,00100000b
        DB      00000000b,01000000b
        DB      00000000b,10000000b
        DB      00000001b,00000000b
        DB      00000010b,00000000b
        DB      00000100b,00000000b
```

```
        DB      01000000B,00000010B     ;LABEL
2
        DB      01010100B,00000011B     ;LOOP
4 TIMES THE BLOCK UP TO LABEL 3
        DB      00000100b,00000000b     ;DIS-
PLAY DATA
        DB      00000110B,00000000B
        DB      00000111B,00000000B
        DB      00000111B,10000000B
        DB      00000111b,11000000b
        DB      00000111b,11100000b
        DB      00000111b,11110000b
        DB      00000111b,11111000b
        DB      00000111b,11111100b
        DB      00000111b,11111110b
        DB      00000111b,11111111b
        DB      01000000B,00000011B     ;LABEL
3

        DB      01010010B,00000100B     ;LOOP
2 TIMES THE BLOCK UP TO LABEL 4

        DB      00000100b,00000001b     ;DIS-
PLAY DATA
        DB      00000010B,00000010B
        DB      00000001B,00000100B
        DB      00000000B,10001000B
        DB      00000000b,01010000b
        DB      00000000b,00100000b
        DB      01000000B,00000100B     ;LABEL
4

        DB      11110000b,00000000b     ;RE-
SET, STARTS FROM BEGINNING

;********PAGE-2 DISPLAY DATA********

PAGE2:   ;CUSTOM PAGE (WITH THE USE OF BLINK AND
SPEED COMMANDS)
        DB      01010100B,00000101B     ;LOOP
4 TIMES THE BLOCK UP TO LABEL 5
        DB      00100000B,00000000B     ;SPEED
IS 0
        DB      00010000b,00000000B     ; N O
BLINK
        DB      00000100b,00000000b     ;DIS-
PLAY DATA
        DB      00000010B,00000000B
        DB      00000001B,00000000B
        DB      00000000B,10000000B
        DB      00000000b,01000000b
        DB      00000000b,00100000b
        DB      00000000b,00010000b
        DB      00000000b,00001000b
        DB      00000000b,00000100b
        DB      00000000b,00000010b
        DB      00000000b,00000001b
        DB      00010000b,00000001B     ;BLINK
        DB      00000000b,00000001b     ;DIS-
PLAY DATA
        DB      00000000b,00000010b
        DB      00000000B,00000100B
        DB      00000000B,00001000B
        DB      00000000B,00010000B
        DB      00000000b,00100000b
        DB      00000000b,01000000b
        DB      00000000b,10000000b
        DB      00000001b,00000000b
        DB      00000010b,00000000b
        DB      00000100b,00000000b
        DB      01000000B,00000101B     ;LABEL
5

        DB      01010011B,00000110B     ;LOOP
3 TIMES THE BLOCK UP TO LABEL 6
        DB      00100000B,00010100B     ;SPEED
IS 20
        DB      00010000b,00000000B     ; N O
BLINK
        DB      00000100b,00000001b     ;DIS-
PLAY DATA
        DB      00000010B,00000010B
        DB      00000001B,00000100B
        DB      00000000B,10001000B
        DB      00000000b,01010000b
        DB      00000000b,00100000b
        DB      01000000B,00000110B     ;LABEL
6

        DB      11110000b,00000000b     ;RE-
SET, STARTS FROM BEGINNING

********PAGE-3 DISPLAY DATA********

PAGE3:;showing nested LOOP

        DB      00100000B,00000000B     ;SPEED
IS 0
        DB      00010000b,00000000B     ; N O
BLINK
        DB      01010011B,00000111B     ;LOOP
3 TIMES THE BLOCK UP TO LABEL 7
        DB      01010010B,00001000B     ;LOOP
2 TIMES THE BLOCK UP TO LABEL 8
        DB      00000100b,00000000b
        DB      00000110B,00000000B
        DB      00000111B,00000000B
        DB      00000111B,10000000B
        DB      00000111B,11000000B
        DB      00000111B,11100000B
        DB      00000111B,11110000B
        DB      00000111B,11111000B
        DB      00000111B,11111100B
        DB      00000111B,11111110B
        DB      00000111B,11111111B
        DB      00000111b,11111111B
        DB      00000111B,11111110B
        DB      00000111B,11111100B
        DB      00000111B,11111000B
        DB      00000111B,11110000B
        DB      00000111B,11100000B
        DB      00000111B,11000000B
        DB      00000111B,10000000B
        DB      00000111B,00000000B
        DB      00000110B,00000000B
        DB      00000100B,00000000B
        DB      00000000B,00000000B
        DB      01000000B,00001000B     ;LABEL
8
        DB      00010000b,00000001B     ;BLINK
        DB      01000000B,00000111B     ;LABEL
7

DB      11110000b,00000000b     ;RESET, STARTS
FROM BEGINNING
```

```
;********PAGE-4 DISPLAY DATA********

PAGE4:
         DB       00100000B,00000000B         ;SPEED
IS 0
         DB       01100000B,00001001B         ;Call
subroutine with label 9
         DB       00100000B,00110000B         ;SPEED
IS 48
         DB       01100000B,00001010B         ;Call
subroutine with label 10            D     B
00100000B,00010000B         ;SPEED IS 16
         DB       00110000B,01100000B         ;DELAY
(96D[01100000B]x100ASec = 9.6Sec)
         DB       00010000b,00000001B         ;BLINK
         DB       01100000B,00001001B         ;Call
subroutine with label 9
         DB       11110000b,00000000b         ;RESET,
STARTS FROM BEGINNING

         DB       01000000B,00001001B         ;Sub-
routine with label 9
         DB       00000100b,00000001b         ;DIS-
PLAY DATA
         DB       00000010B,00000010B
         DB       00000001B,00000100B
         DB       00000000B,10001000B
         DB       00000000b,01010000b
         DB       00000000b,00100000b
         DB       01110000B,00000000B         ;Return
         DB       01000000B,00001010B         ;sub-
routine with Label 10
         DB       01010100B,00001011B         ;LOOP
4 TIMES THE BLOCK UP TO LABEL11
         DB       00000100b,00000000b         ;DIS-
PLAY DATA
         DB       00000110B,00000000B
         DB       00000111B,00000000B
         DB       00000111B,10000000B
         DB       00000111b,11000000b
         DB       00000111b,11100000b
         DB       00000111b,11110000b
         DB       00000111b,11111000b
         DB       00000111b,11111100b
         DB       00000111b,11111110b
         DB       00000111b,11111111b
         DB       01000000B,00001011B         ;LABEL
11
         DB       01110000B,00000000B         ;Return

;********PAGE-5 DISPLAY DATA********

PAGE5:   ;YOU CAN ADD MORE DISPLAY PAGES HERE ON-
WARDS

;********PAGE-6 DISPLAY DATA********

PAGE6:

;********PAGE-7 DISPLAY DATA********

PAGE7:

;********PAGE-8 DISPLAY DATA********

PAGE8:

END
```

AT89C51-BASED MOVING-MESSAGE DISPLAY

■ **PANKAJ KISHOR VARMA**

LED-based moving-message displays are becoming popular for transmitting information to large groups of people quickly. These can be used indoors or outdoors. We can find such displays in areas like railway platforms, banks, public offices, hotels, training institutes, nightclubs and shops.

Compared to LEDs, liquid-crystal displays (LCDs) are easy to interface with a microcontroller for displaying information as these have many built-in functions. But these can't be observed from a distance and large-size LCDs are very costly.

LED-based displays can be of two types: dot-matrix and segmental. If you implement a moving-message display with multiplexed dot-matrix LEDs, it will be very costly for displaying 16 characters or more at a time. Moreover, programming will require a lot of data memory or program memory space. An external RAM may be needed to complement a microcontroller like AT89C51.

However, if you use alphanumeric (16-segment LED) displays for the above purpose, programming burden is reduced and also it becomes highly cost-effective. You can make your own display panel consisting of 16 alphanumeric characters at a much lower cost.

The circuit presented here uses 16 common-anode, single-digit, alphanumeric displays to show 16 characters at a time. Moreover, programming has been done to make the characters move in a beautiful manner. A message appears on the panel from the right side, stays for a few seconds when the first character reaches the leftmost place and then goes out from the left side. It displays 16 different messages to depict different occasions, which can be selected by the user through a DIP switch.

PARTS LIST

Semiconductors:

IC1	- AT89C51 microcontroller
IC2, IC3	- 74LS138 3-to-8 decoder
IC4	- 7805 5V regulator
T1-T16	- BC558 pnp transistor
D1-D4	- 1N4007 rectifier diode
DIS1-DIS16	- KLA51 common-anode alphanumeric display
LED1	- 5mm LED

Resistors (all ¼-watt, ±5% carbon):

R1-R16	- 2.2-kilo-ohm
R17-R32	- 120-ohm
R33-R37	- 10-kilo-ohm
R38	- 220-ohm

Capacitors:

C1, C2	- 33pF ceramic disk
C3	- 2200µF, 25V electrolytic
C4	- 1µF, 16V electrolytic
C5	- 10µF, 16V electrolytic
C6	- 0.1µF ceramic disk

Miscellaneous:

X1	- 220V AC primary to 9V, 500mA secondary transformer
X_{TAL}	- 11.0592MHz crystal
S0-S3	- 4-pin DIP switch
S4	- Push-to-'on' switch

Circuit description

Fig. 1 shows the circuit of the microcontroller-based moving-message display. It comprises microcontroller AT89C51, three-to-eight decoder 74LS138, common anode alphanumeric displays, regulator 7805 and a few discrete components.

At the heart of the moving-message display is Atmel AT89C51 microcontroller (IC1). It is a low-power, high-performance, 8-bit microcontroller with 4 kB of flash programmable and erasable read-only memory (PEROM) used as on-chip program memory, 128 bytes of RAM used as internal data memory, 32 individually programmable input/output (I/O) lines divided into four 8-bit ports, two 16-bit programmable timers/counters, a five-vector two-level interrupt architecture, on-chip oscillator and clock circuitry.

Ports P0 and P2 of the microcontroller have been configured to act as a common data bus for all the 16 alphanumeric displays whose corresponding data pins have been tied together to make a common 16-bit data bus. Port-2 provides the higher byte of data, while port-0 provides the lower one to light up a character on the display. Port pins P1.2-P1.4 and P1.5-P1.7 of the microcontroller have been used as address inputs for decoder IC3 and IC4 (74LS138) to enable one of the fourteen alphanumeric displays (DIS3 through DIS16) at a time, respectively. However, displays

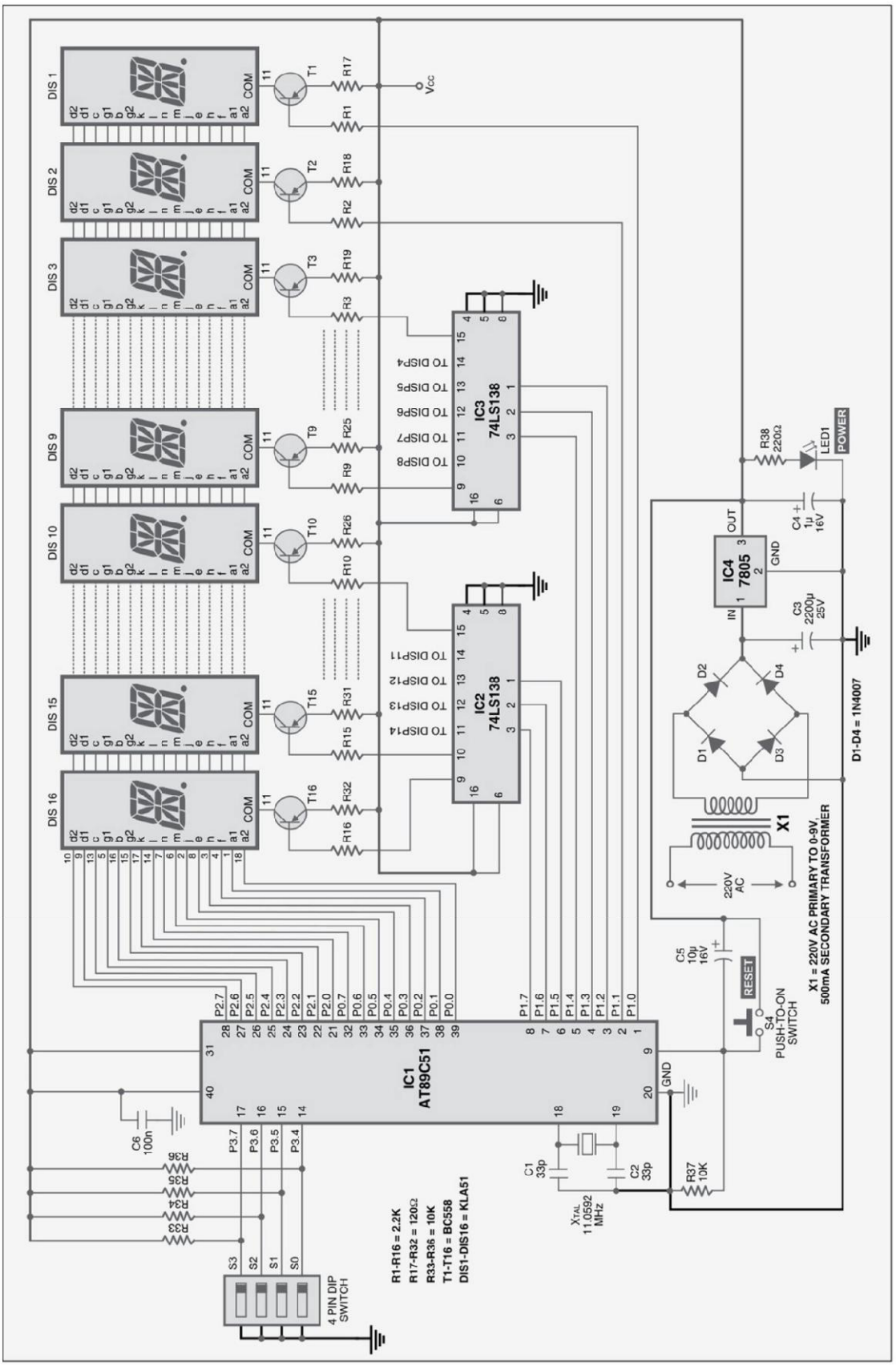


Fig. 1: Circuit for microcontroller-based moving-message display

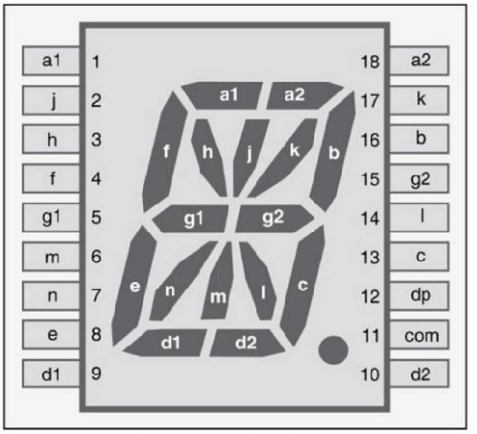


Fig. 2: Pin configuration of alphanumeric display

DIS1 and DIS2 are enabled or disabled directly by port pins P1.0 and P1.1. Pins 4 and 5 are grounded and pin 6 is made high to enable decoder 74LS138.

Fig. 2 shows the pin configuration of the common-anode alphanumeric display.

All the corresponding data pins Dis1 through DIS16 of alphanumeric displays have been tied together, while the common anode of each display is separately powered via a BC558 transistor which switches 'on' or 'off' as required, through outputs of 74LS138 ICs and pins P1.0 and P1.1 of IC1. The higher nibble of port P3 (P3.4 through P3.7) is used as a selection bus to select one of the 16 previously stored messages using the 4-bit binary value present on these pins. This value can be changed through a 4-pin DIP switch (S0 through S3).

Selection pins P3.4 through P3.7 are pulled high via resistors R36 through R33, respectively. When the switch connected to a given pin is open the value is high (1), and when it is closed the pin is held low and the value becomes '0.' In this way, by using a 4-bit number you can select any of the 16 messages shown in the Table.

Capacitor C5 and resistor R37 form the power-'on' reset circuit, while a push-to-connect switch has been used for manual reset. An 11.0592MHz crystal generates the basic clock frequency for the microcontroller. To change the message being displayed while the circuit is working, first change the number present at the selection bus, then press 'reset' key.

The 220V AC mains is stepped down by transformer X1 to deliver the secondary output of 9V, 500 mA. The output of the transformer is rectified by a full-wave bridge rectifier comprising diodes D1 through D4, filtered by capacitor C3 and then regulated by IC 7805 (IC4). Capacitor C4 bypasses any ripple present in the regulated power supply. LED1 acts as the power-'on' indicator.

List of Messages Which can be Selected

S3S2S1S0	Message selected
0 0 0 0	Happy Birthday
0 0 0 1	Happy New Year
0 0 1 0	*Happy Diwali*
0 0 1 1	Merry Christmas
0 1 0 0	*Happy Holi*
0 1 0 1	*Eid Mubarak*
0 1 1 0	Happy Dashehra
0 1 1 1	Happy Wedding
1 0 0 0	Happy Janmashtmi
1 0 0 1	*Happy Rakhi*
1 0 1 0	*Happy Pongal*
1 0 1 1	Happy Mothers Day
1 1 0 0	*Happy Ramjan*
1 1 0 1	*Happy Lohri*
1 1 1 0	*Happy Easter*
1 1 1 1	Welcome to All

Construction

Fig. 3 shows an actual-size, single-side PCB layout for the microcontroller-based moving-message display circuit, except displays DIS1 through DIS16, transistors T1 through T16 and resistors R17 through R32. Component layout for this PCB is shown in Fig. 4.

Fig. 5 shows the PCB for displays DIS1 through DIS8, transistors T1 through T8 and resistors R17 through R24. Component layout for this PCB is shown in Fig. 6. You need to use an additional PCB as shown in Fig. 5 for DIS9 through DIS16, so as to configure 16 alphanumeric displays. For this PCB, the corresponding components will be transistors T9 through T16 and resistors R25 through R32 in addition to displays DIS9 through DIS16. Corresponding connector are provided to make a proper connection. Connectors CON2, CON4 and CON6 of Fig. 3 are connected to CON2, CON4 and CON6 of Fig. 5, respectively, through external wires to interface DIS1 through DIS9. Connectors CON3, CON5 and CON7 of Fig. 3 are connected to CON2, CON4 and CON6 of Fig. 5, respectively, through external cable to interface DIS9 through DIS16.

Software and its working

The source code 'movmsg.asm' is written in Assembly language and assembled using cross-compiler. It is well commented and easy to understand. Timer 1 has been used to generate a delay of around 1 ms

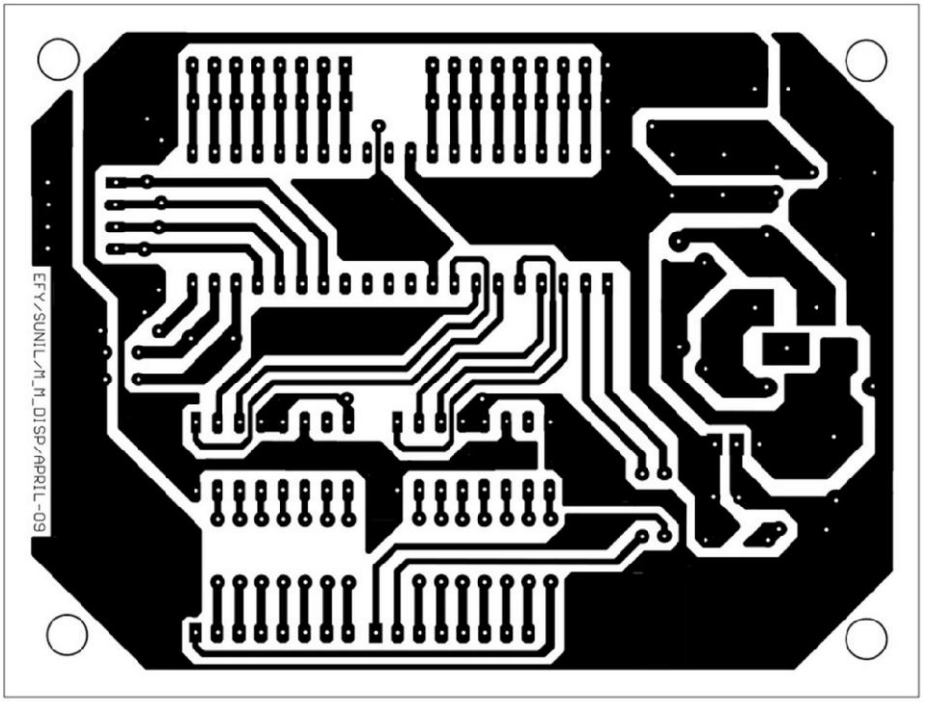


Fig. 3: Actual-size, single-side PCB for the microcontroller-based moving-message display circuit, except alphanumeric display and associated components

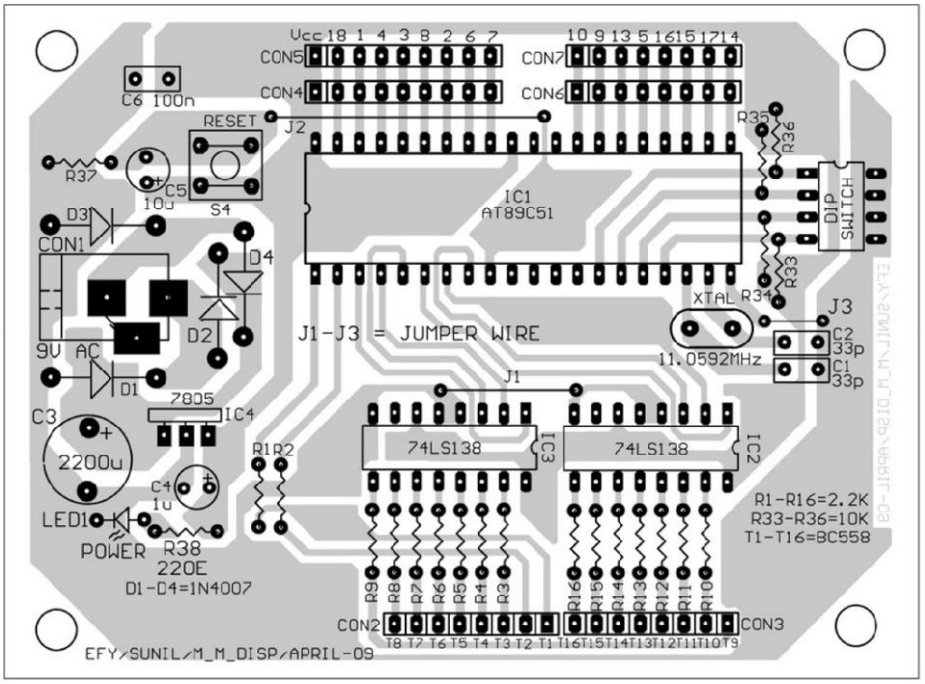


Fig. 4: Component layout for the PCB shown in Fig. 3

for the switching gap between two consecutive displays. Thus, each display is enabled for 1 ms while displaying a message. The length of this cycle depends upon the length of the message string. The cycle repeats after a '0' is encountered at the end of each message stored in the look-up table at the end of the program.

Each time, to display a character at a given display, first two bytes (16 bits) of data are sent to Port-2 and Port-0, then the desired display is enabled by sending its address to Port-1. Thereafter, a delay of 1 ms (slightly more than that) is generated by timer 1. Upon timer overflow, the entire display panel is refreshed by passing 'FFFFH' to the data bus. Then the next character at the next display is passed in the similar manner. The cycle frequency is variable (depending upon the length of the message) but always high enough so that the message appears continuous to the human eye.

Timer 0, with its interrupt enabled, is used to change the starting address of the message in cyclic manner so that the characters scroll from left to right with a proper gap between each shift. Meanwhile, the interrupt service sub-routine also checks for the starting address of DIS16 (right-most display). As soon as the first character reaches DIS16, the message stays for a longer time so that the entire message (message length not longer than 16 characters) can be easily read. Thereafter, characters again start scrolling rightwards, so the entire message goes out and disappears after a while to reappear from left side.

All the messages are stored in the form of a look-up table in the program memory (ROM) itself. When the circuit is switched 'on' (or reset), the monitoring program first checks for the binary number present at the selection

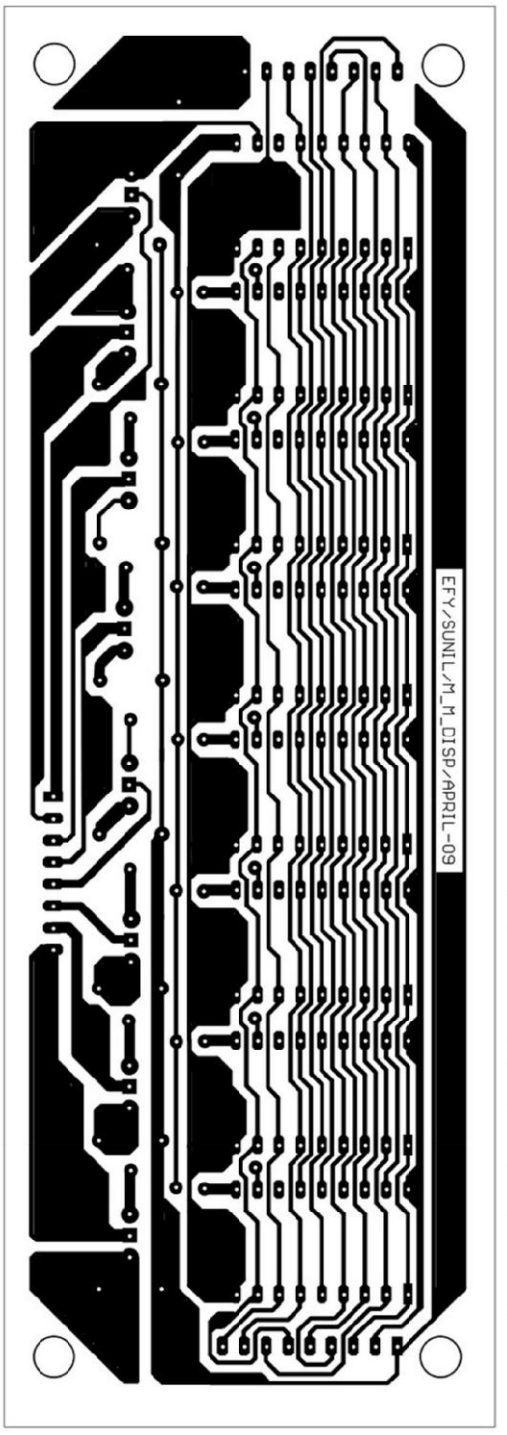


Fig. 5: Actual-size, single-side PCB for alphanumeric display

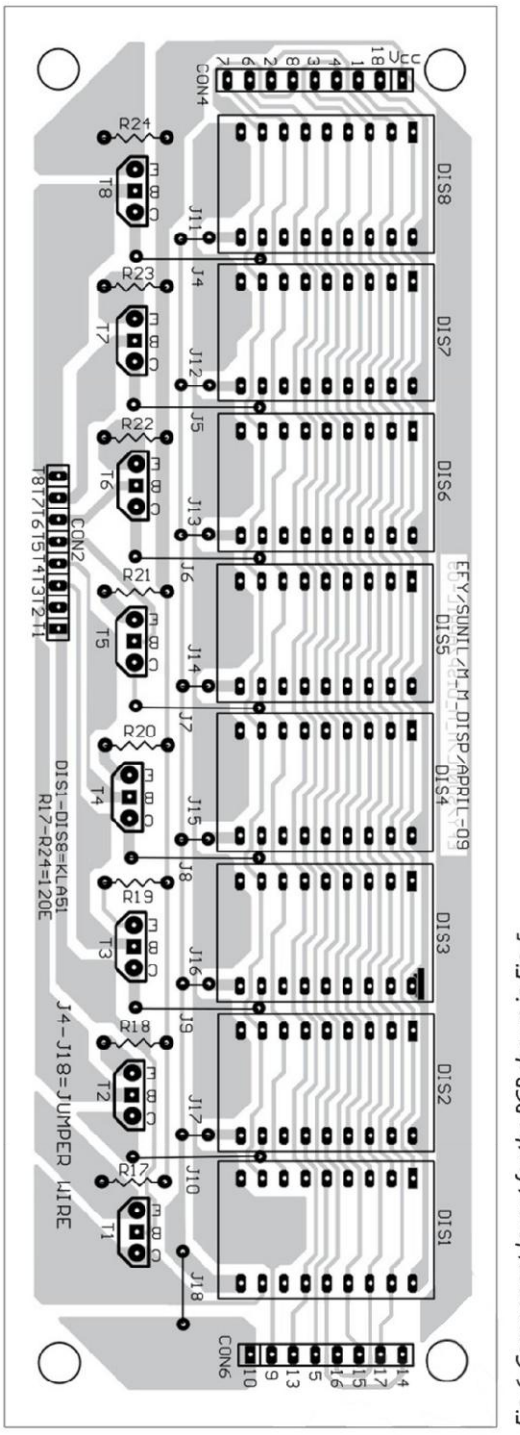


Fig. 6: Component layout for the PCB shown in Fig. 5

bus and according to that, the ROM address of the starting character of the selected message is loaded into the data-pointer. Thereafter, on-chip ROM reading is used to read the entire message over there.

Note that each character is represented in the look-up table of the source code by two bytes. For example, 'S' is represented by 'Sh' and 'Sl' separated by a comma. In addition to the alphabets, Arabic numerals and a few special characters have been defined in the program. For instance, a blank space is represented by 'bsh, bsl.' Thus, it is very easy to modify the program.

Suppose you want to display "HOUSE NO 401-H" in place of message '0.' First, open the source code in the

editor. Delete the old string and write the new string as below:

```
msg0: db Hh, Hl, Oh, Ol, Uh, Ul, Sh,
Sl, Eh, El, bsh, bsl, Nh, Nl, Oh,
Ol, bsh, bsl, fourh, fourl, zeroh,
zerol, oneh, onel, msh, msl, Hh, Hl, 0
```

(Please note that the assembler is case-insensitive. Still, upper and lower cases have been used for clarity.)

Future enhancements

Many more messages would be possible if complete Port-3 is used for message selection. Pins RxD, TxD, INT0 and INT1 have been kept free, so that these can be used for interfacing with the serial port of the PC. Also, interrupt pins can be used to display some message and sound an alarm in the case of an emergency. For example, a fire sensor can be connected to 'INT0' and a vibration detector to 'INT1.' These pins can also be used to send signals to synchronise a similar system that displays another related message at the same time, so a 16-character, two-line display is made possible.

Download Source Code: http://www.efymag.com/admin/issuepdf/Microcontroller-Based%20Message%20Display%20System.zip

MOVMSG.ASM

```
  $mod51
   DBH    equ p2 ; Higher byte of
Data Bus
   DBL    equ p0 ; Lower byte of
Data Bus
  ADB   equ p1 ; Address Bus
  input equ p3 ; message select
input

  ;** codes for decimal digits
are given below:
  ; ('h' refers to higher byte,
'l' to lower one)
  zeroh  equ 17h
  zerol  equ 0e8h

  oneh   equ 0d7h
  onel   equ 0ffh

  twoh   equ 23h
  twol   equ 0ech

  threeh equ 2bh
  threel equ 0fch

  fourh  equ 0c3h
  fourl  equ 0fbh

  fiveh  equ 0bh
  fivel  equ 0f8h

  sixh   equ 0bh
  sixl   equ 0e8h

  sevenh equ 0d7h
  sevenl equ 0fch

  eighth equ 03h
  eightl equ 0e8h

  nineh  equ 03h
  ninel  equ 0f8h

    ;** codes for alphabets are
given below:

  Ah equ 0c3h
  Al equ 0e8h

  Bh equ 0bh
  Bl equ 0ebh

  Ch equ 3fh
  Cl equ 0e8h

  Dh equ 03h
  Dl equ 0efh

  Eh equ 2bh
  El equ 0e8h

  Fh equ 0ebh
  Fl equ 0e8h

  GH equ 1bh
  Gl equ 0e8h

  Hh equ 0c3h
  Hl equ 0ebh

  Ih equ 0ffh
  Il equ 9fh

  Jh equ 17h
  Jl equ 0ffh

  Kh equ 0ech
  Kl equ 0ebh

  Lh equ 3fh
  Ll equ 0ebh

  Mh equ 0d5h
  Ml equ 0e3h

  Nh equ 0d6h
  Nl equ 0e3h

  Oh equ 17h
  Ol equ 0e8h

  Ph equ 0e3h
  Pl equ 0e8h

  Qh equ 06h
  Ql equ 0e8h

  Rh equ 0e2h
  Rlw equ 0e8h

  Sh equ 0bh
  Sl equ 0f8h

  Th equ 0ffh
  Tl equ 9ch

  Uh equ 17h
  Ul equ 0ebh

  Vh equ 0fdh
  Vl equ 6bh

  Wh equ 17h
  Wl equ 0abh

  Xh equ 0fch
  Xl equ 77h
```

```
  Yh equ 0e3h
  Yl equ 0bbh

  Zh equ 3dh
  Zl equ 7ch

  ;** codes for few special characters:

  strh   equ 0e8h ;for star sign (asterisk)
  strl   equ 17h

  plsh   equ 0ebh ;for '+' sign
  plsl   equ 9fh

  mnsh   equ 0ebh ;minus sign
  mnsl   equ 0ffh

  _h     equ 3fh  ;underscore sign
  _l     equ 0ffh

  bsh    equ 0ffh ;blank space
  bsl    equ 0ffh

  pieh   equ 0eah ;for pie
  piel   equ 7fh

  mueh   equ 0e3h ;for micro (mu)
  muel   equ 0ebh

  org 0000h
  sjmp main

  org 000bh             ;timer0 interrupt vector ad-
dress
      clr tr0           ;clear timer0 run bit
      mov tl0,#00h
       mov th0,#00h      ;reload timer0 with initial
count
      djnz r7,a1
      mov r7,#46
      cjne r1,#60h,a5  ;check to again start entering
from left-side
      sjmp a4
   a5: cjne r1,#50h,a2 ;check for display to stay on
reaching display-16
      sjmp a3
   a2: inc r1
      sjmp a1
   a3: djnz r6,a1
      inc r1
      sjmp a1
   a4: mov r6,#10
      mov r1,#41h
   a1: setb tr0         ;set timer0 run bit
       reti              ;return from timer0 ISR and
clear tf0

  main: mov ie,#00h
        setb ea             ;set global interrupt bit
        setb et0            ;enable timer0 interrupt
          mov tmod,#01h         ;timer0 configured in
mode 1
        mov tcon,#00h
        mov tl0,#00h
          mov th0,#00h         ;set initial count to
0000H
          mov r7,#46      ;provides gap between each
shift
        mov r6,#10     ;

          mov r0,#60h
   blank: mov @r0,#0ffh ;initialize the pointed loca-
tion by null address
          dec r0
          cjne r0,#2fh,blank
            mov r1,#41h    ;load address-pointer with
initial address


          mov  50h,#0dfh  ;address  for  16th  Display
(rightmost)
          mov 4fh,#0bfh ;address for 15th Display
          mov 4eh,#9fh  ;address for 14th Display
          mov 4dh,#7fh  ;address for 13th Display
          mov 4ch,#5fh  ;address for 12th Display
          mov 4bh,#3fh  ;address for 11th Display
          mov 4ah,#1fh  ;address for 10th Display
          mov 49h,#0fbh ;address for 9th Display
          mov 48h,#0f7h ;address for 8th Display
          mov 47h,#0f3h ;address for 7th Display
          mov 46h,#0efh ;address for 6th Display
          mov 45h,#0ebh ;address for 5th Display
          mov 44h,#0e7h ;address for 4th Display
          mov 43h,#0e3h ;address for 3rd Display
          mov 42h,#0fdh ;address for 2nd Display
          mov  41h,#0feh  ;address  for  1st  Display
(leftmost)

    chk:  mov a,input     ;load accumulator with value
at P3
        orl a,#0fh      ;mask lower nible to get se-
lection bus value
        cjne a,#0ffh,chk0
         mov dptr,#default ;load dptr with starting
address of defalt message
        sjmp read         ; now start reading

  chk0: cjne a,#0fh,chk1
         mov dptr,#msg0 ;load dptr with starting ad-
dress of msg0
        sjmp read         ; now start reading

  chk1: cjne a,#1fh,chk2
        mov dptr,#msg1
        sjmp read

  chk2: cjne a,#2fh,chk3
        mov dptr,#msg2
        sjmp read

  chk3: cjne a,#3fh,chk4
        mov dptr,#msg3
        sjmp read

  chk4: cjne a,#4fh,chk5
        mov dptr,#msg4
        sjmp read

  chk5: cjne a,#5fh,chk6
        mov dptr,#msg5
        sjmp read

  chk6: cjne a,#6fh,chk7
        mov dptr,#msg6
        sjmp read
```

```
  chk7: cjne a,#7fh,chk8
        mov dptr,#msg7
        sjmp read

  chk8: cjne a,#8fh,chk9
        mov dptr,#msg8
        sjmp read

  chk9: cjne a,#9fh,chk10
        mov dptr,#msg9
        sjmp read

  chk10: cjne a,#0afh,chk11
         mov dptr,#msg10
         sjmp read

  chk11: cjne a,#0bfh,chk12
         mov dptr,#msg11
         sjmp read

  chk12: cjne a,#0cfh,chk13
         mov dptr,#msg12
         sjmp read

  chk13: cjne a,#0dfh,chk14
         mov dptr,#msg13
         sjmp read

  chk14: mov dptr,#msg14
         sjmp read

  read: mov r3,dph
        mov r2,dpl
        setb tr0
   rd1: mov r0,01h
   rd2: clr a
        movc a,@a+dptr
        jz down
        mov DBH,a
        clr a
        inc dptr
        movc a,@a+dptr
        mov DBL,a
        mov ADB,@r0
        acall timer
        dec r0
        inc dptr
        sjmp rd2
  down: mov dph,r3 ;reload dph
        mov dpl,r2 ;reload dpl
        sjmp rd1

  timer: mov tmod,#10h ;set mode 1 for timer1
         mov th1,#0fch ;FC66H will generate a delay
of 1ms with 11.0592MHz Xtal
         mov tl1,#66h
         setb tr1
         jnb tf1,$ ;wait until timer1 overflows
         clr tr1
         clr tf1
         mov DBH,#0ffh
         mov DBL,#0ffh
         ret

  ;** look-up table starts from here:
   msg0:     db Hh,Hl,Ah,Al,Ph,Pl,Ph,Pl,Yh,Yl,bsh,bsl
,Bh,Bl,Ih,Il,Rh,Rlw,Th,Tl,Hh,Hl,bsh,bsl,Dh,Dl,Ah,Al
,Yh,Yl,0
   msg1:     db Hh,Hl,Ah,Al,Ph,Pl,Ph,Pl,Yh,Yl,bsh,bsl,
Nh,Nl,Eh,El,Wh,Wl,bsh,bsl,Yh,Yl,Eh,El,Ah,Al,Rh,Rlw,0
   msg2:     db strh,strl,bsh,bsl,Hh,Hl,Ah,Al,Ph,Pl,Ph
,Pl,Yh,Yl,bsh,bsl,Dh,Dl,Ih,Il,Wh,Wl,Ah,Al,Lh,Ll,Ih,I
l,bsh,bsl,strh,strl,0
   msg3:     db Mh,Ml,Eh,El,Rh,Rlw,Rh,Rlw,Yh,Yl,bsh,b
sl,Ch,Cl,Hh,Hl,Rh,Rlw,Ih,Il,Sh,Sl,Th,Tl,Mh,Ml,Ah,Al
,Sh,Sl,0
   msg4:     db strh,strl,bsh,bsl,Hh,Hl,Ah,Al,Ph,Pl,P
h,Pl,Yh,Yl,bsh,bsl,Hh,Hl,Oh,Ol,Lh,Ll,Ih,Il,bsh,bsl,
strh,strl,0
   msg5:     db strh,strl,bsh,bsl,Eh,El,Ih,Il,Dh,Dl,bs
h,bsl,Mh,Ml,Uh,Ul,Bh,Bl,Ah,Al,Rh,Rlw,Ah,Al,Kh,Kl,bsh
,bsl,strh,strl,0
   msg6:     db Hh,Hl,Ah,Al,Ph,Pl,Ph,Pl,Yh,Yl,bsh,bsl
,Dh,Dl,Ah,Al,Sh,Sl,Hh,Hl,Eh,El,Hh,Hl,Rh,Rlw,Ah,Al,0
   msg7:     db Hh,Hl,Ah,Al,Ph,Pl,Ph,Pl,Yh,Yl,bsh,bsl,W
h,Wl,Eh,El,Dh,Dl,Dh,Dl,Ih,Il,Nh,Nl,Gh,Gl,0
   msg8:     db Hh,Hl,Ah,Al,Ph,Pl,Ph,Pl,Yh,Yl,bsh,bsl
,Jh,Jl,Ah,Al,Nh,Nl,Mh,Ml,Ah,Al,Sh,Sl,Hh,Hl,Th,Tl,Mh
,Ml,Ih,Il,0
   msg9:     db strh,strl,bsh,bsl,Hh,Hl,Ah,Al,Ph,Pl,Ph
,Pl,Yh,Yl,bsh,bsl,Rh,Rlw,Ah,Al,Kh,Kl,Hh,Hl,Ih,Il,bsh
,bsl,strh,strl,0
   msg10:    db strh,strl,bsh,bsl,Hh,Hl,Ah,Al,Ph,Pl,Ph
,Pl,Yh,Yl,bsh,bsl,Ph,Pl,Oh,Ol,Nh,Nl,Gh,Gl,Ah,Al,Lh,L
l,bsh,bsl,strh,strl,0
   msg11:    db Hh,Hl,Ah,Al,Ph,Pl,Ph,Pl,Yh,Yl,bsh,bsl
,Mh,Ml,Oh,Ol,Th,Tl,Hh,Hl,Eh,El,Rh,Rlw,Sh,Sl,Dh,Dl,A
h,Al,Yh,Yl,0
   msg12:    db strh,strl,bsh,bsl,Hh,Hl,Ah,Al,Ph,Pl,Ph
,Pl,Yh,Yl,bsh,bsl,Rh,Rlw,Ah,Al,Mh,Ml,Jh,Jl,Ah,Al,Nh,
Nl,bsh,bsl,strh,strl,0
   msg13:    db strh,strl,bsh,bsl,Hh,Hl,Ah,Al,Ph,Pl,Ph
,Pl,Yh,Yl,bsh,bsl,Lh,Ll,Oh,Ol,Hh,Hl,Rh,Rlw,Ih,Il,bsh
,bsl,strh,strl,0
   msg14:    db strh,strl,bsh,bsl,Hh,Hl,Ah,Al,Ph,Pl,Ph
,Pl,Yh,Yl,bsh,bsl,Eh,El,Ah,Al,Sh,Sl,Th,Tl,Eh,El,Rh,R
lw,bsh,bsl,strh,strl,0
   default: db Wh,Wl,Eh,El,Lh,Ll,Ch,Cl,Oh,Ol,Mh,Ml,Eh
,El,bsh,bsl,Th,Tl,Oh,Ol,bsh,bsl,Ah,Al,Lh,Ll,Lh,Ll,0
            end
```

LED LIGHT CHASER FOR FIVE LIGHTING EFFECTS

■ A.M. BHATT

Light chaser circuits can be used to create lighting animation sequences. These have been used in the past to attract attention for advertising, event promotion (such as the marquees at the local movie theatres) and decoration. These can also be used to produce pleasing effects for entertainment.

The technology has now reached the point where everyone can afford to build light chasers, which may be just what you need for your next gathering or party. Most of them have some preprogrammed sequences which change automatically according to your choice, while others are programmed to generate particular patterns like pictures and graphical designs.

Here is one such application based on AT89C51 microcontroller that generates five different lighting effects using 120 LEDs connected to 24 input/output (I/O) lines of the microcontroller. The salient features of this circuit are:

1. Each of the 24 I/O lines having fives LEDs connected to it in series
2. Changeable time delay to increase or decrease the speed of effect
3. Eight pushbutton switches for different operations like selection of light effect, stop and change of delay
4. Reprogrammable because of AT89C51 microcontroller
5. Software can be extended for many different effects

Circuit description

Light chasers consist of several lighting circuits (usually three or four) strung together. Every first light in the string is turned on and then off, followed by the next light turning on and then off, and so on. There are 24 strings of lights consisting of five LEDs each, and 24 output lines from IC1 for controlling these lights, to create different lighting effects.

Fig. 1 shows the circuit diagram of the microcontroller-based LED chasing effect generator. At the heart of this circuit is microcontroller AT89C51. The AT89C51 is a low-power, high-performance 8-bit microcontroller with 4kB flash programmable and erasable read-only memory. The on-chip flash allows the program memory to be reprogrammed in-system. It is a powerful microcontroller used in many embedded control applications. It has 128 bytes of RAM, 32 I/O lines, two 16-bit timers/counters, a five-vector two-level interrupt architecture, a full-duplex serial port, on-chip oscillator and clock circuitry. The power-down mode saves the RAM contents but freezes the oscillator disabling all other chip functions until the next hardware reset.

Port 1 is an 8-bit, bidirectional I/O port with internal pullups. Port pins P1.0 through P1.7 are connected to push-to-on switches S1 through S8, respectively. Switches S1 through S5 are used for five different light effects, while switches S6, S7 and S8 are used to stop all the light effects, decrement the speed of the current effect and increase the speed of the current effect, respectively.

Port 0 is an 8-bit, open-drain, bidirectional I/O port. It is pulled up through 10-kilo-ohm resistor network RNW1. Port-0 pins P0.0 through P0.7 are connected to pins 1 through 8 of ULN2803 (IC2), respectively. When any of the port-0 pins goes high, the respective output of ULN2803 goes low to drive the connected LEDs.

The ULN2803 consists of eight npn Darlington-connected transistors, which are ideally suited for interfacing between the low-logic-level digital circuitry and the higher current/voltage requirement circuits of lamps, relays or other similar loads for a broad range of computer, industrial and consumer applications. It features open-collector outputs with free-wheeling clamp diodes for transient suppression.

Similarly, when any of port-2 or port-3 pins goes high, the respective output of ULN2803 goes low to drive the connected LEDs. Resistors R2 through R25 limit the current through the LED lines. As mentioned before, a total of five LEDs are connected in series to each output line of ULN2803.

A 12MHz quartz crystal is connected to XTAL1 and XTAL2 of the microcontroller to provide the basic clock. XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator. The output of the internal inverting amplifier is divided by two flip-flops. A high on reset pin 9 of the microcontroller for two machine cycles, while the oscillator is running, resets the device. Power-on reset signal is provided by the combination of capacitor C3 and resistor R1. Using switch S9, you can manually reset the microcontroller.

Fig. 2 shows the circuit diagram of the power supply. The 230V AC mains is stepped down by transformer X1 to deliver the secondary output of 18V, 1A. The transformer output is rectified by a full-wave bridge rectifier comprising diodes D2 through D4, filtered by capacitor C4 and then

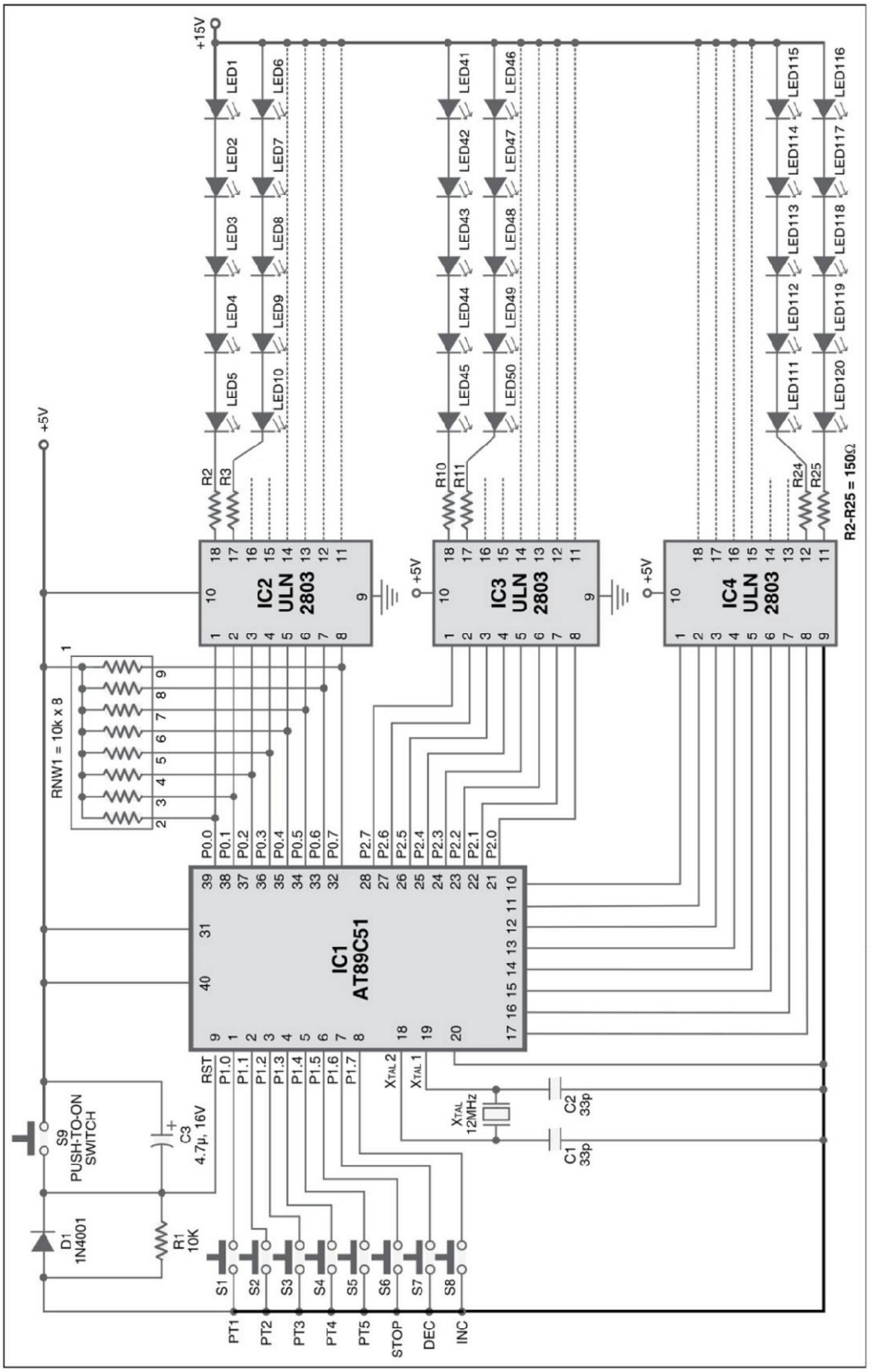


Fig. 1: Circuit of microcontroller-based LED light chaser

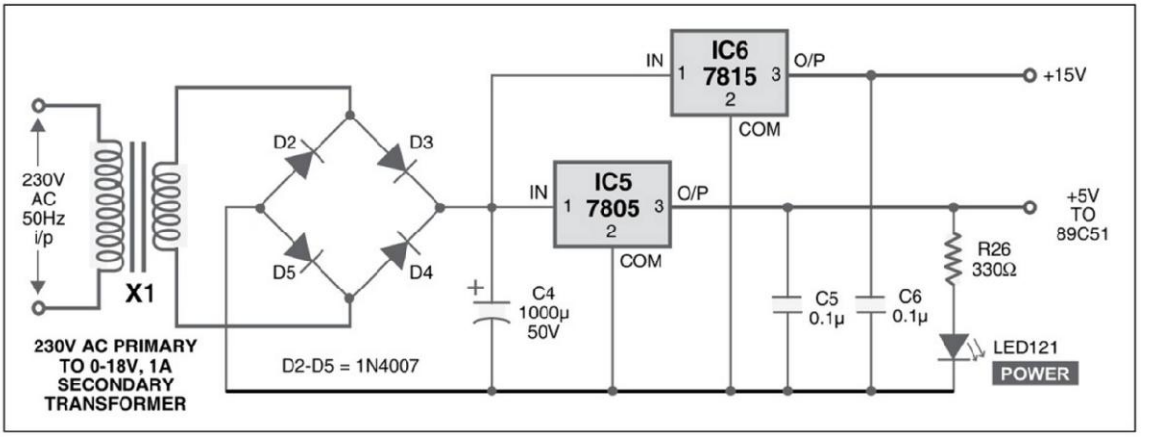


Fig. 2: Power supply circuit

regulated by IC 7805 (IC5) and 7815 (IC6). Regulator ICs 7805 and 7815 provide regulated 5V and 15V, respectively. Capacitors C5 and C6 bypass the ripples present in the regulated power supply. LED121 acts as the power-'on' indicator and resistor R26 limits the current through LED121.

An actual-size, single-side PCB for the microcontroller-based LED chaser is shown in Fig. 3 and its component layout in Fig. 4.

Operation

The complete operation depends upon the eight pushbuttons. Refer to the table for the functions assigned to every pushbutton.

Initially, after power-'on' reset or manual reset, all the LEDs blink for five times and then turn off. Now, if you press any of switches S1 through S5, that effect restarts. This will continue until you press any other key. One can change the effect in between by pressing a button other than S1 through S5. During the effect, if you press switch S7, the speed will decrement, while pressing switch S8 will increment the speed. The speed can be decreased/ increased in five steps. Pressing switch S6 will make all the LEDs blink again for five times and then turn them off. This operation is totally based on the software embedded in 89C51 chip.

PARTS LIST

Semiconductors:

IC1	- AT89C51 microcontroller
IC2-IC4	- ULN2803 eight Darlington array
IC5	- 7805, 5V regulator
IC6	- 7815, 15V regulator
D1	- 1N4001 rectifier diode
D2-D5	- 1N4007 rectifier diode
LED1-LED121	- 5mm LED

Resistors (all ¼-watt, ±5% carbon):

R1	- 10-kilo-ohm
R2-R25	- 150-ohm
R26	- 330-ohm
RNW1	- 10-kilo-ohm

Capacitors:

C1, C2	- 33pF ceramic disk
C3	- 4.7µF, 16V electrolytic
C4	- 1000µF, 50V electrolytic
C5, C6	- 0.1µF ceramic disk

Miscellaneous:

X1	- 230V AC primary to 18V, 1A secondary transformer
X_{TAL}	- 12MHz crystal
S1-S9	- Push-to-on switch

Software

The software is written in 'C' language and compiled using Keil µVision 3. The step-by-step procedure to build a project using Keil µVision 3 IDE is as follows:

1. Open Keil program from 'start' menu or from the desktop icon. Three windows, namely, project workspace, editing window and output window, appear. The project workspace window shows all the relevant files for your project. Editing window is the place where you can edit the code. The output window shows the output when you compile, build or run your project.

2. Now open 'project' option and select 'new project.' Here, first create a new folder named 'chaser' and give a project name 'led-chaser.'

3. Now select the target device '8051' from 'generic' option and click 'ok' button.

4. A message asking whether to add start up the code or not ap-

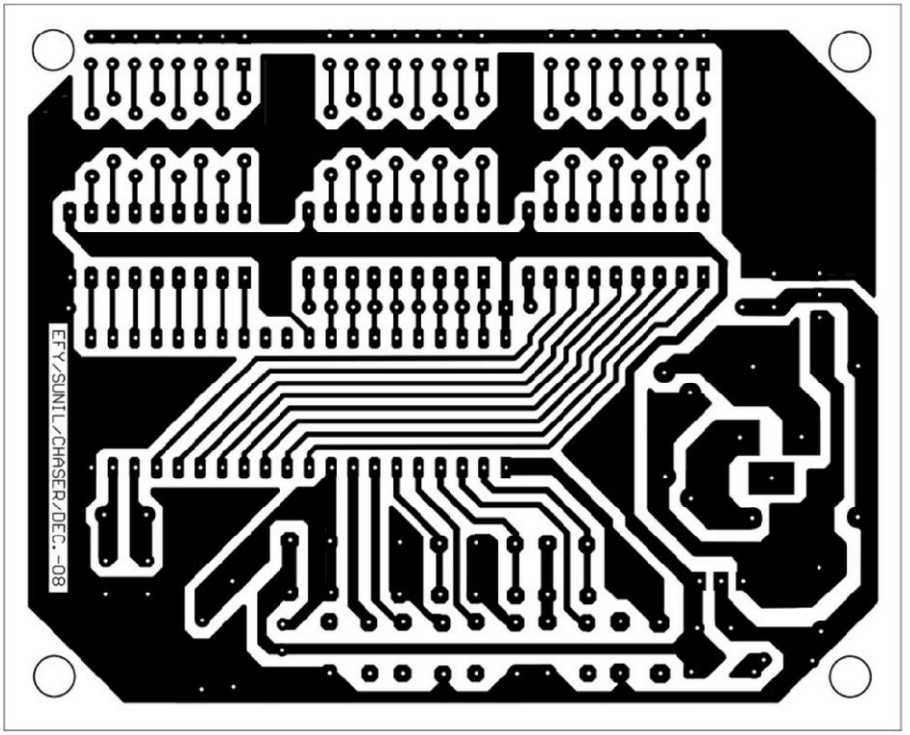


Fig. 3: A single-side, actual-size PCB layout for LED light chaser

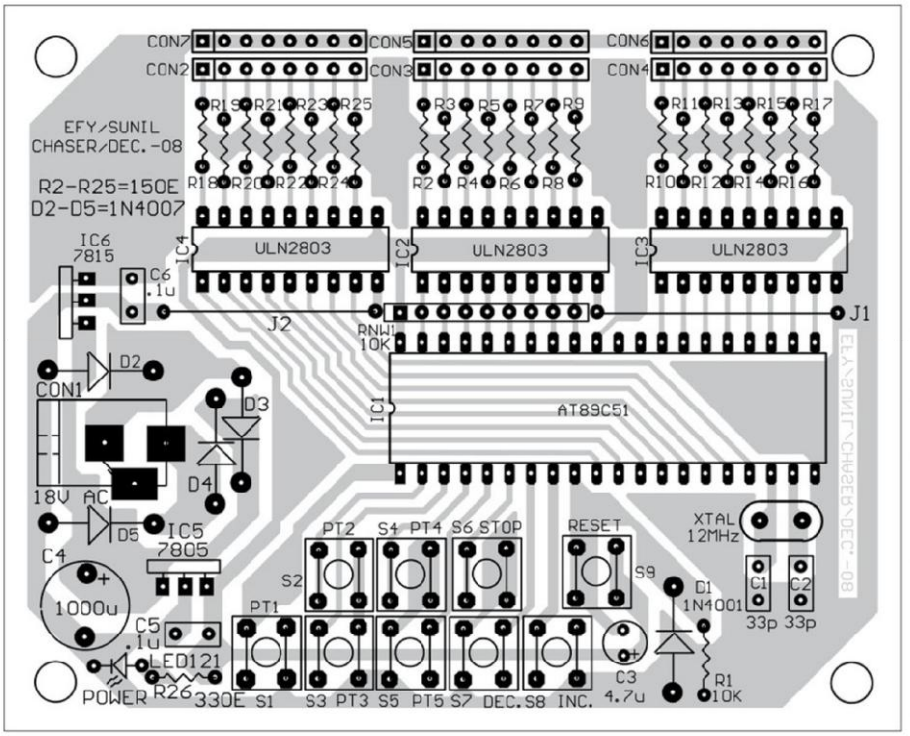


Fig. 4: Component layout for the PCB

pears. Click 'yes' and go ahead.

5. Now from 'file' menu, select 'new file.' The editor window opens, where you can start writing your 'C' code.

6. Right-click the source file in workspace window and click 'add files' in the pop-up menu. Then add file 'reg51.h.' Thereupon, all the special-function registers are initialised.

7. After writing the program, save the file as '.c' extension. Now in project workspace window, right-click 'source group 1.' Select 'add files to source group 1' from pop-up menu. Next, select '.c' file and click 'add'. Now you can see your 'c' file added in the source group.

8. To compile the project, click 'project' menu and select 'build target.' You can see the progress in output window. If there is any compilation error, your target will not be created. So rectify all the errors and then build the target again until you get '0 – Errors, 0 - warnings' message and '.hex' code is generated for burning into the microcontroller.

In initialisation part, the program initialises all the ports as either input or output, then it blinks all the LEDs for five times to check whether all the LEDs are okay or not. Thereafter, the program enters the continuous loop (while), to check whether any key is pressed or not.

In key detection part, the switch statement is used to find out which of the eight keys is pressed. Depending upon the key pressed, a particular case is executed.

Pattern generation is the main part of the whole program. One pattern is assigned with first five keys and for each pattern there is one function. So whenever the key (out of S1 through S5) is pressed, that particular function is called. Then that function is executed until any other key is pressed.

The functions used in this program are:

Effect1( ): Rotates bit '0' on all port pins. It starts from P0.0 to P0.7, then from P2.0 to P2.7 and then from P3.0 to P3.7. Next, it reverses from P3.7 to P0.0. So all the LEDs are 'on,' but at a time one line is low.

Effect2( ): Same as above, except that here bit '1' is rotated. That means all the LEDs are 'off' but at a time one line is high.

Effect3( ): The simplest function. It makes all the LEDs blink alternatively. That means first all the odd lines are high and even lines are low, then even lines are high and odd lines are low.

Effect4( ): Rotates bit '0' forward and backward simultaneously on all three ports. That is, at a time bit '0' is rotated from P0.0, P2.0, P3.0 to P0.7, P2.7, P3.7 and then from P0.7, P2.7, P3.7 to P0.0, P2.0, P3.0.

Effect5( ): Rotates bit '1' from both the ends to the centre and then from the centre to both the ends. Means starting from P0.0 and P3.7 to P2.3 and P2.4. Then in reverse manner from P2.3 and P2.4 to P0.0 and P3.7.

Functions Assigned to Pushbutton Switches

Switch	Function
S1	Start chasing effect 1
S2	Start chasing effect 2
S3	Start chasing effect 3
S4	Start chasing effect 4
S5	Start chasing effect 5
S6	Stop all effects
S7	Decrease the speed of current effect
S8	Increase the speed of current effect

Delay( ): Generates delay of approximately 0.1 second and is used for key debounce.

Dly(int d): The only function with one argument. It generates variable delay according to the value passed to it. This is used to increase or decrease the speed of effect.

Incdly( ): Increases the value of variable 'd' that is passed to dly(int d) function. So delay increases and speed decreases.

Decdly( ): Decreases the value of variable 'd' that is passed to dly(int d) function. So delay decreases and speed increases.

Download Source Code: http://www.efymag.com/admin/issuepdf/Light%20Chaser.zip

INTERFACING NOKIA COLOUR LCD WITH AVR MICROCONTROLLER

■ ARUN DAYAL UDAI

Adding an elegant-looking colour LCD to a project is a dream for robotics enthusiasts. Usually, colour LCDs are costly and it's difficult to find the technical information for their interfacing.

Using the colour LCD of your old mobile phone could be a solution. Here we describe how to use the colour LCD of a Nokia handset (model 6100, 7210, 6610, 7250 or 6220) with Philips PCF8833 chipset through ATmega2560 ATMEL AVR microcontroller. These LCDs are readily available and inexpensive even if you buy a new one. The author's prototype is shown in Fig. 1. With this project, you can easily load a colour picture of 132×132 pixels, in 12-bit RGB (red-green-blue) format, to the colour LCD through your PC's serial port.

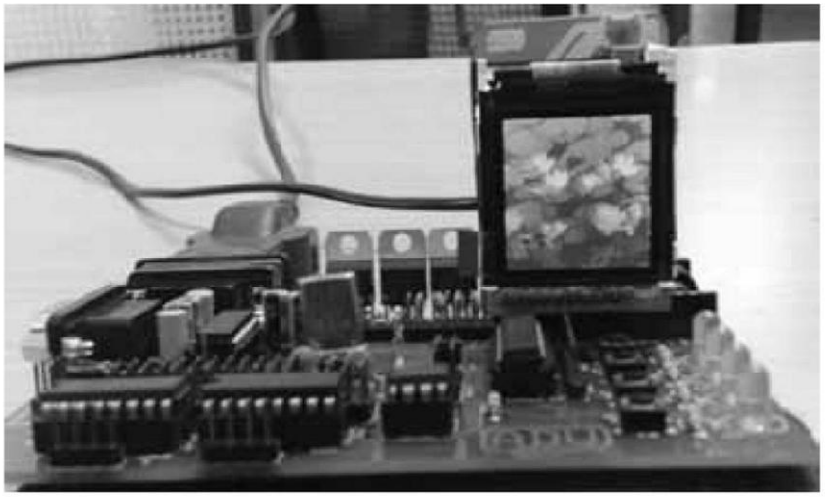


Fig. 1: Author's prototype

ATmega2560 microcontroller

The ATmega2560 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. The AVR core combines a rich instruction set with 32 general-purpose working registers. All the 32 registers are directly connected to the arithmetic logic unit, allowing two independent registers to be accessed in one single in-

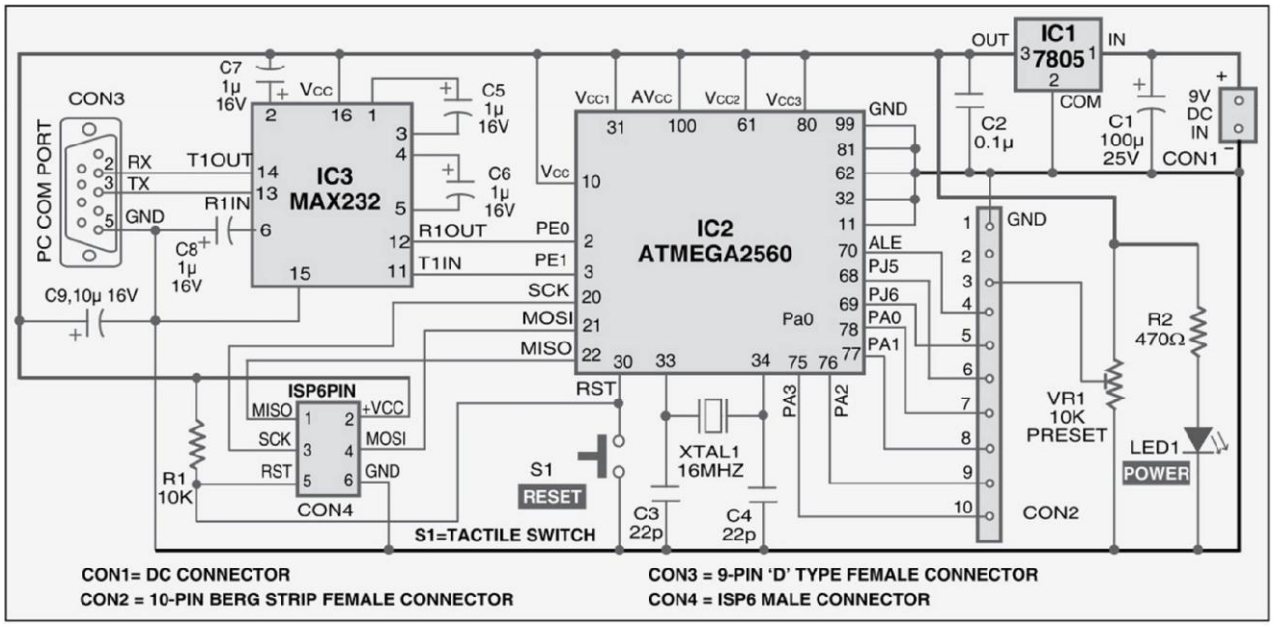


Fig. 2: Circuit for interfacing the Nokia colour LCD with avr microcontroller

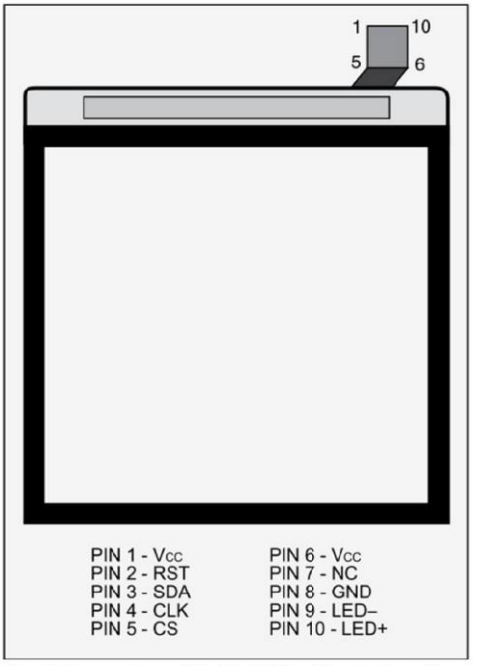


Fig. 3: Front view of Nokia LCD with pin details of the connector

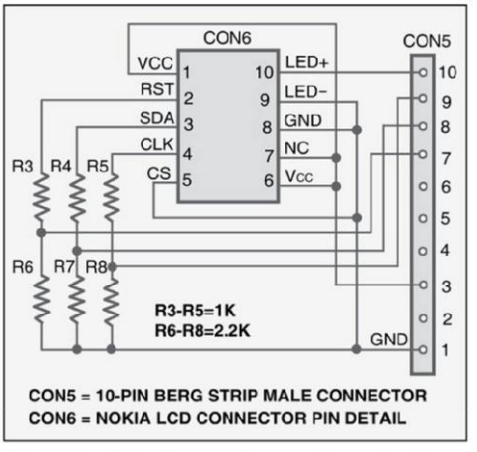


Fig. 4: Circuit for Nokia lcd connector extension

R	R	R	R	G	G	G	G	RED AND GREEN FOR 1ST PIXEL
B	B	B	B	R	R	R	R	BLUE FOR 1ST AND RED FOR 2ND PIXEL
G	G	G	G	B	B	B	B	GREEN AND BLUE FOR 2ND PIXEL

Fig. 5: Standard 12-bit image

Fig. 6: GUI created using VB.net for image transfer from the PC

struction executed in one clock cycle. The resulting architecture is more code-efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The microcontroller has 256 kB of in-system programmable Flash with read-while-write capabilities, 4kB EEPROM, 8kB SRAM, 86 general-purpose input/output (I/O) lines, real-time counter, six flexible timers/counters with compare modes and pulse-width modulation (PWM), four USARTs, a byte-oriented two-wire serial interface, a 16-channel, 10-bit analogue-to-digital converter (ADC) with optional differential input stage with programmable gain, programmable watchdog timer with internal oscillator, a serial peripheral interface (SPI) port, IEEE standard 1149.1 compliant JTAG test interface (also used for accessing the on-chip debug system and programming) and six software-selectable power-saving modes.

PCF8833 LCD driver

The PCF8833 is a single-chip, low-power CMOS LCD controller driver, designed to drive colour super-twisted nematic displays of 132 rows and 132 RGB columns. All the necessary functions for the display are provided in a single chip, including the display RAM which has a capacity of 209 kbits. The PCF8833 uses multiple-row addressing technique in order to achieve the best optimal performance at the lowest power consumption. It offers two types of microcontroller interfaces: 8080 system interface and 3-line serial interface.

The PCF8833 communicates with the host using an 8-bit parallel interface or 3-line serial interface. Here a 3-line serial interface is implemented for communication between the microcontroller and the PCF8833 chip. The three lines are chip-select (CS) or enable pin, serial clock (SCLK) and serial data (SD).

Processing the instructions and data sent to the interface does not require the display clock. The display clock and interface clock are independent of each other. The display clock is derived from the built-in oscillator. The serial data pin of PCF8833 is connected to the SDA pin of the microcontroller.

Circuit description

Fig. 2 shows the circuit for interfacing the Nokia colour LCD with Atmega2560 microcontroller. The circuit is powered from a standard 9V DC source. The 9V DC is converted into 5V DC using a 7805 regulator (IC1). The glowing of LED1 shows the presence of power in the circuit. The regulated 5V supply powers the circuit including the Atmega2560, MAX232, LCD connector (CON2) and ISP6 connector (CON4). CON3 is a 9-pin, D-type COM port connector used to interface with the PC for picture file transfer to the LCD through the microcontroller (IC2). ISP6 connector is used for programming the microcontroller using STK500 board.

Whenever the program doesn't function properly, you can reset the microcontroller by momentarily pressing reset switch S1.

Pin details of Nokia's 10-pin LCD connector are shown in Fig. 3. A manual implementation of the SPI with the microcontroller is done through SDA, CLK and CS pins on the LCD connector. First, the clock pin is cleared, then the data pin is set or cleared depending on the data bit and the clock pin set. SDA and CLK pins of the LCD connector are connected to pins 77 (PA1) and 76 (PA2) of the microcontroller, respectively.

The 8-bit data is transferred through the SPI in eight clock cycles. CS pin of the LCD chip is permanently made low. Backlight LED pin for the LCD may be connected to a PWM pin to have a dimming effect depending on the duty cycle of PWM.

The LED and Reset pins of the LCD are connected to pins 75 (PA3) and 78 (PA0) of the microcontroller, respectively. When PA0 is made low, it resets the LCD. All the lines PA0 to PA3 is made an output pin by setting the data direction register (DDRA) bits high.

This connector is too small to handle and solder using a normal soldering iron. You can extend the connections to a standard-size berg strip male connector as shown in Fig. 4. The LCD connector extension may be developed with resistors R3 through R5 (1 kilo-ohm each) and R6 through R8 (2.2 kilo-ohms each) forming the voltage divider circuits to obtain 3.3V signals (acceptable by the colour LCD) from 5.5V TTL signals.

Note that this connector extension is made for use with a colour LCD as well as standard character LCD. The LCD receives 3.3V supply through preset VR1. VR1 is also used to adjust the brightness of the colour LCD. Contrast is adjusted through software program.

Make a PCB layout and glue the board on the back side of the LCD module. Now you can easily fit the LCD onto the 10-pin berg strip socket in your ATmega2560 application board.

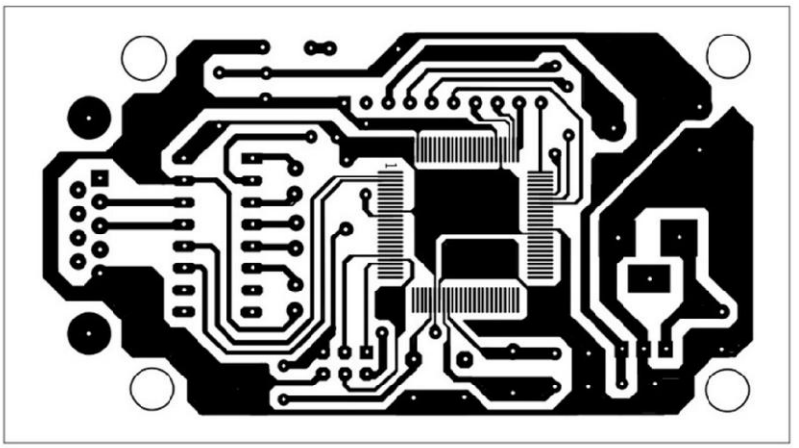


Fig. 7: An actual-size, single-side PCB for interfacing the Nokia colour LCD with the AVR microcontroller circuit

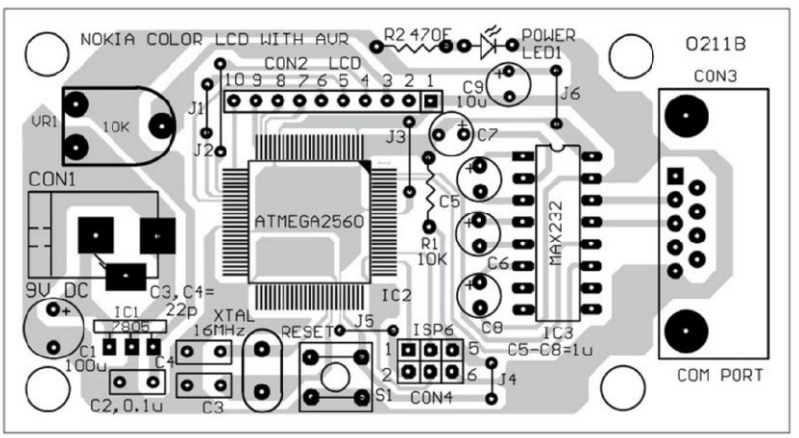


Fig. 8: Component layout for the PCB in Fig. 7

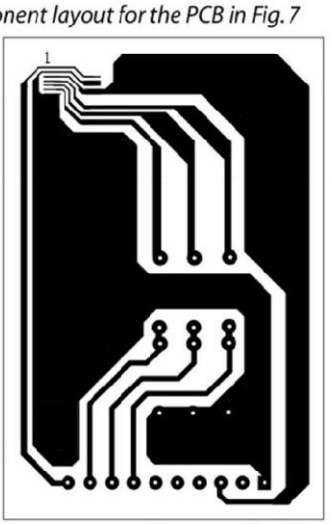


Fig. 9: An actual-size, single-side PCB for the LCD extension circuit

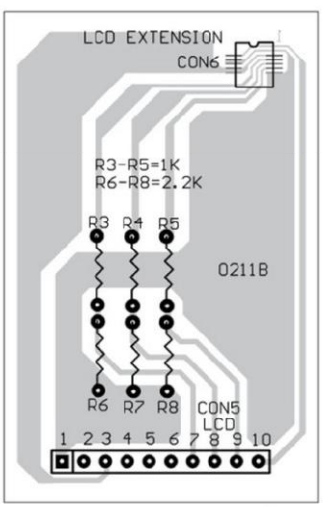


Fig. 10: Component layout for the PCB in Fig. 9

Software program

The software for the microcontroller is written in 'C' language using the IAR Embedded Workbench integrated development environment. IAR Embedded Workbench is being developed by IAR Systems and ATMEL developers in parallel and hence it generates the optimised code which uses full 'C' coding capabilities of AVR devices. AVR development tools for embedded systems can

PARTS LIST

Semiconductors:

IC1	- 7805, 5V regulator
IC2	- ATmega2560 microcontroller
IC3	- MAX232 RS-232 driver
LED1	- 5mm light-emitting diode

Resistors (all ¼-watt, ±5% carbon):

R1	- 10-kilo-ohm
R2	- 470-ohm
R3-R5	- 1-kilo-ohm
R6-R8	- 2.2-kilo-ohm
VR1	- 10-kilo-ohm preset

Capacitors:

C1	- 100µF, 25V electrolytic
C2	- 0.1µF ceramic
C3, C4	- 22pF ceramic
C5-C8	- 1µF, 16V electrolytic
C9	- 10µF, 16V electrolytic

Miscellaneous:

CON1	- DC connector
CON2	- 10-pin berg strip female connector
CON3	- 9-pin D-type female connector
CON4	- ISP 6-pin male connector
CON5	- 10-pin berg strip male connector
CON6	- 10-pin connector for Nokia LCD
S1	- Tactile switch
	- Nokia 6100 colour LCD module

be downloaded for free from IAR website www.iar.com. For details of IAR Embedded Workbench, you may refer to 'A Beginners' Guide to ATMEL AVR Development' article published in January issue of EFY.

To start the application, click 'IAR Embedded Workbench' icon in 'All Programs' menu of Windows. From the main menu, select Project→Create New Project and start an AVR Studio compatible C project. Generate the Intel hex file for burning the code into Atmega2560 microcontroller as follows: Project→Options→Linker (under Category)→Extra Output→Generate Extra Output (to click)→Override Default (to click)→Type Project Name. hex→Output Format→intel-standard (to select).

Press 'ok' after all these settings are done. The source code for LCD interfacing (main.c) along with comments for instructions is included in this month's EFY-CD. Import the main.c file from the CD into the editor screen by clicking 'Add Files...' option in Project menu. Click 'Compile' option from Project menu to compile and generate main. hex code.

To send a command, a low bit is sent first before sending an 8-bit command, forming a 9-bit SPI command signal. Similarly, before sending an 8-bit data, a high bit is sent forming a 9-bit SPI data. You may refer to the datasheet of PCF8833 for various instructions used in the LCD driver chip. The code for this project (main.c) has been programmed as per these instructions.

If you can draw a point with a colour at any given location on the LCD screen, you can implement various graphics algorithms to draw a circle, line, filled rectangle, etc. You can have the AVR microcontroller fitted with a 16.00MHz external crystal oscillator, a serial RS-232 driver connecting a serial port and a reset arrangement to run the code.

Downloading the code into Atmega2560. To program the Atmega2560, connect the 6-wire cable between ISP connectors of the STK500 board (marked as ISP6) and the target board (marked as ISP con). Connect a serial cable from 'RS232 CTRL' connector on the STK500 board to a COM port on the PC. Now start AVR Studio 4.0 without opening any project file.

Optionally, you may proceed as follows: Main Menu→Tools→Program AVR→Select AVR Programmer. Press 'Connect...' after selecting STK500 or AVRISP in the platform window and COM port (say, COM1) in the port window. Next, select Atmega2560 as 'Device,' ISP as 'Programming Mode' and browse your project hex file from Project→Debug→Exe folder. Press 'Program' button on the STK500 dialogue box to burn the hex file into your microcontroller.

Converting the image file. You may convert any image file into a .raw file of 132×132 size, in 12-bit RGB colour, using Adobe Photoshop.

A standard 12-bit image encodes the two pixels in three bytes of data (24 bits) as shown in Fig. 5. In a 12-bit image, each pixel has three colour components of 4-bit value (0 to 15). So each of red, green and blue can vary from 0 to 15. Now, as a single COM port communication or any serial communication has 8-bit data and 12 bits are one-and-a-half byte, the two pixels are transferred so that 12×2 = 24-bit value is sent in three successive transfers of 8-bit value. Same is the case when storing these pixels in a file or a memory which stores a byte value only. The two pixels having 24 bits can be stored in three bytes of data.

A front-end (user-interface) program for transferring a picture file to the AVR microcontroller may be designed using a suitable software.

Using GUI to transfer the image. The user-interface program for transferring the picture file (.raw) from your PC to the Nokia LCD is designed in Visual Basic.NET platform. The send_picture.exe file is already included in the EFY-CD of this issue. When you click this file, you will see a GUI window as shown in Fig. 6.

In the first drop-down menu, select the appropriate com port. The image to be transferred should be saved in *.raw format. Select that image using 'Browse' option in the second pane and press 'Send' button to transfer the image to the graphical LCD. Now the image is displayed on the LCD.

Writing a text to the screen is similar to drawing a bitmap with predefined height, width and colour of characters.

Construction

An actual-size, single-side PCB for interfacing the Nokia colour LCD with the AVR microcontroller circuit (Fig. 2) is shown in Fig. 7 and its component layout in Fig. 8. An actual-size, single-side PCB for the LCD extension circuit (Fig. 4) is shown in Fig. 9 and its component layout in Fig. 10.

Mount all the components as shown in the PCB layouts. The Atmega2560 used in the prototype comes in a 100-pin, surface-mount TQFP package. By using surface-mount devices (SMDs) in your project, you don't have to drill holes in the PCB and the board size will be much smaller. But it may be difficult to handle and solder SMDs without an SMD soldering station. However, with a normal soldering iron, tweezers and some basic knowledge, you can solder SMD components without much problem. You need good eyes, a steady hand and a soldering iron with a small, clean tip.

Make sure that the PCB is clean. You may use copper polish material to remove all kinds of oxidation and acetone (nail polish remover) to remove all kinds of contaminations on the PCB.

On the PCB, figure out the correct location for placing the SMDs. Fixate the SMD at the top right corner and the bottom left corner. Carefully solder SMD pins one by one. SMD components are very sensitive to heat, so allow your SMD to cool down after every step. For more tips on soldering SMD components, you may refer to http://hem.passagen.se/communication/pcbsmd.html and www.engadget.com/2006/03/07/how-to-make-a-surface-mount-soldering-iron/ websites.

After soldering the components on the PCBs, insert the LCD connector extension board in the main interfacing PCB using the connector marked 'CON2.' With the LCD screen facing towards you, the leftmost pin of the berg strip is pin 1. Make sure that the orientation of the LCD connector is correct.

Now connect 9V DC source to the circuit and switch on the power supply. LED1 should glow to indicate the presence of power in the circuit. At this point, you will see a test colour band drawn by itself on the LCD screen and then waits for serial port to get 12-bit RGB image from PC at 115.2 kbps baud rate. Now run the send_picture.exe file and send the picture file (.raw) to the LCD from your computer.

Download source code: http://www.efymag.com/admin/issuepdf/NOKIA-Color-LCD-Interfacing.zip

Robotics

CELLPHONE-OPERATED LAND ROVER

■ SUSRAM RAHUL K. & RAGHAVENDRA PRASAD

Conventionally, wireless-controlled robots use RF circuits, which have the drawbacks of limited working range, limited frequency range and limited control. Use of a mobile phone for robotic control can overcome these limitations. It provides the advantages of robust control, working range as large as the coverage area of the service provider, no interference with other controllers and up to twelve controls.

Although the appearance and capabilities of robots vary vastly, all robots share the features of a mechanical, movable structure under some form of control. The control of robot involves three distinct phases: preception, processing and action. Generally, the preceptors are sensors mounted on the robot, processing is done by the on-board microcontroller or processor, and the task (action) is performed using motors or with some other actuators.

PARTS LIST

Semiconductors:

IC1	- MT8870 DTMF decoder
IC2	- ATmega16 AVR microcontroller
IC3	- L293D motor driver
IC4	- 74LS04 NOT gate
D1	- 1N4007 rectifier diode

Resistors (1/4-watt, ±5% carbon, unless stated otherwise)

R1, R2	- 100-kilo-ohm
R3	- 330-kilo-ohm
R4-R8	- 10-kilo-ohm

Capacitors:

C1	- 0.47µF ceramic disk
C2, C3, C5, C6	- 22pF ceramic disk
C4	- 0.1µF ceramic disk

Miscellaneous:

XTAL1	- 3.57MHz crystal
XTAL2	- 12MHz crystal
S1	- Push-to-on switch
M1, M2	- 6V, 50-rpm geared DC motor
Batt.	- 6V, 4.5Ah battery

Project overview

In this project, the robot is controlled by a mobile phone that makes a call to the mobile phone attached to the robot. In the course of a call, if any button is pressed, a tone corresponding to the button pressed is heard at the other end of the call. This tone is called 'dual-tone multiple-frequency' (DTMF) tone. The robot perceives this DTMF tone with the help of the phone stacked in the robot.

The received tone is processed by the ATmega16 microcontroller with the help of DTMF decoder MT8870. The decoder decodes the DTMF tone into its equivalent binary digit and this binary number is sent to the microcontroller. The microcontroller is preprogrammed to take a decision for any given input and outputs its decision to motor drivers in order to drive the motors for forward or backward motion or a turn.

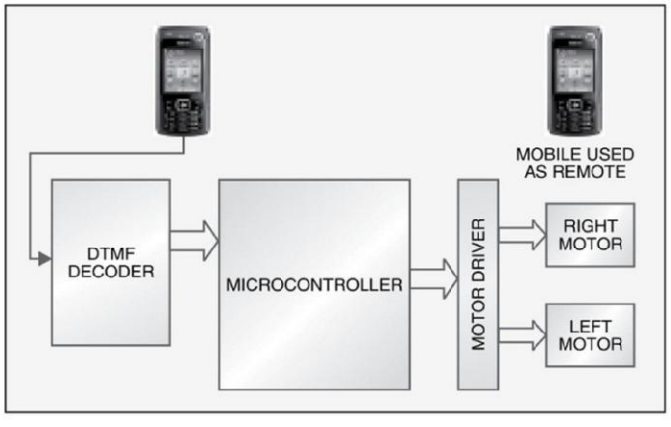


Fig. 1: Block diagram of cellphone-operated land rover

The mobile that makes a call to the mobile phone stacked in the robot acts as a remote. So this simple robotic project does not require the construction of receiver and transmitter units.

DTMF signaling is used for telephone signaling over the line in the voice-frequency band to the call switching centre. The version of DTMF used for telephone tone dialing is known as 'Touch-Tone.'

DTMF assigns a specific frequency (consisting of two separate tones) to each key so that it can easily be identified by the electronic circuit. The signal generated by the DTMF encoder is a direct algebraic summation, in real time, of the amplitudes of two sine (cosine) waves of different frequencies, i.e., pressing '5' will send a tone made by adding 1336 Hz and 770 Hz to

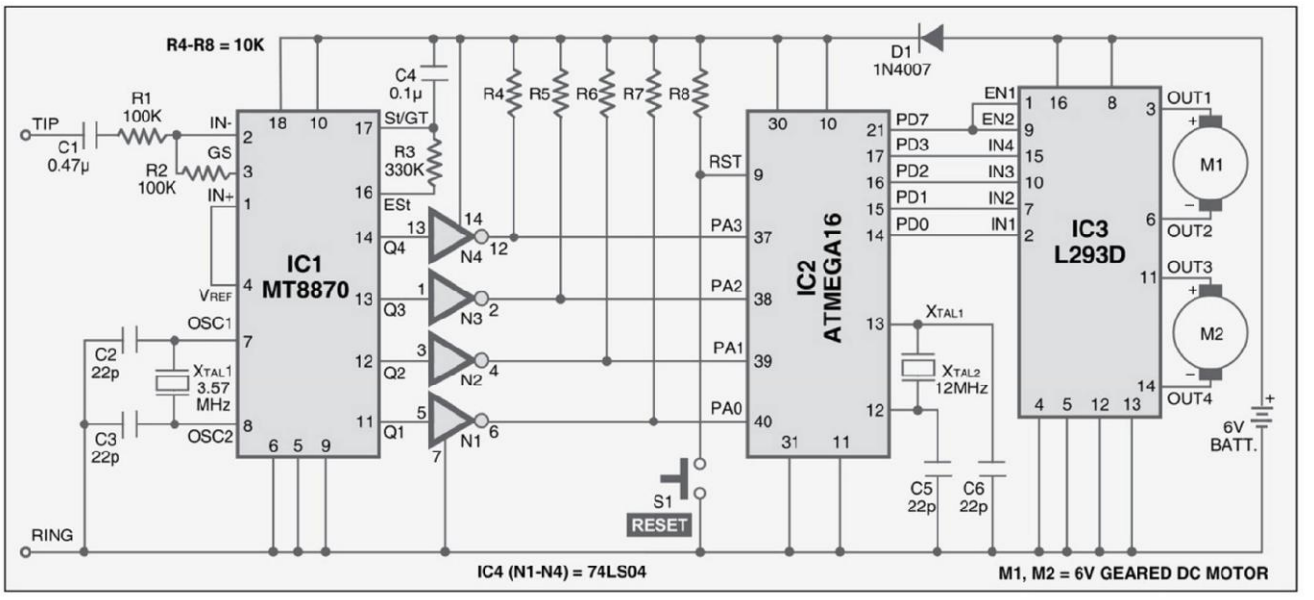


Fig. 2: Circuit diagram of microcontroller-based cellphone-operated land rover

TABLE I

Tones and Assignments in a DTMF System

Frequencies	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

the other end of the line. The tones and assignments in a DTMF system are shown in Table I.

Circuit description

Fig. 1 shows the block diagram of the microcontroller-based mobile phone-operated land rover. The important components of this rover are a DTMF decoder, microcontroller and motor driver.

An MT8870 series DTMF decoder is used here. All types of the MT8870 series use digital counting techniques to detect and decode all the 16 DTMF tone pairs into a 4-bit code output. The built-in dial tone rejection circuit eliminates the need for pre-filtering. When the input signal given at pin 2 (IN-) in single-ended input configuration is recognised to be effective, the correct 4-bit decode signal of the DTMF tone is transferred to Q1 (pin 11) through Q4 (pin 14) outputs.

TABLE II

DTMF Data Output

Low group (Hz)	High group (Hz)	Digit	OE	D3	D2	D1	D0
697	1209	1	H	L	L	L	H
697	1336	2	H	L	L	H	L
697	1477	3	H	L	L	H	H
770	1209	4	H	L	H	L	L
770	1336	5	H	L	H	L	H
770	1477	6	H	L	H	H	L
852	1209	7	H	L	H	H	H
852	1336	8	H	H	L	L	L
852	1477	9	H	H	L	L	H
941	1336	0	H	H	L	H	L
941	1209	*	H	H	L	H	H
941	1477	#	H	H	H	L	L
697	1633	A	H	H	H	L	H
770	1633	B	H	H	H	H	L
852	1633	C	H	H	H	H	H
941	1633	D	H	L	L	L	L
—	—	ANY	L	Z	Z	Z	Z

Table II shows the DTMF data output table of MT8870. Q1 through Q4 outputs of the DTMF decoder (IC1) are connected to port pins PA0 through PA3 of ATmega16 microcontroller (IC2) after inversion by N1 through N4, respectively.

The ATmega16 is a low-power, 8-bit, CMOS microcontroller based on the AVR enhanced RISC architecture. It provides the following features: 16 kB of in-system programmable Flash program memory with read-while-write capabilities, 512 bytes of EEPROM, 1kB SRAM, 32 general-purpose input/output (I/O) lines and 32 general-purpose working registers. All the 32 registers are directly connected to the arithmetic logic unit, allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code-efficient.

Fig. 3: Top view of the land rover

Outputs from port pins PD0 through PD3 and PD7 of the microcontroller are fed to inputs IN1 through IN4 and enable pins (EN1 and EN2) of motor driver L293D, respectively, to drive two geared DC motors. Switch S1 is used for manual reset. The microcontroller output is not sufficient to drive the DC motors, so current drivers are required for motor rotation.

The L293D is a quad, high-current, half-H driver designed to provide bidirectional drive currents of up to 600 mA at voltages from 4.5V to 36V. It makes it easier to drive the DC motors. The L293D consists of four drivers. Pins IN1 through IN4 and OUT1 through OUT4 are input and output pins, respectively, of driver 1 through driver 4. Drivers 1 and 2, and drivers 3 and 4 are enabled by enable pin 1 (EN1) and pin 9 (EN2), respectively. When enable input EN1 (pin 1) is high, drivers 1 and 2 are enabled and the outputs corresponding to their inputs are active. Similarly, enable input EN2 (pin 9) enables drivers 3 and 4.

An actual-size, single-side PCB for cellphone-operated land rover is shown in Fig. 4 and its component layout in Fig. 5.

Software description

The software is written in 'C' language and compiled using CodeVision AVR 'C' compiler. The source program is converted into hex code by the compiler. Burn this hex code into ATmega16 AVR microcontroller.

The source program is well commented and easy to understand. First include the register name defined specifically for ATmega16 and also declare the variable. Set port A as the input and port D as the output. The program will run forever by using 'while' loop. Under 'while' loop, read port A and test the received input using 'switch' statement. The corresponding data will output at port D after testing of the received data.

TABLE III

Actions Performed Corresponding to the Keys Pressed

Number pressed by user	Output of HT9170 DTMF decoder	Input to the microcontroller	Output from microcontroller	Action performed
2	0×02 00000010	0×FD 11111101	0×89 10001001	Forward motion
4	0×04 00000100	0XFB 11111011	0×85 10000101	Left turn Right motor forwarded Left motor backwarded
6	0×06 00000110	0XF9 11111001	0×8A 10001010	Right turn Right motor backwarded Left motor forwarded
8	0×08 00001000	0XF7 11110111	0×86 10000110	Backward motion
5	0×05 00000101	0XFA 11111010	0×00 00000000	Stop

Working

In order to control the robot, you need to make a call to the cell phone attached to the robot (through head phone) from any phone, which sends DTMF

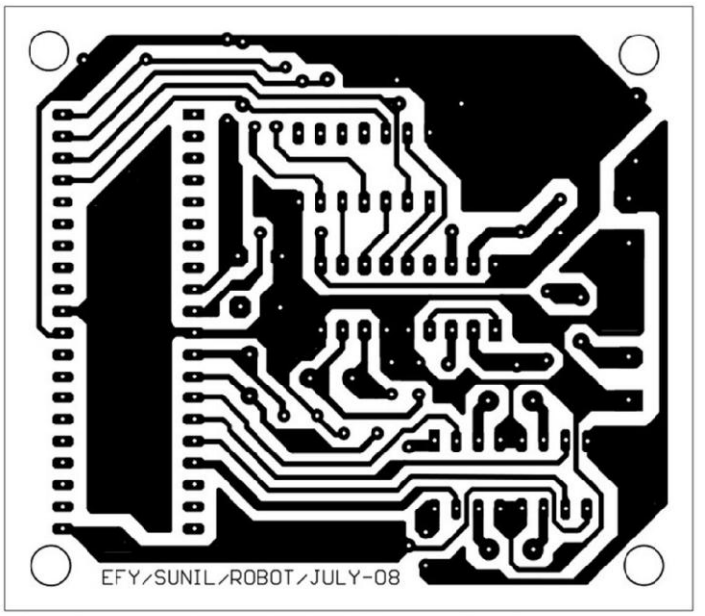


Fig. 4: An actual-size, single-side PCB layout for cellphone-operated land rover

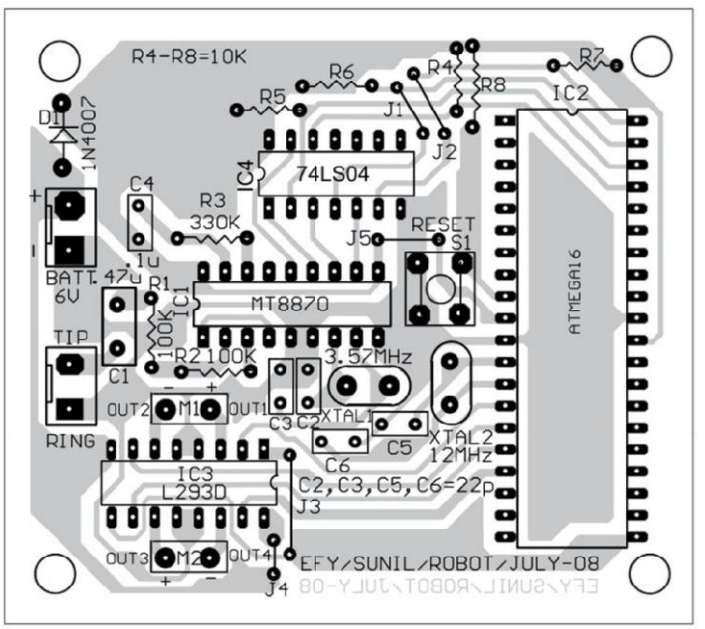


Fig. 5: Component layout for the PCB

tunes on pressing the numeric buttons. The cell phone in the robot is kept in 'auto answer' mode. (If the mobile does not have the auto answering facility, receive the call by 'OK' key on the rover-connected mobile and then made it in hands-free mode.) So after a ring, the cellphone accepts the call.

Now you may press any button on your mobile to perform actions as listed in Table III. The DTMF tones thus produced are received by the cellphone in the robot. These tones are fed to the circuit by the headset of the cellphone. The MT8870 decodes the received tone and sends the equivalent binary number to the microcontroller. According to the program in the microcontroller, the robot starts moving.

When you press key '2' (binary equivalent 00000010) on your mobile phone, the microcontroller outputs '10001001' binary equivalent. Port pins PD0, PD3 and PD7 are high. The high output at PD7 of the microcontroller drives the motor driver (L293D). Port pins PD0 and PD3 drive motors M1 and M2 in forward direction (as per Table III). Similarly, motors M1 and M2 move for left turn, right turn, backward motion and stop condition as per Table III.

Construction

When constructing any robot, one major mechanical constraint is the number of motors being used. You can have either a two-wheel drive or a four-wheel drive. Though four-wheel drive is more complex than two-wheel drive, it provides more torque and good control. Two-wheel drive, on the other hand, is very easy to construct.

Top view of a four-wheel-driven land rover is shown in Fig. 3. The chassis used in this model is a 10×18cm^2 sheet made up of parax. Motors are fixed to the bottom of this sheet and the circuit is affixed firmly on top of the sheet. A cellphone is also mounted on the sheet as shown in the picture.

In the four-wheel drive system, the two motors on a side are controlled in parallel. So a single L293D driver IC can drive the rover. For this robot, beads affixed with glue act as support wheels.

Further applications

This land rover can be further improved to serve specific purposes. It requires four controls to roam around. The remaining eight controls can be configured to serve other purposes, with some modifications in the source program of the microcontroller.

Download source code: http://www.efymag.com/admin/issuepdf/Microcontroller%20Based%20Land%20Rover.zip

ROBOT.C

```
Source program:
Robit.c
#include <mega16.h>
void main(void)
 {
         unsigned int k, h;
         DDRA=0x00;
         DDRD=0XFF;
 while (1)
 {
         k =~PINA;
         h=k & 0x0F;
 switch (h)
         {
         case 0x02: //if I/P is 0x02
         {
         PORTD=0x89;//O/P 0x89 ie Forward
         break;
         }
         case 0x08: //if I/P is      0x08
         {
         PORTD=0x86; //O/P 0x86 ie Backward
         break;
         }
         case 0x04:
         {
         PORTD=0x85; // Left turn
         break;
         }
         case 0x06:
         {
         PORTD=0x8A; // Right turn
         break;
         }
         case 0x05:
         {
         PORTD=0x00; // Stop
         break;
         }
         }
         }
         }
```

AUTOMATED LINE-FOLLOWING ROBOT

■ BISWAJEET PATRA & JAGBANDHU

Line-following robots with pick-and-placement capabilities are commonly used in manufacturing plants. These move on a specified path to pick the components from specified locations and place them on desired locations.

Basically, a line-following robot is a self-operating robot that detects and follows a line drawn on the floor. The path to be taken is indicated by a white line on a black surface. The control system used must sense the line and manoeuvre the robot to stay on course while constantly correcting the wrong moves using feedback mechanism, thus forming a simple yet effective closed-loop system.

Circuit description

Fig. 1 show the block diagram of the automated line-following robot. It consists of mainly four parts: two sensors, two comparators, one decision-making device and two motor drivers. The robot is built using microcontroller AT89C51 (used as the decision-making device), motor driver L293D, operational amplifier LM324 (comparator), phototransistor (sensor) and a few discrete components.

In the circuit, the sensors (phototransistors) are used to detect the white strip on a black background. The sensor output is fed to the microcontroller, which takes the decision and gives appropriate command to motor driver L293D so as to move the motor accordingly.

Sensor. The sensor senses the light reflected from the surface and feeds the output to the comparator. When the sensor is above the white background the light falling on it from the source reflects to the sensor, and when the sensor is above the black background the light from the source doesn't reflect to it. The sensor senses the reflected light to give an output, which is fed to the comparator.

Comparator. The comparator compares the analogue inputs from sensors with a fixed reference voltage. If this voltage is greater than the reference voltage the comparator outputs a low voltage, and if it is smaller the comparator generates a high voltage that acts as input for the decision-making device (microcontroller).

Microcontroller. The microcontroller is programmed

PARTS LIST

Semiconductors:

IC1	- LM324 quad operational amplifier
IC2	- AT89C51 microcontroller
IC3	- L293D motor driver
T1, T2	- L14F1 photo-transistor
D1	- 1N4007 diode
LED1, LED2	- 5mm LED

Resistors (1/4-watt, ±5% carbon, unless stated otherwise)

R1, R2, R5	- 10-kilo-ohm
R3, R4	- 5.6-kilo-ohm
R6	- 330-ohm
R7	- 220-ohm
R8	- 1-kilo-ohm
VR1, VR2	- 10-kilo-ohm preset

Capacitors:

C1	- 10µF, 16V electrolytic
C2, C3	- 33pF ceramic disk
C4	- 47µF, 16V electrolytic
C5	- 0.1µF ceramic disk

Miscellaneous:

S1	- On/off switch
S2	- Push-to-on switch
XTAL	- 12MHz crystal
M1, M2	- 20 rpm, 6V DC geared motor
Batt.	- 6V, 4.5AH battery
	- Two side brackets for mounting motors
	- One castor wheel (for front wheel)
	- Two wheels for the rear
	- Chassis

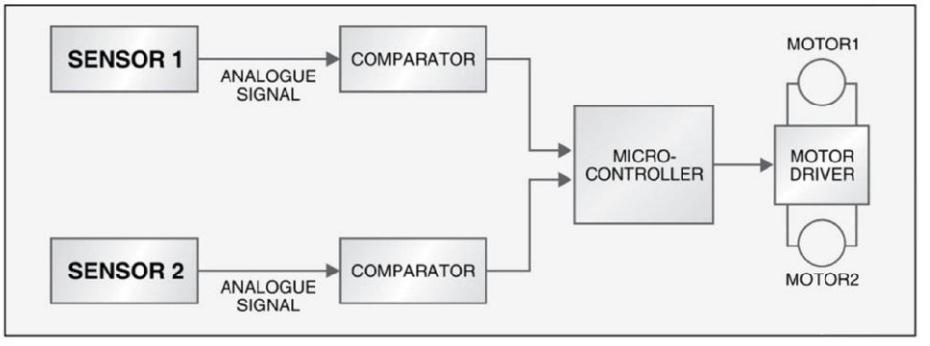


Fig. 1: Block diagram of automated line-following robot

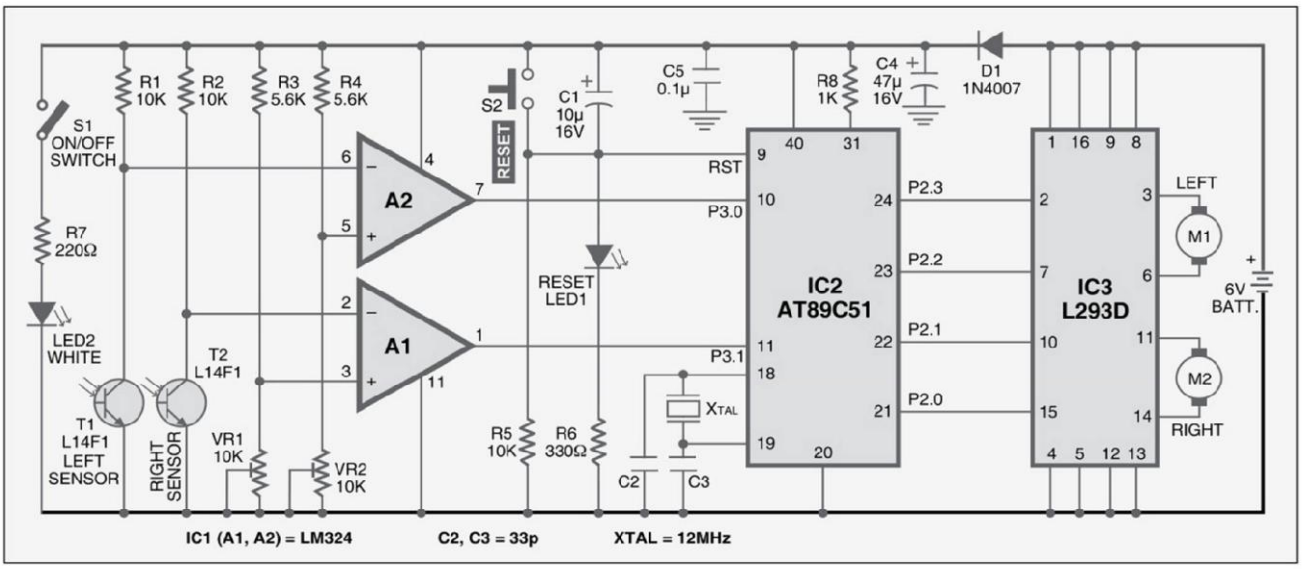


Fig. 2: Automated line-following robot circuit

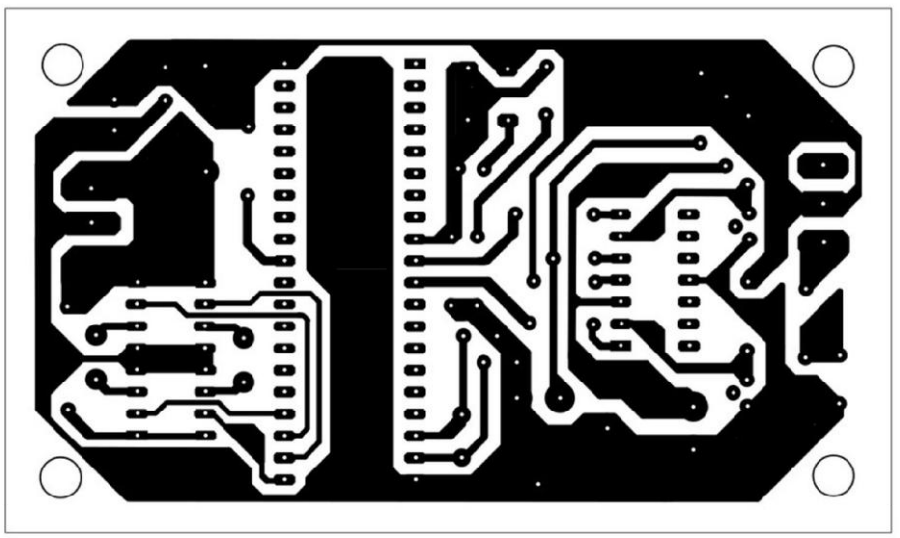


Fig. 3: Actual-size, single-side PCB for the automated line-following robot

to make the robot move forward, turn right or turn left based on the input coming from the comparator. The outputs of the microcontroller are fed to the motor driver.

Motor driver. The current supplied by the microcontroller to drive the motor is small. Therefore a motor-driver IC is used. It provides sufficient current to drive the motor.

Fig. 2 shows the circuit of the automated line-following robot. When light falls on the phototransistor (say, T1), it goes into saturation and starts conducting. When no light falls on the phototransistor, it is cut-off. A white LED (LED2) has been used to illuminate the white path on a black background. Phototransistors T1 and T2 are used for detecting the white path on the black background.

Collectors of phototransistors T1 and T2 are connected to the inverting inputs of operational amplifiers A2 and A1. The signal voltage at the inverting input of the operational amplifier is compared with the fixed reference voltage, which is formed by a potential divider circuit of 5.6-kilo-ohm resistor and 10-kilo-ohm preset. This reference voltage can be adjusted by changing the value of the 10-kilo-ohm preset.

When sensor T2 is above the black surface, it remains cut-off as the black surface absorbs virtually all the

Action Corresponding to the Microcontroller Outputs

Inputs		Outputs				Action
P3.0	P3.1	P2.3	P2.2	P2.1	P2.0	
0	0	1	0	1	0	Forward
0	1	0	0	1	0	Left
1	0	1	0	0	0	Right
1	1	0	0	0	0	Stop

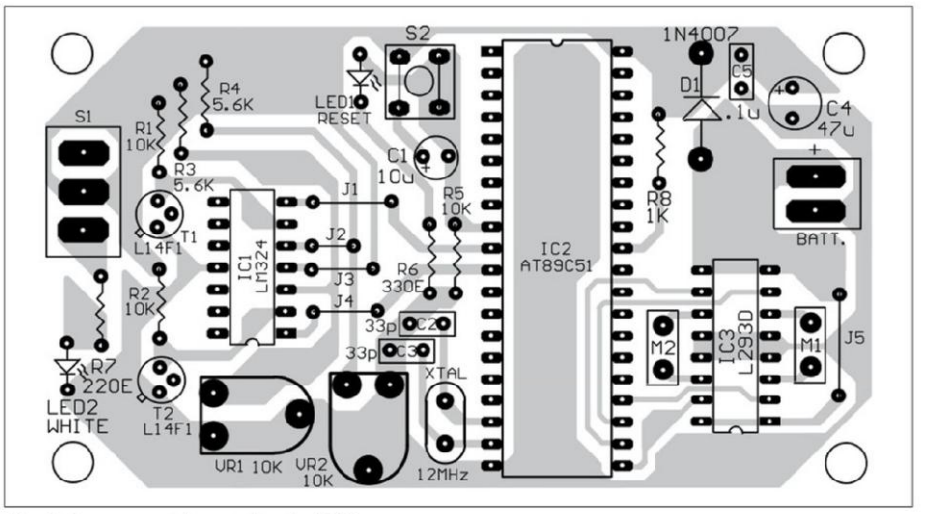


Fig. 4: Component layout for the PCB

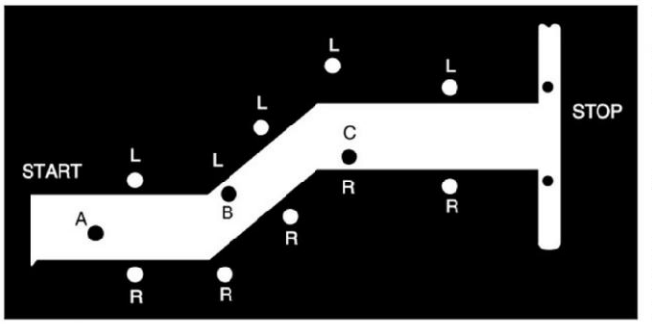


Fig. 5: Path of automated line-following robot

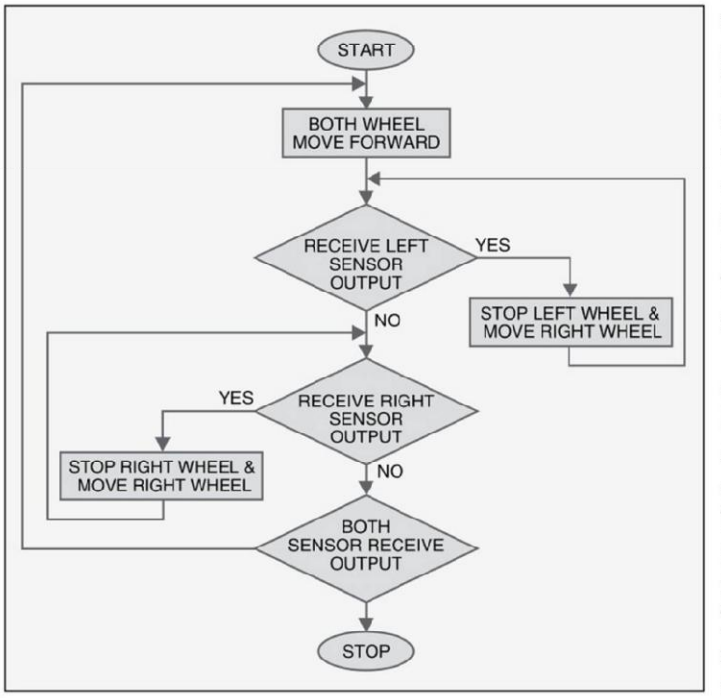


Fig. 6: Program flow-chart for automated line-following robot

light falling from LED2 and no light is reflected back. The voltage at the inverting input (pin 2) of operational amplifier A1 is higher than the reference voltage at its non-inverting input (pin 3) and therefore the amplifier output at pin 1 becomes zero.

When sensor T2 is above the white line, the light gets reflected from the white surface to fall on phototransistor T2. Phototransistor T2 goes into saturation and conducts. The inverting input (pin 2) of operational amplifier A1 goes below the reference voltage at its non-inverting input (pin 3) of operational amplifier A1 and therefore output pin 1 goes high. This way, the comparator outputs logic '0' for black surface and logic '1' for white surface.

Similarly, comparator A2 compares the input voltage from phototransistor T1 with a fixed reference voltage.

The outputs of operational amplifiers A1 and A2 are fed to microcontroller AT89C51. The AT89C51 is an 8-bit microcontroller having 4 kB of Flash, 128 bytes of RAM, 32 I/O lines, two 16-bit timers/counters, a five-vector two-level interrupt architecture, on-chip oscillator and clock circuitry. A 12MHz crystal is used for providing the basic clock frequency. All I/O pins are reset to '1' as soon as RST pin goes high. Holding RST pin high for two machine cycles while the oscillator is running resets the device. Power-on reset is derived from resistor R5 and capacitor C1. Switch S2 is used for manual reset. The microcontroller, based on the inputs from sensor T1 (say, left) and sensor T2 (say, right), controls the motor to make the robot turn left, turn right or move forward.

Port pins P2.0, P2.1, P2.2 and P2.3 are connected to pins 15, 10, 7 and 2 of motor driver L293D. Port pins P2.0 and P2.1 are used for controlling the right motor, while port pins P2.2 and P2.3 are used for controlling the left motor. Three wheels can be used for this robot—one on the front and two at the rear. Front wheel can rotate in any direction as specified by the rear wheel. To make the robot turn left, the left-side motor should stop and the right-side motor should rotate in the clockwise direction. Similarly, to make the robot turn right, the right-side motor should

stop and the left-side motor should rotate in clockwise direction. For forward motion, both the motors should rotate in clockwise direction.

Working

An actual-size, single-side PCB for the automated line-following robot is shown in Fig. 3 and its component layout in Fig. 4. Fig. 5 shows the path of the line-follower robot, where 'L' is the left sensor and 'R' is the right sensor.

At the start, when the robot is at point 'A,' sensors T1 and T2 are above the black surface and port pins P3.0 and P3.1 of the microcontroller receive logic '0.' As a result, the robot moves forward in straight direction.

At point 'B,' a left turn is encountered, and the left sensor comes above the white surface, whereas the right sensor remains above the black surface. Port pin P3.0 of the microcontroller receives logic '1' from the left sensor and port pin P3.1 receives logic '0' from the right sensor. As a result, the left motor stops and the right motor rotates, to make the robot turn left. This process continues until the left sensor comes above the black background.

Similarly, at point 'C,' where a right turn is encountered, the same procedure for right turn is executed. When both the sensors are at the white surface, the robot should stop. The output of the microcontroller (IC2) depends on the inputs received at its port pins P3.0 and P3.1 as shown in table.

Software

The source program for the project is written in Assembly language and assembled using Metalink's ASM51 assembler, which is freely available on the Internet for download. It is well commented for easy understanding and works as per the flow-chart shown in Fig. 6. The hex file 'robot.hex' is to be burnt into the microcontroller.

Download source code: http://www.efymag.com/admin/issuepdf/Automated%20Line%20Following%20Robot.zip

ROBOT.ASM

```
$MOD51

        ORG 0000H
        LJMP MAIN
        ORG 0030H
 MAIN:  SETB P3.0 ;Input for left sensor
        SETB P3.1 ;Input for right sensor
AGAIN:  JB P3.0,NEXT
        JB P3.1,GO
        CLR P2.0
        SETB P2.1
        CLR P2.2
        SETB P2.3
        SJMP AGAIN
  GO:   CLR P2.0
        SETB P2.1
        CLR P2.2
        CLR P2.3
        SJMP AGAIN
 NEXT:  JB P3.1,GO1
        CLR P2.0
        CLR P2.1
        CLR P2.2
        SETB P2.3
        SJMP AGAIN
 GO1:   CLR P2.0
        CLR P2.1
        CLR P2.2
        CLR P2.3
        SJMP AGAIN
HERE:   SJMP HERE
        END
```

ARDUINO-BASED RF CONTROLLED ROBOT

■ ROBIN CHALANA

Here we present a simple Arduino-board based robot that can be driven remotely using an RF remote control. This robot can be built very quickly in a small budget. The RF remote control provides the advantage of a good controlling range (up to 100 metres with proper antennae) besides being omnidirectional.

Circuit description

The block diagram of the robot is shown in Fig. 1. It has two major sections: (a) transmitter and (b) receiver and motor driver. The transmitter circuit (Fig. 2) is built around encoder IC HT12E (IC1), 433MHz RF transmitter module (TX1) and a few discrete components. The receiver and motor driver circuit (Fig. 3) is built around Arduino UNO board (BOARD1), decoder IC HT12D (IC2), 433MHz RF receiver module (RX1), motor driver IC L293D (IC3), regulator IC 7805 (IC4) and a few discrete components.

Arduino UNO board. The heart of the robot is Arduino UNO board. Arduino is an Open Source electronics prototyping platform based on flexible, easy-to-use hardware and software. It is intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

The Arduino Uno board is based on the ATmega328 microcontroller. It consists of 14 digital input/output pins, six analogue inputs, a USB connection for programming the on-board microcontroller, a power jack, an ICSP header and a reset button. It is operated with a 16MHz crystal oscillator. It contains everything needed to support

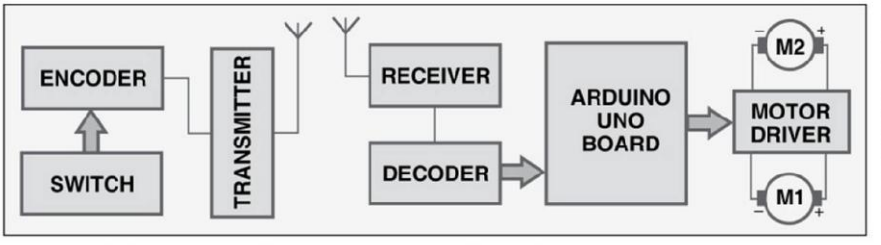


Fig. 1: Block diagram of Arduino-based RF controlled robot

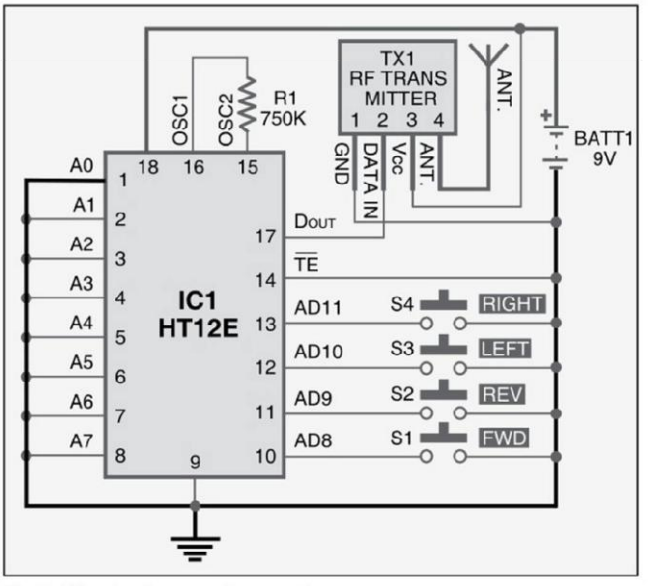


Fig. 2: Circuit of transmitter section

Technical Specifications of Arduino UNO Board

Details	Specifications
Microcontroller	ATmega328
Operating voltage	5V
Input voltage—recommended	7-12V
Input voltage—limits	6-20V
Digital I/O pins	14 (of which six provide PWM output)
Analogue input pins	6
DC current per I/O pin	40 mA
DC current for 3.3V pin	50 mA
Flash memory	32 kB (ATmega328), of which 0.5 kB is used by bootloader
SRAM	2 kB (ATmega328)
EEPROM	1 kB (ATmega328)
Clock speed	16 MHz

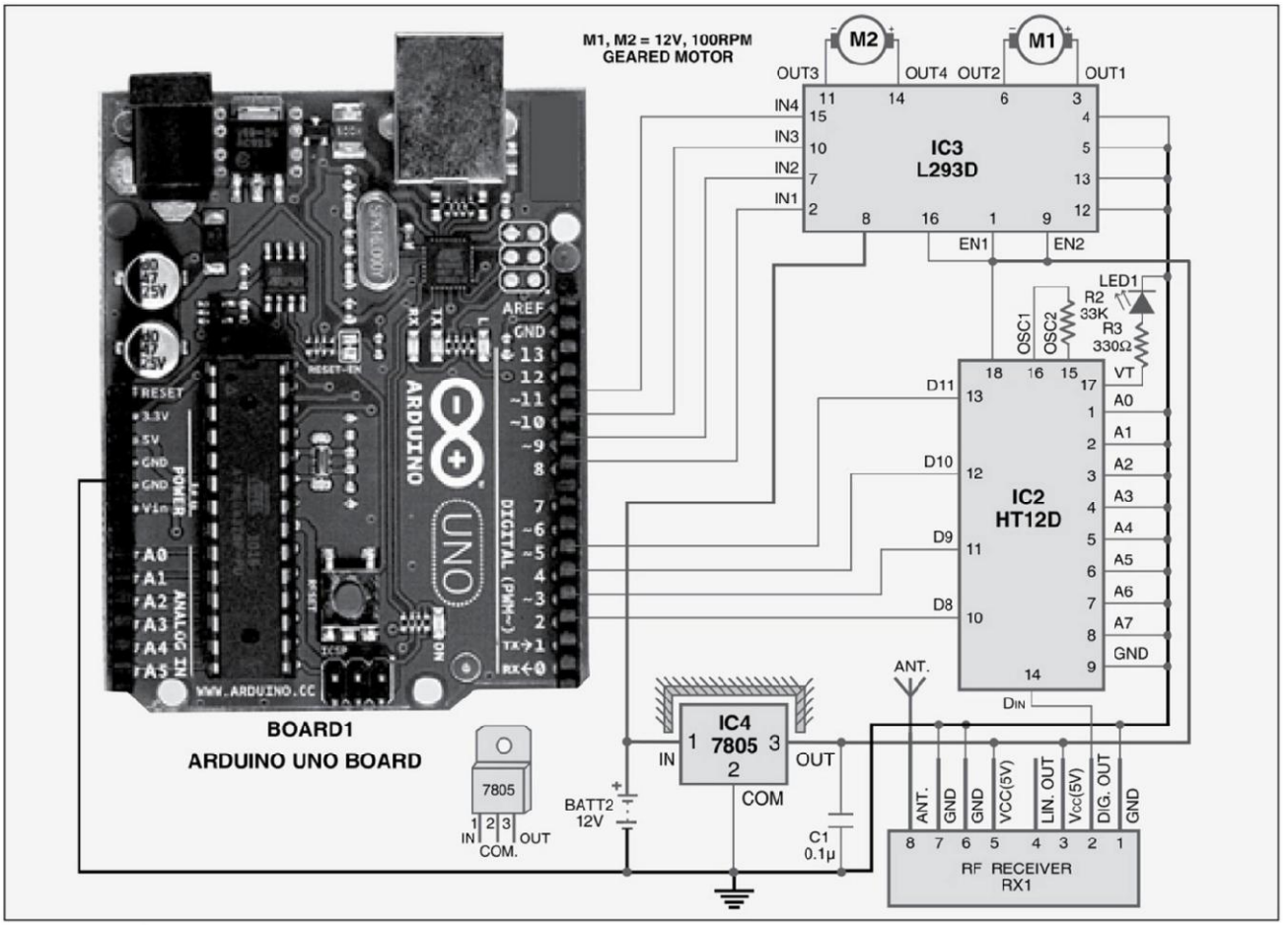


Fig. 3: Circuit of receiver and motor driver

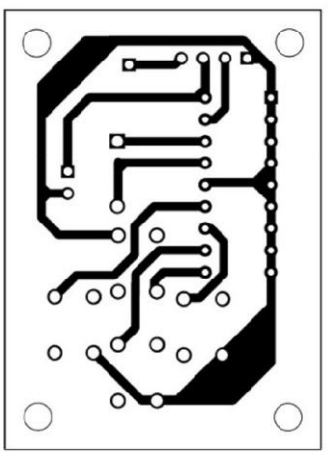


Fig. 4: An actual-size, single-side PCB for the RF transmitter

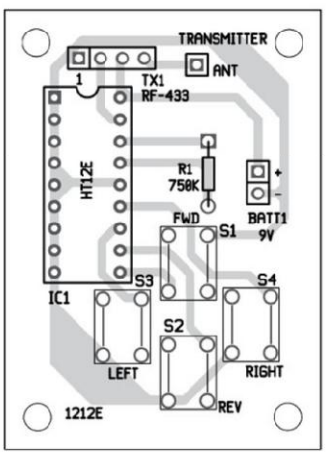


Fig. 5: Component layout for the PCB in Fig. 4

the microcontroller. It is very user-friendly; simply connect it to a computer with a USB cable to get started. The microcontroller on the board is programmed using the Arduino programming language and the Arduino development environment.

Remote control. For controlling the robot remotely, Holteks' encoder-decoder pair (HT12E and HT12D) together with a 433MHz transmitter-receiver pair is used.

HT12E and HT12D are CMOS ICs with working voltage ranging from 2.4V to 12V. Encoder HT12E has eight address and another four address/data lines. The data set on these twelve lines (address and address/data lines) is serially transmitted when transmit-enable pin TE is taken low. The data output appears serially on D_{OUT} pin.

The data is transmitted four times in succession. It consists of differing lengths of positive-going pulses for '1' and '0,' the pulse-width for '0' being twice the pulse-width for '1.' The frequency of these pulses may lie between 1.5 and 7 kHz depending on the resistor value between OSC1 and OSC2 pins.

The internal oscillation frequency of decoder HT12D is 50 times the oscillation frequency of encoder HT12E. The HT12D receives the data from the HT12E on its D_{IN} pin serially. If the address part of the data received matches the

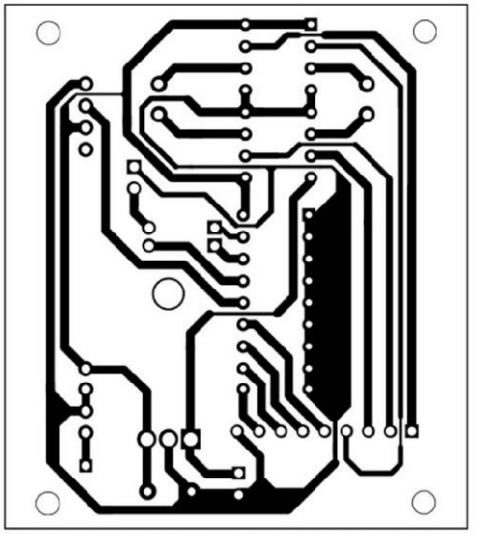


Fig. 6: An actual-size, single-side PCB for the RF receiver

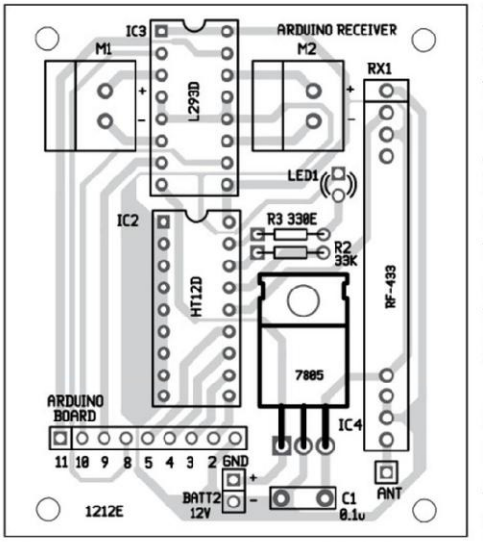


Fig. 7: Component layout for the PCB in Fig. 6

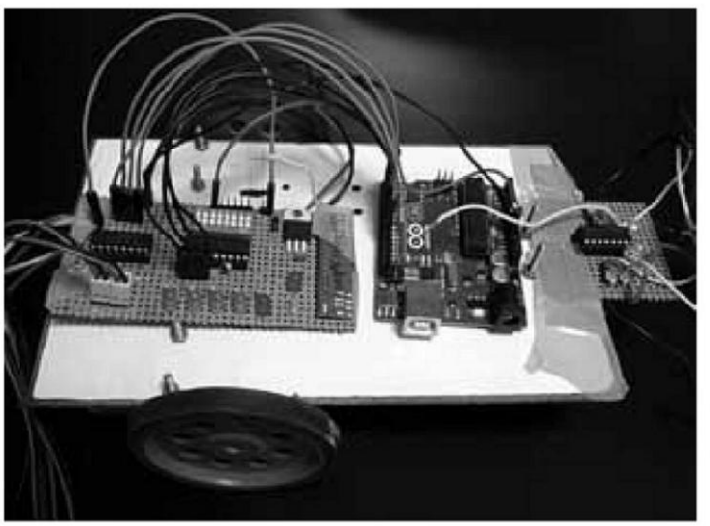


Fig. 8: Author's prototype

levels on A0 through A7 pins four times in succession, the valid transmission (VT) pin is taken high. The data on pins AD8 through AD11 of the HT12E appears on pins D8 through D11 of the HT12D. Thus the device acts as a receiver of 4-bit data (16 possible codes) with 8-bit addressing (256 possible channels).

Transmitter circuit. Switches S1, S2, S3 and S4 are interfaced with AD8 through AD11 of encoder HT12E for forward (FWD), reverse (REV), left (LEFT) and right (RIGHT) motions, respectively. Resistor R1 is connected between oscillator pins 15 and 16 to set the transmitter frequency.

HT12E is permanently enabled for transmission by connecting its $\overline{TE}$ pin to ground. When any switch, say, S1, is pressed, the corresponding data is serially transmitted from D_{OUT} pin through the RF ASK transmitter module. A 9V battery is used to power the circuit.

Receiver and motor driver circuit. Assuming that address pins on the encoder and the decoder are identical, when any of the switches on the transmitter (marked as FWD, REV, RIGHT, LEFT) is pressed, the corresponding data pin of the decoder goes low. The data outputs from D8 through D11 of HT12D (IC2) are fed to pins 2 through 5 of Arduino UNO board to generate appropriate logic outputs from pins 8 through 11 of Arduino UNO board.

Outputs from pins 8 through 11 of Arduino Uno board are fed to IN1 through IN4 of L293D (IC3) to drive both the motors M1 and M2 as shown in Fig. 3. Outputs OUT1 and OUT2 drive motor M1, and outputs OUT3 and OUT4 drive motor M2. Enable pins EN1 (pin 1) and EN2 (pin 9) are connected to Vcc for always enabled output. Regulator 7805 (IC4) provides regulated 5V to the receiver.

Construction

An actual-size, single-side PCB for the RF transmitter (Fig. 2) is shown in Fig. 4 and its component layout in Fig. 5. The PCB for the receiver (Fig. 3) is shown in Fig. 6 and its component layout in Fig. 7. A suitable connector arrangement has been made on the RF receiver PCB in order to extend connections to the drive motors and the battery mounted on the chassis of the RF robot.

Software

The source code file (RFROBOT.INO) for this project is listed at the end of this article. The Arduino Uno is programmed with Arduino IDE software. The ATmega328 on Arduino Uno comes pre-burned with a bootloader

that allows you to upload new code to it without using an external hardware programmer. It communicates using the original STK500 protocol. You can also bypass the bootloader and program the microcontroller through the ICSP (in-circuit serial programming) header but using the bootloader makes the programming quick and easy.

Select Arduino Uno from the Tools→Board menu (according to the microcontroller on your board) in the Arduino IDE and burn the program through the standard USB port in the computer.

Download source code: http://www.efymag.com/admin/issuepdf/ArduinoRobot.zip

RFROBOT.INO

```
//RF Robot
int sw1 =2;
int sw2 =3;
int sw3 =4;
int sw4 =5;
int out1=8;
int out2=9;
int out3=10;
int out4=11;
void setup()
{
  pinMode(sw1,INPUT);
  pinMode(sw2,INPUT);
  pinMode(sw3,INPUT);
  pinMode(sw4,INPUT);
  pinMode(out1,OUTPUT);
  pinMode(out2,OUTPUT);
  pinMode(out3,OUTPUT);
  pinMode(out4,OUTPUT);
}
void loop()
{
  if ((digitalRead(sw1)==LOW) &&
(digitalRead(sw2)==HIGH) && (digitalRead(sw3)==HIGH) &&
(digitalRead(sw4)==HIGH))
  {
    fwd();
  }
  else if ((digitalRead(sw1)==HIGH) &&
(digitalRead(sw2)==LOW) && (digitalRead(sw3)==HIGH) &&
(digitalRead(sw4)==HIGH))
  {
    bwk();
  }
  else if ((digitalRead(sw1)==HIGH) &&
(digitalRead(sw2)==HIGH) && (digitalRead(sw3)==LOW) &&
(digitalRead(sw4)==HIGH))
  {
    lft();
  }
  else if ((digitalRead(sw1)==HIGH) &&
(digitalRead(sw2)==HIGH) && (digitalRead(sw3)==HIGH) &&
(digitalRead(sw4)==LOW))
  {
    rgt();
  }
  else
  {  digitalWrite(out1,LOW);
     digitalWrite(out2,LOW);
     digitalWrite(out3,LOW);
     digitalWrite(out4,LOW);
  }
}
void fwd()
{  digitalWrite(out1,HIGH);
   digitalWrite(out2,LOW);
   digitalWrite(out3,HIGH);
   digitalWrite(out4,LOW);
}
void bwk()
{   digitalWrite(out1,LOW);
    digitalWrite(out2,HIGH);
    digitalWrite(out3,LOW);
    digitalWrite(out4,HIGH);
}
void lft()
{   digitalWrite(out1,LOW);
    digitalWrite(out2,HIGH);
    digitalWrite(out3,HIGH);
    digitalWrite(out4,LOW);
}

void rgt()
{   digitalWrite(out1,HIGH);
    digitalWrite(out2,LOW);
    digitalWrite(out3,LOW);
    digitalWrite(out4,HIGH);
}
```

PC-BASED WIRELESS CONTROL FOR TOY CAR

■ BODHIBRATA MUKHOPADHYAY, GOURAB SIL, SUBHAJIT MAZUMDAR

Here we show how you can control a toy car through your PC's serial port using a pair of ASK transmitter and receiver modules. The received signal is decoded by a P89V51RD2 microcontroller and fed to the motor driver circuitry to move the toy car in forward, backward, right or left direction. All the signals are in RF domain.

Circuit description

Fig. 1 shows the block diagram for PC-based wireless control of a toy car. The different stages for wireless control are:

1. Transmission of the control signals from a PC's serial port
2. RF transmitter and receiver
3. Decoding of the received signals using the microcontroller
4. Motor drivers

Asynchronous serial communication is established between the computer and P89V51RD2 microcontroller through wireless RF link. The microcontroller and computer are both synchronised with each other. The baud rate of data transfer is 1200.

Transmission of the control signal through PC's serial port. The first part of the project is transmission of control signals through the serial port of the PC. The control signals are W, S, D, A, Q, E, C, Z and U to control the toy car in forward, backward, right drift, left drift, sharp forward left turn, sharp forward right turn, sharp backward left turn, sharp backward right turn and stop, respectively. Each of the control signals is fed from the keyboard and sent through the serial com port. The signal is then transmitted wirelessly by the ASK transmitter module.

PC's serial com port. There are many ports available at the base of your PC in order to send data to the connected peripherals. Serial port, parallel port and USB port are some of the ports for connecting to the peripherals.

The serial port transmits or receives the data serially (1-bit data per TX or RX clock pulse). It is based on IEEE RS-232 standard, which defines voltages and baud rates for serial communication between devices connected to it. Most desktop computers have an RS-232 serial port as it has a very simple circuitry and is cheap and easy to handle.

Windows-based 9-pin serial port DB9 connector has the configuration as shown in Table I.

The RS-232 standard serial port has nine pins having different functions for transmitting and receiving data. Of these, only three pins (pins 2, 3 and 5) are mostly used for sending and receiving data. Only pins 3 and 5 are used in this project. The RS-232 standard has specific voltage levels for data logic 0 and logic 1 (-3 to -15V for logic 1, and +3 to +15V for logic 0). But the microcontroller defines logic 0 and logic 1 by voltage levels 0-0.5V and 4.5-5V, respectively. So you have to convert the RS-232 standard signal level into the microcontroller signal level. For that purpose, we have used a MAX232 converter.

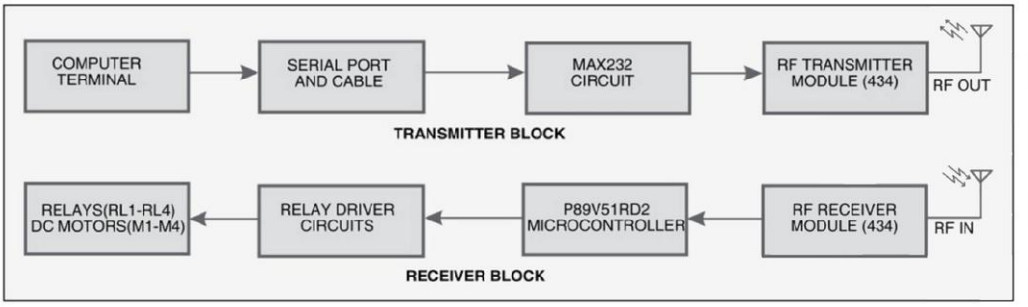


Fig. 1: Block diagram for PC-based wireless control of a toy car

TABLE I
Serial Port

Pin	Functions
1	Data-carrier detect
2	Received data
3	Transmitted data
4	Data-terminal ready
5	Signal ground
6	Data-set ready
7	Request to send
8	Clear to send
9	Ring indicator

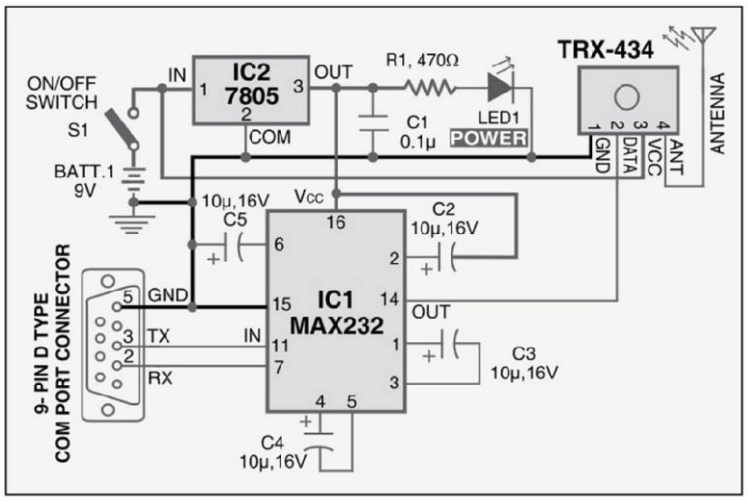


Fig. 2: Circuit of transmitter section

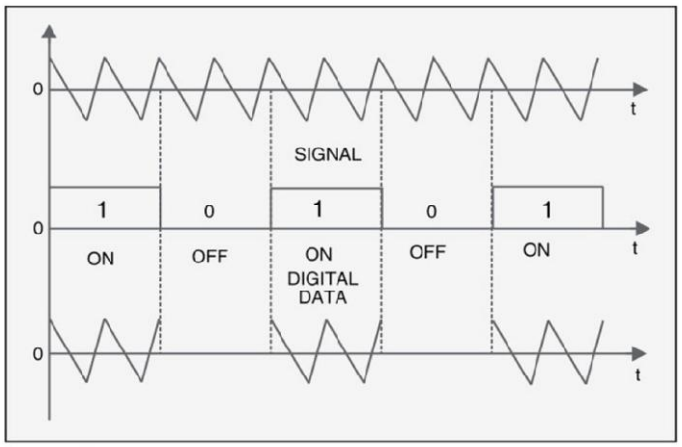


Fig. 3: ASK-modulated signal

For signal-level conversion, MAX232 requires four capacitors (10µF) and a 5V supply. Since we are using 9V supply, a single voltage source voltage regulator IC 7805 is required to supply power to MAX232. Now, the serial port signal, which is at RS-232 logic, can be converted into 5V TTL-level signal.

The transmitter section built around MAX232 is shown in Fig. 2.

RF transmission. The 5V supply from 7805 voltage regulator IC powers the RF ASK transmitter module through its pin 3. This module actually collects the binary signal from the data pin (which is connected to the MAX232) and modulates this binary signal with amplitude-shift keying (ASK) digital modulation scheme by a carrier frequency of 434 MHz and transmits the data through the antenna. The concept of the ASK signal is shown in Fig. 3.

RF reception. The ASK receiver module (Fig. 4) receives the 434MHz signal and demodulates it. The baseband signal is a 10-bit digital signal with the first bit as start bit, the following eight bits as data bits and the last bit as stop bit. The received data is sent to the microcontroller. The RXD pin (p3.0) of P89V51RD2 microcontroller is connected to the data pin of the 434MHz ASK module receiver. So the P89V51RD2 detects the transmitted data at its pin 10 (RXD).

Decoding of the received signal with P89V51RD2. The P89V51RD2 is an 80C51 microcontroller with 64kB flash, 1024 bytes of data RAM, 32 input/out-

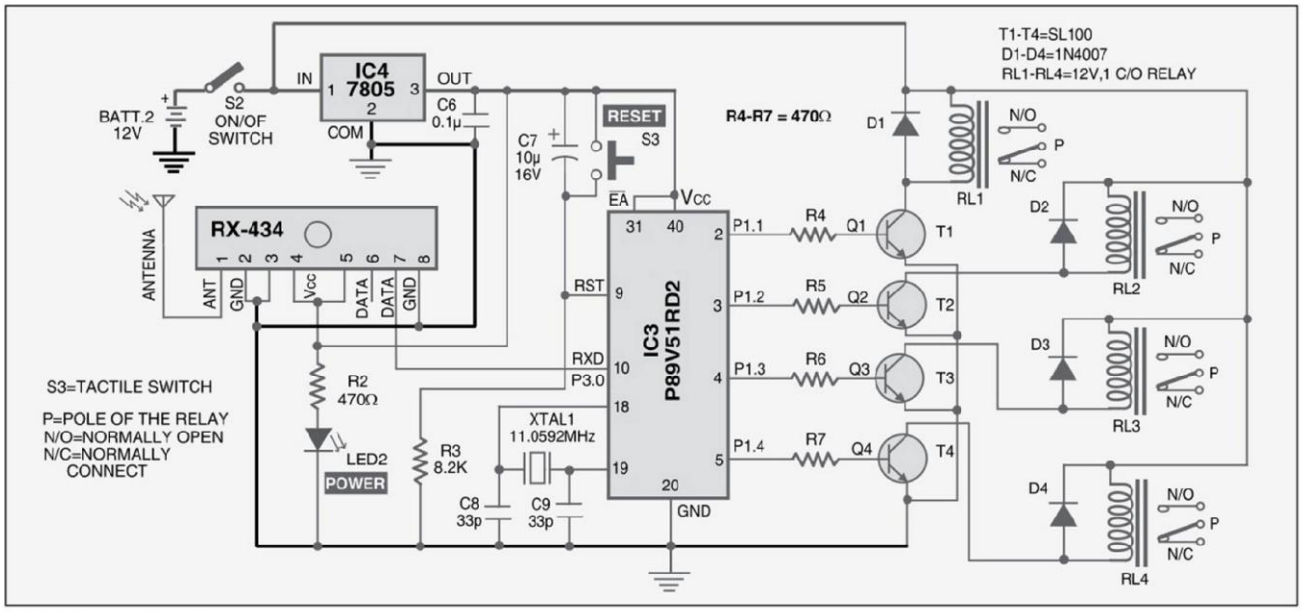


Fig. 4: Circuit of receiver section

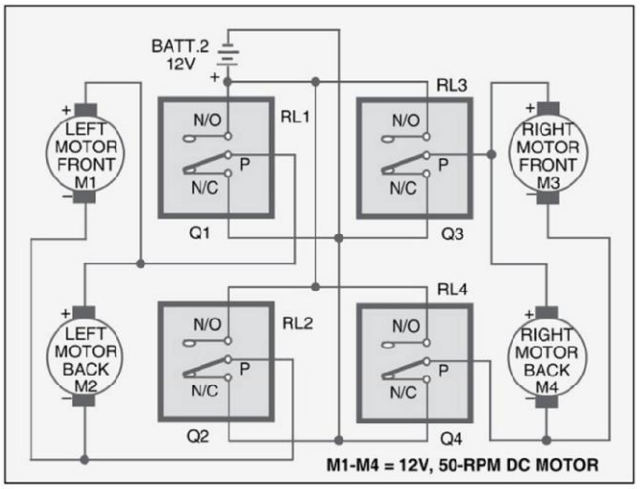


Fig. 5: Relay connections to motors

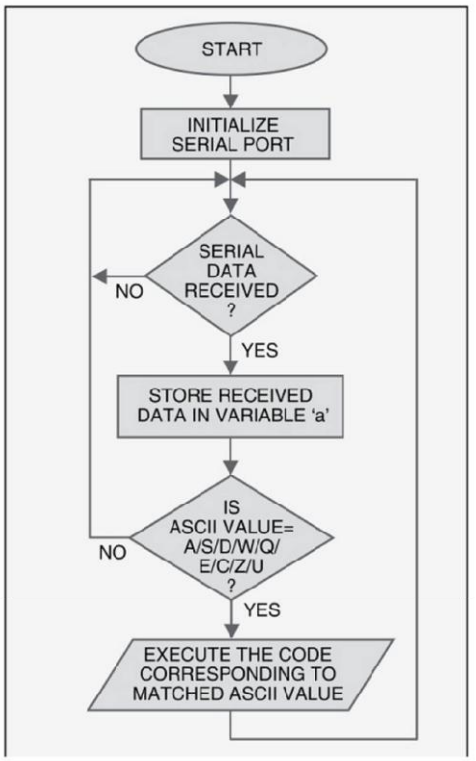


Fig. 6: Flow-chart of the program

put (I/O) ports, three 16-bit timers/counters and two pins for serial data transmission and reception. Timer 1 is used for serial communication. It is operated with an 11.0592MHz crystal.

A key feature of the P89V51RD2 is its X2 mode option. You can choose to run the application with the conventional 80C51 clock rate (12 clocks per machine cycle) or select the X2 mode (six clocks per machine cycle) to achieve twice the throughput at the same clock frequency.

The Flash program memory supports both parallel programming and in serial in-system programming (ISP). It is also in-application programmable (IAP), allowing the Flash program memory to be reconfigured even when the application is running.

In this project, timer 1 (TH1) is used in mode 2 (8-bit auto-reload). It is used to set the baud rate. Here it is loaded with a value of E8 hex (or -24) and so the baud rate is set at 1200. The SCON register is loaded with a hex value of 50, indicating serial mode 1, where 8-bit data is framed with a start bit and a stop bit.

After timer TH1 is set, it starts running until the P89V51RD2 microcontroller is made off. The P89V51RD2 waits until it receives a start bit. After receiving the start bit, it receives the 8-bit data and places the data in SBUF register. Then the framing error is checked. If there is framing error, the byte received is discarded. Otherwise, the content of SBUF is compared with the ASCII code of alphabets W, S, D, A, Q, E, C, Z and U. When a match is found, the operation related to each alphabet is executed. Then the toy car stops or moves in a particular direction as per this value.

Driving the DC motors. Port pins p1.1 through p1.4 of the microcontroller drive four relays through a relay-driver circuitry comprising transistors T1 through T4. The four relays, in turn, control four motors of the toy car (Fig. 5). Two relays control the forward and reverse rotations of a motor. The left two motors are connected in parallel and so are the two motors of the right. So two motors are controlled simultaneously using two relays.

Each of the four relays is 12V, single-changeover electromagnetic type to control the PMDC motor (12V, 50-rpm). The relays play an important role in isolating the controlling circuit and PMDC motors to protect the microcontroller and other low-current devices from the relatively high-current-driven motors. Basically, these are switches that connect or disconnect the motors from the 12V supply. Control signals from the microcontroller energise or de-energise the relays. That is, when a control signal makes a pin of P89V51RD2 high, the transistor connected to it conducts to energise the corresponding relay.

Here +12V terminal of the battery is connected to normally-open (N/O) contacts of all the relays and the ground terminal is connected to normally-closed (N/C) contacts of the relays.

This means when all the relays are not energised, positive and negative terminal ends of all the motors connect to the ground terminal of the battery, so the motors will not rotate. If all the relays energise, ends of the motors connect to +12V and the motors don't rotate. If any of the relays energises, one end of the respective motor connects to +12V and the other end to the ground. This makes the motor rotate. Energisation of the relay decides clockwise or anticlockwise movement of the motor.

Fig. 7: Screenshot of 'Options for Target 1' window

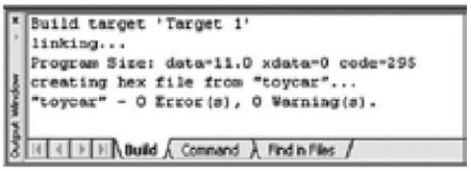


Fig. 8: Program compilation output screen

Fig. 9: Screenshot of 'Flash Magic' window

Q1 output from the microcontroller is fed to the base of transistor T1. When Q1 is high, transistor T1 conducts and relay RL1 energises to make the pole (P) shift towards N/O contact. This connects +12V to the positive terminal of motor M1 on the front left of the toy car (refer Fig. 5).

When Q2 output is high, transistor T2 conducts and relay RL2 energises to make P shift towards N/O contact. This connects +12V to the positive terminal of motor M2 on the left back of the car.

When Q3 output is high, transistor T3 conducts and relay RL3 energises to make P shift toward N/O contact. This connects +12V to the positive terminal of motor M3 on the right front.

When Q4 output is high, transistor T4 conducts and relay RL4 energises to make P shift toward N/O contact. This connects +12V to the positive terminal of motor M4 on the right back.

Controlling the toy car

'W' forward movement (Q1=1, Q2=0, Q3=1, Q4=0). The toy car moves forward when all the motors move clockwise. To achieve this, the output logic at Q1, Q2, Q3 and Q4 should be high (1), low (0), high (1) and low (0), respectively. Character 'W' is defined in the code to give 1010 bits in the output. That is, when you type 'W' character from the keyboard, the microcontroller generates 1010 bits at its port pins p1.1 through p1.4. This signal is sent to relay-driver section T1 through T4.

'S' backward movement (Q1=0, Q2=1, Q3=0, Q4=1). Backward movement takes place when all the motors move anticlockwise. So signal 0101 is sent to relay-driver section T1 through T4. Character 'S' is defined in the code to give 0101 bits in the output.

'D' right drift (Q1=1, Q2=0, Q3=1, Q4=0 (for 54 ms) and Q1=1, Q2=0, Q3=0, Q4=0 (for 108 ms)). Right drift is possible by rotating the left motors at a high speed and the right motors at a low speed. This is possible with the pulse-width-modulated (PWM) pulse given to the right motors. The right motors are given a pulse train of 33 per cent duty cycle so that these rotate at one-third the speed of the left motors. The car takes a right turn in forward direction resulting in a drift. So signal 1010 is sent for 54 ms and a signal of 1000 for the next 108 ms to transistors T1 through T4. Character 'D' is defined in the code to generate the 1010 and 1000 signals with 54 ms and 108 ms delays, respectively.

'A' left drift (Q1=1, Q2=0, Q3=1, Q4=0 (for 54 ms) and Q1=0, Q2=0, Q3=1, Q4=0 (for 108 ms)). Left drift is possible by rotating the right motors at a high speed and the left motors at a low speed. This is possible with the PWM pulse given to the left motors. The left motors are given a pulse train of 33 per cent duty cycle so that these rotate at one-third the speed of the right motors. The car takes a left turn in forward direction, resulting in a drift. So signal 1010 is sent for 54 ms and signal 0010 for the next 108 ms to transistors T1 through T4. Character 'A' is defined in the code to generate 1010 and 0010 signals with 54ms and 108ms delays, respectively.

'Q' sharp-forward left turn (Q1=0, Q2=0, Q3=1, Q4=0). The toy car moves to the left sharply in the forward direction when the left motors are static and the right motors move clockwise. So signal 0010 is sent to transistors T1 through T4. Character 'Q' is defined in the code to generate 0010 bits in the output.

'E' sharp-forward right turn (Q1=1, Q2=0, Q3=0, Q4=0). The car moves to the right sharply in the forward direction

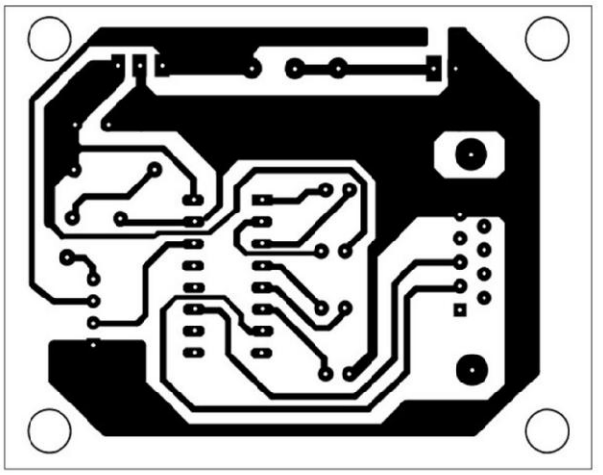


Fig. 10: An actual-size, single-side PCB of the transmitter circuit

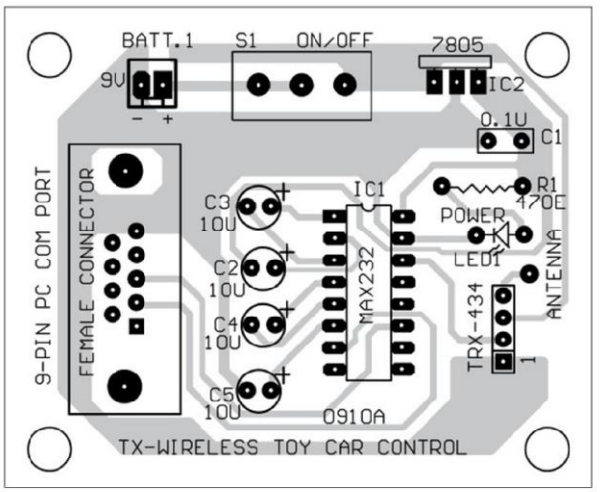


Fig. 11: Component layout for the PCB shown in Fig. 10

when the left motors move clockwise and the right motors are static. So signal 1000 is sent to transistors T1 through T4. Character 'E' is defined in the code to generate 1000 bits in the output.

'C' sharp-backward left turn (Q1=0, Q2=0, Q3=0, Q4=1). Sharp left turn takes place when the left motors are static and the right motors move anticlockwise. The car moves to the left sharply in the backward direction. So signal 0001 is sent to transistors T1 through T4. Character 'C' is defined in the code to generate 0001 bits in the output.

'Z' sharp-backward right turn (Q1=0, Q2=1, Q3=0, Q4=0). The car moves to the right sharply in the backward direction when the left motors move anticlockwise and the right motors are static. So signal 0100 is sent to transistors T1 through T4. Character 'Z' is defined in the code to generate 0100 bits in the output.

'U' stop (Q1=0, Q2=0, Q3=0, Q4=0). To stop the car, all the motors should be static. This is achieved by sending signal 0000 to the output of the microcontroller at its port pins p1.1 through p1.4. Character 'U' is defined in the code to generate 0000 bits in the output to stop the toy car.

Software program

Fig. 6 shows flow-chart of the program. The program is written in 'C' and compiled using Keil C software. The hex code generated using Keil software is burnt into the chip using Flash Magic programming software from NXP (Philips) Semiconductors.

Keil C µVision3 operations. 1. Run Keil µVision3 application from the desktop. From 'Project' menu, select 'New Project' option. Name the project as 'efytoy.uv2' and save it.

2. Select microcontroller P89V51RD2 from the database under NXP (Philips) option.

3. Right-click 'Source Group 1' option in 'Project Workspace' window on the left-hand side of the screen. Click 'Add Files to Source Group 1' option to add the toycar.c file.

4. Right-click 'Target 1' option from 'Project Workspace' and select 'Options for Target 1.' The window appears as shown in Fig. 7.

5. Change the X_{TAL} (MHz) value to 11.0592 as used in the project. Click 'Output' menu and tick the button against 'Create HEX File' option.

6. Now close the window and go to the main window. Compile the project by clicking 'Build Target' option. The program will be compiled with the message as shown in Fig. 8.

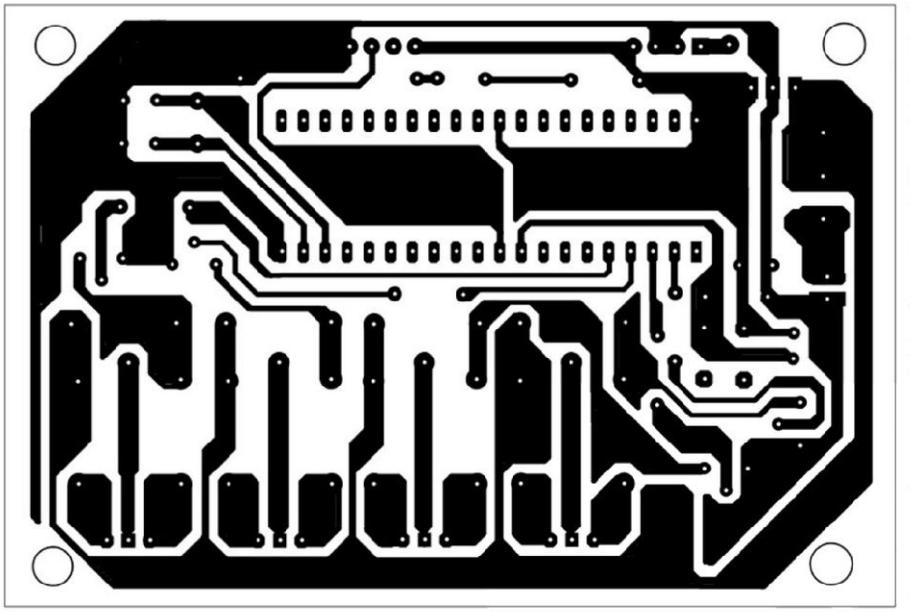


Fig. 12: An actual-size, single-side PCB of the receiver circuit

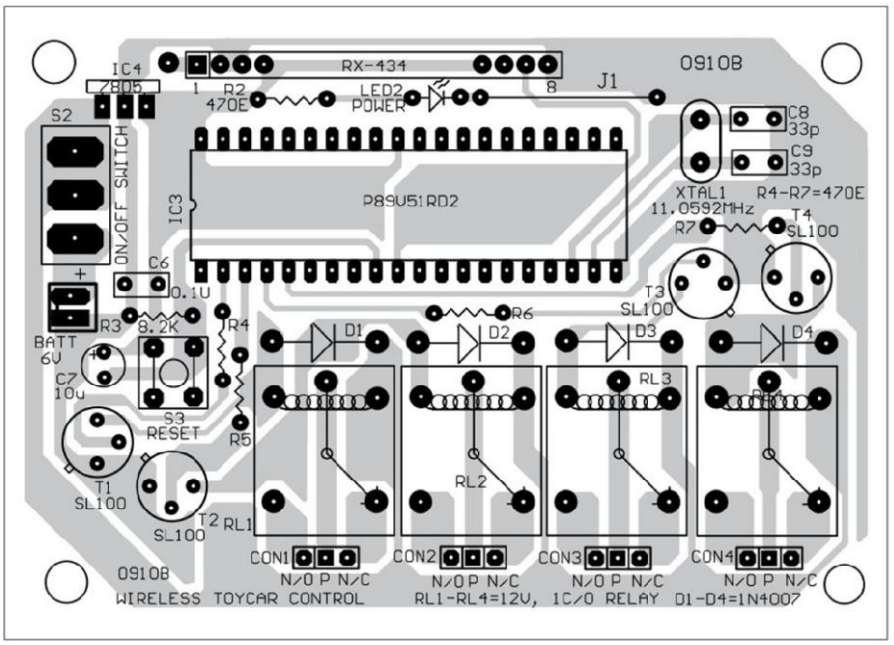


Fig. 13: Component layout for the PCB shown in Fig. 12

PARTS LIST

Semiconductors:

IC1	- MAX232 RS-232 level converter
IC2, IC4	- 7805, 5V regulator
IC3	- P89V51RD2 microcontroller
T1-T4	- SL100 npn transistor
TX-434	- 434MHz RF transmitter module
RX-434	- 434MHz RF receiver module
D1-D4	- 1N4007 rectifier diode
LED1, LED2	- 5mm light-emitting diode

Resistors (all ¼-watt, ±5% carbon):

R1, R2, R4-R7	- 470-ohm
R3	- 8.2-kilo-ohm

Capacitors:

C1, C6	- 0.1µF ceramic
C2-C5, C7	- 10µF electrolytic
C8, C9	- 33pF ceramic

Miscellaneous:

X_{TAL1}	- 11.0592MHz crystal oscillator
BATT.1	- 9V PP3 battery
BATT.2	- 12V, 4.5Ah, rechargeable SMF battery
S1, S2	- On/off toggle switch
S3	- Tactile switch
RL1-RL4	- 12V, single-changeover relay
M1, M2	- 12V, 50-rpm DC motor
	- Whip antenna
	- 9-pin D-type female connector
	- Metal chassis
	- Nuts and bolts

Now, the efytoy.hex code will be generated in the directory where the project file efytoy.uv2 is located. This hex code is used for programming the chip.

Programming the chip using Flash Magic

1. Run the Flash Magic software and select the device as P89V51RD2 and 'Erase all Flash' option as shown in Fig. 9.

2. In the 'Advanced Option,' uncheck 'Use DTR to control RST' and 'Assert DTR and RST while COM port open' options.

3. Browse the efytoy.hex file from your PC and load it onto the main screen. Press 'Start' button to program the chip.

Construction and testing

An actual-size, single-side PCB of the transmitter circuit is shown in Fig. 10 and its component layout in Fig. 11. An actual-size, single-side PCB of the receiver circuit is shown in Fig. 12 and its component layout in Fig. 13.

Burn the code into P89V51RD2 and mount the components on the receiver PCB.

Mount the populated PCB, 12V battery and four motors on a suitable chassis, preferably a 20×15cm^2 metal chassis. Use proper nuts and bolts to fix them firmly on the chassis. Four wheels, each of 7cm dia., are attached to shafts of respective motors.

After mounting the components on the transmitter PCB, connect the circuit to the serial port of the computer using a serial cable. If your PC has a USB port only, you may use a USB-to-serial converter. Now switch on the 9V power supply using switch S1.

Before sending the data from your computer, check which com port is connected to the circuit. Set the baud rate of the com port to 1200, stop bit to 1 and data bits to 8. Go to 'Device Manager' option from your desktop to do these settings. Now run a serial communication software such as Terminal v1.9b, select the com port and make the afore-mentioned settings. After completing all the settings and component assembly, switch on the power supply to the transmitter and receiver circuits. Run Terminal v1.9b software and activate the connection by pressing 'Connect' button followed by W, S, D, A, Q, E, C, Z or U key on the keyboard. The ASCII code corresponding to that key will be transmitted serially in a 10-bit (1 start bit+8 data bits for ASCII code+1 stop bits) binary datastream. The toy car will move as per the input from the keyboard.

Download source code: http://www.efymag.com/admin/issuepdf/PC-Based_Wireless_Control_for_Toy_car.zip

TOYCAR.C

```
#include<reg51.h>
sbit m1a = P1^1;//output 1(Q1)
sbit m1b = P1^2;//output 2(Q2)
sbit m2a = P1^3;//output 3(Q3)
sbit m2b = P1^4;//output 4(Q4)
void RXDATA(unsigned char);unsigned char RX(void);void
MSDELAY(unsigned char);void A();void S();void D();void
W();
void Z();void C();void U();void E();void Q();

void main( )
{
unsigned char a;
SCON=0X50;
TMOD=0X20;
TH1=-24;
TR1=1;
PCON=0X40;
P1=0X00;
          while(1)
          {
           a=RX();
          RXDATA(a);

          }
}

void RXDATA(unsigned char a)    //Receiving a byte data
through serial port
{
          switch(a)
          {
                    case('A'):
                     A( );
                    break;
                              case('S'):
                     S( );
                    break;
                              case('D'):
                     D( );
                    break;
                              case('W'):
                     W( );
                    break;
                              case('Q'):
                     Q( );
                    break;
                              case('E'):
                     E( );
                    break;
                              case('U'):
                     U( );
                    break;
                              case('C'):
                     C( );
                    break;
                              case('Z'):
                     Z();
                    break;
          }
}

void D( )   //Right drifts
{
unsigned char i;
P1=0;
m1a=1;
for(i=0;i<=9;i++)//PWM for 1/3 duty cycle
{
m2a=1;
MSDELAY(15);
m2a=0;
MSDELAY(30);
}
P1=0;
}

void S( )  //Backward movement
{
P1=0;
m1b=1;
m2b=1;
}

void A()  //Left drift
{
unsigned char i;
P1=0;
m2a=1;
for(i=0;i<=9;i++)//PWM for 1/3 duty cycle
{
m1a=1;
MSDELAY(15);
m1a=0;
MSDELAY(30);
}
P1=0;
}

void W( )  //Forward movement
{
P1=0;
m1a=1;
m2a=1;
}

void U()  //Stop
{
P1=0;
}

void E()  //Sharp forward right turn
{
P1=0;
m1a=1;
}

void Q( )  //Sharp forward left turn
{
P1=0;
m2a=1;
}

void Z( )  //Sharp backward right turn
{
P1=0;
m2b=1;
}

void C()  //Sharp backward left turn
{
P1=0;
m1b=1;
}

unsigned char  RX() // Function to receive a data
{
          EMERGENCY:
           RI=0;
          while(RI==0);
          if(SM0==1)//Check for framming error
          {
          SM0=0;
          goto EMERGENCY;
          }
          return(SBUF);
}

void MSDELAY(unsigned char b)  //Delay of 3.6Ms
{
unsigned char i;
unsigned int j;
for(i=0;i<=b;i++)
for(j=0;j<=310;j++);
}
```

About the Book

This book is a great resource for electronics professionals and enthusiasts who form the exciting Do-It-Yourself (DIY) community. It can also provide great reference material for academicians who work in the field of electronics.

The book presents some of the best microcontroller-based projects published in *Electronics For You* magazine in recent years. The 51 DIY projects in this book have been tested at the EFY Labs, and include applications such as security systems, domestic and industrial devices, display systems and even robotics. Besides the schematics of the circuits, the book provides detailed explanations, along with the parts list, image of the PCB layout, and links to software code that can be downloaded from our website (www.efymag.com). The book also helps you gain experience on popular microcontrollers from Texas Instruments, Microchip, Freescale and Atmel.

This book is an upgrade on the first edition, which featured 26 projects, and is part of a continuing series of DIY books on electronics. For more information, please go to www.efyindia.com.

About *Electronics For You* Magazine

Started in 1969, the magazine (print edition) is read by over half-a-million electronics professionals and enthusiasts from India. Another half-a-million professionals, from all across the globe, access its Web portal, www.electronicsforu.com, every month. And now the ezine version of the magazine, which is available on tablets, mobile devices, smartphones, desktops and laptops, is gaining popularity. The magazine is adored for its focus on technology trends, coupled with a lot of Do-It-Yourself content.

EFYGROUP
Technology Drives Us

EFY Enterprises Pvt. Ltd,
D-87/1, Okhla Industrial Area, Phase 1, New Delhi 110020
Ph: 011-26810601/2/3; E-mail: info@efyindia.com
Website: www.efyindia.com

ISBN: 978-81-88152-25-4
PUBLISHED BY EFY