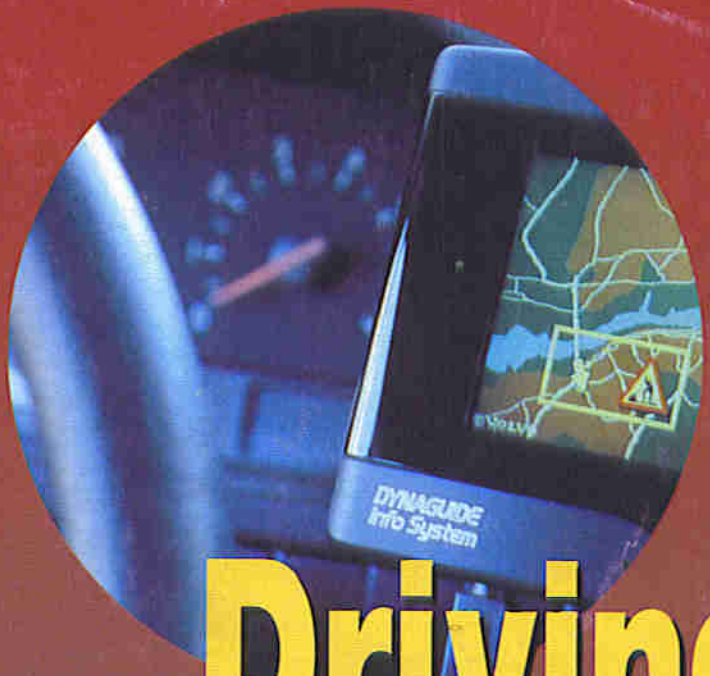
ETI
ELECTRONICS TODAY INTERNATIONAL

TOMORROW'S TECHNOLOGY TODAY



Driving into the future

Electronics behind 21st century cars

Radio control by computer

Developing ETI Basic microcontroller applications

NO COVER DISK

THEN ASK YOUR NEWSAGENT

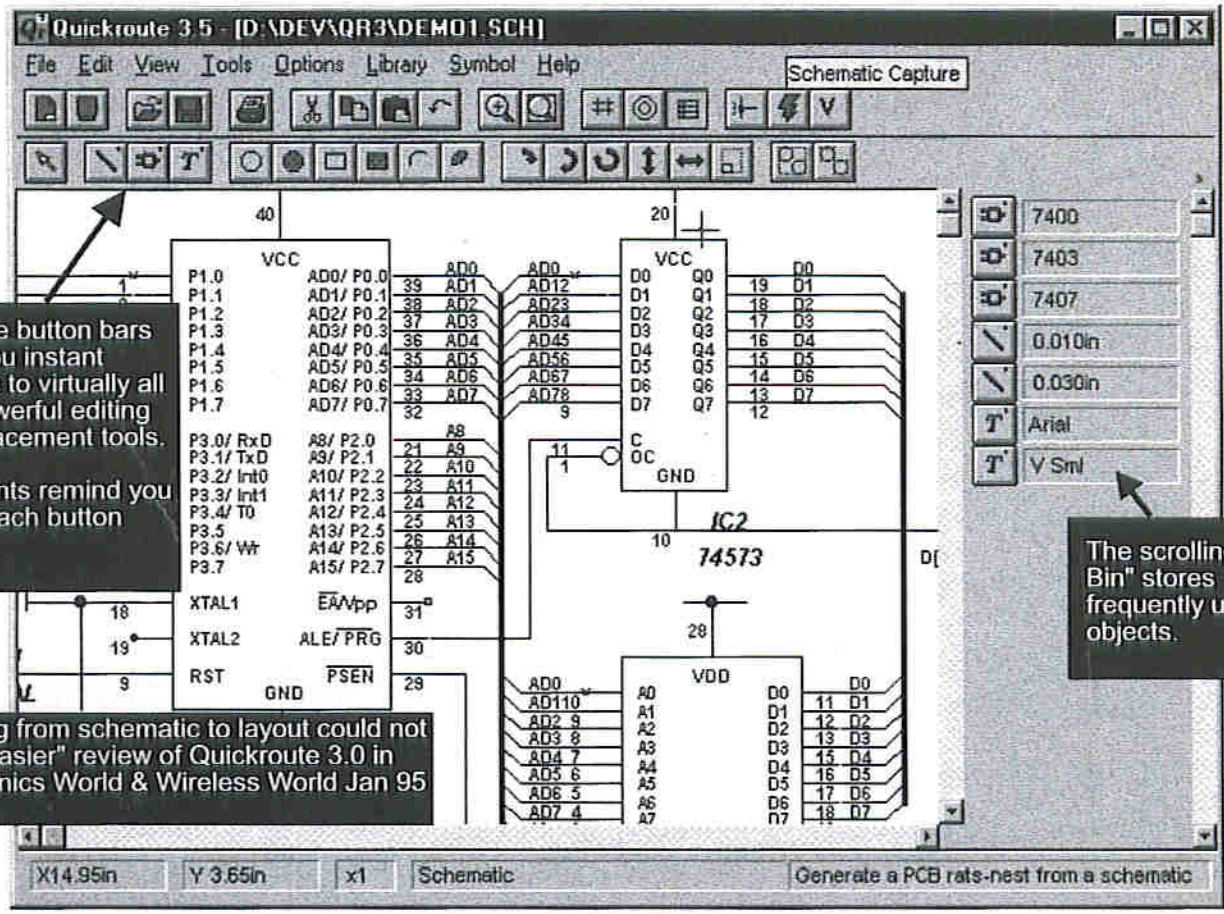
NOVEMBER 1995 £2.50

9 770142 722108 11

NEXUS

NEW

quickroute 3.5

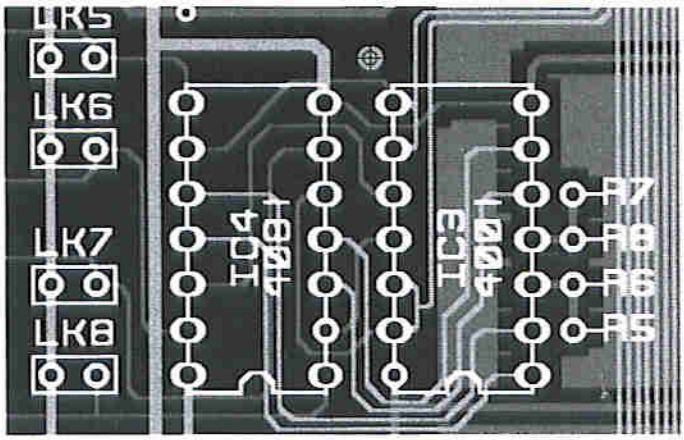


Multiple button bars give you instant access to virtually all the powerful editing and placement tools.

Tool hints remind you what each button does.

The scrolling "Parts Bin" stores frequently used objects.

"moving from schematic to layout could not be easier" review of Quickroute 3.0 in Electronics World & Wireless World Jan 95



Contact Quickroute Systems for a free demonstration pack & brochure. Prices quoted exclude post & packing, and V.A.T.

quickroute SYSTEMS Limited
14 Ley Lane, Marple Bridge, Stockport, SK6 5DD. UK.
Tel/Fax 0161 449 7101
email info@quicksys.demon.co.uk

MasterCard VISA

Personal £68

The Personal Edition gets you started with schematic & PCB drafting for just £68.00. The full set of editing & placement tools are included, with support for 2 copper layers & 1 silk screen layer. You can also create your own custom symbols. Note that the manual is provided on disk.

Designer £149

The Designer Edition of QR 3.5 includes schematic capture & automatic generation of a PCB 'rats-nest'. Routes can then be manually routed, and checked against the schematic. The full manual is also included.

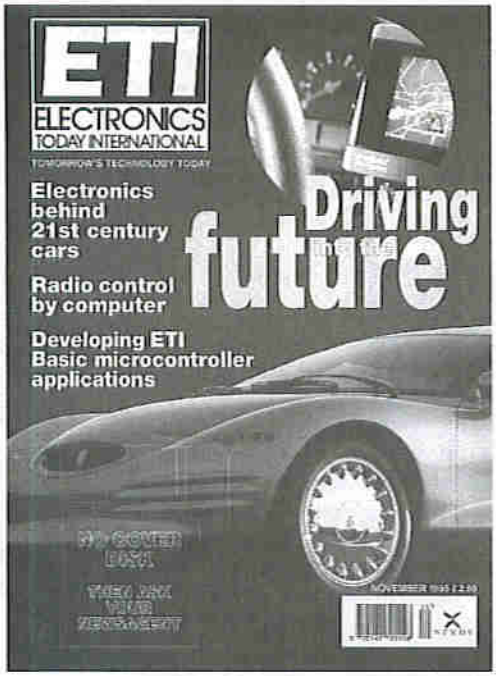
PRO £249

The PRO Edition is our base professional product with support for 8 copper layers. It includes schematic capture, automatic PCB rats-nest generation, an auto-router and support for a range of CAD-CAM outputs. Also included is our extended library pack (CMOS, Surface mount PCB symbols, etc).

PRO+ £399

PRO+ is our full professional product. It supports advanced schematic capture (global nets), copper fill, enhanced auto-routing, and a range of export and import capabilites including GERBER import, and SPICE & SpiceAge support.

Contents



Volume 24 No.11

The ETI Basic microcontroller 22

ETI's new premier project for the autumn, this is a PIC microcontroller system which can be programmed in the widely known BASIC computer language. This month, Robin Abbott shows how to develop applications for this system using the development software provided free on the cover of this issue

Computerised radio control 32

Dr Pei An has developed a simple system which allows a PC to remotely control devices such as robots using a very low power wireless link

Single stepping the 8088 single board computer 47

This month Richard Grodzik looks at how to add this useful feature to ETI's project from earlier this year

Versatile bench power supply 52

In Part 2 of this project, Tim Parker looks at the construction of a practical upgradable regulated bench power supply specifically designed for the electronics enthusiast

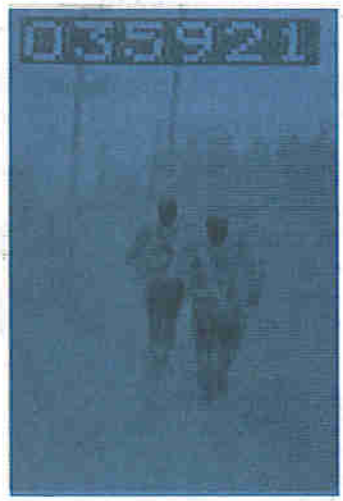
Designing a PIC microcontroller based project 62

In Part 6 of this short tutorial series, Bart Trepak continues his look at the circuit and design of a PIC based alarm clock

& Features Projects

Driving into the future 10

Douglas Clarkeson takes a look at how developments in electronics are set to revolutionise the way in which we use and drive the cars and lorries of the 21st Century



Regulars

- News 6
- PCB foils 68
- Roundup 74

been verified and adopted widely and are 'legal' for use, the much denser traffic of automotia presents a completely different scenario. Even if the car of 2050 was ready to drive on the road of 1995, its extensive 'smart' features would be on uncertain ground within the framework of current Road Traffic Acts.

There are, however, some parallels between the uptake of current technology and how future technology will be utilised. In the case of air bags, while there have been instances of accidental inflation and failure of systems to inflate, the significant number of lives saved and reduction in severity of injuries with successful inflations have given this technology wide acceptance. It is probable, therefore, that this will also be the pattern of uptake with the new auto safety technologies.

The PROMETHEUS framework

PROMETHEUS stands for 'PROgraM for a European Traffic with Highest Efficiency and Unprecedented Safety'. The programme was initiated in 1986 and the conclusion of the primary phase of the project was reported on in Paris in October 1994. This first phase was the pre-competitive research programme and involved a wide uptake within the European car manufacturers. The nature of work undertaken tended to include partnerships where even major manufacturers joined together to develop aspects of PROMETHEUS technology. Many of the schemes within the PROMETHEUS framework, however, rely on the development of a co-ordinated European transport policy with the goal of a common traffic infrastructure. Thus, hopefully, vehicles crossing national borders within Europe will be able to operate within data frameworks and data links which are the same all over Europe.

There is also a basic division within the PROMETHEUS work of vehicle-autonomous and infrastructure-based systems. A system for, example, which recommends appropriate speed based on road conditions and associated vehicles is an example of a vehicle autonomous system while a system for route guidance depends on an infrastructure being in place for transmission and updating of road details and conditions.

Various vehicle-autonomous systems now exist as highly developed prototypes within specific test vehicles. Some of the infrastructure based systems have been demonstrated in specific geographical areas.

Safe driving technology

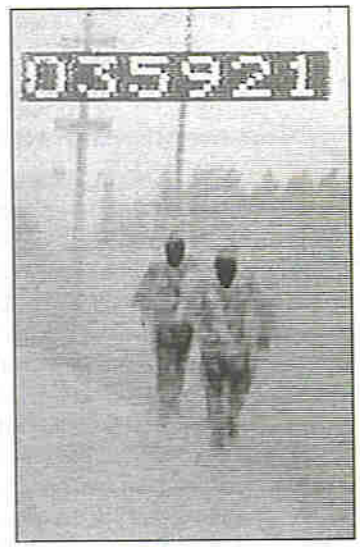
The balance in car design between safety and looks/appearance is hopefully placing more emphasis on intrinsic safety. Considering, however, how long cars have been manufactured in Europe, the trend towards safety as a prime feature is slow, though not unwelcome in coming. A range of specific developments in this sector is now examined.

Vision enhancement

A range of companies are developing ultraviolet headlamps to improve visibility at night. In developments, for example with SAAB Automobile AB, a separate company, Ultralux, has been formed specifically to develop UV systems. Clothing of pedestrians, for example, is more easily picked out with UV radiation. It is planned to use discharge lamp technology for such a development.

Accidents tend to take place more frequently in bad weather under conditions of reduced visibility. Vision enhancement systems are being developed which use infra-red illuminators and infra-red image cameras to provide to the driver a 'clear' image of the road ahead. Such an image could be either presented to the driver as a head up display which matched the normal field of view or on a separate flat panel display. Figure 1a shows what the driver would normally see on in bad weather conditions. Figure 1b shows how infra-red imaging can provide a clearer image in conditions of such poor visibility. Renault SA have demonstrated such an infra-red illumination system to improve visibility in bad weather conditions.

RIGHT: Normal driver image in conditions of poor visibility with driver assisted by impaired headlamps
BELOW: Same field of view but with image obtained using infra-red imaging technology

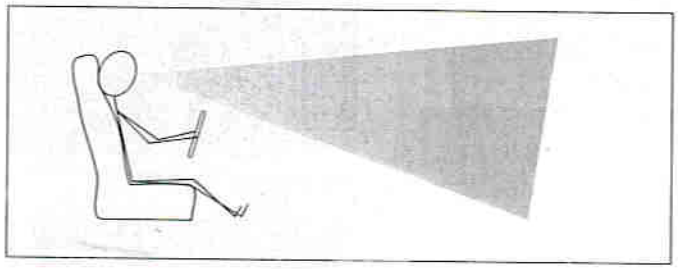


Within this sector, Fiat have developed an ant-glare system based on pulsed headlamps and a liquid crystal panel placed in front of the driver. This ensures, by detecting oncoming light, that the optical density of the liquid crystal panel is rapidly increased with the onset of glare before it dazzles the vision of the driver. A similar technology, for example, has been developed with visors to protect welders from UV radiation from high intensity arcs of light with a high ultra-violet content.

Figure 2: Opel Concept Car: Display indicates measured visibility derived from infra-red backscatter signals

Visibility range monitoring

The Opel Concept Car has implemented an infra-red fog sensor which measures backscatter and derives a visibility range as an image on a separate dashboard display as shown in figure 2. Volkswagen has implemented such as system with a head-up display on the driver screen. There are, however, major implications for the assigning of safe speeds to visibility measurements, e.g. as indicated on a display in the vehicle. Such estimations of speed are principally derived for an 'open road' scenario.



Friction monitoring and vehicle dynamics

As a vehicle travels along a road, the driver is typically aware of only a very small subset of data indicating the vehicle status. While speed, engine RPM, and engine temperature are typically present, parameters relating to friction monitoring and vehicle dynamics are usually not available. Drivers are typically aware only of major differences in road conditions - ranging from high resistance grip surfacing at traffic junctions to travelling on hard packed snow. It is likely, however, that future car systems will provide monitoring of road friction and alert the driver to any reduced margin of safety.

Such 'smart' systems would be able to monitor the car's function in routine braking conditions and assess the efficiency of the vehicle's braking system with relation to the prevailing road conditions. Such systems should be able, for example, to characterise the braking function of each individual wheel.

Again, there is the process of recommending vehicle speed in respect of available road traction information. This has implications of product liability. A new defence of the driver in court could be that he (or she) was driving with due care and attention and driving at the speed recommended by their on-board car management system. If that was faulty - then the driver could be found innocent and the vehicle manufacturer guilty.

Data is also scheduled to be captured for monitoring and control of rear wheel camber angle during lateral manoeuvres - e.g. where there is a risk of loss of sideways traction. Excessive side swing of rear of vehicle can result in skidding of rear wheels and loss of vehicle control. Product development in this field is due to commence during 1995. There is also the advantage of communicating data relating to surface friction into an information infrastructure so that other vehicles can be alerted of the road surface conditions prior to encountering it. In such a system, it would be desirable to communicate such data only to vehicles in a zone of several km.

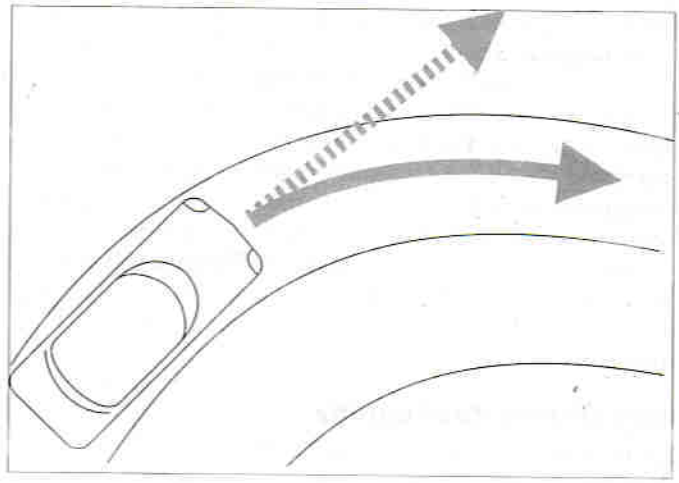
One of the major players in this field is Porsche which is co-operating with Darmstadt University and IPG Karlsruhe. Porsche is developing a multisensor system to detect wet and slippery road conditions for all driving situations. Such data will be interfaced into a variety of systems whose software algorithms will determine a speed limit and inform the driver using adaptive strategies.

Lane keeping support

Increasingly, accidents are caused by drivers becoming drowsy at the wheel and drifting out of their intended lane. A range of systems are actively being developed to assist the driver to maintain the lane selected.

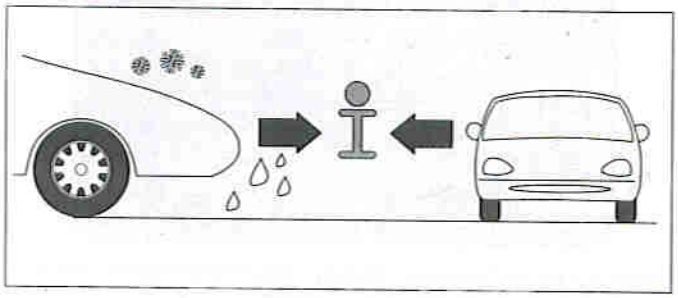
One option is to include sensing systems which recognise lane markings and control a torque motor on the steering column to help the driver maintain the current lane. Matra has developed such a system with an additional screen on the dashboard which provides a TV image of the road ahead and the degree of compliance of the driver with lane control. A system has also been developed by Jaguar which provides lane support through a similar system. For such systems to be effective, however, lane markings require to be established more clearly on the road.

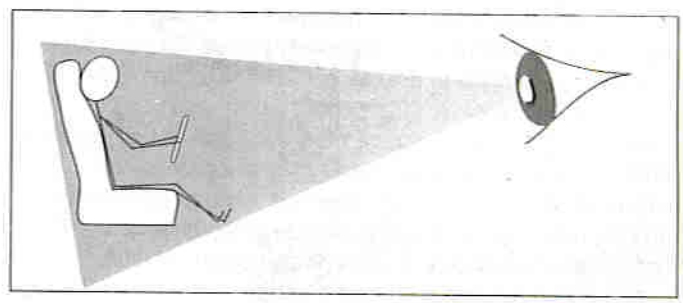
Also, monitoring of lane keeping may reduce driver strain and reduce tiredness on longer journeys.



Driver status monitoring

It is estimated that there are more road fatalities as a result of drivers falling asleep at the wheel than due to the effects of impaired driver performance due to alcohol. A number of research organisations have been studying this phenomenon and several companies within the PROMETHEUS initiative have been developing prototype demonstration systems.



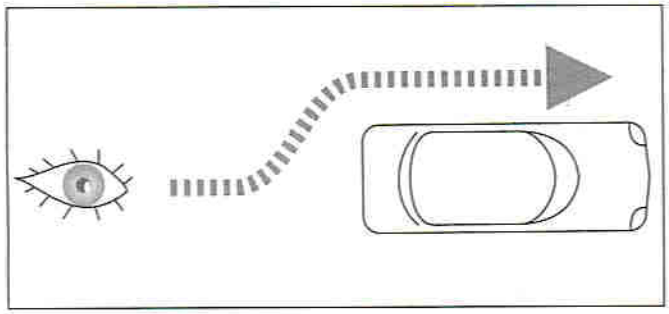


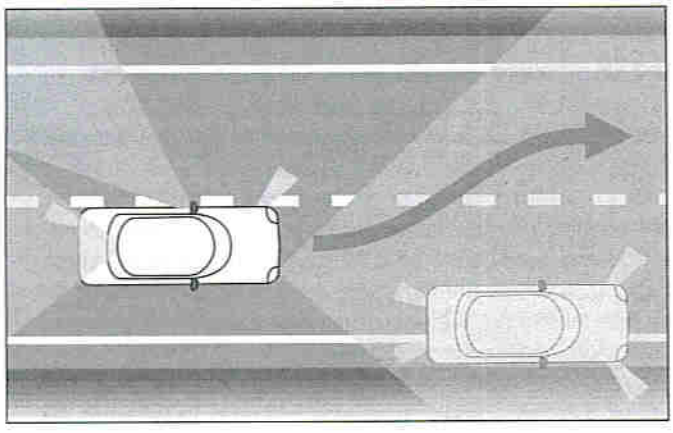
Look no hands: DaimlerBenz advanced VITA II Vision Technology Application for collision avoidance. The technology allows for lane change under control of the collision avoidance system

Accidents often result from driver impairment caused by reduced stimulation. This is why driving on uniform motorways where there is little for the driver to do can lead to drowsiness and even falling asleep at the wheel. It may be argued that the best way to beat the problem is through better education about driving techniques and advising people not to drive for excessive periods. While this may help in some respects, there is a strong argument to introduce technology to monitor the status of the driver.

In such a system, the responses of the driver to various situations is continuously monitored and compared with data considered representative of 'alert' driving. In the context of lane keeping, for example, usually corrective steering for lane discipline will have a 'sharp' type of response - e.g. 90% of all corrective procedures taking place within 0.5 second of the need for correction. If the time frame for this correction increases significantly, e.g. to 1 or 2 seconds. then this is a good indication that driver fatigue is setting in.

There are a range of technologies that could be used to monitor driver status. Systems with self training neural networks may be particularly appropriate. Renault are, in fact, combining their lane keeping system with driver status monitoring.





Collision avoidance

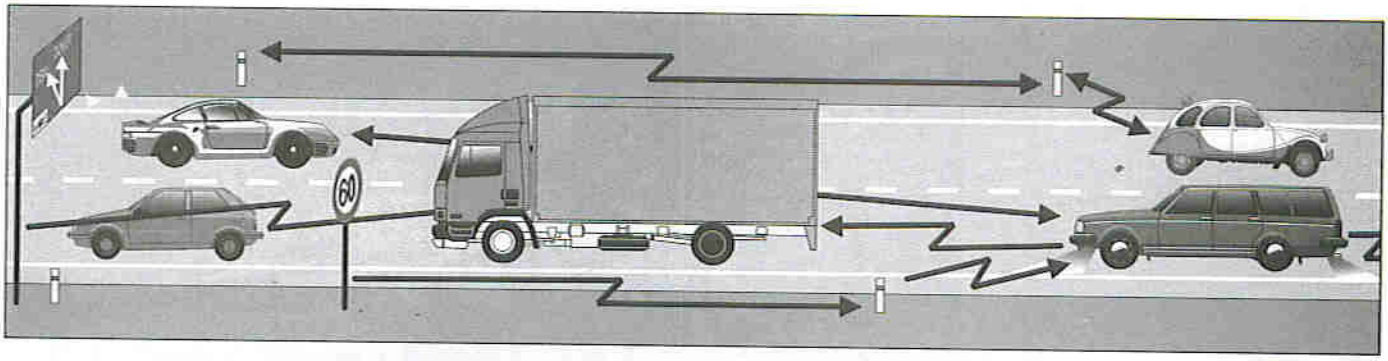
Accidents may be described as unintentional collisions. While not all accidents may be prevented with the introduction of new technology, it is self-evident that a large number could be. Take the simple case of a car which fails to brake on time when it encounters a stationary queue of traffic. Or in the car park where one car reverses into another. Arrays of sensors could easily be deployed to detect 'safe' environments around vehicles and, in certain situations, control vehicles so that collisions did not take place. A typical array of sensing systems is shown in figure 3.

Often the prime sensor systems for collision avoidance can be used for other functions such as intelligent cruise control and lane support. In a Jaguar demonstrator vehicle, for example, the output of an AICC radar sensor (replacing one of the twin nearside headlamps) and a lane support video camera are used to provide a reliable and robust forward looking collision warning system.

On the Fiat ALERT demonstrator, a microwatt radar sensor provides for automatic braking in the presence of obstacles in the vehicle path. For parking, the vehicle is equipped with a microwave antenna in the rear bumper.

Perhaps the collision avoidance technology is most highly developed in the DaimlerBenz VITA II Vision Technology Application which can demonstrate autonomous automated driving. It can, for example, overtake automatically, undertaking lane change manoeuvres based on object detection and recognition. The driver can have his foot on the accelerator pedal but no hands on the wheel. A variation of the system includes vehicle following with adaptive distance control. Such a system is shown in action in figure 4

A similar system is being developed by BMW. The BMW demonstrator provides a Conflict Zone Monitoring (CZM) system and provides integrated driver support to facilitate both longitudinal and lateral control. Figure 5 shows a short range collision avoidance system developed by Jaguar.



Short range collision avoidance is a feature developed by Jaguar. Low-cost, high-performance, infra-red sensors are fitted to the rear and side of the vehicle. One sensor warns of obstacles while reversing and another monitors the 'blind spot' for overtaking vehicles

Co-operative driving

In its broadest sense, Co-operative Driving is the principle whereby relevant information that becomes available to one driver could improve the safety of other drivers in the vicinity. Thus, if one car detects an icy road surface by measuring change of friction dynamics, this information is of value to approaching cars who have the opportunity to adjust their speed and vehicle following distances accordingly. This is the type of technology, therefore, that requires an information infrastructure in place before it can be fully implemented. Figure 6 shows the communication links utilised by the Matra-MANET system.

Communication can be between vehicles and between a network of beacons which are in turn connected to traffic management centres. Such a system is considered to provide for increased traffic safety and efficiency.

Again, however, there are key issues of standards and legislation. The standards, for example, would relate to wavelengths used for local transmission and the format and content of the message traffic. Some basic dialect-free 'language of the car' is required which can be readily implemented by a range of vehicle manufacturers

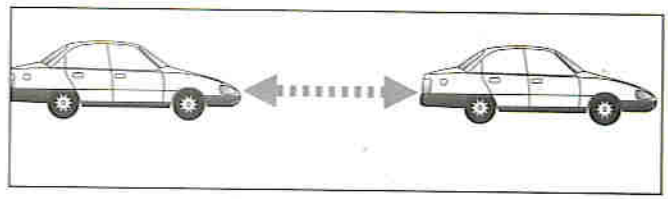
It is likely, however, that the potential economic benefits of co-operative driving will have to be assessed very thoroughly before investment went ahead into such expensive infrastructure developments.

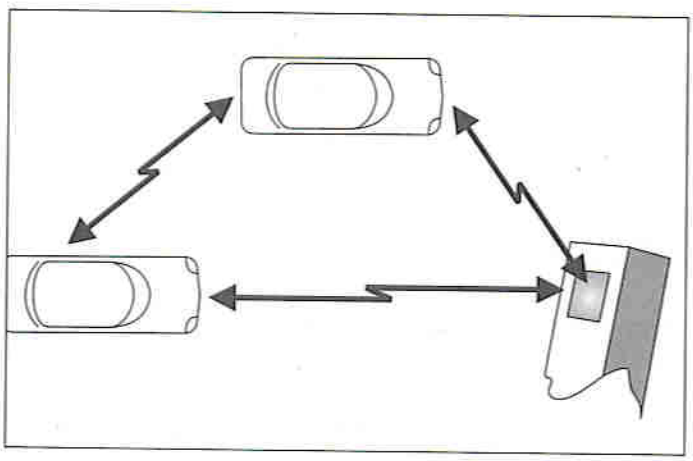
At a practical level, there is often the need to send a (polite) message to a car in front - e.g. 'Your rear left tyre is under inflated.' or 'Your mountain bike is ready to fall off the back of your car'. Presumably, these messages could be coded and sent via the co-operative driving system. The key element in such a system would be the man/machine interface for the driver information display.

The Fiat demonstrator vehicle provides for such a bi-directional data flow. In the case of a crash it automatically sends an emergency call via the cellular telephone.

The Matra demonstrator with co-operative driving facility includes Emergency Warning (EW) and Medium Range Pre-Information (MRP) integrated in the vehicle to vehicle and vehicle infrastructure communications system developed my Matra (MANET: MAtra NETwork). It is anticipated that such systems will be especially effective on motorways in the prevention of pile-ups. A co-operative driving feature has also been implemented by Opel/GM in its Opel Concept Car.

Renault S.A. has implemented a system claimed to be compatible with ADAMS (Automatic Debiting Application for New Motorway Services) developed in partnership with COFIROUTE - the French motorway operator.

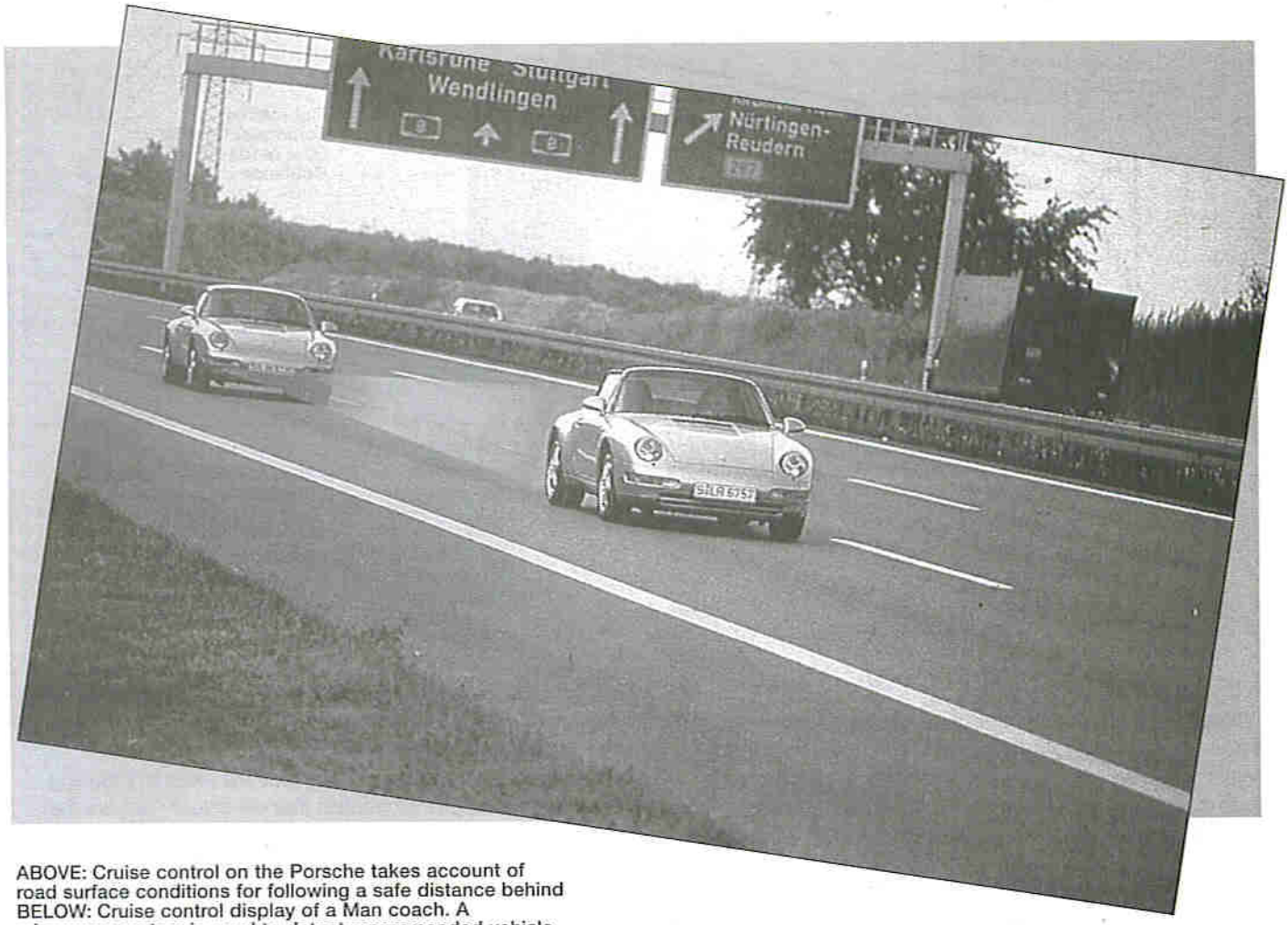




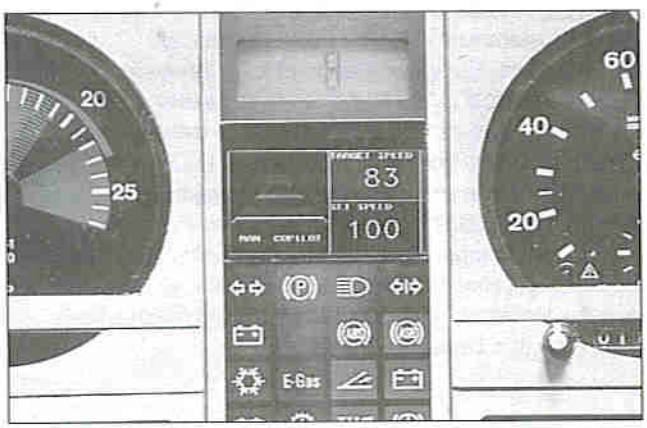
Autonomous Intelligent Cruise Control

This facility takes components from previous 'Safe Driving' elements to provide automatic control of speed and distance in relation to the preceding vehicle. A key part of such Cruise Control is advising the driver on keeping a safe distance behind leading traffic. Such a facility is also considered to provide improved comfort for the driver and reduce the strain of driving. There is, perhaps, with this facility, an increased risk that with less to do, the driver may lose concentration at the wheel more easily on long journeys.

A range of systems have already been demonstrated. Renault S.A. have employed a laser telemeter to measure the



ABOVE: Cruise control on the Porsche takes account of road surface conditions for following a safe distance behind
BELOW: Cruise control display of a Man coach. A microwave system is used to detect recommended vehicle distance separations

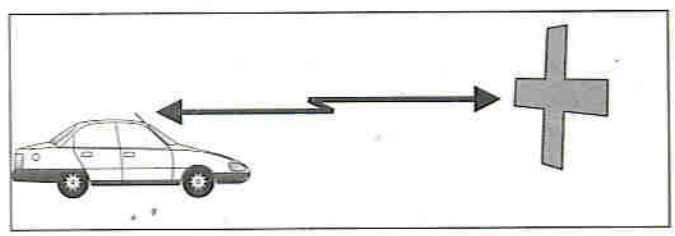


distance to the leading vehicle. In this system, the driver fixes the speed and the car takes this as a maximum and then regulates the car speed and separation with respect to following vehicles. The Matra CCD based Lidar sensor, which complies with NF EN 60825 for Class 1 safety acceptance, is an example of a laser system with intrinsically safe characteristics.

A large part of the development of PROMETHEUS technology is providing the driver with new information in an acceptable way. The intelligent speed and distance control feature developed by Porsche provides a conventional speedometer with zones highlighted in red to warn where speed is above recommended values.

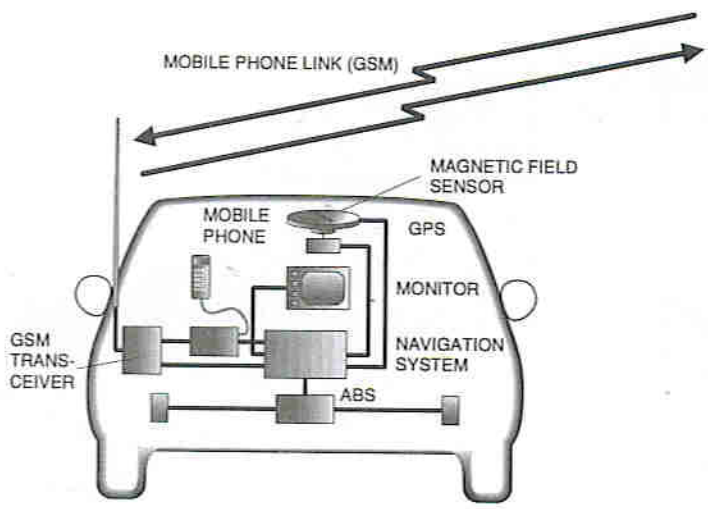
The Opel Concept Car utilises both microwave RADAR and infra-red forward looking sensors for its intelligent cruise control system. In the Volkswagen AICC demonstrator, data on road condition is taken into account is determining the safe following distance. Data is presented in the format of a head-up display.

Figure 7 shows the field of sensor beams used by Porsche to implement an AICC system. Figure 8 shows the AICC display of a Man coach which uses a microwave sensor to measure vehicle separations.



Automatic Emergency Call

The Automatic Emergency Call facility is designed for use in emergency situations where vehicles involved in accidents are automatically located and rescue measures are initiated by specially equipped mobile phones (GSM). The major benefits are considered to be faster rescue operation and higher efficiency in faster clearing of traffic around the accident site. Figure 9 indicates how vehicles involved in a collision could relay information to roadside beacons to alert other vehicles along the road and also to regional/subregional networks linked



Fiat Interlink demonstrator with Dual Mode Route Guidance

to the emergency services.

Already, however, a significant number of road accidents are reported to the Emergency Services by conventional mobile phone calls from motorists. This is an area in which groups such as the RAC and the AA in the UK could have a significant technical input into designing such automatic emergency call technology.

An on-going development is the extension of the GSM function for vehicle location, so that a distress call can be detected and its location immediately pinpointed.

There is also the potential disadvantage of such systems - they could be activated as false alarms by time-wasting individuals or for trivial problems such as running out of fuel. Fake calls at this level should be considered equivalent to fake 999 emergency service calls.

Dual mode route guidance

In this mode, as shown in figure 10, the vehicle receives data from a range of inputs. The heart of the system would be an on-board computer with a digital road map of the area on CD-ROM and which supported an autonomous vehicle navigation system. Updates on traffic conditions would be received on GSM cellular radio or RDS/TMC standard radio broadcast. The on-board computer would assimilate the relevant data sets and derive the appropriate route for the vehicle. Such systems would have the ability to allow drivers to rapidly locate alternative routes. The reduction of traffic jams would have significant economic and environmental advantages.

The technology has already been demonstrated by a number of car manufacturers. BMW has developed a system using components indicated in figure 11. The CARIN system provides the basic logistics using GPS satellite positioning. This is updated with data transmitted on radio under RDS-TMC or on a cellular link using mobile phone GSM standard.

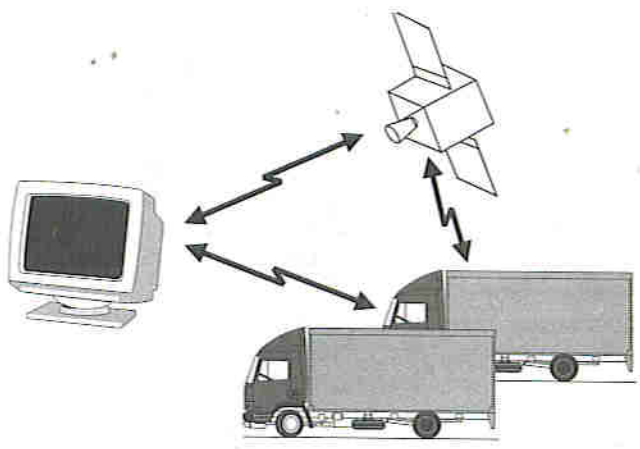
The Fiat interlink demonstrator utilises a similar system as shown in figure 12. The prime guidance system is provided for by GPS satellite links and additional 'best route' data is provided via RDS-TMC radio. Such systems usually provide a route map prompt for the driver directly in a central position on the dashboard.

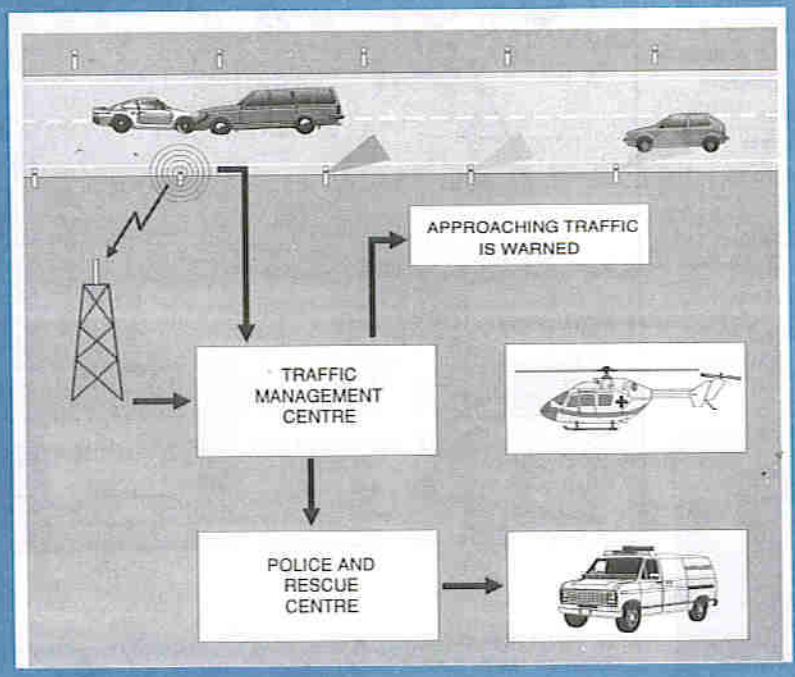
DaimlerBenz has developed a system which utilises a digital road map with updates transmitted from RDS/TMC radio. In terms of the driver interface, both optical (colour screen) and audio information systems are provided.

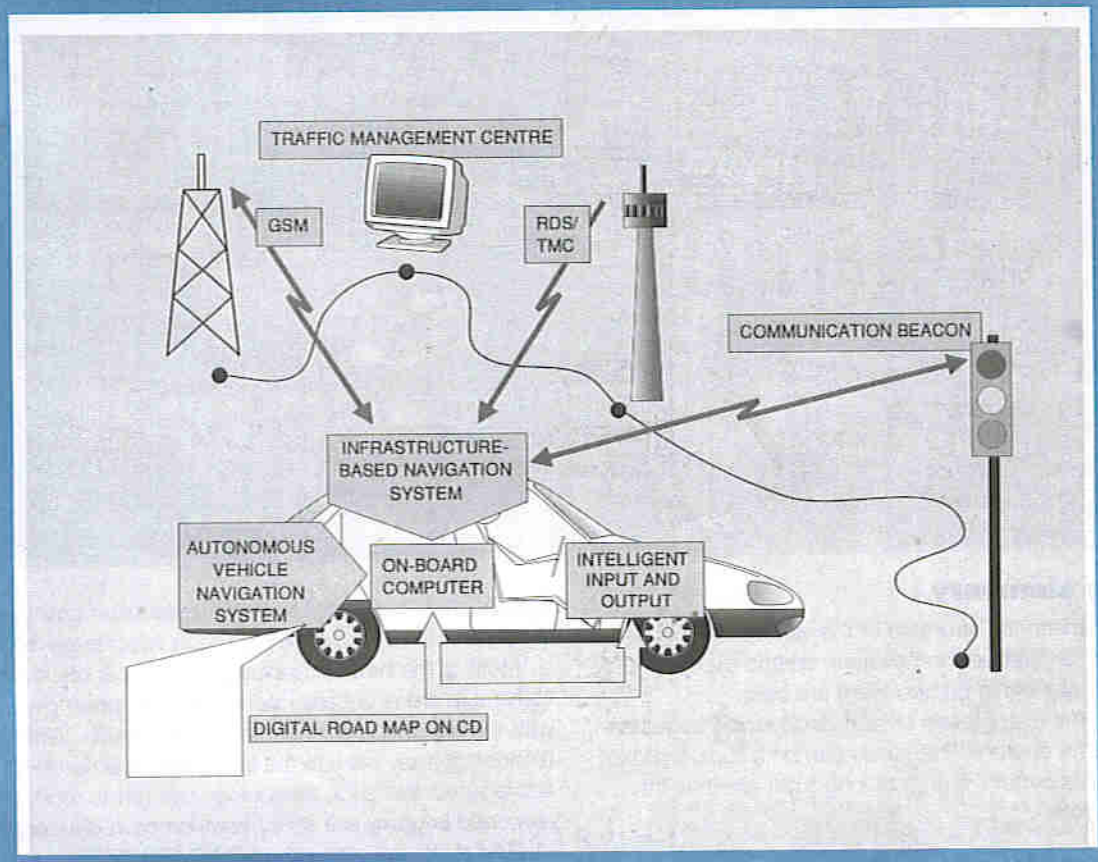
The benefits, however of such systems relate to providing the driver with useful information that will actually prevent the driver from driving into traffic jams. The approach of Volkswagen with their Route Guidance System has been to intelligently interface planned route with traffic problem data so that effective route changes can be made at an early enough stage.

Volvo has, in turn, developed a range of products. DYNAGUIDE, as shown in figure 13, uses RDS-TMC radio links to drive an in-car display to graphically show where traffic incidents are. The second product, SOCRATES, interfaces a navigation system with real time traffic information.

Within Europe, European Geographic Technologies BV (EGT) based at Best in the Netherlands is generally recognised as the industry leader in the development of navigable map data bases for route guidance applications. EGT is likely to play a leading role in providing the digital infrastructure data for dual mode route guidance systems. EGT has extensive involvement in a range of parallel European initiatives. During 1995, EGT's detailed city data bases will cover all German cities with more than 100,000 inhabitants and major cities in Italy, the U.K., the Benelux and France.

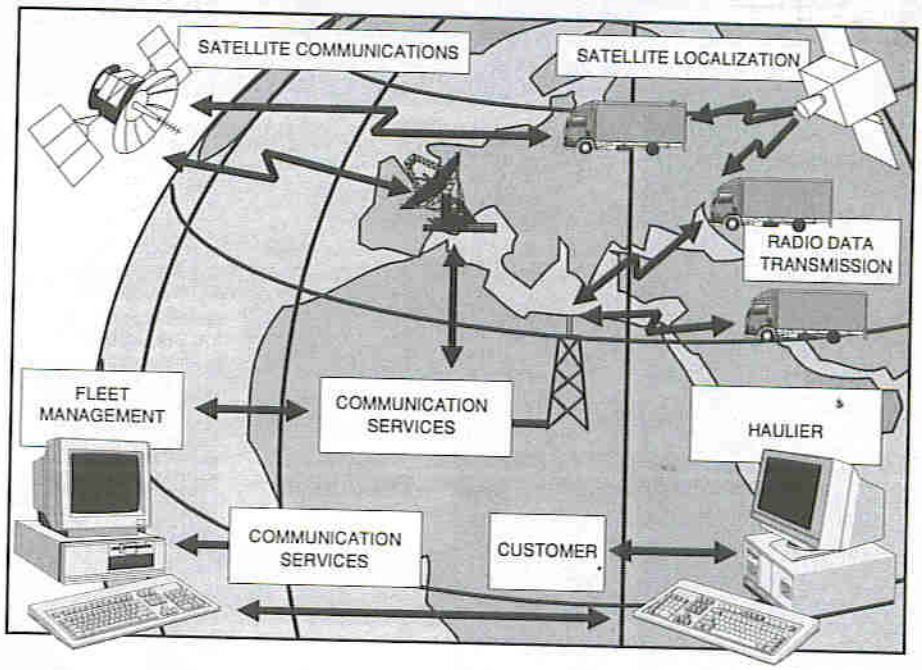



APPROACHING TRAFFIC IS WARNED
TRAFFIC MANAGEMENT CENTRE
POLICE AND RESCUE CENTRE


TRAFFIC MANAGEMENT CENTRE
GSM
RDS/ TMC
COMMUNICATION BEACON
INFRASTRUCTURE-BASED NAVIGATION SYSTEM
AUTONOMOUS VEHICLE NAVIGATION SYSTEM
ON-BOARD COMPUTER
INTELLIGENT INPUT AND OUTPUT
DIGITAL ROAD MAP ON CD

Fleet management

Fleet Management has become one of the 'actual' technologies to be widely taken up. A summary of the complex data links of such a system is shown in figure 12. The key to rapid development in this particular field has been the wide deployment of mobile cellular phones. While some systems have been implemented which utilise satellite systems for wide area coverage, for short to medium distance lorry operators are favouring systems utilising mobile telephones. The MOBIGUIDE system developed by Volvo, for example, utilises data transfer capabilities of various mobile networks such as NMT and G



Volvo DYNAGUIDE system. A road map is presented together with updates on traffic systems and possible delays

Travel and traffic information systems

This initiative seeks to transmit general travel information via existing FM transmitters without interrupting the audio programme. The data pathways in such a system are shown in figure 15. Messages are coded (RDS/TMC) to be silent and language independent. It is anticipated that car receivers will be presented with this information using speech, text or graphics. While the data will be multilingual, the driver will be able to select information in his own language. Such systems have already been deployed within Europe.

An extended version of this type of information system is being developed for use with a range of devices linked to bi-directional communication links such as GSM. It is planned that such systems will provide information on a broad range of topics including public transport information and schedules, park and ride information, parking availability and traffic reports. It is anticipated that better information about public transport will encourage its use and consequently decrease urban traffic.

While such technology can be readily demonstrated, there is the requirement for deployment of harmonised data networks Europe-wide. Also, there is a considerable role for service providers to update databases and provide connectivity to the appropriate networks.

Technology summary

Figure 16 summarises the structure of the various sectors of the PROMETHEUS initiative. Somewhere, spread over a goodly number of filing cabinets in Europe there are draft specifications of the major levels of standardisation that will be required in the 21st century. This surely can be a topic that can be looked at with a certain degree of long sight which is free from political carrots.

Lateral thoughts

In terms of cost of additional PROMETHEUS components, one argument is that this will add significantly to the cost of a car. The counter argument would be that over the lifetime of the car, there will be the reduced cost of motoring accidents.. However, estimations cannot effectively be undertaken until there is a significant number of such high-tech vehicles on the road.

Most of the new technology, however, is placed in the vulnerable areas of cars - behind the front bumper or shared with the headlamp/sidelight lamp array areas. While the new technology may work in the laboratory, it is no doubt appreciated that such technology has got to work with a car drenched in grimy salt spray from lorries in dark conditions.

One of the key aspects of PROMETHEUS is to determine what basic features cars of the future should incorporate to allow a range of PROMETHEUS technologies to be implemented. It may be appropriate, for example, that the rear of cars includes reflective RADAR and LIDAR devices.

One of the basic unmistakable characteristics of road traffic

is that the use of the system is basically unplanned. This is unlike the rail network or commercial air space where all journeys are filed and approved and 'known' about via a central scheduling system. In contrast, the road network has the freedom of 'lack of such controls'.

It is being realised, perhaps late in the day, that the solution to an improved road traffic system could be improved road management rather than more miles of motorway or dual carriageway.

Looking down on a busy motorway at night the scene typically displays a swath of headlights and tail lights winding on for miles and miles. If various of the concepts of PROMETHEUS are deployed, then a new range of signals will surround the motorway - radar, laser and microwave and cellular radio as information traffic travels car to car, beacon to beacon along the lanes of the motorway.

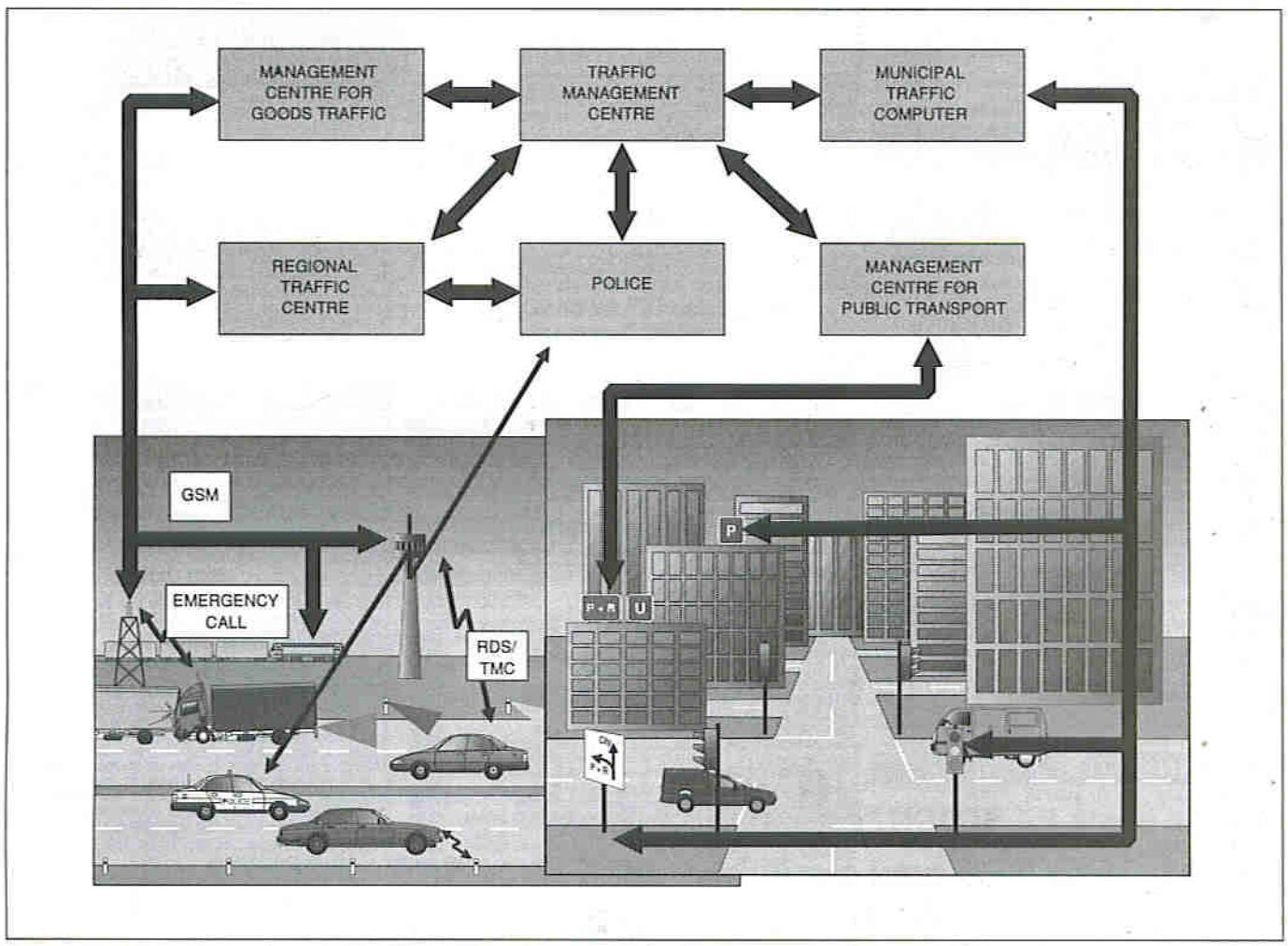
There is also, on the far horizon, the potential for the use of PROMETHEUS-like technology to be used in trade wars within the motor car industry. Trading blocks could, for example, specify specific technology criteria that cars would have to include as standard before they were could be sold within a given trading bloc.

As yet, the clarification of cost for the PROMETHEUS-type technology has not been identified in the 'glossy' brochures issued from the project. There is, however, the need to look at costs on a wider scale than simply the cost of an on-board system and the infrastructure cost of support systems per km. More effective traffic flow, more economical use of fuel, prevention of accidents, also appear on the balance sheet.

There is considerable importance, also, for the U.K economy in being at the heart of forthcoming standards for in-vehicle and vehicle to infrastructure systems for car safety. The potential market for such technology is certainly large and will continue to grow with the future planned expansion of the European Community.

In the short space of eight years, during which the PROMETHEUS project has been running, technology has developed extensively in many of the core areas required to implement such technology. Systems for image recognition, for example, derived from automated control in conventional industrial manufacturing now can demonstrate many of the key requirements of the PROMETHEUS systems.

The scope of PROMETHEUS is considerable when the range of products and services related to its function are considered. Within mainland Europe, there appears to be considerable commitment to the project's themes of working for safer and less congested roads due to the implementation of appropriate technology. It is certainly in the interests of the U.K. car manufacturing and component manufacturing industry to take such concepts on board as the cars of the 21st century begin to take shape on design screens all over Europe.



Basic for the pic microcontroller

The project can be programmed using the development software contained in this month's free cover mounted disk.The software will run on any PC running Windows 3.1

The developer of this unique project, Robin Abbott, describes how to programme and use the ETI PIC based controller module, a very small and cheap computer which runs BASIC and is intended as a building block for any project requiring program control.

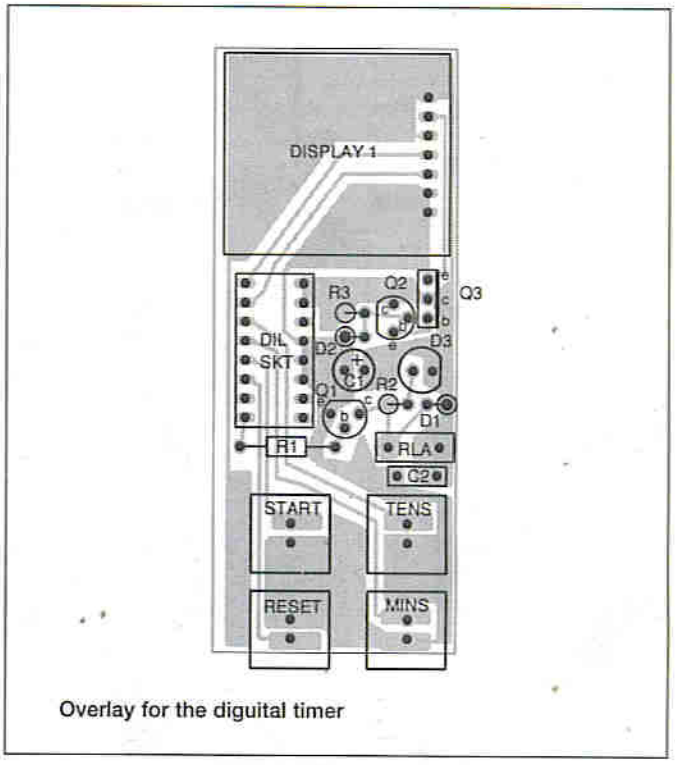
Last month we looked at the PIC BASIC controller module for 18 pin devices. This month we will look at the development system for PIC BASIC programs. PIC BASIC programs are developed on a PC using a development system which runs under Windows 3.1. The development system is included free on this month's cover disk.

The key features of the development system are shown in Figure 1.

To install the development system, copy the files from the cover disk to a temporary directory. Most PCs have a directory called C:\temp, this is a good location to copy the files. Now uncompress the files by by typing

from DOS. Now run the installation program from within Windows. This may be achieved from the program manager by using the File I Run menu option and typing C:\TEMP\INSTALL.EXE into the Command Line box if you used the TEMP directory to uncompress the files. The installation program will now create the program directory, and the program group in the program manager.

Run the development system by double-clicking the PICBASDE icon with the mouse. The development environment will now start up. If you have a module connected then you should define the communications port to be used with the Module Communications menu option which will allow the port used for the BASIC module to be defined and the baud rate used to be set up. Use a baud rate of 9600bps for the 4MHz 16C84; the actual bit rate used depends on the PIC in use.



Overlay for the diguital timer

Using the development system

The development system is a complete environment for the development of PIC Controller BASIC programs. The system defines the files and options which make up a program in a special file type called a project. When a project file is saved, all the open files which are being edited are saved, the list of BASIC files which are to be compiled and assembled are saved and all the options in use are also saved. Figure 2 shows the development environment in use on a simple project which only has one BASIC file as part of the final program.

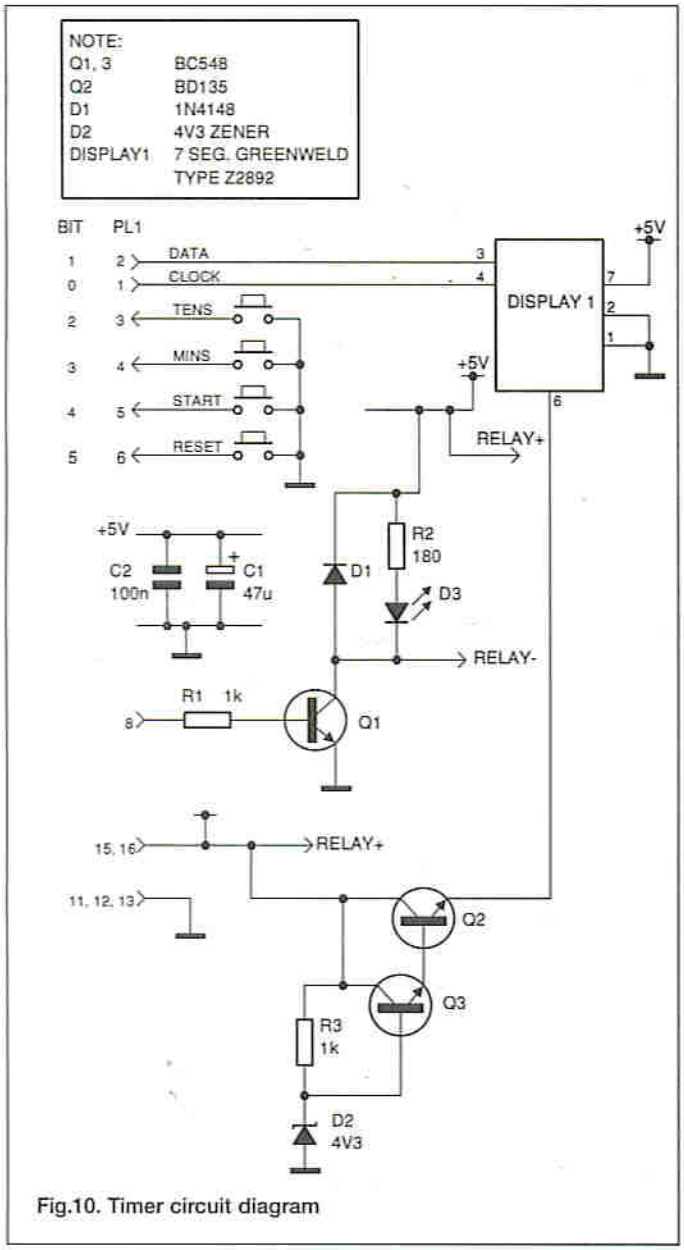


Fig.10. Timer circuit diagram

At the top of the main window is a row of buttons which may be used for rapid access to some of the functions of the program. Hopefully the icons on the buttons should be fairly self explanatory - but my drawing skills sometimes leave something to be desired! However, if the mouse pointer is moved over a button and left for a second or so, then a small yellow "bubble" will appear with a help hint for the button.

At the bottom of the screen is a status bar. This contains a Saved/Modified flag for the current file window, the position of the cursor in the current file window, the caps lock and numeric keypad lock keys. The status bar also displays help hints for the buttons and the menu options. When scrolling through errors the status bar shows the current error.

There are five permanently open windows. These are the information, terminal, debugging, message and project windows.

The project window holds a list of files which make up the final program. The files can be PIC Controller BASIC files which have an extension of .BAS, or PCODE files which have an extension of PCD. The use of PCODE files is more advanced and will be covered in a later article; however, they can be used to make up libraries of useful routines, or to optimise code.

The Message window lists errors when a file is compiled. If any errors are found, the file and location of the error will be shown in the Message window. If an error is double-clicked with the mouse pointer, then the file will be opened and the cursor positioned at the location of the error. The error will be displayed in the scroll bar and then jumping from error to error is possible using the Compile I Next error and Compile I Prev error menu options.

The information window shows copyright messages, and the total code size of the compiled program. The information window also shows the amount of EEPROM used by the program in graphical form with a bar chart.

The debugging window shows a complete list of all the memory variables in the system. Whenever the PIC Controller BASIC program executes a MONITOR() or DEBUG() instruction then the debugging window is updated to display the current value of all the variables in the system. Later versions of the system (with the larger PICs) will allow single stepping of programs to aid debugging.

Finally, the terminal window shows information sent by the module over the serial link. When the terminal window is active, keypresses in the terminal window will be sent to the module.

Example

As an example of the use of the development system we'll take a look at a very simple program. This test program will prompt the user for a keypress which should be a number from 1 to 9 and will then count to this number on the terminal window.

Use the Project I Open/New project menu item and type in the name of the project which will be test. All open files will be closed. Now use the File I New menu option to create a new file. Save it immediately by using the File I Save As menu option. Type in the file name TEST.BAS. Now enter the program shown in figure 3; this has a deliberate error in it. The comments in the file which on each line follow the semi-colon do not need to be entered. Use lower or upper case, they are treated identically. Now use the Project I Add Item menu command to add an item to the project window. In this case, we are adding TEST.BAS to the project, so select this file.

Now to compile and assemble the test file use the Compile I Make Project menu command. A dialog box will appear. Enter the EEPROM size in to the EEPROM box and select the PIC in use (16C84 in this case). Ignore the other options in the dialog box which are used for advanced projects. Click OK and the file will compile. If you have typed it in as shown in figure 3, there will be one error. Now to correct the error, double-click on the Message Window where the error is displayed. This will bring the file TEST.BAS to the fore and the cursor will be shown on the line with the error. The status bar will also display the error so that it is visible when windows are maximised. The line shown should be :

```
serout('\n')                ; Print a carriage return
```

To find out what is causing this error, make sure the cursor is on the word SEROUT and press Control and F1. This will bring up the help file with the topic for SEROUT. Reading this topic it is clear that we must identify the port and pin to which the serial information is to be written. In this case we are using the default serial port, so correct the line to read:

```
serout('\n',defserout)   ; Print a carriage return
```

Now compile and assemble the project again. This time it should compile with no problems and should occupy about 80

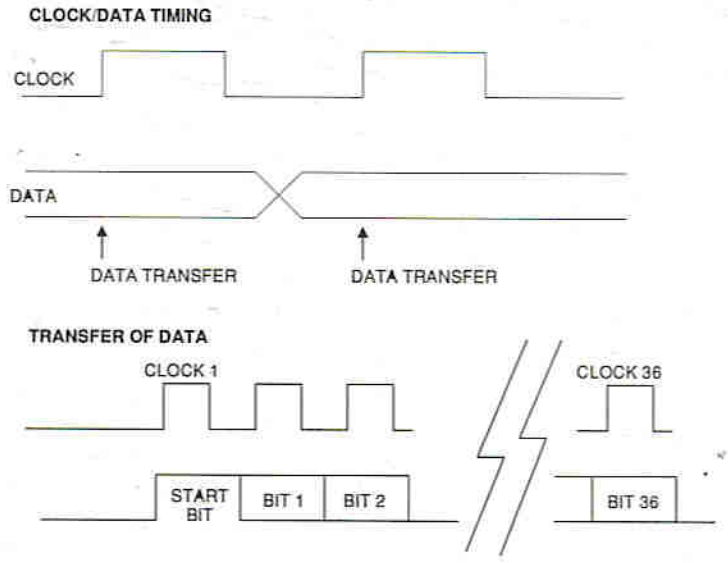


Fig.11. Interface to 7-segment display module

bytes depending on the PIC in use.

To program the module, make sure that it is powered up and connected. Use the Module I Program Module menu command to download the program to the module. A check box will appear with the number of times that the module has been programmed already. Click on OK and the program will be downloaded to the module and then verified. A bar graph will show the progress of the download.

Once successfully downloaded, the program can be run. Use the Module I Run menu command to start the program and click on the terminal window. The terminal should show a prompt requesting a key press. Press a few letter keys to check that nothing happens and then press any number key between 1 and 9 inclusive. The terminal should show numbers counting up to the digit that was pressed and then the debug window will flicker as the program stops and returns to the monitor. Scroll up and down the debug window to find the variables i and key, which should be set to the correct values from the program run.

This is a very brief overview of the use of the development system. Much of the operation of the program is covered in the help file. Most menu commands have keyboard equivalents which are shown on the menus and in the help file. Some also have equivalents on the button bar; for instance the Module I Run command is also the tool bar option which shows a stick man "running" !

A look at the PIC Controller BASIC language

The controller language is shown fully in the file PICDEF.RTF which is included with the software. However, we will look at the language here with an overview of the commands, functions and variables available with the PIC16C84 version of the controller. Other versions of the language for other PICs have extended capabilities and we will look at some of these in later articles.

Numbers and variables

All calculations in PIC Controller BASIC for the 16C84 are performed in 15 bits. This means that calculations can contain any integer value from -16384 to +16383. However, numbers can be stored in variables in less bits than this to save on memory, which is at a premium in the 16C84.

Variables can be defined in size from 1 bit to 8 bits and 16 bits. Variables are packed together to save on memory, so that four 2 bit variables occupy the same space as one 8 bit variable. With 16 bit variables, one of the bits is unused. When used in calculations, all variables are expanded to 15 bits. When a number which is too big for a variable is assigned to that variable, then the variable simply "chops off" the unused bits. Thus, if the number -5 is assigned to a 2 bit variable, the result will be -1. Figure 4 shows the number of bits available for variables and the range of numbers which can be stored. As in most BASIC languages, it is possible to define a variable just by using it. Examine figure 3 and look at the line which reads:

```
key=0 ; Receives from serial port
```

In this case, key has not been used before, so it is defined as an 8 bit varaible. To define a variable as being of any other number of bits, the command DIM is used as follows:

```
DIM flag.1, x.8, result.16
```

This defines a variable flag which is one bit long, x which is 8 bits long and result which is 16 bits long.

Variables can also be held in EEPROM. These variables are always 16 bits long. Note that these variables can be read indefinitely, but can only be written 10,000 times because of the limited number of write cycles in the EEPROM. For example, to set SAVETIME to be a variable in program EEPROM use the following line:

```
DIM SAVEDATE.E, SAVETIME.L
```

Arrays can be defined and used for normal variables and for EEPROM variables. Arrays for normal variables have a maximum size of 16, and for EEPROM variables have a maximum size of 1024, they always start at 0. For example, the following code defines an array of four 2 bit variables, and sets them all to 0:

```
dim array[4].2
for i=0 to 3
 array[i]=0
next
```

To assist in understanding the operation of the program it is possible to define constants. Constants are used anywhere in the program that a number can be used. For example, consider this code:

```
const min='1',max='9'

key=0
while (key<'1') or (key>'9')
etc.
```

This can often be an easier way of understanding a program. There is one important use of constants. This is for look-up tables. If a constant array is defined and used in the program then it will be stored with the program in EEPROM and does not occupy normal memory space. For instance, consider this code:

```
const patterns[10]
patterns[0]=3fh
patterns[1]=06h
patterns[2]=5bh
patterns[3]=4fh
patterns[4]=66h
patterns[5]=6dh
patterns[6]=7dh
patterns[7]=07h
patterns[8]=7fh
patterns[9]=6fh...                    ; other code
here
display=patterns[i]
```

In this case, patterns is a constant array with ten members. Each member contains the pattern for a 7 segment display. Thus patterns[2] holds the 7 segment display pattern for the numeral 2. Then the variable display is set to the pattern to be sent to the display. This type of look-up table is used in the example timer project described below.

The ALIAS command sets up one variable to have more than one name. Consider the line :

```
alias OutputPort=portb
```

This sets up the variable OutputPort so that every time that it is used then the variable portb is used instead; this can make the program easier to understand and is illustrated in the example timer project.

Program control, loops, subroutines and functions

PIC Controller BASIC has a wide range of program control commands. FOR/NEXT loops are fairly standard. For example :

```
for i=-4 to 4 step 2
 number=i
 prtnum()
next
```

This will execute the lines between the FOR command and the NEXT command whilst i takes the values of -4, -2, 0, +2 and +4. i finishes at 6, but the loop is not executed whilst i is 6.Note that (unlike in most BASICs) there is no variable after the NEXT.

WHILE/WEND loops perform a series of actions whilst an expression is true. The following example is identical in function to the FOR/NEXT example shown above.

```
i=-4
while i<=4
 number=i
 prtnum()
 i=i+2
wend
```

IF/ELSE/THEN/ENDIF are conditional statements. IF is followed by an expression. If the expression is true, then the code between THEN and ELSE is executed, if false then the code between ELSE and ENDIF is executed. GOTO causes the program to go to a label and carry on executing from there. The following code is identical again in action to the WHILE/WEND and FOR/NEXT examples, but is harder to understand (and slower) :

```
I=-4
loop:
if i<4 then
 number=i
 prtnum()
else
 goto loopend
endif
i=i+2
goto loop
loopend:
```

Note that there is another form of IF/THEN statements. Consider the following line from a number printing programme:

```
if number<) then number=-number : serout ('-
',defserout)
```

In this case there is no ELSE or ENDIF. The other statements on the same line as the IF are the only statements exectuted if the expression is true.

Subroutines and functions are very similar. A subroutine is a fragment of code which can be called to perform a specific function. A function is similar, but it returns a value which can be used in calculations. Subroutines and functions must be defined before they are used using TYPESUB and TYPEFUNC. The PIC Controller BASIC definition in PICDEF.DOC describes functions and subroutines in much greater detail. The code in figure 5 shows conversion of fahrenheit into centigrade. This is included on disk under the project name of "FTOC.PRJ". This project makes use of a utility routine called PRTNUM, which prints the number in the variable called "number" to the serial port. To use this routine, we include the file PRTNUM.PCD in the list of project files using the PROJECT I ADD ITEM menu option and include the definition of the prtnum subroutine. This is done using the line:

```
include "util.inc"
```

Open this project and examine the code to see how it has been done

Controlling the PIC

So far we have looked at the PIC Controller BASIC language and variables, but have not yet looked at interfacing to the outside world, or to the PIC's peripheral devices. The ports of the PIC are represented by built in variables which do not have to be defined. For instance, port B is represented by a variable called "portb", an array of eight 1 bit variables called "B" and eight one bit variables called B0, B1, to B7. There is also an array called "PORT" which is an array of 8 bit variables; port[0]

represents port A, port[1] represents port B etc. If this all seems confusing then it's only because all these variables are provided for flexibility, they are fully documented on the disk! The following four examples all set bit 4 of port B to 1, whilst leaving all the other bits of port B as they are:

```
1)    portb=portb | 16
2)    port[1]=port[1] | 16
2)    b4=1
3)    b[4]=1
```

In the first and second examples, the 'I' is a bitwise OR function. Similar variables are defined for the other ports.

Now on power-up, the PIC sets all of the ports to be read only. To set them to output, the TRIS() functions are used. TRISB() sets outputs on port B, TRISC() on port C etc. The TRIS() functions need one parameter which is an 8 bit number with a 0 in each bit which is an output. For example, the following code sets the bottom four bits of PORT B to output, the top four bits, are input and sets all the bits of port C as outputs.

```
trisb(11110000n)
trisc(00000000n)
```

The n after the number indicates that it is binary.

Similar variables are defined for all of the PIC's main registers. For instance, RTCC is the real time clock register and the OPTION() function sets the PIC's internal option register.

PIC interrupts

PIC Controller BASIC fully supports PIC interrupts - provided that they don't occur too frequently. These are fully discussed in the example project.

PIC Controller BASIC keyword summary

Figure 6 shows a summary of all the PIC Controller BASIC keywords and built in variables for the 16C84 version of the project. They are fully described with examples in the documentation supplied on disk and in the help file.

Example Project - Universal timer

To illustrate the use of the PIC BASIC Controller in a real application, we will have a look at a timer project which can be used as a photographic darkroom timer, or as a timer for an EEPROM eraser or PCB UV exposure box. This project illustrates the use of i/o on port B, and also shows the use of interrupt routines for timing functions. The project will have the following features:

- Counts down from 99 to 0 in hours, minutes or seconds.
- Timer will have a 2 digit, 7 segment display to show the remaining time.
- Output drive will be suitable for a relay for control of mains voltage.
- Control by 4 push buttons.
- Timer will use interrupts for time keeping.
- Stores the time last set through power down and powers up with this as the default time.

In practice, the circuit shown is suitable for general purpose I/O for the PIC BASIC module. The 2 digit, 7 segment display can easily be extended to 4 digits and the push buttons and relay output can be used for any function.

The timer has four buttons: tens, units, start and reset. When it powers up, the unit enters the time setting phase, the relay is off and the display shows the last time that was programmed into the timer. The tens and units buttons are used to set the time; if they are held down, the time will increment rapidly. If the time reaches 99, then the next timer set button pressed will increment the time to 01. When the correct time is showing, the start button will turn on the relay and start the timing period. The time will count down on the display to 0, at which time the relay will turn off and the unit will enter the time setting phase again. The reset button will turn the relay off and return to the time setting phase.

Circuit Description

The circuit diagram is shown in figure 10. The interface to the PIC BASIC module is through a 16 pin IC socket. The interface cable is 16 way IDC cable with a 16 pin DIL header at each end.

The 7 segment display used is a dual-digit, 7 segment display module which has a serial interface which cuts down on the I/O requirement from the BASIC module. The module used is a TSM 5052, from III/V systems. These are currently available from Greenweld Electronics; almost any of the modules in the series is suitable. The module requires a standard 5V supply, a 3V supply for the LED display and a data and clock input. Figure 11 shows the clock and data interface to this module. The start bit is high, and should be followed by the data bits for the first two displays, digit 1 first, then digit 2. Each display has the data for the A bit sent first, then the B bit etc; the 8th bit is not used. Following the data for the displays are sent 19 clocks where data should be set to 0, at which point the display will show the transferred data.

TR2 and TR3 together with R3 and D2 provide a crudely regulated 2.7V supply for the LED input of the module. Further modules can be driven from the same supply.

The pushbuttons pull down the port B inputs. By default, Port B is set to have 10K pull-ups on each of its pins (to turn these off use the OPTION command) and the push buttons pull down the inputs to the PIC BASIC module.

The relay output is a straightforward transistor switch, D3 is an LED which shows when the output is turned on, the relay is driven by two outputs from the board and is shunted by D1 which quenches the reverse induced voltage when it is turned off.

Program description - use of interrupts

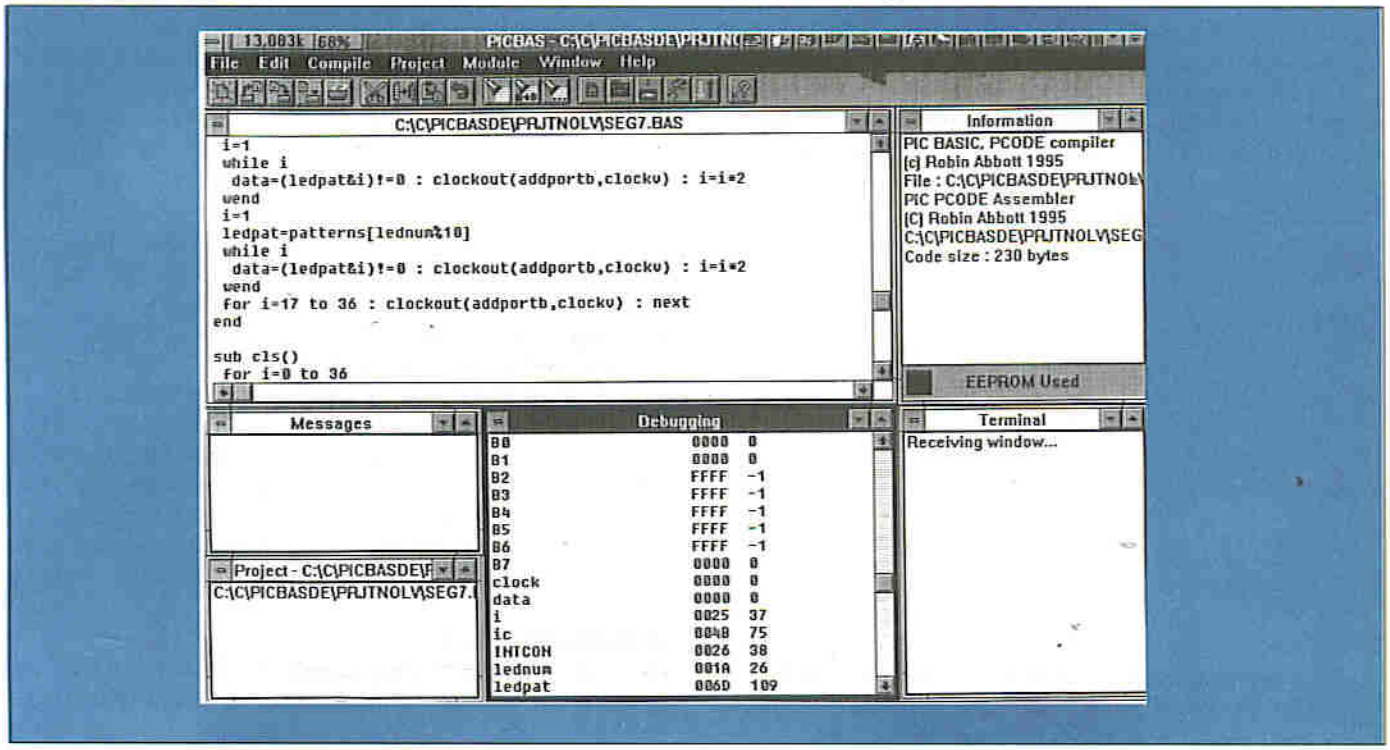
Figure 12 shows the BASIC program used to implement the timer. This compiles to around 400 bytes of EEPROM space.

The initial part of the program sets up the varaibles and the i/o channels of the PIC. Note the use of the alias command to set up meaningful names for the bits of port B. The variable "mins" stores the remaining time.

The label "reset" is the start of the program, the display is cleared, and the outputs on port B are defined. The EEPROM variable "savemins" saves the last value set to the timer and if this is not zero then it is used as the initial timer value. The time is set by the subroutine "settime()" which turns on the relay when it finishes. Consider then the WHILE/WEND loop;

```
while(mins and breset)
  if (flag) then displed() : flag=0
wend
```

This waits until the time is 0. Every time that one minute passes then the interrupt routine sets flag to 1. At this point the displed() subroutine updates the display and the flag is reset. This loop will terminate when the mins variable reaches zero, when the relay will be turned off and the program loops back to the start. Note that if the reset button is pressed, then the loop will also terminate.



The settime() subroutine checks for button presses. The variables btens, bmins, bstart and breset represent the buttons. These are set up using the ALIAS command, and are single bit variables. When a button is pressed, the variable will have the value 0, if it is released then the variable will have the value -1. The WHILE/WEND loop in the subroutine waits for a keypress and then takes action on the key that was pressed. When the bstart variable is zero the loop is completed and the subroutine exits by turning on the relay and by saving the time in the EEPROM variable "savemins".

The 7 segment display must have the patterns for each digit defined and this is done with a constant array called patterns. The bottom bit (bit 0) of each element of the array is segment A, bit 1 is segment B etc. Ten elements of this array are defined, one for each digit from 0 to 9. The display is driven through two subroutines, cls() and displed(). The subroutine cls() clears the display by writing 0 to each digit, while displed() writes the contents of the variable "number" to the display. In this case, only from 00 to 99. The subroutine displed() uses a variable called ledpat which is set to the pattern for the digit being driven. The while/wend loop checks each bit of ledpat and if it is set then the data line is driven high, or low otherwise. There are two while/wend loops, one for each digit, followed by a for/next loop which sends out the 19 clocks at the end of the data for the digits. Note that the "number" variable is set to be the same as the "mins" variable using an alias command at the top of the program.

Interrupts

Interrupts are a facility provided on main microprocessors and microcontrollers which allow a special subroutine to be executed when some external event occurs. Such events can be the change of state of an input pin, or a timer register reaching a preset value. When the event occurs the program stops executing at its current location and calls the interrupt routine. When the routine is completed, the program continues at the point at which it was interrupted.

Those PIC processor implementations which support interrupts allow the BASIC program to run a special subroutine every time that an interrupt occurs. These PICs are the 61, 64, 71, 74,8 4 and all upcoming PIC devices such as the 16C62X series. The special subroutine is called "interrupt()" and it takes no parameters. The interrupt subroutine should be as short as possible because, while it is executing, any other interrupts will be ignored.

There is one special PIC Controller BASIC function which is provided to deal with interrupts; the EI function. The EI function takes one parameter which is either 0 to disable interrupts, or is a combination of values to enable interrupts. In addition to this, there is one variable called INTCON which represents the interrupt control register in the PIC.

To enable interrupts the EI function is called with the value corresponding to the interrupts to be enabled. The following values are allowed for the 16C84 implementation:

- GIE — General interrupt enable, set to enable all interrupts.
- TIMERIE — Set to enable the interrupt when RTCC overflows.
- INITIE — Set to enable an interrupt on port B, bit 0.
- BCHGIE — Set to enable an interrupt when port B, bits 4:7 change.

Thus to enable an interrupt on port B, bit 0 then the function :

```
EI(GIE+INTIE)
```

should be called. Once in the interrupt routine then the INTCON variable will show which interrupt source causes the interrupt routine to be called. This will only be needed if there are several interrupt sources. The following sources will be shown as bits in INTCON:

- TIMERIF — Set if the interrupt was caused by timer overflow.
- INTIF — Set if the interrupt was caused by Port B, bit 0.
- BCHGIF — Set if the interrupt was caused by Port B changing.

If the INTCON variable is used then the bit which caused the interupt should be set to 0 before the interrupt routine finishes. If it is not used then it may be ignored.

During the interrupt routine any further interrupts will be

ignored. When the interrupt routine returns, interrupts are enabled again, hence the interrupt routine must be as short as possible.

The following example shows a simple delay program which waits for 100 internal clock overflows and then returns to the monitor. This takes 6.55S with a 4Mhz clock. Note that in this example we check and clear the intcon variable in the interrupt routine, however this would only be necessary if there were more than one interrupt source enabled.

```
typesub interrupt()

ei(gie|timerie)              ; Enable timer
interrupts
time=0
while(time<100)
wend
ei(0)                                  ; Disable all
interrupts
monitor()                              ; Return to
monitor
sub interrupt()
  if (intcon & timerif) time=time+1
  intcon=intcon & ~timerif                    ;
clear flag
end
```

For the timer program, we count one minute's worth of interrupts. Each timer overflow interrupt occurs as a default every 262144 main PIC clocks. This is every 0.065536 seconds with a 4MHz clock. Thus in one minute there will be 915.52 interrupts, the constant intcnt is set to 916 which will count roughly once per minute with an error rate of 0.05%.

The interrupt subroutine looks like:

```
sub interrupt
  ic=ic-1
  if (not ic) then ic=intcnt : mins=mins-1 :
flag=1
end
```

The ic variable is initially set to intcnt (916), and then counts down to 0. When it reaches 0, it is reset to intcnt, the mins variable is decremented and the flag variable is set to 1 to show that a minute has passed. The flag variable is used in the main program to update the display.

Changing timer operation to hours or seconds

To operate the timer in seconds, the clock should be changed to 4.194304MHz. This will result in exactly 16 interrupts per second. However, serial port timing will then be less accurate. This should not be a major problem. The variable intcnt can then be changed to 16 and the module will operate in seconds.

To operate in hours then the variable intcnt should be set to 54960. However, this is too large for the variable size (recall that a 15 bit variable can only hold numbers up to 16383). In this case, another variable should be used to count up to 60 minutes before the flag is set.

Construction and testing

There is a small PCB for this project. Figure 13 shows the overlay for the PCB and Figure 14 shows the component list. Construction is straightforward. Insert resistors first, then transistors, IC socket, veropins, and finally the display module and push buttons. Construction of the case installation will depend on the application. Any 6V relay with a coil resistance of greater than 50R can be driven by the module. Use 16 way IDC cable with 16 pin DIL headers to connect the PCB to the main BASIC module. IDC headers do not need expensive assembly equipment, they can be fixed to the cable by soldering a 16 pin IC socket to a small piece of scrap vero board. Assemble the header around the cable, plug it into the socket, and press the fixing clip on to the cable using a vice or mole wrench.

The BASIC program for the timer is included in the project file "TIMER.BAS" which is included on the disk. Open this project, compile it, download to the module and set the autoboot flag. Reset the module by removing and then reapplying power and the module should now be tested and should operate as described above.

Taking the timer further

This is a good project to experiment with PIC Controller BASIC; try doing more in the interrupt routine and try to improve the efficiency of the various subroutines used, both to make the program faster and to occupy less space. Examining Pcode files can be helpful here. It is also instructive to implement the timer using only two push buttons, but providing the same functionality. This can be achieved by making the buttons dual purpose according to where the program is operating.

Obtaining PICs

PIC16C84 devices programmed with PIC Controller BASIC are available from the author, Robin Abbott, at 37 Plantation Drive, Christchurch, Dorset, BH23 5SG. Please state the EEPROM type to be used (24LC16 and compatibles (2Kx8), or 24LC65 and compatibles (8Kx8)) as they have different addressing schemes. Please also state the clock frequency to be used (4MHz or 10MHz) as this affects the timing loops for the serial interface and EEPROM interface. 10MHz clocks can only be used with 16C84/10 devices. Send a SAE together with a blank 16C84 and a cheque for £15.00, or send £25.00 if you wish the device to be supplied as well.The author is happy to answer questions, or suggestions for enhancements to the language - particularly for future devices which have more code space. Send comments to the above address, or on Compuserve to 100023,535, or on the Internet to 100023,535@compuserve.com.

Figure 1 - Key features of the development system

- Projects Allow all the parameters of the program under development to saved under a common file system
- Editing The program allows a number of files to be opened and edited simultaneously.
- Key word help Positioning the cursor on a keyword and pressing Ctrl+F1 brings up the help file entry for that keyword
- Making files System holds a list of all the files which make up a project and automatically compile and assemble them all on request.
- Debugging Debugging support shows all variables in use at selected points in the running program
- Terminal Display of information received from module and allows key presses to be sent over the serial port to the program currently running on the module.
- Error tracking Errors can be traced and the file with the error opened with the cursor at the error location; errors can be followed through the files which make up the project.
- Module communication Programs can be downloaded to the module. The module can be verified and programs uploaded for duplication to other modules. Programs can be run on the module and the module can be set to autoboot programs so that they run automatically when power is applied to the module.

Figure 3 - BASIC test program - with Error !

```
serout(12,defserout)      ; Clear the terminal
seroutstring("Enter a digit >",defserout)    ;
Prompt
key=0 ; Receives from serial port
while((key<'1') or (key>'9')) ; Wait until key in
range
 key=serin(defserin,0)
wend
serout('\n')              ; Print a carriage return
for i='1' to key
 serout(i,defserout)      ; print the number
 serout('\n',defserout)   ; Print a carriage return
next
monitor()                 ; return to the monitor
```

Figure 4
Range of size of variables and the numbers which they hold

Number of bits in the variable	Minimum value the variable can hold	Maximum value the variable can hold
1	-1	0
2	-2	1
3	-4	3
4	-8	7
5	-16	15
6	-32	31
7	-64	63
8	-128	127
16	-16384	16383

Figure 5 - Example program using subroutines and functions

```
include "util.inc"
dim temp.16               ;temperature to convert
typefunc ftoc()           ;defines the function
ftoc
serout(12,defserout)      ; clear terminal
seroutstring("0 fahrenheit in celsius is
",defserout) ; message
temp=0
 number=ftoc()            ;convert 0'f to 'c
 prtnum()
serout('\n',defserout)
seroutstring("212 fahrenheit in celsius is
",defserout)              ; message
temp=212 : number=ftoc() : prtnum() :
serout('\n',defserout)
monitor()
;this is the function which converts temp to
centigrade
func ftoc()
 return (temp-32)*5/9
end
```

Figure 6 - Summary of BASIC keywords and built in variables

;	Introduces a comment
A[4],B[8],C[8] etc.	Arrays of single bit varibles representing port bits
A0,A1,A2,A3,B0-B7,C0-C7 etc.	Single bit variables representing port bits
ADDPORTA,ADDPORTB etc.	Address of ports
ALIAS name=variable	Defines substitute names for variables.
ASC(string$)	Return ASCII code of first character of string
BCHGIF	Constant - Port B upper change flag for interrupts
CLOCKOUT(PORT,MASK)	Produce a high going clock pulse on a port pin
CONST	Define a constant
DEBUG()	Update the debugging window on the host PC
DEFSERIN	Default serial input port definition
DEFSEROUT	Default serial output port definition
DIM var[array].size.[E/L]	Define variables and arrays
EI(mask)	Enable interrupts
ELSE	Used with IF
END	Used with functions and subroutines
ENDIF	Used with IF
FALSE	Constant, equal to 0
FOR var=start TO end [STEP value]	Loop function
FUNC name()	Used at start of a function
GOTO label	Transfers program to a label
IF expression THEN	Conditional program control
INCLUDE	Include another source file
INTCON	Interrupt control register
INTIF	Constant - Interrupt (B0) flag for interrupts
MID$(string$)	Shorten a string
MONITOR()	Return to monitor control from the PC
NEXT	Used with FOR
OK()	Debugging only (advanced)
OPTIONS(value)	Set the PIC options register
PEEK(address)	Return contents of PIC memory address
POKE(address,value)	Set the contents of PIC memory
PORT	Array of 8 bit variables representing ports on the PIC
PORTA,PORTB etc.	Variables representing ports on the PIC
REM	Remark statement
RETURN value	Returns a value from a function
RTCC	Real time clock register
SERIN(port,wait)	Read a byte using serial protocol
SEROUT(value,port)	Write byte to port using serial protocol
SEROUTSTRING(string,port)	Write a string to a port using serial protocol
STEP	Used with FOR
SUB name()	Introduces a subroutine
TIMERIF	Constant - Timer flag for interrupts
TRISA(value),TRISB(value), etc.	Define output pins on port
TRUE	Constant, set to 1
TYPEFUNC name()	Defines a function
TYPESUB name()	Defines a subroutine
WEND	Used with WHILE
WHILE expression	Loop control

Fig.12 - Code for Timer.Bas

```
; BASIC code for PIC Controller Module
; Timer project for ETI November issue
;
; Define the subroutines
;
typesub displed()          ; Display variable
"number" on 7 seg
typesub cls()              ; Clear 7 segment display
typesub interrupt()        ; Deal with interrupts
typesub settime()          ; Set the time
dim i.8,ledpat.8,number.8    ; index & LED pattern
const intcnt=916           ; interrupts/minute
dim savemins.E             ; Saves selected time in
EEPROM
dim ic.16,flag.1           ; interrupt counter &
flag
alias mins=number          ; Mins counts the time
alias clock=b0,data=b1,relay=b7       ; Clock,data,
& relay in & out
const clockv=1,datav=2,relayv=80h     ; Values of
bits for i/o
alias btens=b2,bmins=b3    ; Push button bits
alias bstart=b4,breset=b5
const patterns[10]         ; 7 segment patterns
patterns[0]=3fh            ; Pattern for digit 0
patterns[1]=06h            ; Pattern for digit 1
patterns[2]=5bh            ; Pattern for digit 2
patterns[3]=4fh            ; Pattern for digit 3
patterns[4]=66h            ; Pattern for digit 4
patterns[5]=6dh            ; Pattern for digit 5
patterns[6]=7dh            ; Pattern for digit 6
patterns[7]=07h            ; Pattern for digit 7
patterns[8]=7fh            ; Pattern for digit 8
patterns[9]=6fh            ; Pattern for digit 9
;
; Start of program, jump here when time period is
over
;
reset:
ei(0) ; interrupts off
data=0 : clock=0 : relay=0     ; Turn off all
outputs
trisb(~(datav+clockv+relayv)) ; Set outputs on
port B
mins=10                    ; Default time
if ((savemins>0) and (savemins<100)) then
mins=savemins               ; Recall
 last time
settime()                  ; Set the time & start
clock
ei(gie|timerie)            ; Enable timer interrupts
while(mins and breset)     ; Loop until end, or
reset
 if (flag) then displed() : flag=0    ; 1 minute
has passed
wend
relay=0                    ; Time is over, reset
relay
goto reset                 ; and loop back to start
;
; Set the time
;
sub settime()
 cls() : displed()         ; Clear 7 seg
 while(1)                  ; Loop until start
  while(btens+bmins+bstart+breset=-4): wend  ; Any
button pressed?
  if (not bstart) then goto timeset  ; Start
button
  if (not btens) then mins=mins+10    ; Tens button
  if (not bmins) then mins=mins+1     ; Minutes
button
  if mins>99 then mins=1 ; Loop round at 99
  displed()                ; Update display
 wend
timeset:                   ; We set the time
successfully & want to start
 relay=1                   ; Turn on relay
 if (mins!=savemins) then savemins=mins      ; Save
new default time
 ic=intcnt                 ; & reset interrupts
end
;
; Interrupt, Set a flag when 1 minute passes
;
sub interrupt()
 ic=ic-1
 if (not ic) then ic=intcnt : mins=mins-1 : flag=1
end
;
; Display 8 bit variable "number" on the LED
;
; This routine is optimised for speed as the LED
needs 35 clocks

sub displed()
 data=1 : clockout(addportb,clockv)   ; start bit
 ledpat=patterns[number/10]    ; Pattern for tens

 i=1  ; Display tens digit
 while i
  data=(ledpat&i)!=0 : clockout(addportb,clockv) :
i=i*2
 wend
; Display minutes digit
 i=1
 ledpat=patterns[mins%10]
 while i
  data=(ledpat&i)!=0 : clockout(addportb,clockv) :
i=i*2
 wend
; And finally the dummy clocks
 for i=17 to 36 : clockout(addportb,clockv) : next
end
;
; Clear display, clock out a 1 and then 36 0's
;
sub cls()
 for i=0 to 36
  data=not i
  clockout(addportb,clockv)
 next
end
```

PARTS LIST

Resistors

- R1, 3 — 1K
- R2 — 180R

Capacitors

- C1 — 47uF, 10V
- C2 — 100n Ceramic

Semiconductors

- TR1, 3 — BC548
- TR2 — BD135
- D1 — 1N4148
- D2 — 4V3, Zener diode
- Display 1 — 7 segment display module, Greenweld type Z2892 etc

Miscellaneous

- 16 pin DIL IC SKT
- Case
- PCB
- 16 way IDC cable
- 16 pin DIL IDC header x2
- 6V Relay
- veropins x2
- SPST pushbuttons x4

Computerised Digital Radio Control System

The combination of radio transmission and reception technology with computer technology allows us to open up a whole range of new fascinating, and potentially very useful applications and ideas. Dr Pei An looks at one type of application, the computerised radio control system

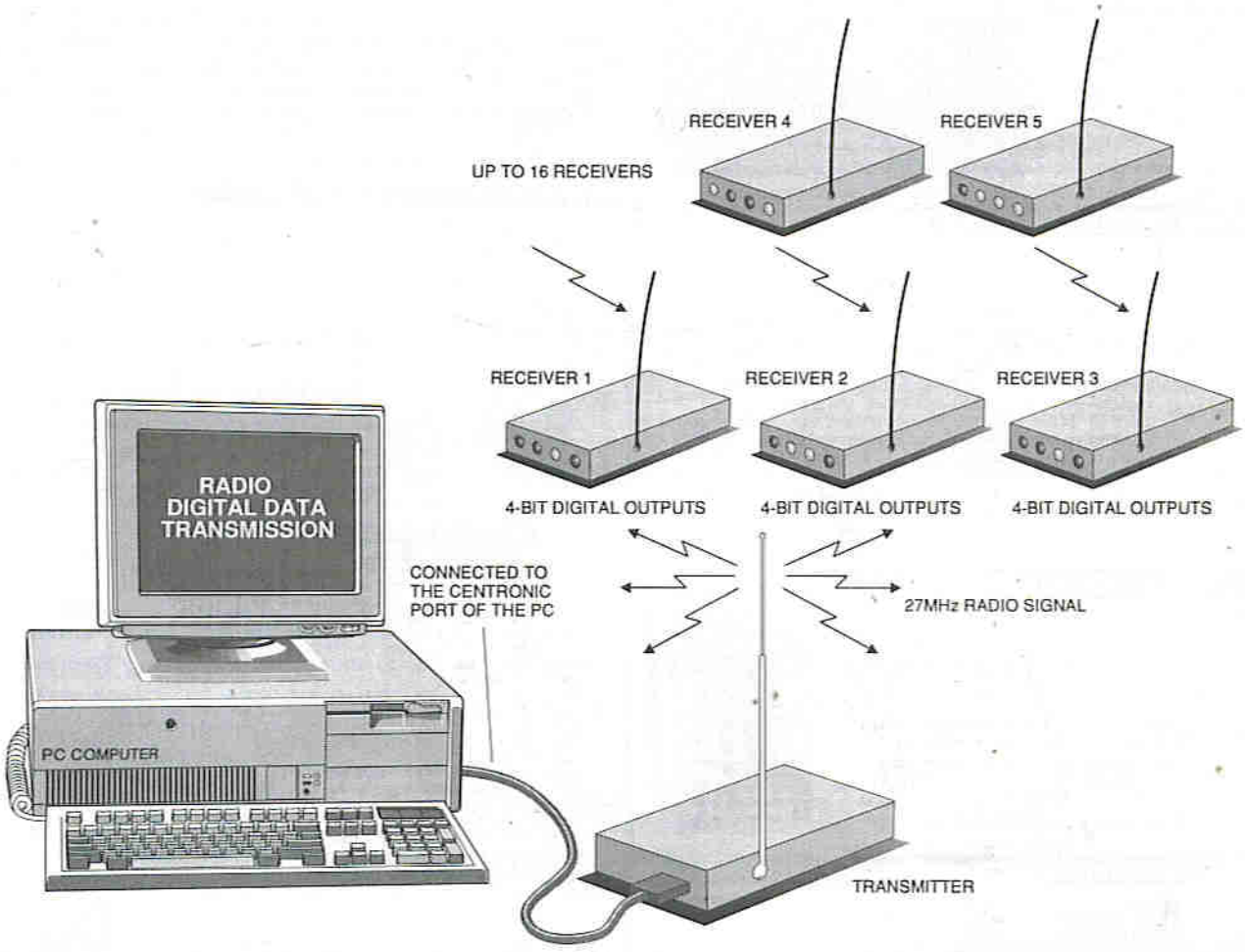


Fig.1. Computerised radio digital data transmission systyem

Wireless local area networks have been on the market for a couple of years and their use demonstrates the potential for transmitting computer data using low power radio communications circuits. However, for the experimenter a far more interesting application is the use of a computer to add intelligence to a conventional radio control system. Just think of how one could use a computer to switch in a pre-programmed complex aerobatic sequence. Or use the computer to simulate the actual aircraft control so that one can get a more realistic feel of flying the model aircraft. Or how about having a radio-controlled mobile robot?

These are just some of the potential ways in which the project described in this article can be used. In fact, the aim of this article is to introduce a project which demonstrates how digital data is transferred between two devices via the radio link. The project consists of a transmitter and a receiver. The transmitter is plugged to the Centronic port of the PC and allows the receiver to input a sequence of 4-bit data commands issued by the computer. The radio link operates at 27MHz legal band with a maximum communication distance of about 20 metres indoors and at least 30 metres outdoors. A schematic showing the complete system is given in Figure 1.

Electronics technology has provided us with the 'state-of-art' radio transmitter/receiver modules. These devices are small and exhibit superb performance. They are still a bit expensive (a pair usually costs about £30). The radio transmitter and receiver system in this application, however, utilises conventional transistor circuits, which are economical to construct. It also offers an opportunity for constructors to understand fundamental aspects of radio communication circuits.

The first step in understanding the design of this project is to look at how digital signals issued by computers are transmitted to the receivers via the radio link. For this, we need to know how radio broadcasting works.

Broadcasting

Radio broadcasting utilises high frequency sinewave (called the carrier signal) to carry audio signals around. There are two common methods of adding information to the carrier signal. The first is called the Amplitude Modulation (AM) and the other one is the Frequency Modulation (FM). Amplitude Modulation involves changing the amplitude of the carrier in sympathy with the modulating signal (see Figure 2). Amplitude Modulation is easy to be recovered. In Frequency Modulation, instead of changing the amplitude of the carrier, the frequency of the carrier is shifted a little higher or lower in sympathy with the modulating signal (see Figure 2). Detection of the FM signal is slightly complicated. In this project, Amplitude Modulation is used, thus, it is explained in detailed.

A schematic showing the principle of AM broadcasting is shown in Figure 3. In radio stations (diagram on the left-hand side), the amplitude of the carrier signal is modulated by audio signals which may be a piece of music or speech. The modulated carrier signal is amplified by a powerful amplifier and sent out to the atmosphere from a tall antenna. The diagram on the right shows your radio receiver (and you). The radio signal is picked up by the antenna of the receiver. It is detected (i.e. the high frequency carrier signal is removed) and the original audio signal is reproduced. The signal is then amplified by an audio amplifier and sent out from the speaker.

Radio frequency is divided into several spectral regions: From 30KHz to 300KHz is the Long Wave (LW); from 300KHz to 3 MHz the Medium Wave (MW); from 3MHz to 30 MHz the Short Wave (SW); from 30MHz to 300MHz the Very High Frequency (VHF) and from 200MHz to 3 GHz is the Ultra High Frequency (UHF). The LW radio frequency is used for long-range and submarine communications. MW is used for broadcast. SW is used for broadcasting, model control and CB radio. VHF is used for stereo radio and UHF used for TV broadcast. 27MHz and 35MHz frequency bands are legally available for radio control in the U.K. The former is suitable for model boats, cars and most types of radio-control models. 35MHz band is used for aircraft models.

The digital radio link

The present digital radio transmission system utilises the 27MHz legal band and applies the amplitude modulation principle. The modulating signal, instead of being an analogue audio signal, is a digital one (either '1' or '0'). The transmitter sends the carrier signal (27MHz) out when the modulating signal is '1' and keeps silent when the signal is '0'. This system consists of a master transmitter controlled by a PC and up to 15 receivers (see Figure 1). The transmitter is able to address

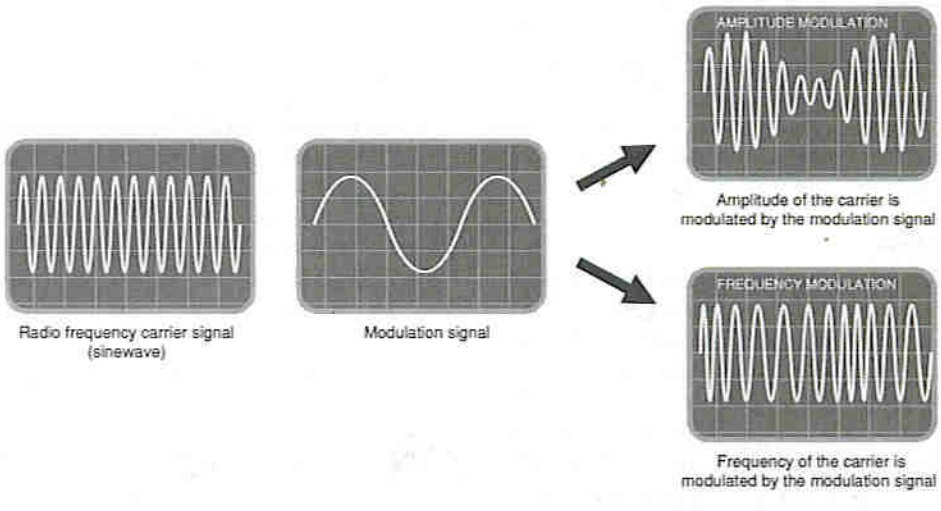


Fig.2. Amplitude Modulation (AM) and Frequency Modulation (FM)

each receiver individually and transmits a 4-digit data to the receiver via the radio link.

Inside the transmitter (see Figure 4a), there is a 9-bit parallel-to-serial data encoder which converts a 9-bit digital data into a specially-encoded serial digital data. The 9-bit data which is supplied by the computer contains a 5-bit address and 4-bit data. The encoded digital signal is used to modulate the 27MHz carrier signal. When the encoded signal is in logic 1, the transmitter will give a burst of 27MHz sinewave. When the encoded signal is 0, the transmitter remains silent. Therefore after modulation, the encoded digital data is transformed into a series of bursts of radio carrier signal.

Inside the receiver (see Figure 4b), the radio receiver circuit detects the radio signals and re-produces the encoded serial data: When a 27MHz radio burst is received, logic 1 will be produced at the output. When no burst is detected, logic 0 is produced. This signal is amplified and fed into the decoder which is in pair with the encoder in the transmitter and converts the serial data back into the parallel data. Each decoder has a 5-bit address and 4-bit data. If the address bits of the received data match the preset address of the decoder, it latches the 4-bit data to its outputs. If the address bits do not match, the receiver will ignore the transmitted data and retain the previous one.

The first one manages the interfacing between the encoder IC and the Centronic port of the computer. The encoder unit converts the 9-bit parallel data into the serial data. This is used as the modulating signal for the radio transmitter unit. The modulated radio signal is sent out from an aerial. The block diagram of the transmitter is given in Figure 4a and the complete circuit diagram is given in Figure 5. The function of each unit is explained in detail as follows.

The computer interfacing unit

This circuit is based on the 74LS164 8-bit serial-to-parallel shift register (see Figure 5). In operation, when the CLOCK input (Pin 8) goes to logic 1 from 0, the logic level currently on the DATA-a and -b (Pins 1 and 2) is transferred to Q1. The next clock pulse also transfers it to Q1, and the logic level previously present on Q1 is shifted to Q2. The third clock pulse transfers the data to Q1, and in the same time shifts the old data on Q1 to Q2, and that on Q2 to Q3. 8 clock pulses will enable an 8-bit data to be fully loaded into the register serially. In the present circuit, DATA-a and -b are both connected to the DB0 of the DATA port of the Centronic port (Pin 2 of the Centronic port). The CLOCK is connected to the DB1 of the DATA port (Pin 3 of the port). A software controlling the data loading will be discussed later in the article.

The first 4 bits of the shift register outputs (Q1 to Q4) supply the address (A0 to A3) to the encoder and the other four bits (Q5 to Q8) supply the data (D0 to D3).

The encoder unit

This circuit is built around a M145026 encoder IC. A detailed introduction has been given in articles entitled 'Smart Mains Controller' in the January and February issues of ETI magazine. In brief, the IC is able to convert a 5-bit address and a 4-bit data into a serial data form. This data is output from the DATA OUT (Pin 15), provided that the -TE input (Transmit Enable, Pin 14) is set to low. In the present circuit, this pin is connected to the DB2 of the DATA port of the Centronic port (Pin 4 of the Centronic port). While loading the data into the 74LS164 shift register, DB2 is set to '1' to inhibit the encoder to output the serial data. After the data has been loaded successfully into the shift register, DB2 becomes '0', and enables the encoder to transmit data. The data is then fed into the Radio Transmitter Unit to modulate the radio-frequency carrier signal.

It is noted that although the encoder has 5 address input lines, in this circuit, only 4 lines (A0 through to A3) are used. A4 is permanently connected to the ground. This simplifies the circuit a bit, however it has the drawback that the maximum number of addresses issued by the computer is reduced from 31 to 15.

The Radio Transmitter Unit

From Figure 5 it can be seen that the unit can be further divided into 5 circuits, namely, the carrier signal oscillator circuit, voltage translator, carrier modulation circuit, power amplification and the aerial output circuit.

The carrier signal oscillator circuit generates carrier signals at a precise and stable frequency. The frequency of the carrier must be precise and stable enough in order to guarantee that the transmitter is operating within the legal band limits. To achieve this a crystal-controlled oscillator circuit is used. This circuit is around Tr1, R1, R2, R3, L1, C1 and TX. It produces a single-frequency sinewave at a frequency determined only by the crystal, regardless of the precise value of other components.

The crystal used is the third overtone type. It means that its normal operating frequency is one third of the marked frequency, but it is designed to work at the marked frequency if an oscillation circuit resonant at this frequency is included in the circuit to suppress oscillation at the fundamental frequency. These crystals are specially designed for radio control applications. They are often supplied in pairs: one used in the transmitter and the other used in the receiver. The resonant frequencies of the two crystals in a pair are slightly different. The one used in the receiver has a frequency 455KHz lower than that used in the transmitter (the reason for that will be given later). The available crystal pairs are listed in the following table.

The amplitude of the carrier signal is modulated by the signal from the encoder. The amplitude modulation circuit is based on Tr2, C2, C3, R4 and a tuned coil L2. The modulating signal is fed into the emitter of Tr2 and the amplitude of the carrier signal is linear to the voltage level at the emitter. The use of a tuned coil (resonant frequency of which is tuned at the oscillation frequency of the oscillator) at the collector is to improves the efficiency. L2 in the circuit is made by the constructors. Please refer to the construction section for details.

The encoded signal from the encoder swings from 0 to 5V (as the encoder uses a 5V DC power supply). A voltage translation circuit is then used to expand the signal span to 0 to 9V (the radio transmitter circuit utilises a 9V DC power supply). This enables the modulated radio frequency signal at the antenna output to have the maximum voltage swing.

The modulated carrier signal from the modulator is fed to a power amplification circuit. A high power transistor (Tr4, BFY51) is used here. The output power is boosted to about 400 mW.

The amplified signal is finally fed into a tuned aerial output circuit. The two LC circuits are both tuned to the carrier frequency to improve efficiency. The two induction coils are made by constructors. Please refer to the construction section for details. The capacitor C11 is used to couple the circuit and

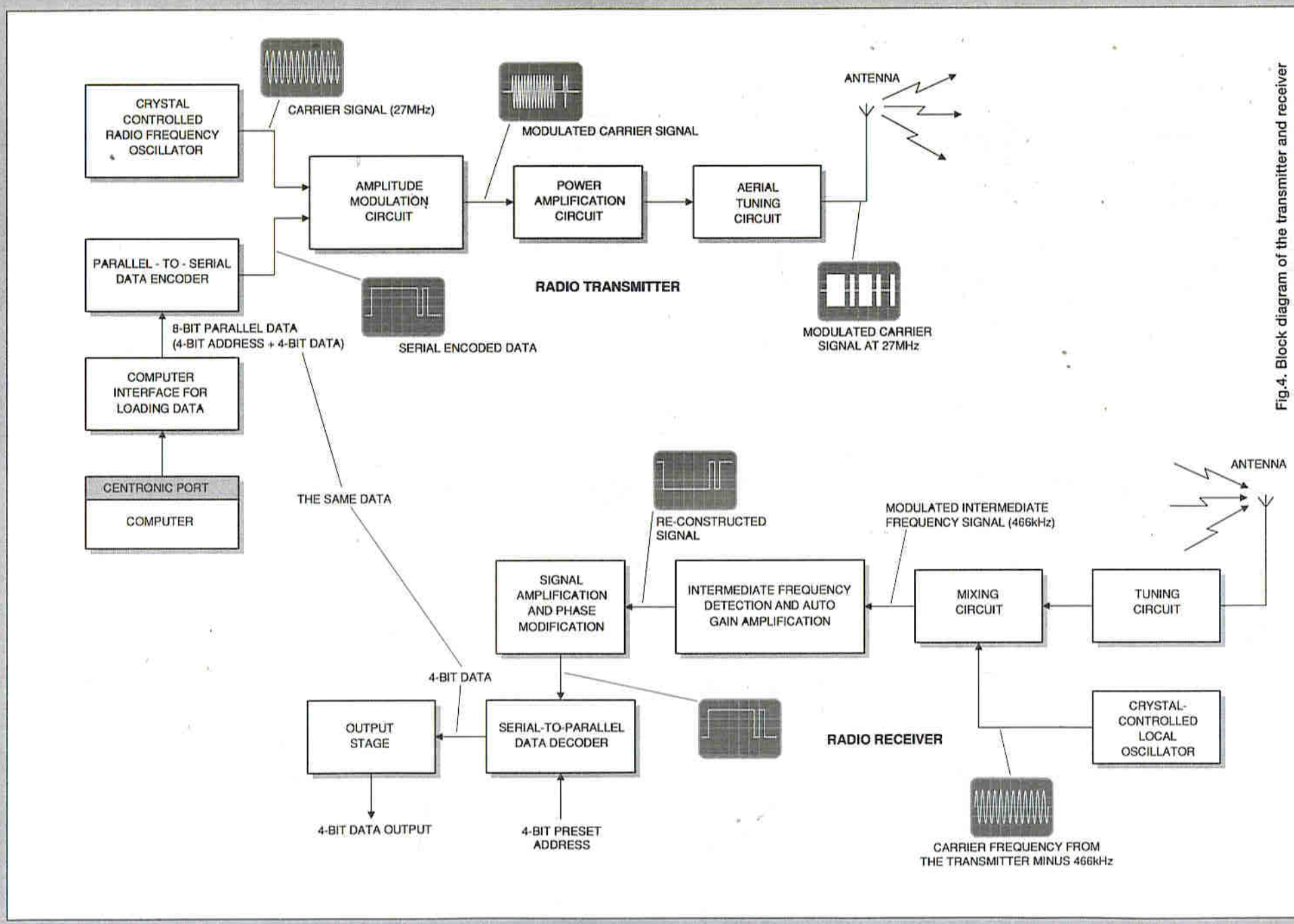


Fig.4. Block diagram of the transmitter and receiver

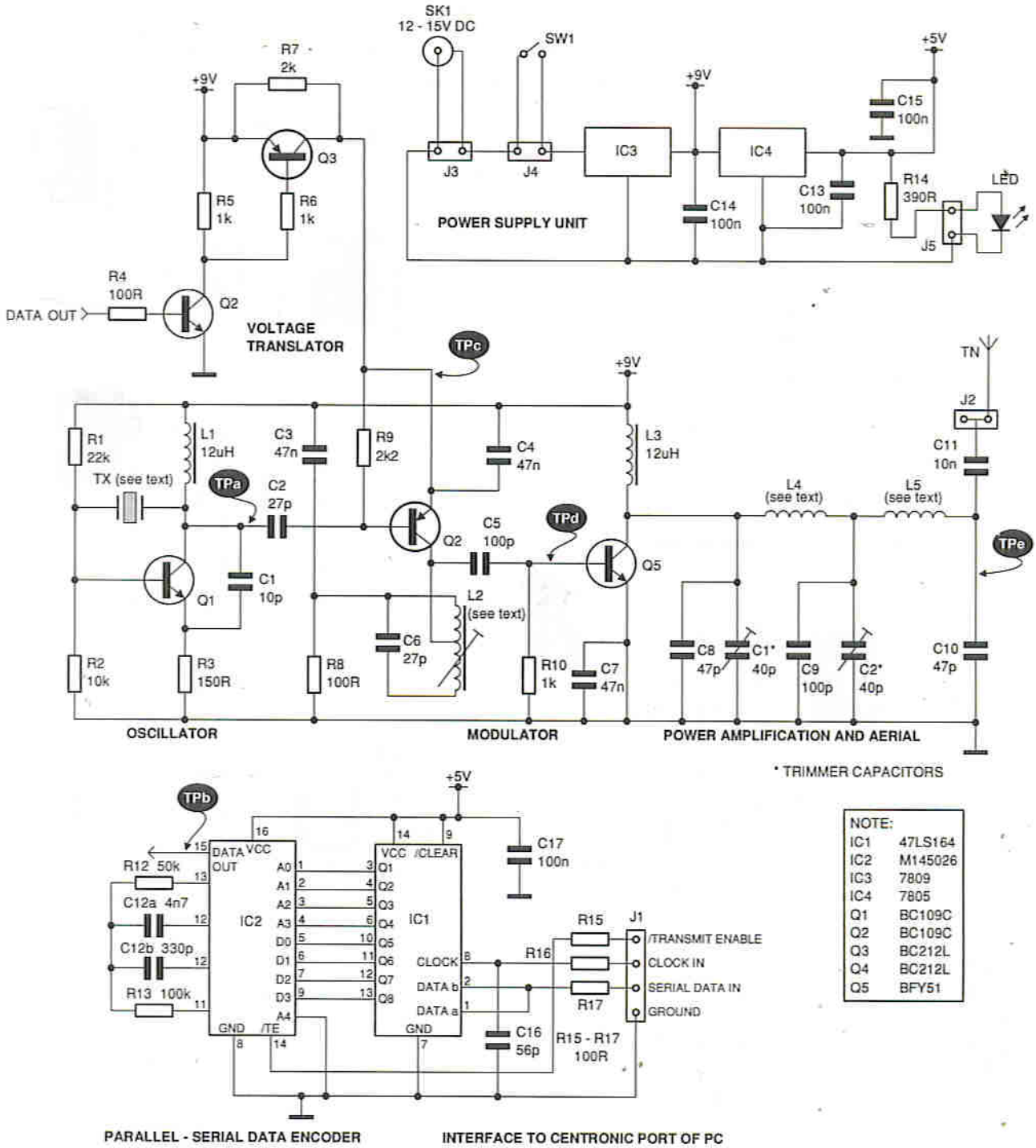


Fig.5. Circuit diagram of the transmitter

the aerial. It does not play any role in the functioning of the circuit; however, if the aerial is accidentally shorted to the transmitter's ground, this would short-circuit the output of the power transistor (Tr4) and damage it. The length of the aerial has to be specially chosen to maximise the efficiency of the radio transmission. The lengths, in theory, should be 1, 1/2, 1/4, 1/8 or 1/16 of the wavelength of the carrier. If the carrier has a frequency of 27MHz, according to the following equation,

lambda = c over f

in which lambda is the wavelength (in meter), C the propagation speed of electro-magnetic waves which is 300,000,000 meters /second (i.e. the speed of light) and f the frequency of the carrier in Hz, the wavelength is 11.1 meters. Therefore the practical lengths of a aerial should be 1.4 meters or 0.7 meters.

Power supply unit

The power supply unit requires a 12 -15V DC power supply. It has a 7809 +9V DC and 7805 +5V DC voltage stabilisers. The +9V DC powers the radio transmitter circuit and the +5V DC powers the computer interfacing and the encoder circuit.

The radio receiver unit receives the transmitted radio signal, detects it and amplifies it. The signal is then the re-construction of the encoded signal from the encoder in the transmitter. It is finally fed into the decoder unit where it is decoded into parallel data. The block diagram of the receiver is given in Figure 4b and the complete circuit diagram is given in Figure 6.

Third Overtone Radio control crystals

Channel	Transmitter
frequency (MHz)	Receiver
frequency (MHz)	
Brown	26.995 26.540
Red	27.045 26.590
Orange	27.095 26.640
Yellow	27.145 26.690
Green	27.195 26.740
Blue	27.245 26.790

Radio receiver unit

The radio receiver unit is of a superheterodyne (superhet) design. The unit is composed of 5 basic circuits, namely, the LC tuning circuit, the local oscillator, the mixer circuit, the intermediate frequency (IF) detection and automatic gain control (AGC) circuit and finally the signal amplification circuit.

It is a fact that a LC circuit has low impedance at frequencies other than the resonant frequency, and this 'shorts out' unwanted radio frequencies. At the resonant frequency the LC circuit has the highest impedance, so the radio signal at the selected frequency appears across the circuits a small radio frequency alternating voltage. The resonant frequency of a LC circuit is calculated using the following equation

f = 1 over {2 pi sqrt {L C}}

in which L is the inductance of the coil in H and C the capacity in Farad.

In the present circuit, the inductor is an adjustable 15uH coil. The ferric slag inside the coil enables the inductance to be adjusted slightly. The capacitor is a 2.2 pF ceramic disk. This works out that the resonant frequency is about 27.9 MHz. By adjusting the inductance of the coil, a resonant frequency can be achieved. The aerial is directly coupled to the tuned LC circuit. The electro-magnetic field generated by the transmitter induces a tiny oscillating current in the aerial. This signal will be selected by the LC resonant circuit.

The local oscillator is based on Tr2 and has a common collector Crystal oscillator configuration. The crystal is the one in pair with that in the transmitter (refer to Table 1). The frequency of the crystal is exactly 455KHz lower than that of the carrier signal from the transmitter.

The mixing unit is used to convert the incoming radio-frequency signal into an intermediate frequency, 455KHz. A junction gate field effect transistor is used in this stage. The advantage of using a FET transistor rather than an ordinary bipolar transistor, is that the former has an extremely high input impedance and so the tuned circuit can be coupled directly to its input. The output from the oscillator is fed to the gate of Tr1 via C3. As Tr1 does not act as a perfectly linear amplifier, the oscillator signal causes variations in the gain of Tr1, and provides the required mixing actions. The mixing will produce an output that is the difference between the frequencies of the carrier and that of the local oscillator. 455KHz is resulted in our case. This signal is output from the terminals 1 and 3 of the IF (intermediate frequency) transformer, which is tuned at 455KHz. The output of the mixer is therefore a 'carrier' at 455KHz, and this carrier is still modulated with the original audio signal.

The IF signal is fed into the IF detection and amplification circuit which is based on a signal chip AM radio IC, ZN414. This chip is well known to most amateur electronics enthusiasts. It integrates many functions, such as RF amplification, RF detection and AGC (automatic gain control) circuit, on a tiny transistor-like IC and it requires a 1.5V DC power supply with an operation current of about 0.4 mA. Together with a few components, it is able to form a complete AM radio. It operates at frequencies up to 3MHz and therefore it works happily in the circuit. The recovered signal is then fed into the signal amplifier.

The signal amplification circuit amplifies the recovered encoded signal to the voltage level required by the decoder IC. The amplifier is based on the CA3104 operation amplifier which is configured as a non-inverting amplifier. The amplification is set to about 46.

The Decoder unit

The decoder unit is based on the M145027 decoder IC which is in pairs with the M145026 encoder. A detailed introduction to the IC has been given in an article entitled 'Smart Mains Controller' in the January and February issues of ETI magazine. In brief, the IC will decode the received signal. The decoder receives the serial data generated by the encoder, checks it for errors and outputs the received data if it is valid. The first five bits are assumed to be address bits and must be encoded to match the address inputs. If the address bits match, the next four data bits are stored in an internal register and compared to the last valid data stored. If the data matches, the Valid Transmission output will go high on the second rising edge of the 9th bit of the received word. If not, the VT output remains low.

The Power Unit

The power unit requires a 8-15V DC power supply. It incorporates a 7805 +5V voltage regulator which supplies the +5V DC to the circuit.

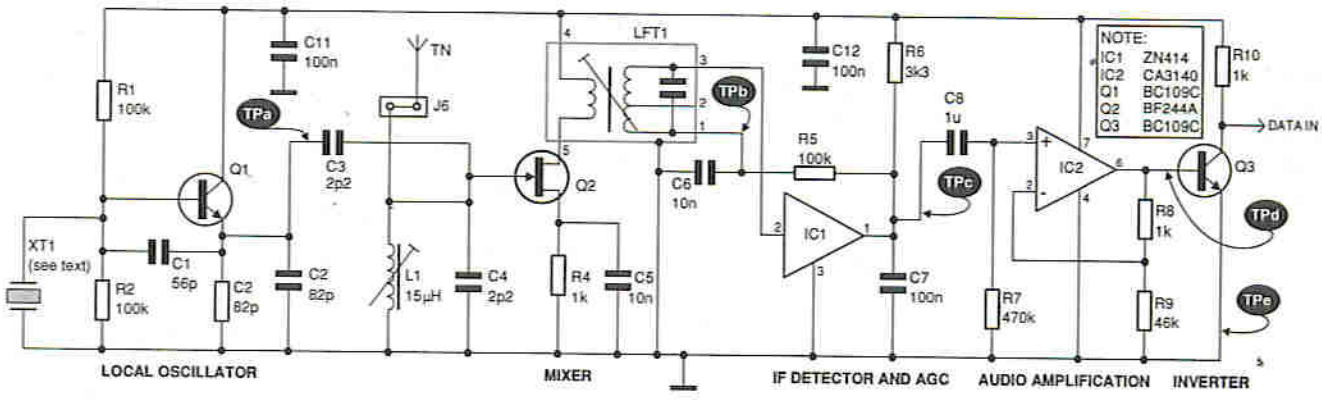


Fig.6a. Circuit diagram of the receiver

CONSTRUCTION

Transmitter

The transmitter is constructed on a single-sided PCB board. The PCB artwork is shown in the PCB foils section at the rear of this issue of ETI, and the component layout is shown in Figure 9.

Before soldering the components on the board, check the PCB board carefully for unwanted links or cuts on the copper tracks. Small components should be mounted on the board before the larger components. When soldering transistors on the board, care should be taken identifying the pins. Sockets are used for all ICs and the crystal oscillator.

There are three coils to be made by the constructors. L4 and L5 are made from 18swg (about 1.22 mm in diameter) enamelled copper wire. They can be wound round a pencil about 8 mm in diameter. The turns of the wire touch each other and 12 turns are needed. Both coils should be wound in the same direction. When mounting the coil on the PCB board, they should stand about 3 mm above the board.

To make L1, first the coil former is fixed onto the board, then one end of a 24 swg wire (approx. 0.559 mm in diameter) is stripped and soldered in the hole marked 's'. Wind 3 1/4

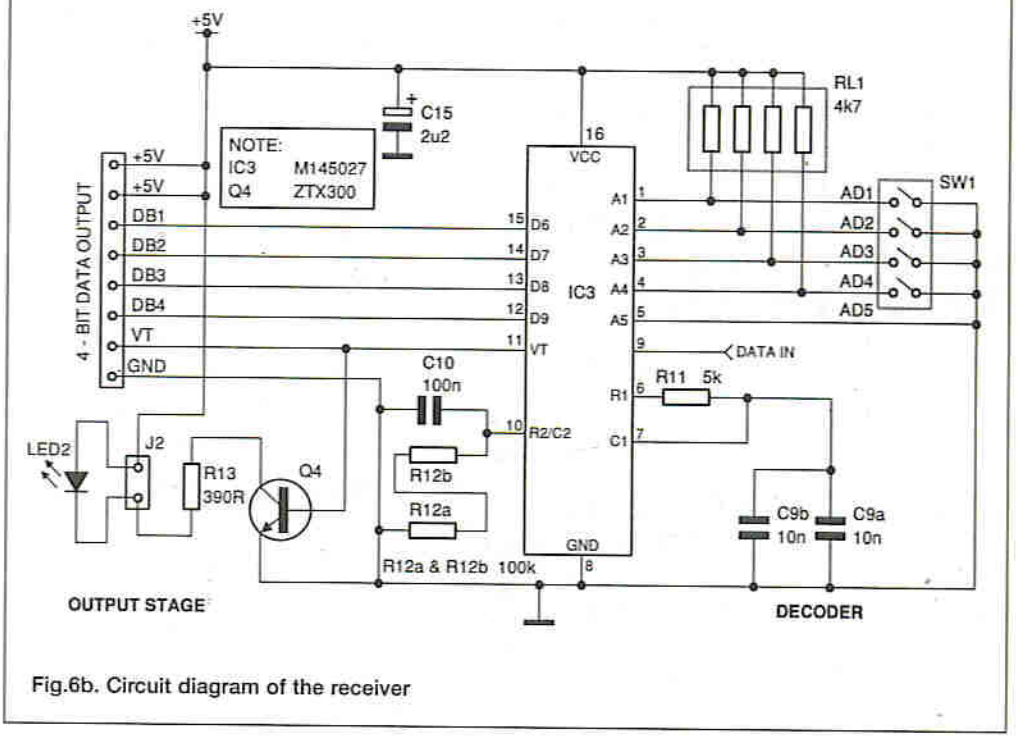


Fig.6b. Circuit diagram of the receiver

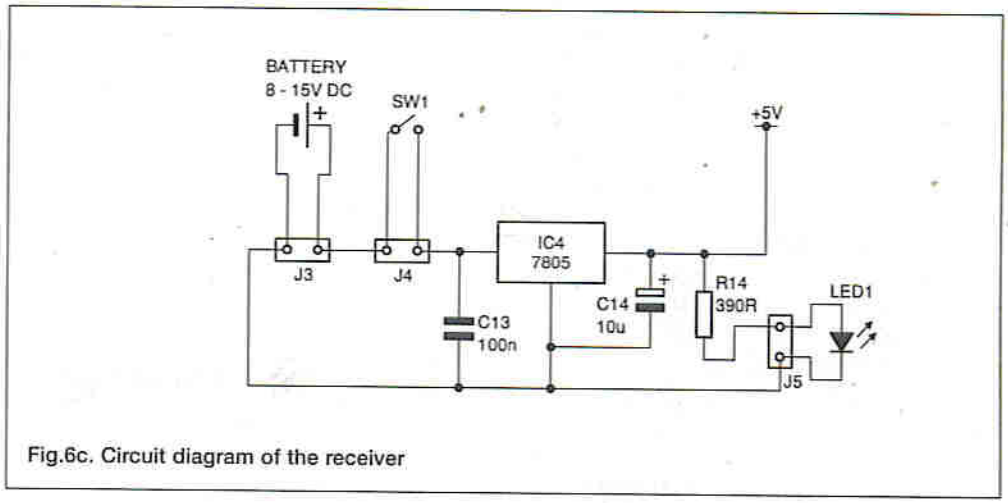


Fig.6c. Circuit diagram of the receiver

turns anticlockwise up the former, and thread the wire down through hole 'm1' and out again through 'm2'. Wind a further 6 1/2 turns anticlockwise up the former. Then the wire is threaded through hole 'f' and soldered in position. Finally, scrape away the insulation of the wire between m1 and m2 underside the PCB board and solder it on the board.

Receiver

The receiver is constructed on a singled-sided PCB board. The PCB artwork is shown in the foils section at the rear of this issue of ETI, and the component layout is shown in Figure 10.

Please follow the precautions and construction procedures as described above. The PCB board is mounted inside a box (see the photograph).

TESTING AND ALIGNMENT

Testing of the transmitter

Testing of the transmitter can be carried out in two stages, the preliminary testing and final testing.

Preliminary testing involves testing all the functions of the individual circuit. Testing of such transistor based circuits is relative simple. What we need to do is to observe the waveforms of the signal at various test points. Starting with the oscillator circuit, you should be able to see a sinewave waveform at test point -a. The frequency of the wave should be around 27.094MHz (see Figure 11a). At test port -b . You will see the waveform of the encoded serial data. The waveform shown in Figure 11b is for the case where all the 9 inputs are set to logic 0. If the transmitter is connected to the computer, by running the sample software supplied, you should be able to see the waveform for other encoded data. The amplitude of the signal is about 5V. Any problems related to the Centronic interface and the encoder circuit should be found at this stage. The waveform at test point-c (the output of the voltage translator circuit) should be similar to that at test point-b, except that the amplitude increases from 5V to about 8.5V (Figure 11c). The waveform at test point-d shows the modulated carrier signal and is shown in Figure 11d. If this waveform is not observed, the position of the core in L3 needs to be adjusted. A special trimmer tool should be used to do so, as an ordinary screwdriver will induce an extra inductance and make the adjustment impossible. Adjust the core until you get the maximum amplitude of the waveform. Test point e should have a similar waveform as that at test point d, however, with bigger amplitude. If all these have been achieved, the final stage of testing can proceed.

The final stage of testing ensures that the transmitter works at its maximum efficiency. To do so, the following RF detector can be used (Figure 12). The radio signal induces a small current in the antenna wire. The signal is rectified by a germanium diode OA91 and then measured by a 500 uA ammeter. In testing, connect the power to the transmitter and bring the RF meter close up to the fully extended aerial. Adjust the core of L3 with a plastic trimmer. Adjust the core for maximum deflection of the meter. Now adjust the output capacitors, C1* and C2*, to increase the deflection to the maximum. The above adjustment should be carried out while moving the RF meter further away from the aerial, until no further improvement is obtained. This ensures that all the tuned circuits are set as accurately as possible to 27.095 MHz. An oscilloscope can also be used as a RF meter. The test lead can form an aerial and the waveform of the radio signal received can be seen on the CRT of the oscilloscope. Follow the adjusting procedures as mentioned above until the amplitude of the received signal on the CRT reaches the maximum.

Testing and alignment of the receiver

When testing the receiver, the transmitter should be switched on. The testing of the receiver is carried out in stages.

Firstly, Make sure the telescope antenna is fully extended and the output from the local oscillator (test point a) is tested. The waveform is shown in Figure 13a - simply it is a 26.790MHz sinewave. At test point b, a waveform of a 466KHz sinewave (intermediate frequency signal) with its amplitude still modulated should be observed (see Figure 13b). If this waveform is not observed, adjustment of the cores in L1 and in the IFT is needed. Again, a special trimming tool should be used. The waveform from the IF detection and AGC circuit should be observed at test point c. This should be the inverted encoded serial data from the transmitter (a dual-trace oscilloscope is very useful in checking this). Now adjust the cores in L1 and IFT for the maximum amplitude of the signal. The amplified signal can be observed at test point d and the signal, after being inverted, is shown at test point e. This signal should have an amplitude of about 5V and is the 100% reproduction of the encoded signal from the transmitter.

Programming

The interfacing between the transmitter and the computer is fairly simple. DB0, DB1 and DB2 of the DATA port of the Centronic port control the whole operation of the transmitter. However, to program the transmitter, readers should know a little bit about the works of the Centronic port.

When loading the address and data into the shift register, first DB2 (which is connected to the -TE input of the encoder) is set to high to stop the IC to output serial data. Then the 1st MSB of the parallel data is put on DB0 while DB1 is low. A low-to-high transition on DB1 (which is connected to the CLOCK input of the shift register IC) will enable the data to be shifted to the 1st register output. The 2nd MSB, 3rd MSB. are then put to DB0 in sequence. The clocks will shift the 8-bit serial data into the register outputs.

After the data is successfully loaded to the register. DB1 is brought low which enables the encoder IC to send out the encoded serial data.

A sample program written in Turbo Pascal 6 is listed in this article. The program will ask for an address first (0-15), then the first time delay (T1) and the second time delay (T2). After this, the program will send '1' for all the data lines. It will delay a time period T1 and then send '0' for all the data lines. After another time delay T2, the transmitter will send '1' again. This will continue until RETURN is pressed.

This is just a simple example, the rest is up to you and your imagination!

Caution

This 27MHz transmitter and receiver system has not received official certification, and thus may affect, or be affected by, other radio systems working at the same frequency. However, because of its very limited range there is unlikely to be any practical problems facing its use. But, readers are advised to seek advice before making any attempt to increase the range and transmitting power of this project

This frequency is allotted to radio control systems with limited transmitting power and range. Nearly all the standard

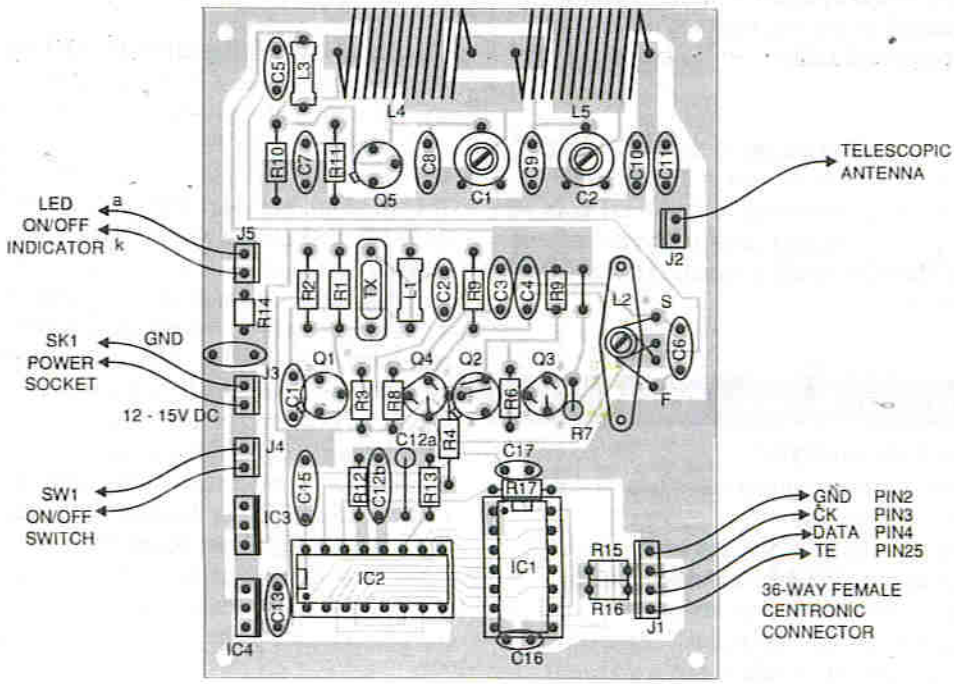


Fig.9. Component layout of the transmitter

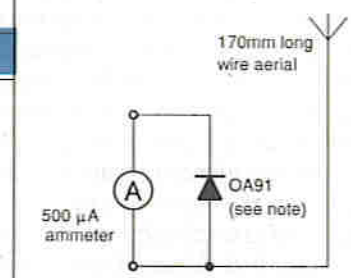


Fig.11. A radio frequency detector for testing the 27MHz radio transmitter

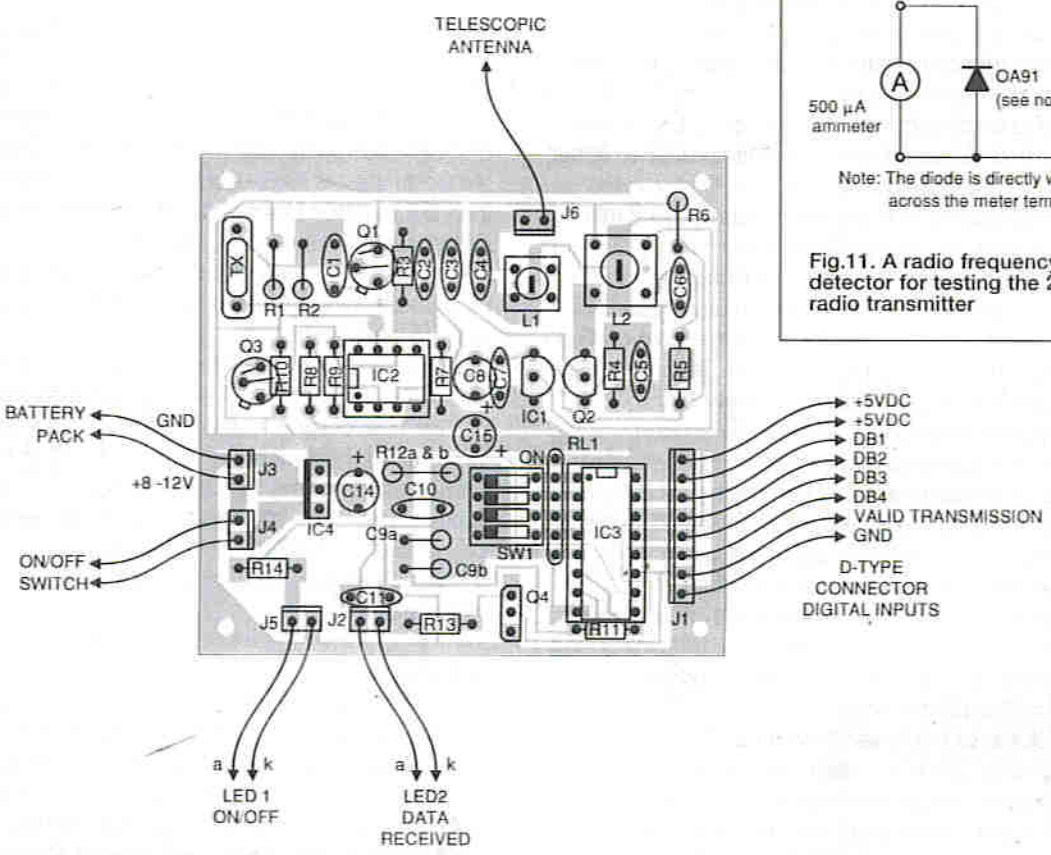
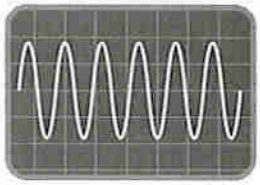
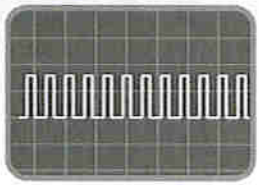


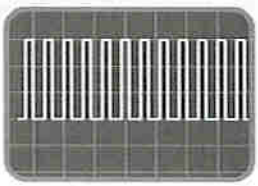
Fig.10. Component layout of the receiver



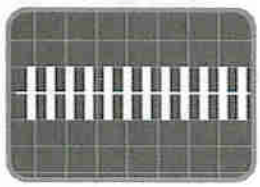
(a) Waveform at test point a, showing the 27MHz sinewave carrier



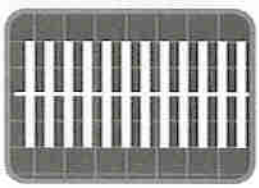
(b) Waveform at test point b. This is the encoded serial data from the encoder IC. The 9 parallel data inputs are set low. The amplitude of the serial data is 5V.



(c) Waveform at test point c. This is the amplified serial data. The amplitude of the data is 8.5V.

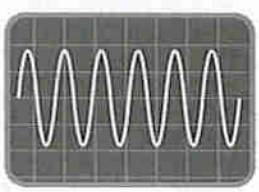


(d) Waveform at test point d. The amplitude of the modulated carrier is about 8V.

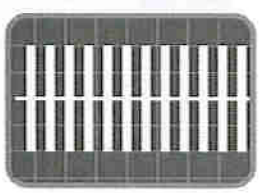


(e) Waveform at test point e. This is the waveform at the antenna. The amplitude of the modulated carrier is about 8V.

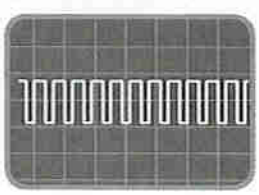
Fig.12. Waveforms at various test points of the transmitter



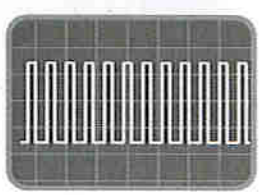
(a) Waveform at test point a. This is the sinewave of the local oscillator. The frequency is 27MHz - 455kHz.



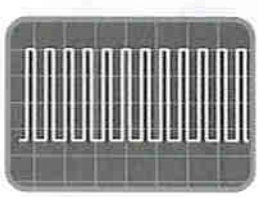
(b) Waveform at test point b. This is the signal received by the antenna. Frequency of the carrier is the IF frequency, 455kHz.



(c) Waveform at test point c. High frequency carrier is removed.



(d) Waveform at test point d. Amplified signal.



(e) Waveform at test point e. The signal is inverted then fed into the decoder.

Fig.13. Waveforms at various test points of the receiver

commercial systems use digital pulse code modulation to transmit commands over a communications channel, most conventional systems have four channels. the only difference between this project and a conventional radio control system is that the commands are generated by a computer rather than by an individual moving a couple of joysticks.

If you do experience problems, or have any doubts about the use of this project then contact your local radio modeller's society, or the radio communication experts at the Radio Communications Agency which is part of the Department of Trade and Industry - they can supply you with more detailed information on this sometimes rather complex subject.

Program listing 1

```
program radio_transmission_system;
{Pascal demonstration program for
driving the radio data communication
system
designed by Dr. Pei An, 17/3/95}
{74LS164 latches the data sent
serially by the computer's Printer
port.
  DB0, DB1, DB2 and DB3 are loaded
with address A0, A1, A2 and A3
  DB4, DB5, DB6 and DB7 are loaded
with data     D0, D1, D2 and D3}
uses
  dos,crt;
var
  address,i,j,swaddress:integer;
  weight:array[1..12] of integer;
  delaytime,lighttime:real;
Procedure bit_weight;
{Find the weight of the binary bits}
begin
  weight[1]:=1;
  for i:=2 to 12 do
weight[i]:=weight[i1]*2;
end;
Procedure
send_address(address:integer);
{Send the address to the 74LS164
shift register}
{When sending the address, the
Transmit Enable (T)must be high to
stop transmit}
{During loading, (1) DB0 is loaded
with the data sw[i],
(2) DB1 (CLOCK) is made from low to
high then low
(3) DB2 (transmit enable) is kept
        high all thetime}
var
  sw:array[1..12] of byte;
begin
  for i:=8 downto 1 do
        begin
              sw[i]:=0;
              if
address<=weight[i] then begin

address:=addressweight[i];
                        sw[i]:=1;
              end;
        end;
  {loading address and data into the
74LS164 registers}
  for i:=8 downto 1 do
        begin
              port[888]:=sw[i]+4;
{DB0=sw[i], DB1=0, DB2=TE=1}
              delay(1);{a delay}
```

PARTS LIST

Transmitter
Resistors (All resistors are 0.25W, 1% metal film resistors)

R1	22K
R2	10K
R3	150R
R4,R5,R6	1K
R72K	
R8,R10	100R
R9	2K2
R11	2R7
R12	50K
R13	100K
R14	390R

Capacitors

pt:C1	10pF ceramic disc
C2	27pF ceramic disc
C3,C4,C7	0.022uF ceramic disc
C5,C9	100pF ceramic disc
C6	27pF ceramic disc
C8,C10	47pF ceramic disc
C11	10nF ceramic disc
C12a	4n7/16V 1% polysty
C12b	330pF/16V 1% polysty
C13,C14	100nF ceramic disc
C1*,C2*	65pF trimmer capacitors
TX	27.245MHz radio control crystal with socket

Inductors

L1,L3	12uH RF choke
L2	Coil former diameter 7 mm with Iron dust core and coil made from 24 swg copper wire
L4,L5	Air coil made from 18 swg copper wire

Semiconductors

Tr1,Tr2	BC109C, NPN bipolar transistor
Tr3,Tr4	BC212L PNP bipolar transistor
Tr5	BFY51, NPN power transistor
IC1	74LS164 octal serial loading latch register
IC2	M145026 encoder
IC3	7809 +9V DC regulator
IC4	7805 +5V DC regulator

Others

J1	4-way PCB plug
J2,J3,J4,J5	2-way PCB plugs
PCB	

Optional

SW1	SDSP switch
SK1	Power socket
TN	1.2 meter telescope aerial
LED	LED

```
      port[888]:=sw[i]+2+4;  {DB0=sw[i],
      {DB1=1(loading into register), DB2=TB=1}
      delay(1); {a delay for loading the bit}
      port[888]:=sw[i]+4;{DB0=sw[i],DB1=0,
      {DB2=TB=1}
   end;
end;
```

PARTS LIST

Receiver
Resistors

R1,R2,R5,	
R9,R10	100K
R3,R4	1K
R6	6.8K
R7	680K
R8	1.2K
R11	50K
R12	200K
R13,R14	390R
RL	SIL resisitor array, 5-way, 4K7

Capacitors

C1	56pF ceramic disc
C2,C5,C6	10nF ceramic disc
C3,C4	2.2pF ceramic disc
C7,C8	220nF ceramic disc
C9a,C9b	10nF/16V 1% polystr
C10	100nF/16V 1% polystr
C11,C12	100nF ceramic disc
TX	26.790 MHz radio receiver crystal (in pair with the one in the transmitter) with socket

Semiconductors

Tr1,Tr3	BC109C NPN bipolar transistor
Tr2	BF244A FET transistor
Tr4	ZTX300 NPN bipolar transistor
IC1	ZN414 AM radio
IC2	CA3140 op-amp
IC3	M145027 decoder IC
IC4	7805 +5V regulator

Inductors

L1 119ANA5873HM	15uH adjustable coil,
L2 YRCS11099	IFT transformer, TOKO

Others

J1	8-way PCB connection plug
J2,J3,J4,J5	
J6	2-way PCB connection plug
PCB	

Optional

LED1, LED2	LEDs
SK1	Power socket
SW1	SDSP switch
TN	0.8 meter telescope aerial

```
Procedure transmit(flag:boolean);
{Start or quit the encoded data transmitting
depending on FLAG}
begin
 if flag then port[888]:=0 else port[888]:=4;
end;
Procedure intialization;
begin
 clrscr;
 writeln('      Radio Digital Data Communication
       System');
writeln;
writeln('     This program shows that 15 digital
      data receivers are controlled by');
writeln('         a transmitter  which is
      controlled by the Centronic of the PC');
writeln;
writeln;
write('Input the address of the receiver (1
```

```
      through to 15):     '); readln(swaddress);
 write('Input the light OFF period (in second,
      minimum: 0.1 s): '); readln(delaytime);
 write('Input the light ON  period (in second,
      minimum: 0.1 s): '); readln(lighttime);
 if delaytime[9999]0.1 then delaytime:=0.1;
 if lighttime[9999]0.1 then delaytime:=0.1;
end;
******Main Program********************}
begin
 intialization;
 bit_weight;
repeat
 transmit(false); {stop transmission}
 send_address(swaddress+16+32+64+128);       {loading address and data
(light on) to the shift register}
 transmit(true); {start transmission}
 delay(30); {transmission lasts 20 ms}
 transmit(false); {stop transmission}
 gotoxy(35,23); writeln('Light On ');
 delay(round(lighttime*100030)); {delay a specified
      time period1}
 send_address(swaddress+0); {loading address and
      data (light off) to the shift register}
 transmit(true); {start transmission}
 delay(30); {transmission lasts 20 ms}
 transmit(false); {stop transmission}
 gotoxy(35,23); writeln('Light off');
 delay(round(delaytime*100030)); {delay a specified
      time period2}
until keypressed;
 readln;
 end.
Procedure transmit(flag:boolean);
   {Start or quit the encoded data transmitting
   depending on FLAG}
begin
 if flag then port[888]:=0 else port[888]:=4;
end;
Procedure intialization;
begin
 clrscr;
 writeln('   Radio Digital Data Communication
   System');
 writeln;
 writeln('     This program shows that 15 digital
   data receivers are controlled by');
 writeln('   *     a transmitter  which is
   controlled by the Centronic of the PC');
 writeln;
 writeln;
 write('Input the address of the receiver (1 through to 15):     ');
readln(swaddress);
 write('Input the light OFF period (in second,
      minimum: 0.1 s): '); readln(delaytime);
 write('Input the light ON  period (in second, minimum: 0.1 s): ');
readln(lighttime);
 if delaytime[9999]0.1 then delaytime:=0.1;
 if lighttime[9999]0.1 then delaytime:=0.1;
end;
{*********Main Program************}
begin
 intialization;
 bit_weight;
repeat
 transmit(false); {stop transmission}
 send_address(swaddress+16+32+64+128); {loading
      address and data (light on) to the shift
      register}
 transmit(true);       {start transmission}
 delay(30); {transmission lasts 20 ms}
 transmit(false); {stop transmission}
 gotoxy(35,23); writeln('Light On ');
 delay(round(lighttime*1000-30)); {delay a
      specified time period-1}
 send_address(swaddress+0); {loading address and
      data (light off) to the shift register}
 transmit(true); {start transmission}
 delay(30); {transmission lasts 20 ms}
 transmit(false); {stop transmission}
 gotoxy(35,23); writeln('Light off');
 delay(round(delaytime*1000-30)); {delay a
      specified time period-2}
until keypressed;
 readln;
end.
```

Single stepping

the 8088 in real time

Further ideas on using the 8088 Interrupt-based SBC by Richard Grodzik

A logic analyzer is a very powerful tool which enables the examination of the data bus and/or address bus every machine cycle. However, it cannot "see" inside the microprocessor to ascertain what information is being manipulated in the various registers. To overcome this problem, software can be used to 'single-step' the user software and a trace of the register contents can be produced for every program instruction.

This article details the software requirements for implementing the software to achieve a single-stepping trace facility.

Once a program is written and downloaded to the emulator for execution, some means must be made available to verify that the program is running correctly and that the program executes as the programmer intended. For a program that outputs data on a port, the simplest way to debug the program is to insert long delay loops and provide some visual indication i.e. LEDs on the port pins via suitable buffers. If the program runs as expected, the delay loops can then be edited out and the program re-assembled.

As the complexity of a program increases,another means must be available which will slow down the execution of a program so that individual registers can be inspected and the results of each line of program instruction execution can be verified. Single-stepping is a software orientated debugging facility which controls the execution of a program so that the programmer can trace every single instruction execution of the program in real time.

The trap flag

The 16-bit flag register of the 8088 reports various status conditions after the execution of every instruction that determine the progress of a program. Bit 8 of this flag register (the TRAP flag), when set, allows the single-stepping of a program whereby one instruction is executed at a time allowing the inspection of the registers and therefore aiding the user in the debugging of a program. When single-step via the trap flag is implemented, the processor is directed via a type 1 interrupt to a subroutine after the execution of each instruction.

To effect single-stepping, the TRAP flag is set. Anytime this flag is set, the 8088 automatically generates a type 1 interrupt after the execution of an instruction. However,the interrupt is not generated until after the next instruction is executed. Once this is complete, a type 1 interrupt is generated. The flags, CS(code segment) and IP (instruction pointer) register values are pushed onto the stack and then the trap flag is cleared. The interrupt service routine (ISR) that the type 1 interrupt generates ends with a IRET instruction which pops the IP, CS and flag values back into their respective registers returning the TRAP flag to the set condition. This sequence is repeated i.e. a type 1 interrupt is issued after every instruction execution.

If a simple delay routine is included in the interrupt routine, the time interval between execution of each line of program determines the time between each instruction and can be slowed down to allow the 8088's register contents to be sent to the PC via the serial port. A terminal emulation program then writes the register contents to the screen at a rate dependent on the delay between instructions. Besides providing a line by line visual indication of processor activity, the screen display can then be simultaneously printed out to hard copy. (see TRACE).

Since the contents of the flag register cannot be directly modified by an instruction, the following routine uses the base pointer to access the flag register and set the TRAP flag:

```
PUSHF
MOV BP,SP
OR BYTE PTR [BP]+1,01H
POPF
```

How the trap program works

After one instruction is executed, a type 1 interrupt is issued and the ISR takes over. All the registers are pushed on stack and then are individually accessed by the BP register. Since each register occupies two address locations in stack memory, each byte needs to be converted to ASCII format to be displayed on the PC's screen.This is the function of procedure CONVERT. The user program that is single-stepped

is a trivial example, incrementing the AL register but it serves to demonstrate the single-step facility:

```
      MOV AL,0FH
CYCLE ADD AL,1
      JMP CYCLE
```

The resulting display will be as follows:

```
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 02 FF 80 00 14 00 F7 01 0F 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 12 FF 80 00 16 00 F7 01 10 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 12 FF 80 00 14 00 F7 01 10 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 06 FF 80 00 16 00 F7 01 11 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 06 FF 80 00 14 00 F7 01 11 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 06 FF 80 00 16 00 F7 01 12 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 06 FF 80 00 14 00 F7 01 12 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 02 FF 80 00 16 00 F7 01 13 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 02 FF 80 00 14 00 F7 01 13 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 06 FF 80 00 16 00 F7 01 14 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 06 FF 80 00 14 00 F7 01 14 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 02 FF 80 00 16 00 F7 01 15 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 02 FF 80 00 14 00 F7 01 15 01 0A 00 63 00 00 00 FD 00 0B E8 E8
FLAGS  C_S   I_P   S_P  AH AL BH BL CH CL DH DL  B_P   S_I   D_I
F1 02 FF 80 00 16 00 F7 01 16 01 0A 00 63 00 00 00 FD 00 0B E8 E8
```

If we examine the software listing (TRACE.LST), it can be seen that the main user program commences at start address 0012H.The I_P and the AL contents in the above trace verify that the program is executing correctly i.e the AL registers is being incremented and the instruction pointer oscillated between addresses 0012H/0014H.

```
(Trace.lst)
     ;MAIN PROGRAM
0012  B0 0F              MOV AL,0FH  ;INCREMENT THE AL REGISTER
0014  04 01              CYCLE:ADD AL,1
0016  EB FC              JMP CYCLE   ;Repeat forever
```

It can be seen that the 8088's register values are displayed after the execution of each instruction. The stack pointer is loaded with an initial value of 0FFH. Everytime a TYPE1 interrupt occurs, the FLAG register, Code segment and Instruction Pointer registers are automatically placed on stack and the stack pointer decrements downwards two places for each register, pushing the registers into successively lower address memory locations. The remaining registers of the 8088, some of which are used by the ISR are themselves pushed onto the stack to preserve their contents. The contents of the stack, once all the 8088 register's have been pushed and the relative SP offset from the top of stack (0FFH), are shown below:

```
      STACK TOP
FLAGS +20              .
CS    +18              .
IP    +16              .
SP    +14              .
AX    +12              .
BX    +10              .
CX    +8               .
DX    +6               .
BP    +4               .
SI    +2               .
DI    +0.............STACK POINTER
```

Of course, all these operations are transparent to the user program and can be considered as a background program running which does not modify any registers or the stack pointer. If no CALL or PUSH/POP instructions are used in the user program the value of the Stack Pointer (SP) will remain at F7H: even though the ISR program will modify registers, including the IP and SP; these are all returned to exactly the same values at the point in the program at which the single step TYPE1 interrupt occurred. If the single step program in any way changed the user program execution, the exercise would be totally invalidated.

Procedures TRANSMIT should present no difficulty to the reader, since it has been used previously in example 'RS232.ASM'.

Any terminal emulator such as 'TELIX' or 'TERMINAL' in Windows may be used configured for the following RS232 protocol:1200 baud, 1 stop bit, 8 data bits, no parity. Connect the port A (PA0) line via a 2K resistor to the RXD pin of the COM1 serial port of the PC.Link, the CTS/RTS and DSR/DTR pins, and connect a common 0 volt line between pin 5 of the 9-way D-type socket and the 0 volt line of the SBC. Start the terminal emulation and escape to DOS. Meanwhile, load program 'TRACE.COM into the EPROM emulator and, after a few seconds, hold down the RESET switch on the SBC. Return to the terminal emulator by typing 'EXIT'. Releasing the RESET switch will cause the user program to start executing one line at a time, displaying the register contents on the screen.

```
       ;TRACE

CODE SEGMENT
ASSUME CS:CODE,DS:CODE

ORG 0
MOV SP,0FFH                ;Initialise Stack Pointer

CALL PORT                  ;Initialise port A

;INITIALISE SINGLE STEP
CALL VECTOR                ;Load interrupt vector
for Single step

PUSHF ;Set Trap flag
MOV BP,SP
OR BYTE PTR[BP]+1,01H
POPF
NOP

      ;MAIN PROGRAM
 MOV AL,0FH                ;INCREMENT THE AL
REGISTER
CYCLE:ADD AL,1
 JMP CYCLE                 ;Repeat forever

VECTOR PROC NEAR           ;Load INT1 interrupt
vector

 C_S  EQU 0FF80H           ;Base address code
segment
 POINTER EQU  1 * 4 ;IP offset (INT 1 X 4)
MOV WORD PTR DS:[POINTER],OFFSET ISR ;Load IP
ofset of
      ;interrupt service routine (ISR) into
pointer table at
      ;pointer table at address 0004
MOV WORD PTR DS:[POINTER+2],C_S       ;Load CS base
address
      ;(0000) into pointer tabale at address 0006
RET

VECTOR ENDP

PORT PROC NEAR

MOV DX,0                   ;Port command register
MOV AL,0FH                 ;Port A is output port
OUT DX,AL                  ;Bit 0 is Txd pin
RET

PORT ENDP

;SINGLE STEP INTERRUPT ROUTINE
ISR:
 PUSH SP                   ;Save registers
 PUSH AX
 PUSH BX
 PUSH CX
 PUSH DX
 PUSH BP
 PUSH SI
 PUSH DI
```

```
CALL LABELS                ;Print "FLAGS C_S I_P S_P
AH AL BH BL CH CL DH DL B_P S_I D_I "

MOV BP,SP
MOV AX,[BP+20]
 XCHG AL,AH
 CALL CONVERT              ;FLAGS HIGH  BYTE
 XCHG AL,AH
 CALL CONVERT              ;FLAGS LOW  BYTE

MOV BP,SP
MOV AX,[BP+18]

 XCHG AL,AH
 CALL CONVERT              ;CODE SEGMENT HIGH
 XCHG AL,AH
      CALL CONVERT         ;CODE SEGMENT LOW
MOV BP,SP                  ;Get IP from stack
MOV AX,[BP+16]
 XCHG AL,AH
 CALL CONVERT              ;IP HIGH
 XCHG AL,AH
 CALL CONVERT              ;IP LOW

MOV BP,SP
MOV AX,[BP+14]
 XCHG AL,AH
 CALL CONVERT              ;STACK POINTER HIGH
 XCHG AL,AH
 CALL CONVERT              ;STACK POINTER LOW

MOV BP,SP
MOV AX,[BP+12]
 XCHG AL,AH
 CALL CONVERT              ;AL HIGH
 XCHG AL,AH
 CALL CONVERT              ;AH  LOW

MOV BP,SP
MOV AX,[BP+10]
 XCHG AL,AH
 CALL CONVERT              ;BH
 XCHG AL,AH
 CALL CONVERT              ;BL

MOV BP,SP
MOV AX,[BP+8]
 XCHG AL,AH
 CALL CONVERT              ;CH
 XCHG AL,AH
 CALL CONVERT              ;CL

MOV BP,SP
MOV AX,[BP+6]
 XCHG AL,AH
 CALL CONVERT              ;DH
 XCHG AL,AH
 CALL CONVERT              ;DL

MOV BP,SP
MOV AX,[BP+4]
 XCHG AL,AH
 CALL CONVERT              ;BP H
 XCHG AL,AH
 CALL CONVERT              ;BP L

MOV BP,SP
MOV AX,[BP+2]
 XCHG AL,AH
 CALL CONVERT              ;SI HIGH
 XCHG AL,AH
 CALL CONVERT              ;SI LOW

MOV BP,SP
MOV AX,[BP+0]
 XCHG AL,AH
 CALL CONVERT              ;DI HIGH
 XCHG AL,AH
 CALL CONVERT              ;DI LOW

CALL NEXTINSTRUCTION       ;New line
 POP DI                    ;Retreive registers
```

```
 POP SI
 POP BP
 POP DX
 POP CX
 POP BX
 POP AX
 POP SP
ADD SP,2                ;Modify stack pointer
 IRET

      ;Convert Byte to two ASCII  characters

CONVERT PROC NEAR
PUSH AX
PUSH DX
MOV DL,AL               ;Save byte
SHR AL,1                ;High nibble
SHR AL,1
SHR AL,1
SHR AL,1
AND AL,0FH
MOV BX,OFFSET ASCIITABLE
MOV AH,0
MOV SI,AX
MOV AL,CS:[BX+SI]
CALL TRANSMIT

MOV AL,DL               ;Retrieve byte
AND AL,0FH              ;Low nibble

MOV BX,OFFSET ASCIITABLE
MOV AH,0
MOV SI,AX
MOV AL,CS:[BX+SI]       ;Convert to ASCII

CALL TRANSMIT
MOV AL,020H                   ;Space between bytes
CALL  TRANSMIT

POP DX
POP AX

MOV CX,06FFFH
DELAY:LOOP DELAY

RET
CONVERT ENDP

      ;Transmit at 1200 baud

TRANSMIT PROC NEAR

PUSH AX
PUSH DX

 ;MOV CX,0FFFH
 ;DELAY:LOOP DELAY

CALL LONGT              ;delay between characters

MOV DX,0001H            ;Port A address
XCHG BL,AL

 MOV AL,0FFH            ;Start  bit is high
 OUT DX,AL
 CALL DELAYCELL         ;1 bit delay period
 MOV CX,8               ;8 data BITS
 XOR Bx,0FFH            ;invert 8 bit data in BL
SHIFT:SHR BL,1          ;shift right into carry
flag
 LAHF ;Copy CARRY into AH register
 XCHG AL,AH             ;move AH into AL register
 AND AL,01H             ;get bit 0
 OUT DX,AL              ;output bit on port line
 CALL DELAYCELL;.833 ms delay (1 bit period)
 LOOP SHIFT             ;Repeat until 8 data bits
sent


 MOV AL,0H              ;Stop BIT is low at port
PA0
 OUT DX,AL
 CALL DELAYCELL

POP DX
POP AX

 RET

TRANSMIT ENDP

      ;New line

NEXTINSTRUCTION PROC NEAR

MOV AL,0AH                    ;Line feed
CALL TRANSMIT

MOV AL,0AH                    ;Line feed
CALL TRANSMIT

MOV AL,0DH                    ;Carriage return
CALL TRANSMIT
RET

NEXTINSTRUCTION ENDP

LABELS PROC NEAR

 MOV CX,67
 MOV BX,OFFSET TABLE
NEXT:MOV AL,CS:[BX]

 PUSH CX
 push bx
 CALL TRANSMIT
 pop bx
 POP CX

 INC BX
 LOOP NEXT
 RET

LABELS ENDP

DELAYCELL PROC NEAR       ;1 bit period delay
 PUSH CX                   ;Preserve bit count
register
 MOV CX,0A6H              ;.833 ms delay
MILLI:LOOP MILLI
 POP CX
 RET
DELAYCELL ENDP

LONGT PROC NEAR           ;character delay time
 PUSH CX
 MOV CX,0FFH
HERE: LOOP HERE

 POP CX
RET
LONGT ENDP

ASCIITABLE DB '0123456789ABCDEF'
TABLE      DB 'FLAGS  C_S   I_P   S_P  AH AL BH BL
CH CL DH DL  B_P S_I   D_I'

  DB 0DH,0AH

ORG 07F0H
JMP FAR PTR BOOT
ORG 0800H
CODE ENDS

CODEBOOT SEGMENT AT 0FF80H
ASSUME CS:CODEBOOT

ORG 0000H
BOOT PROC FAR
BOOT ENDP
CODEBOOT ENDS

END
```

Versatile regulated power supply unit

A practical upgradeable bench power supply for electronic circuits designed and developed by Tim Parker

PART 2

Last month we looked at the overall design of the power supply, thist month we continue with a look at the practical aspects of constructing the PSU. Follow the component layout diagram of fig 4. As with all PCB construction, begin with the low profile components first - wire links (of which there are 6), diodes, small resistors etc, building up to the high profile ones such as C1 and C2. Take care with diode and electrolytic capacitor polarities. Note that D8 is mounted vertically with its anode closest to the back edge of the PCB. When fitting the power diodes D1 to D4 and the resistors, mount them about 5mm proud of the PCB to allow air flow around them; this will prevent discoloration of the PCB due to heat transfer under high power operating conditions.

The power ON indicator LED1 can be fitted either way round, since it is connected via R1 across the low voltage Ac output of the transformer. This reduces power dissipation on R1 and also gives an immediate indication that the mains supply has been removed, rather than having to wait for the LED to fade out - such would be the case if it were connected across any part of the DC circuits.

The prototype heatsink was a piece of 50mm x 5mm thick aluminium angle bracket, which is sufficient for the power dissipation when used for the low current outputs mentioned earlier, and even better if bolted so as to be thermally coupled (using heatsink compound) to an aluminium or steel base plate - such as the bottom of the enclosure. If you intend increasing the output current then you will need substantially more heatsinking capability (see 'Beefing it up').

In order that the OV rail does not come into contact with the chassis - and therefore the mains earth (the benefit of this will be made clear shortly - use insulating kits and heatsink compound when mounting all four regulators, even though IC3 is the only one that actually requires insulating (its tab is connected to its input terminal, whereas all the others have their tabs connected to the common terminal). Doing this will greatly improve thermal transfer and power dissipation, prevent earth leakage currents between the regulators via the heatsink and also improve noise rejection on the output OV terminal. If a mains earth reference is required, this can be provided separately by fitting a green terminal post on the front or rear panel of the completed unit, and wiring this directly to the mains earthing point on the enclosure.

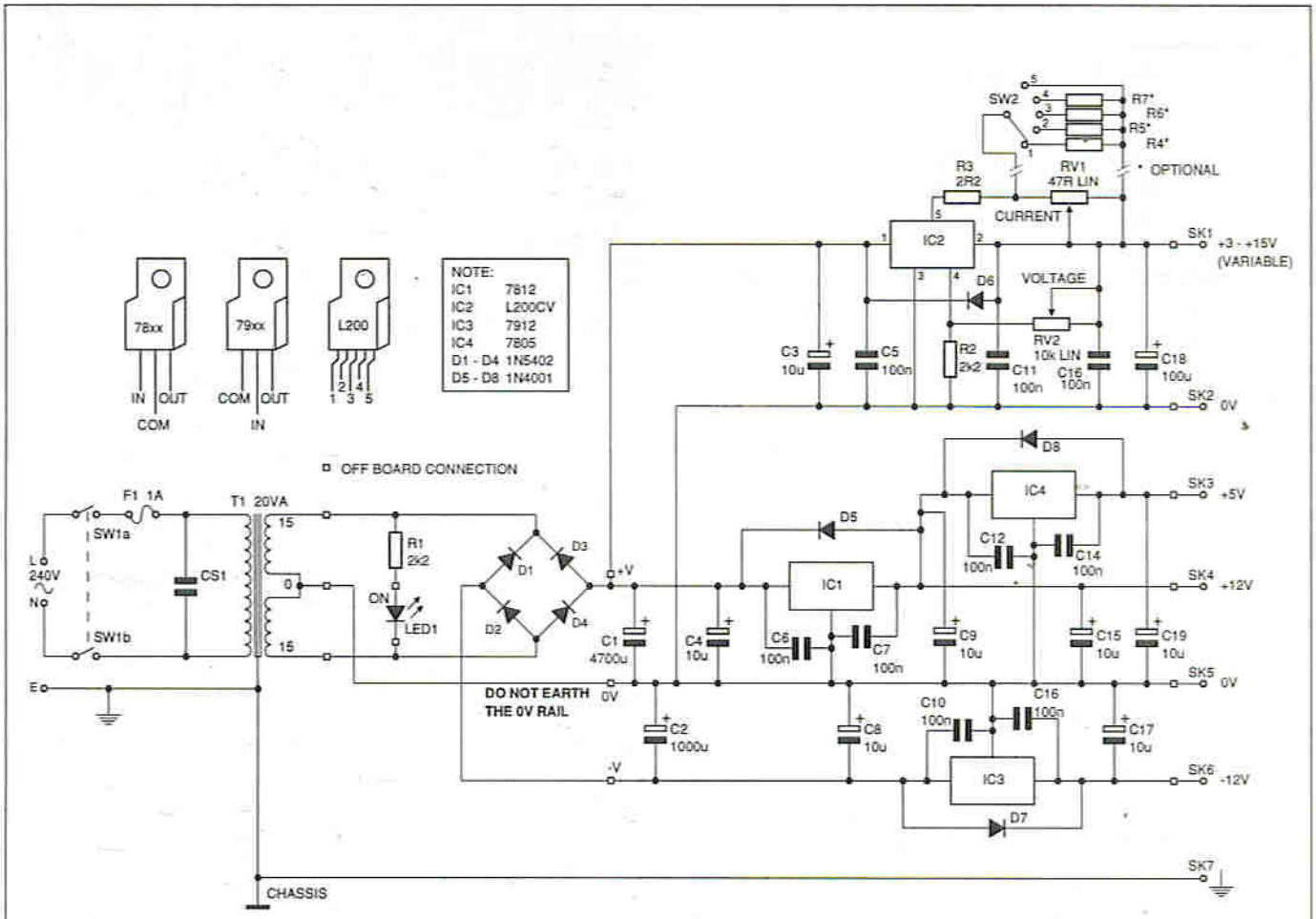
The low voltage wiring should be completed using 16/0.2 or 24/0.2 wire. To further reduce noise and the risk of hum pick-up, all wiring should be as short and neat as possible. The low voltage AC input to the left hand side of the board and regulated DC output wiring should be kept away from each other and routed separately - preferably in opposite directions, and must NOT pass close by the mains transformer.

The diagram of fig 5 shows the mains wiring and all off-board connections to be made. Make sure ALL of the mains connections are sleeved to prevent any accidental contact with the high voltage wiring. This includes the connections to the fuseholder, transformer and on/off switch. Rather than soldering, the latter connections are best made by terminating the mains wiring into fully insulated Lucar connectors to fit over the spade terminals of the switch, but be careful. Due to the changing methods used by manufacturers to retain the spade terminals in place (they used to be twisted at an angle to the body so it was almost impossible to push them back into the switch. More often than not, nowadays they are simply crimped), an awful lot of switches cannot withstand the force exerted when trying to fit the connectors, without the spade terminals being pushed inside the body. The best thing to do is to grip the spade terminal very tightly with a pair of small nosed pliers as close as possible to the body of the switch whilst pushing the receptacles home.

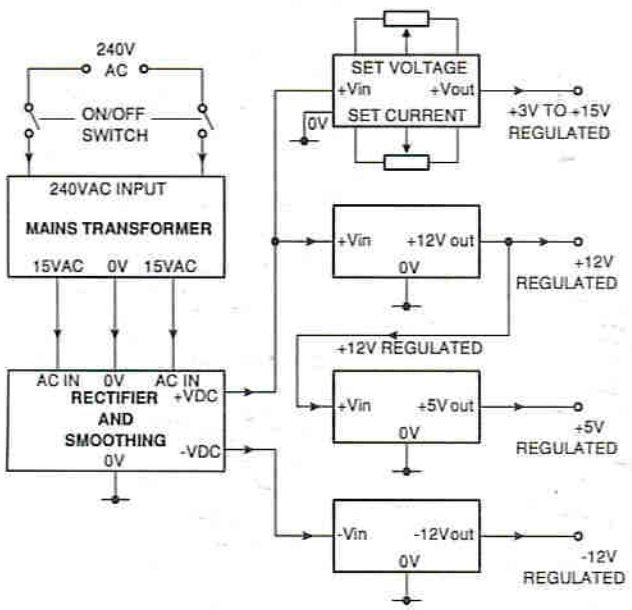
The remainder of the wiring can be protected by liberal use of good quality heat-shrink sleeving, and by using a terminal cover for the mains connections to transformer T1. If used, the contact suppressor CS1 can be soldered directly across the primary input terminals of T1, with any excess lead length being fitted with plastic sleeving.

The fuseholder for F1 can either be fitted with a rubber boot, or the whole of the rear of the fuseholder can be enclosed in a length of large bore heat-shrink sleeving, so as to cover the bottom and side terminations at the same time. If a chassis mounting fuseholder is favoured in place of the panel mounted one shown, then fit a protective cover over the fuse, and sleeve the solder connections to it.

WARNING! For safety reasons, this equipment MUST be



Complete circuit diagram for the regulated PSU



Block diagram of the regulated PSU

earthed to the incoming mains earth. Failure to do so could result in potentially lethal voltages appearing on the chassis should a serious fault condition arise. Please take extra care when building ANY item of equipment which is powered from the 240V (UK) AC mains supply. If you are unsure or have any doubts whatsoever about connecting this type of equipment ask a competent person for help. Play safe, we want you to be able to use this equipment without risk once you've built it.

It is possible to secure the front of the PCB to the front panel by using the mounting bushes of VR1 and VR2 (depending on the type used). Where this is impractical, use the mounting points located at the front corners of the PCB. The back of the board can be secured above the bottom of the case using the heatsink - fitted with mounting spacers if necessary. If the PCB hampers your desired panel layout, the potentiometers can be mounted off-board, with the PCB positioned further back to allow more access to the available front panel area. A word of advice; if you're going to use switched current limiting, the PCB will have to be mounted away from the front panel, due to the large diameter of the switch itself - it will be hampered by the current limiting resistors R4-R7. In this case, a long spindled pot is required for VR2, held in place by a separate angle bracket (see diagram). Also, a smaller hole will be required in the front panel, since only the spindle will need to pass through it.

The front panel layout is a suggestion only and is left to personal preference, which will depend largely on the type of enclosure you intend to use. No calibration scales have been

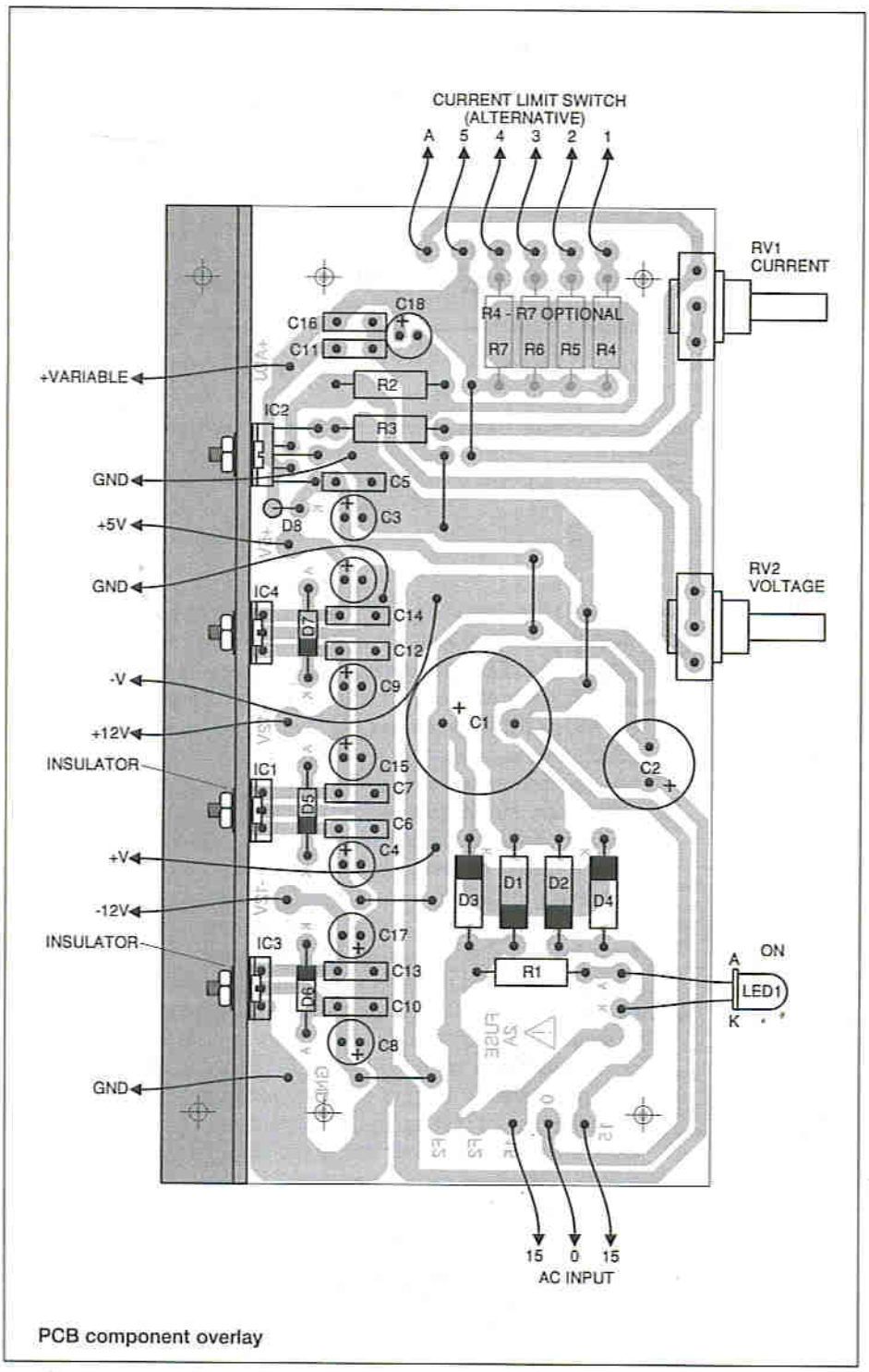
included because this will depend on whether or not the upgraded version is built, and also which type of current limiting is used - switched or variable.

In use

The amount of current that can be drawn from the power supply depends to a great extent on the size of the heatsink fitted to the regulators. It is quite possible to draw up to 1 Amp from any one (and only ONE) of the fixed outputs at any time (the variable output is limited by R3). If more than one output is in use then pay attention to the TOTAL current being drawn; you might think you only have a 500mA load here or a 500mA load there, but spare a though for T1 which has to supply ALL of the current for ALL of the outputs - remember this is a transformer, not an electric fire!

Table 2
Connections required for each fixed voltage output
NOTE: PSU MUST HAVE OV ISOLATED FROM MAINS EARTH

POSITIVE OUTPUTS			NEGATIVE OUTPUTS		
+OUT	+V	GND	-OUT	-V	GND
+5V	+5V	0V	-5V	0V	+5V
+7V	+12V	+5V	-7V	-12V	+5V
+12	+12	0V	-12V	-12V	0V
+17	+5V	-12V	-17V	-12V	+5V
+24V	+12V	-12V	-24V	-12V	+12V



PCB component overlay

transformer, not an electric fire! Remember also that the +12V output sees not only its own load but that of the +5V output too, so a 500mA load connected to each would put the full load capacity on IC1.

The various fixed voltage outputs available should be adequate for most purposes. The +12V and -12V can be used as separate and independent outputs, or as complementary supplies for powering op-amp or small power amplifier based circuits that require split supply rails. In the majority of cases it should be possible to power 9V circuits from the fixed +12V output, but do check the component ratings of the circuit to be connected first.

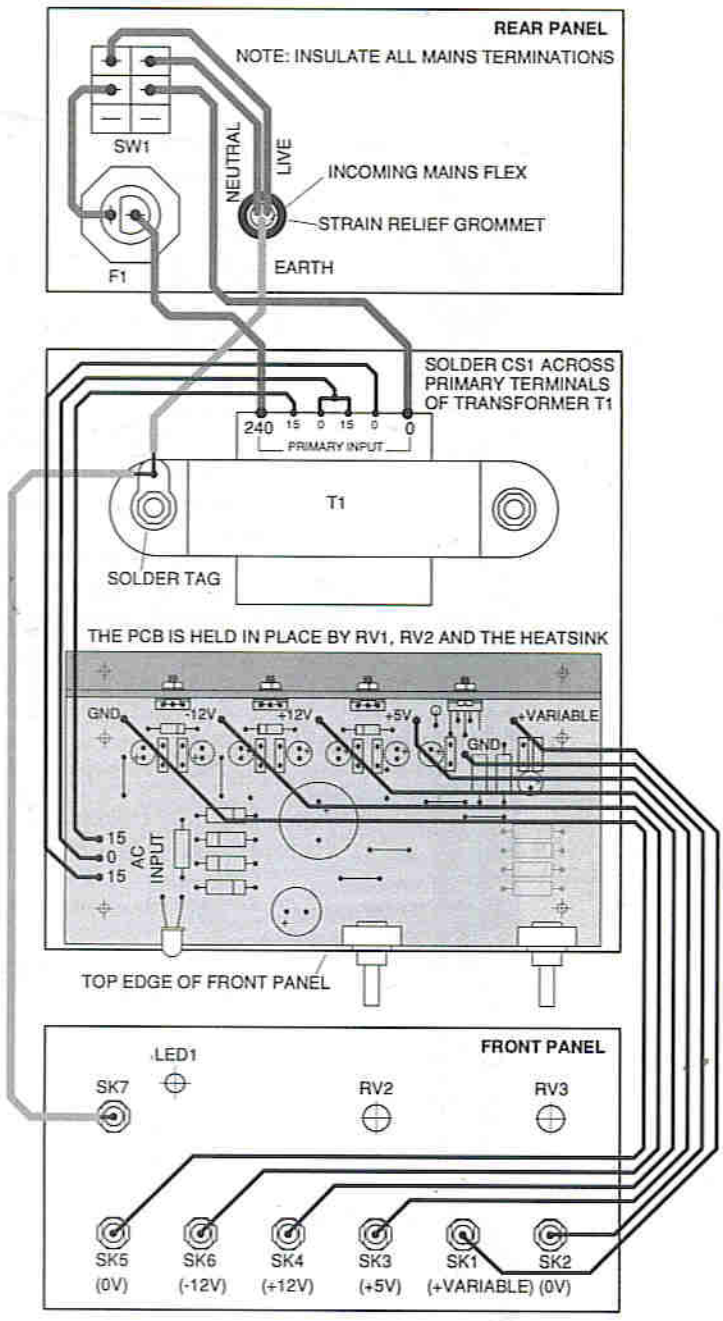
The +5V output is designed (logically - pun intended!) for TTL circuits of all types and there should be sufficient current available to power a small microprocessor board - and we don't mean a 486 motherboard.

The variable output should be able to supply somewhere between 3V and 15V at 200mA. As the current consumption rises, the upper voltage level may fall very slightly, particularly if you already have a load connected across one of the fixed voltage outputs.

More than you thought

Having the OV terminal isolated from the mains earth provides a feature sometimes overlooked or not realised with power supplies of this type; that is the availability of more fixed output voltages than those provided by the regulators alone. These 'extra' voltages are achieved by connecting across two different outputs, rather than simply using one of the fixed outputs and OV. In Table 2 the first column shows the output voltage available, the second column is the terminal to use in order to obtain that voltage and the third column shows which terminal to use as the ground reference. Further more, by

Table 1 Connections for split supplies				
+/- SUPPLIES		+V	GND	-V
+5V	-12V	+5V	0V	-12V
+7V	-5V	+12V	+5V	0V
+7V	-17V	+12V	+5V	-12V
+12V	-12V	+12V	0V	-12V
Obtainable supplies		**Terminal to use for positive**	**Terminal to use for GND reference**	**Terminal to use for negative**



Wiring diagram (Note: front and rear panels shown flat and separated for clarity)

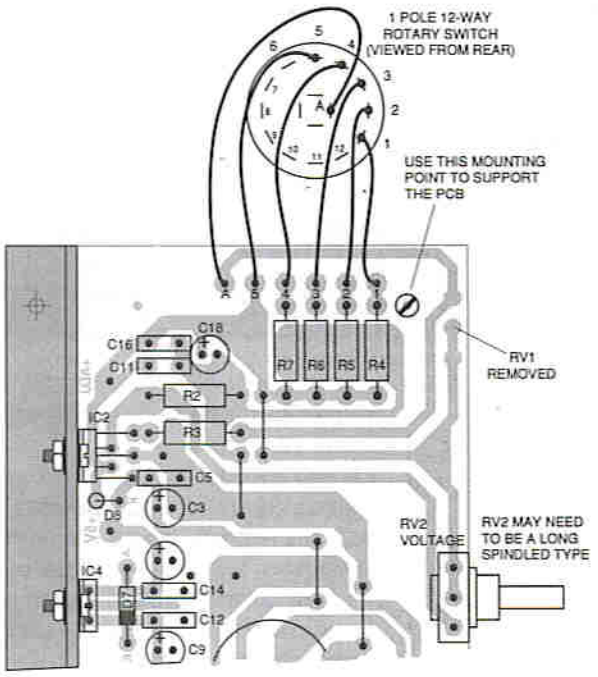
including the variable output in the same manner ANY voltage is possible from -24V to +24V.

It is also possible to obtain various non-symmetrical split rail (dual) supplies by the same method. Table 3 details the connections to be made in order to obtain all of those available. Again, inclusion of the variable output extends the possibilities even further.

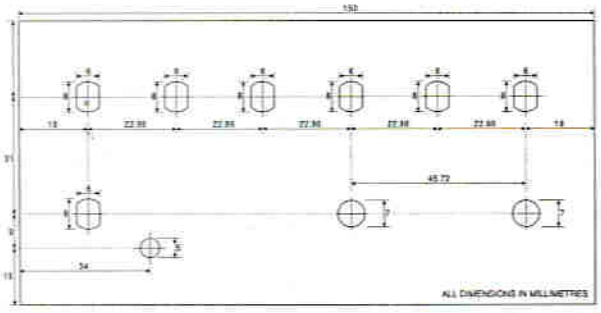
There are many circuit applications where, for example, a split supply of ±9V is specified. You may well find that the circuit will operate just as well if powered from one of the non-symmetrical supplies given in Table 3. Sometimes the voltages are specified for convenience, and a negative supply is needed just to allow the inputs and/or outputs of particular devices to pass above and below a 0V reference.

But (I hear you say) the OV rail doesn't have to be isolated from the mains earth just to be able to do all of this. Well, strictly speaking, no. But, if it wasn't isolated and you were to connect that other valued piece of test equipment - an oscilloscope - to the circuit under test (that's whatever you've got connected to, and being powered from the power supply), the output of the power supply that you are using as your GND reference would be shorted to 0V through the physical chassis GND of the oscilloscope, since most of them have their croc-clip terminal connected internally to the mains earth. Do not confuse the term 'GND' with '0V', - it is quite possible to have these held at different potentials.

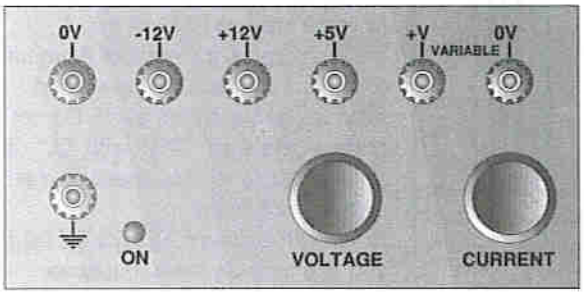
Isolating the power supply's 0V rail allows you to effectively have a positive, negative or zero volt GND, a feature which is not quite as useless as you might at first think. In days of old, there was quite a lot of 'hi-fi' (well we thought it was 'hi-fi' then, in fact anything that reproduced the music louder than the



Connections for switched current limit selection (VR1 must be removed)



Front panel cutting diagram



Front panel legend

PARTS LIST

Resistors

- R1, 2 — 2K2 — 0.5W carbon (2 off)
- R3 — 2n2 — 2.5W wirewound
- R4 - R7 — optional - (selected by user) *
- VR1 — 47n LIN potentiometer
- VR2 — 10Kn LIN potentiometer

Capacitors

- C1 — 4700uF/35V radial electrolytic
- 2 — 1000uF/35V radial electrolytic
- C3,4,8,9,15,17,19 — 10uF/35V radial electrolytic (7 off)
- C5,6,7,10,11,12,13,14,16 — 100nF/50V ceramic or polyester (9 off)
- C18 — 100uF/35V radial electrolytic
- CS1 — 100n+100nF 250V RC contact suppressor

Semiconductors

- D1 - D4 — 1N5402 3Amp rectifier diodes (4 off)
- D5 - D8 — 1N4001 1Amp rectifier diodes (4 off)
- IC1 — 7812 +12V 1Amp voltage regulator
- IC2 — L200CV 2Amp adjustable regulator
- IC3 — 7912 -12V 1Amp voltage regulator
- IC4 — 7805 +5V 1Amp voltage regulator
- LED1 — 5mm standard red LED

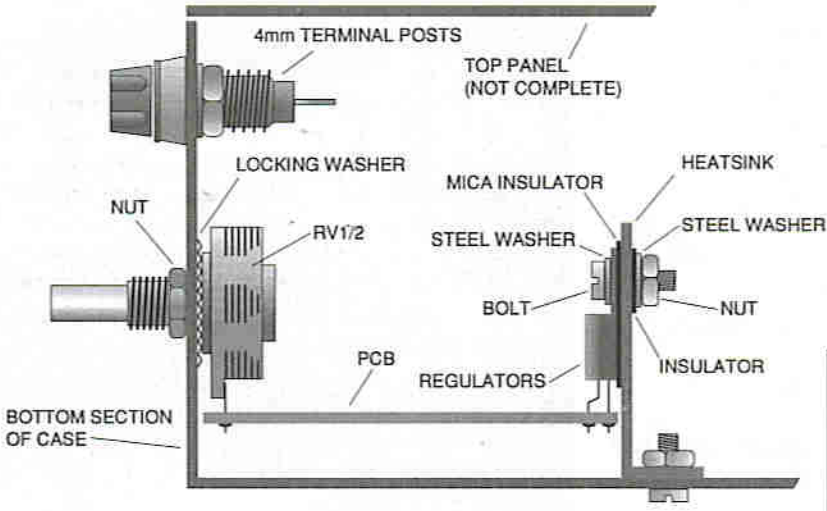
Hardware & miscellaneous

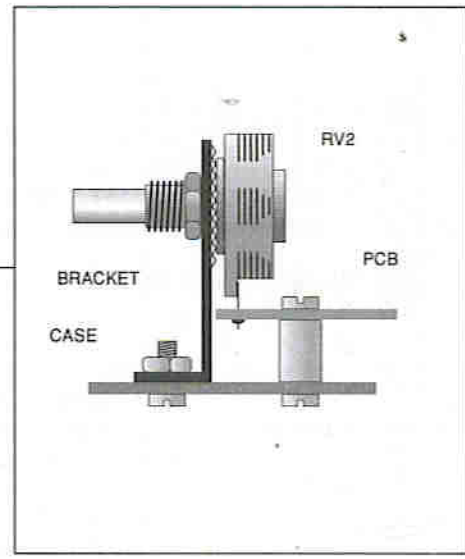
- F1 — 20mm 1Amp fuse
- fuseholder — 20mm panel mounting fuseholder
- SK1,3,4,6 — 4mm red terminal posts (4 off)
- SK2,5 — 4mm black terminal posts (2 off)
- SK7 — 4mm green terminal post
- SW1 — DPST rocker switch rated 6A/250V
- T1 — 0-15-0-15 20VA mains transformer

heatsink thick aluminium angle 150 x 50 x 50
fixings screws, nuts & washers as required
knobs control knobs for VR1 & VR2
insulators insulating kits for IC1 - IC4
terminals lucar connectors for SW1 (4 off)
flex 6A mains flex
grommet strain relief grommet
cover terminal cover for T1
sleeving heat-shrink sleeving
case enclosure to suit *
legend front panel label *

PCB DTE "PSU" printed circuit board £6.50
available from DTE MICRO SYSTEMS

* Not supplied with the kit





buzzing was considered 'hi-fi') equipment which had the positive output from the rectifier connected directly to the chassis, which in turn was connected to the mains earth. Not too dissimilar to some old cars which were classed as positive earth due to the battery's positive lead (rather than the negative one) being connected to the chassis; and anyone who ever owned one of these (like me) will remember just how much of a problem it was trying to fit a standard negative earth radio to it, by reversing the power wires, making sure it was completely isolated from the chassis, removing the earth connection from the speaker (yes, there really was only one speaker fitted. This was when AM meant Any Music, FM meant Fuzzy Music and the insensitivity of VHF meant Very Hard to Find) etc. When you powered it up everything seemed to work fine up to a point, that is, until you tried to plug the aerial in. Yeah - BANG!

Anyway (digression over), there are occasions, albeit possibly rare ones, when a positive or negative GND reference is either useful, desirable or maybe even necessary for the correct operation of some circuits.

Beefing it up

As mentioned, the design presented here is for low current outputs up to about 250mA. In order to take full advantage of each regulator's capabilities, it is necessary to make substantial changes to various components. The changes are supplied here for those who wish to increase the available output current and the voltage range of IC2. These modifications assume that potentiometers VR1 and VR2 are to be used to control current and voltage. If you intend to build the switched current version using R4 - R7 then simply skip item 7 below and use the switch in the same way as before. The changes are:
Transformer T1 secondary should be 20-0-20 and at least 50VA. The mains fuse F1 will need to be increased to 2A.
Ideally, diodes D1 - D4 should be replaced with a 6 Amp bridge rectifier mounted off-board and bolted to the chassis, although this is only required if the TOTAL continuous current consumption of the power supply is anticipated to rise above 2 Amps.

Capacitor C1 should be increased to 10,000uF and ideally have a voltage rating of 50V (absolute minimum of 35V).
Capacitor C2 should be increased to 2200uF/50V. Again, with an absolute minimum working voltage of 35V.
R1 should be increased to 3K3.
R3 should be replaced with a wire link.
VR1 must be a 100n / 2Watt (min) high power type potentiometer.
R2 should be replaced with a 820n / 2.5W wirewound type.
VR2 must be a 10kn / 2Watt (min) high power type potentiometer.
Regulator IC1 should be a 78S12 (2A) or 78T12 (3A) type.
A substantial heatsink rates at 2 to 4 C/W should be fitted, which may result in the regulators being mounted off-board.

Given all these changes it is easy to realise what makes linear high current regulated power supplies such expensive items of equipment. Indeed, these component changes alone will quite easily more than double the cost of the original design. It should go without saying that these modifications will require a large, well ventilated enclosure which allows plenty of air flow in order to keep everything at a reasonable operating temperature. Although the regulators can be quite forgiving when operated beyond their limits (they have output short circuit protection, thermal shutdown etc.) it would be a wise idea to fuse each regulated output separately. This is better than fusing the raw DC supplies, since fuses fitted here would have to be large enough to carry the current for ALL of the outputs under full load conditions, which means that a serious fault on just one of the outputs would not draw sufficient current to blow the fuse.

PARTS LIST

Resistors

3	2n2	2.5W wirewound
R4 - R7	optional - (selected by user) *	
VR1	47n LIN potentiometer	
VR2	10Kn LIN potentiometer	

All resistor 0.25W 5% except R2 and R5 which will be as available - see text.

Semiconductors

D1 - D4	1N5402 3Amp rectifier diodes (4 off)
D5 - D8	1N4001 1Amp rectifier diodes (4 off)
IC1	7812 +12V 1Amp voltage regulator
IC2	L200CV 2Amp adjustable regulator
IC3	7912 -12V 1Amp voltage regulator
IC4	7805 +5V 1Amp voltage regulator
LED1	5mm standard red LED

Hardware & miscellaneous

F1	20mm 1Amp fuse
fuseholder	20mm panel mounting fuseholder
SK1,3,4,6	4mm red terminal posts (4 off)
SK2,5	4mm black terminal posts (2 off)
SK7	4mm green terminal post
SW1	DPST rocker switch rated 6A/250V
T1	0-15-0-15 20VA mains transformer
	heatsink thick aluminium angle 150 x 50 x 50
fixings	screws, nuts & washers as required
knobs	control knobs for VR1 & VR2
insulators	insulating kits for IC1 - IC4
terminals	lucar connectors for SW1 (4 off)
flex	6A mains flex
grommet	strain relief grommet
cover	terminal cover for T1
sleeving	heat-shrink sleeving

Kit available

A kit of parts for the variable low current version of the power supply (using potentiometers) including PCB, 20VA transformer and heatsink (but not including the case - this is left to personal choice) is available from the author by mail order only at the following address:

DTE MICRO SYSTEMS
112 SHOBNALL ROAD
BURTON ON TRENT
STAFFORDSHIRE
DE14 2BB

The price for the kit of parts is:
£39.00 inc.
The PCB is also available separately at:
£ 6.50 inc.
Please add postage and handling (per order):
£ 2.50 for the UK
£ 4.00 elsewhere.

Please make Cheques/Postal Orders payable to 'DTE MICRO SYSTEMS'. If ordering from overseas, payment must be in Pounds Sterling (£) and cheques must be drawn on a British bank.

Goods will normally be dispatched within five working days from receipt of order (subject to availability and cheque clearance), but please allow up to 28 days for delivery.

Designing a Pic Micro controller based project

PART 5

Bart Trepak continues his construction of a PIC based alarm clock

Last month we completed the clock counting circuits and the software presented so far should, if loaded into a micro-controller, count hours and minutes and display them on an LED display. The only problem is that the clock always starts at 2:34 and runs at an enormous speed. The latter problem can be corrected by changing the contents of TBREG (time base register) as mentioned last month. In this month's article we will consider software techniques for reading switches and write the programme for setting the time as well as generating the alarm when required.

A switch in time

Almost every micro-computer application requires some form of input and the most commonly used are switches although the same kind of strategy would be used if the input were connected to a thermostat, phototransistor, logic gate or some other kind of digital input. Because the PIC16C5X series do not support interrupts which could be used to make a programme branch when a switch closure was detected, the switch inputs must be "polled" or read often enough to make sure that any switch closures or inputs are not missed or that the micro does not take an inordinately long time to respond. This can easily be done by writing the software for this as a subroutine which can be called as often as required by the main programme in the same way that the DISPLY programme is called in the clock project. Normally, once every half second or so is fast enough for mechanical switch inputs and any micro-controller would be able to manage this - usually it is done a great deal more often. In this design the subroutine is called every 10mS so there is not much chance of a key closure being missed.

Where only a few functions need to be controlled, the switches are normally connected as shown in Fig. 19 to a previously defined input port and the negative or positive supply rail, with pull-up or pull down resistors as required. The switches are simply "read" when required by using the BTFSS or BTFSC instruction and specifying the I/O port and bit to which the switch is connected. Thus, if the switch was connected to port B3 and the positive rail, the instruction "btfss PORTB,3" followed by a "goto...." would be used. If the switch was not pressed, bit 3 of port B would be clear and the GOTO instruction would be executed, the programme branching to perhaps read the next switch. If, however, the switch had been pressed when the instruction was executed, the GOTO instruction would be skipped and the required function such as incrementing the minutes register or switching on an LED connected to an output port would be executed

Normally, when reading mechanical switches, such as push

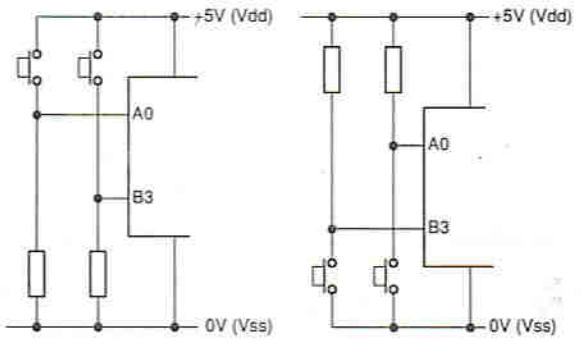


Fig. 19 Connecting switches to I/O ports

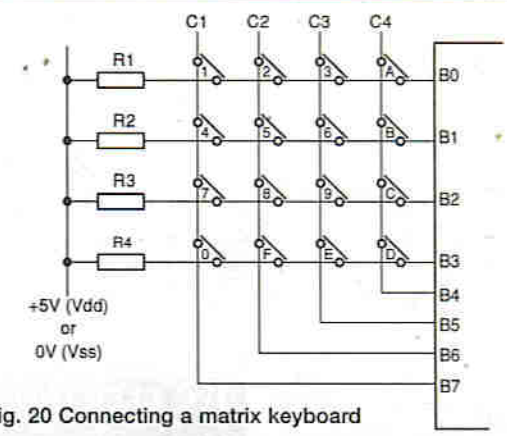


Fig. 20 Connecting a matrix keyboard

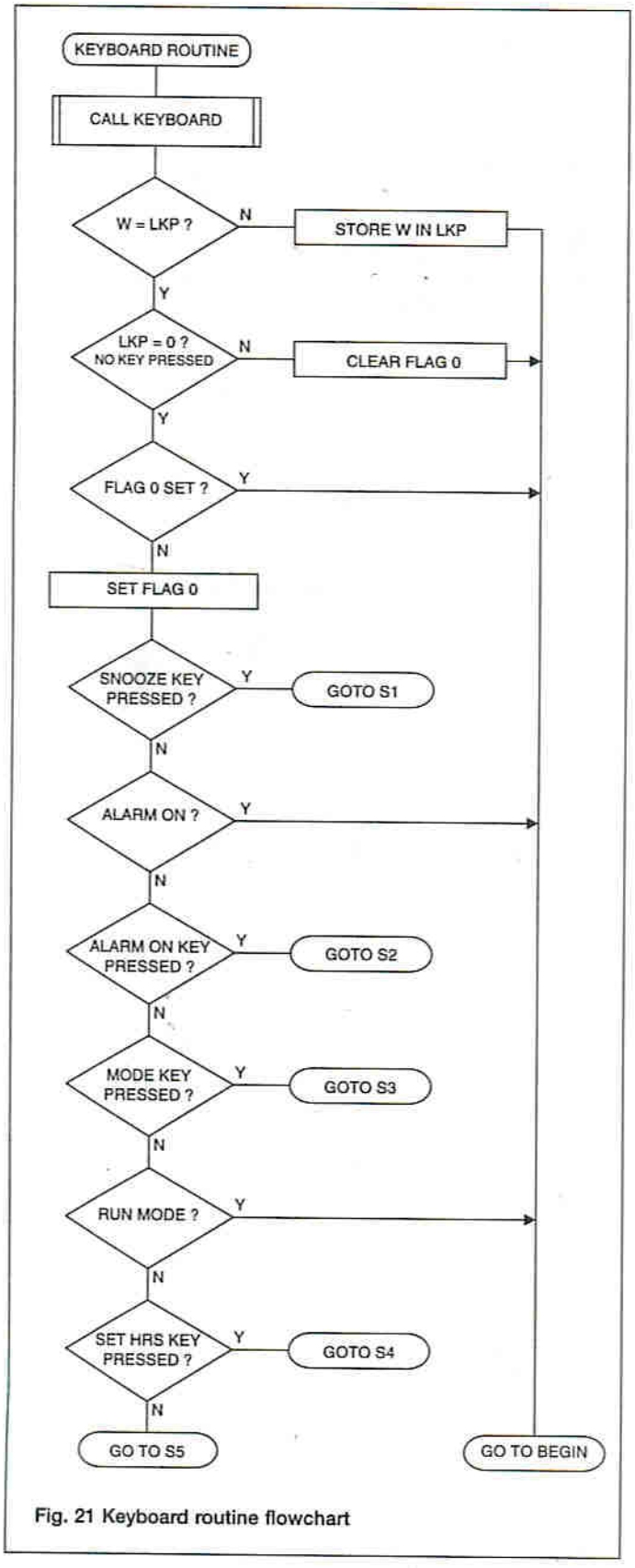


Fig. 21 Keyboard routine flowchart

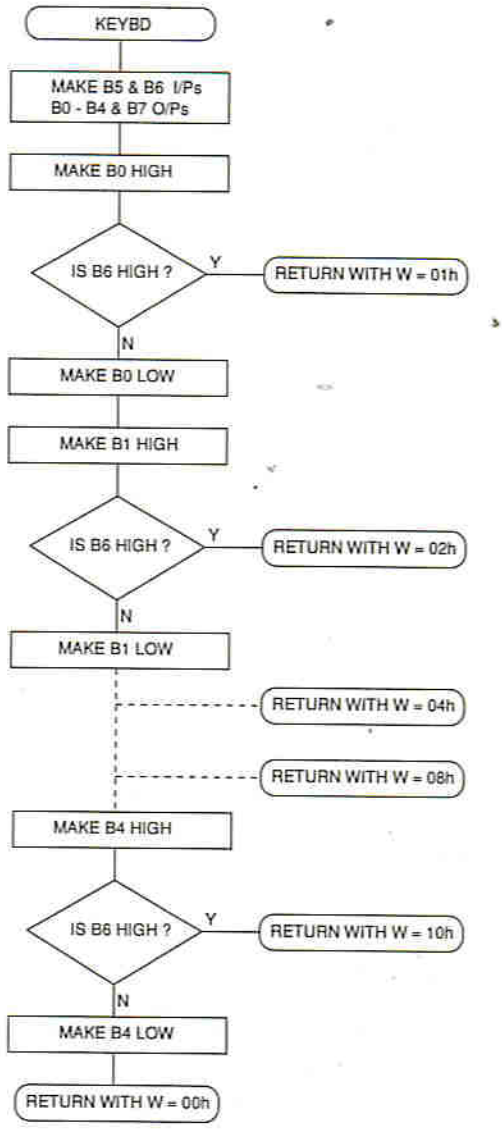


Fig 21a Keyboard subroutine flowchart

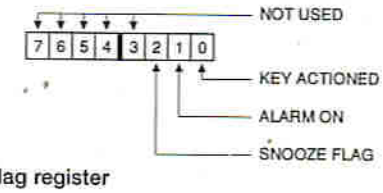


Fig 21b Flag register

buttons, reed switches or relays, the problem of contact bounce must be addressed. If it is not, the programme may read a single switch closure as a number of closures with more or less disastrous consequences depending on the function of the switch. If the switch serves to start a 10 second timer for example, then the first switch closure detected would start the timer and subsequent closures would not alter the situation other than perhaps restarting the timer a number of times during the few mS during which the switch contacts bounce. This would clearly not affect the operation of the unit except that it may be triggered by short noise spikes picked up on the input line - but this could be cured in other ways.

However, the switch function was such that it started the timer but if pressed while the timer was running it stopped it, then it would be a matter of luck as to which state the timer would end up in when the switch was pressed and the

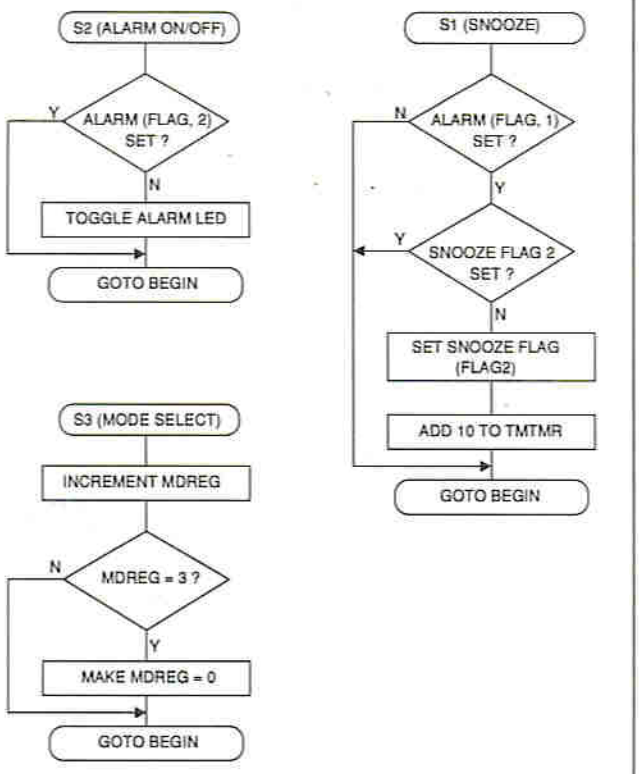


Fig 22 Switch functions flowchart

contacts stopped bouncing. To overcome this problem, a short delay routine is normally introduced so that the switch is read and ,if it has been pressed this is stored (by setting a "flag" bit in a register), and after the delay it is read again. Only if the switch closure is still detected, is the function carried out - if it is not, the flag is cleared. A five to ten milli-second delay is usually sufficient. This can either be generated by a routine similar to the delay routine described earlier or by simply setting a flag and carrying on with the execution of the main programme and calling the switch reading subroutine the next time around if the programme takes about this long to execute.

If the switch is still pressed when it is read again, the function of the switch may be carried out. Similar precautions must also be taken when the switch is released. These comments do not, of course, apply to inputs read from non-mechanical switches such as logic gates or Hall Effect switches where there is no contact bounce.

It must also be remembered that a micro-computer programme runs very fast and the subroutine (even including any delay which has been introduced to take care of contact bounce) may be called many times while the switch is depressed. For example, even if the switch is depressed for half a second - which is a very short time - it is still much longer than any likely key debounce routine and the time taken to perform the switch function, so that the function will be performed many times. The code must therefore be written in such a way that this is not misinterpreted as multiple switch closures by the programme. The simplest way of doing this is to make the programme detect a situation where no switches are closed before it is allowed to carry out any new function or even the same function in response to a switch closure. In this way, a switch must be released before another switch can be read and the length of time for which a switch is depressed is then immaterial.

Keyboards

The above method of connecting switches uses dedicated I/O lines programmed as inputs and is suitable for applications requiring, say, one or two switches. If a larger number of switches is required, as in the case of a keyboard for example, the number of I/O ports required can easily exceed the number available on the chip so a matrix connection is normally used. A 4x4 matrix would use 8 bits of I/O and enable 16 keys to be connected whereas only eight switches could be used if the

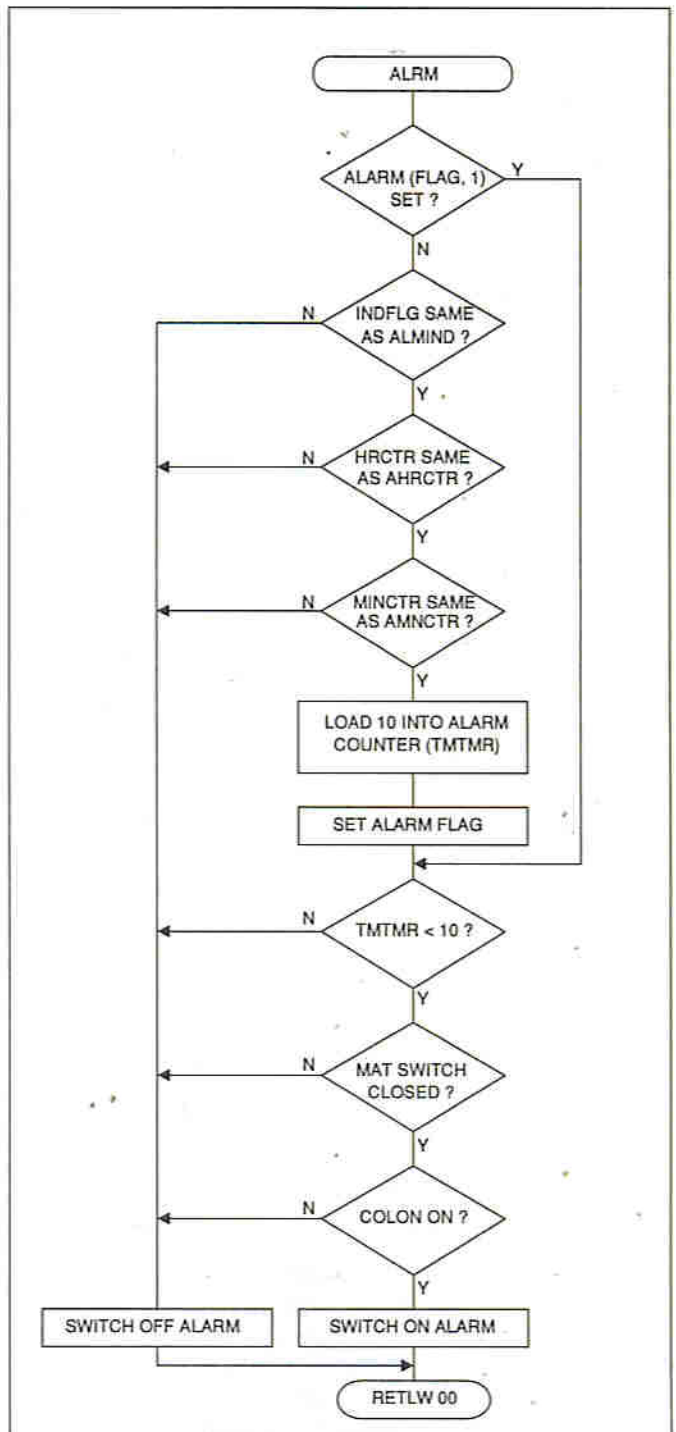


Fig 23 Alarm subroutine flowchart

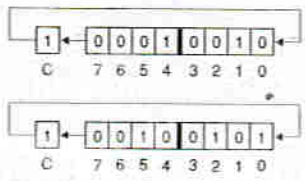
arrangement in Fig 19 were used. A matrix connected keyboard is shown in Fig. 20 and the method of driving this is similar in principle to that used in driving multiplexed displays. In fact, with some careful design, it is often possible to use the same lines for connecting the switches as for driving the displays. By using the digit drive lines (port A) and the segment drive lines in the clock project, a 28-way keyboard organised as a 7x4 matrix could be devised.

There are various algorithms available for reading matrix connected keyboards but perhaps the simplest is to drive each column line high in turn and read the resulting row inputs. In the example shown in Fig. 20, the Rows R1-R4 are connected to ports B0-B3 which are programmed as inputs and the columns C1-C4 are connected to ports B4-B7 which are configured as outputs. The four resistors function as pull-up or pull-down resistors and give either logic 1 or logic 0 levels on B0-B3 when no keys are pressed depending on which supply rail they are connected to. If the negative rail is chosen, then the column drivers B4-B7 will be driven high in turn while if the resistors are connected to the positive rail, B4-B7 should be driven low in turn.

Suppose it is the first case and key 6 has been pressed. When B4 is taken to +5 Volts, the inputs B0-B3 will read zero and this will be interpreted as no keys closed. B4 will then be taken to 0 Volts and B5 to +5 Volts. This time B1 will read high and this can be used to return to the main programme with the value 06h in the W register using the RETLW instruction. Alternatively, the programme could be made to jump to a specified location where the key function would be executed if this key did not have a numerical value.

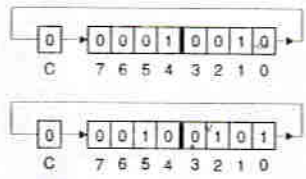
There is a trap here for the unwary, however, and it has to do with the way that the PIC micro-controller reads and writes data to the output ports. The actual write operation to an output port occurs at the end of an instruction cycle whereas, for reading, the data must be present on the input pins at the beginning of the read instruction cycle. In the above example, where B5 is taken high, this could be done either by means of a "movwf PORTB" instruction having first loaded W with the binary value 0010 0000 or by means of a "bsf PORTB,5" instruction which are both write instructions and therefore carried out at the end of the cycle. The next step in the operation would be a "movf PORTB,W" or perhaps a "btfss PORTB,0" instruction to read the resulting pin voltages to determine which, if any, key had been pressed. This read operation will occur immediately after the previous write operation and depending on the circuit values, the new pin voltages will, in all probability, not have had time to stabilise before the pin status is read into the processor. This can result in the wrong values being entered or the wrong operation carried out. It can be very perplexing to the programmer and take many hours to track down. What is worse is that sometimes the programme may appear to work and carry out the required function such as incrementing the hours display, for example, but, on occasion, the minutes display may be incremented instead or the programme fail altogether.

Because the effect is also dependent on the values of pull up or pull down resistors and the circuit capacitances, the prototype may work fine on a breadboard but on the finished unit, which happens to have longer wires from the keyboard, the same programme appears not to function leading the designer to suspect a hardware fault in the final unit and to spend many hours searching for a non-existent fault. For this reason, write followed by read operations on the same port should, as a matter of course, be separated by NOP or other instructions not using this port to enable pin voltages to assume their new values - this can be seen in the final listing of the keyboard programme.



Original contents of file = 12h (18 decimal)
Contents after RLF instruction = 25h
Note: If carry had been 0, contents would have been 24h (36 decimal ie. doubled)

Fig 25a Rotate left through carry



Original contents of file = 12h (18 decimal)
Contents after RRF instruction = 09h (ie. halved)

Fig 25b Rotate right through carry

It is often useful if the computer has an indication that a key has been pressed as this can be used to enter the key debounce routine and also to drive a sounder as this may be required with some keyboards to give the user an indication that the key pressed has been detected. With most of the keys this would not be a problem because the W register will contain a value other than zero which can be tested in the normal way by checking the zero bit of the STATUS register.

The problem is, of course, in numerical keypads when the zero key is hit as the code returned will then also be zero and indistinguishable from that resulting from no keys being pressed. The solution to this is to return from the keyboard routine with, for example, 86h, 83h or 80h when keys 6, 3 or 0 are pressed and 00h only when no keys are operated. The "8" can be easily removed from the returned code by using the "andlw 7Fh" (ie. 0111 1111 binary) instruction which will cause the most significant bit to be reset to zero and leave the numerical value of the key unchanged. Control keys such as "#", "*" etc can return with values greater than 89h or with bits 5 or 6 set and can then be treated differently.

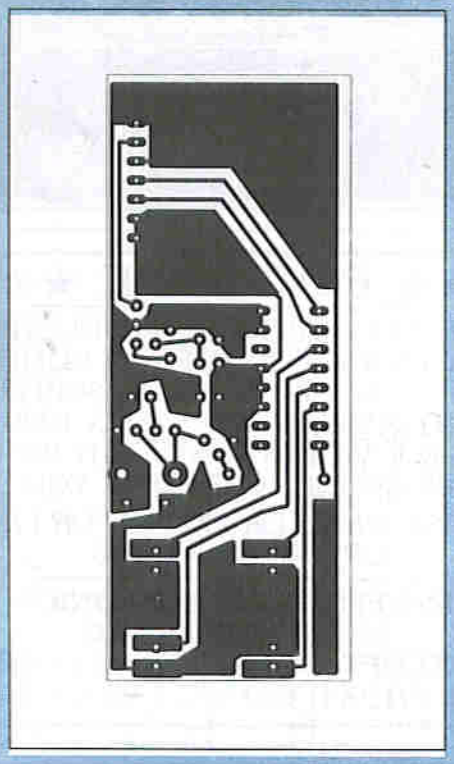
The problem of contact bounce may be solved as mentioned before by reading the key twice, separated by a suitable delay, and checking to see that the same code is returned both times. Another problem which needs to be resolved in the case of multi-way keyboards is what should happen if two or more keys are depressed simultaneously. With a scanned keyboard, it can easily be arranged for the programme to jump out of the keyboard subroutine as soon as it detects any key closure so the value read in the event of two keys being pressed at once will simply depend on the order in which the keys are scanned. This will avoid the programme ever reading more than one key or trying to carry out two functions at once.

Next Month...

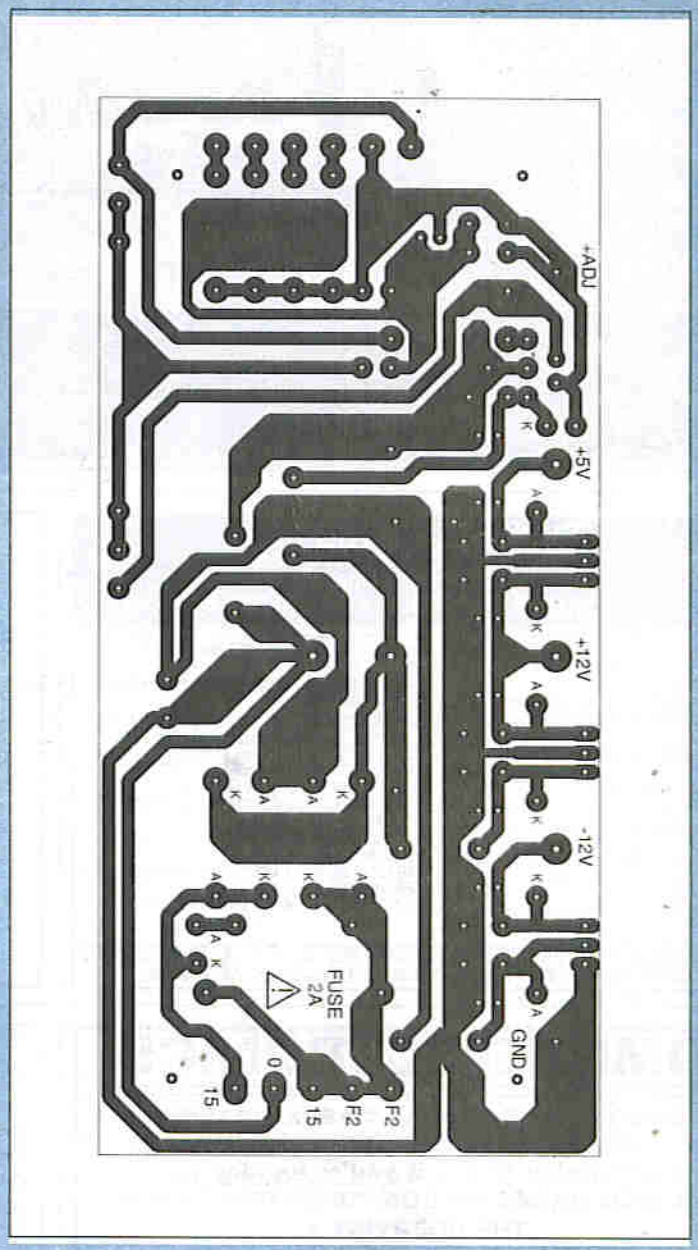
we will conclude this tutorial series with a complete software listing.

ETI
ELECTRONICS
TODAY INTERNATIONAL

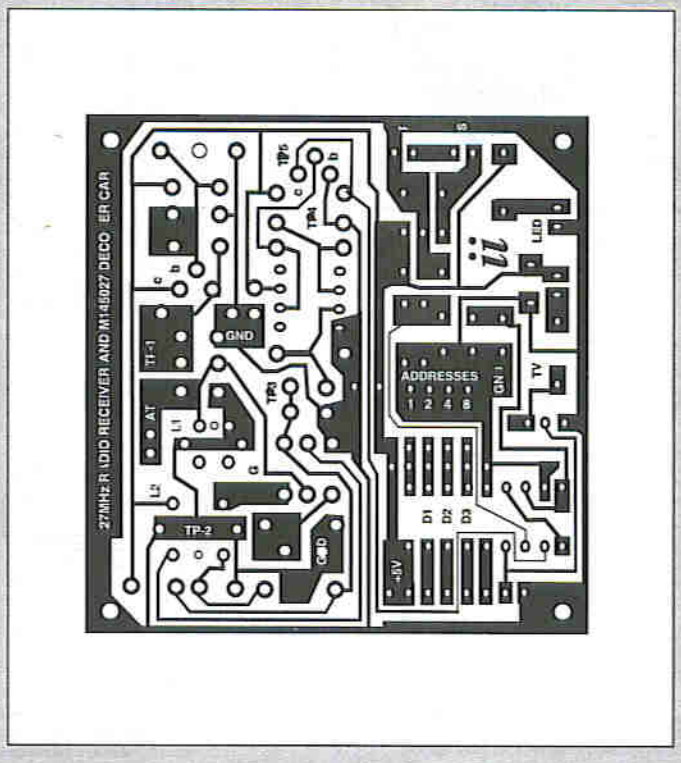
FOILS FOR THIS ISSUE



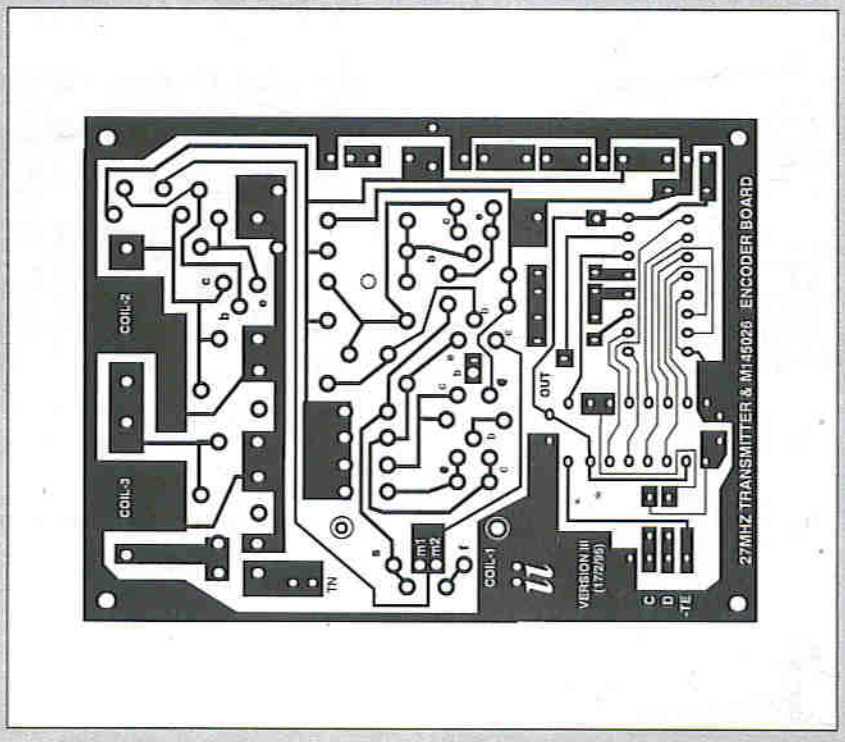
Eti Basic Microcontroller



Bench Power Supply



Computer radio control



Computer radio control

RoundUp

Few people in the industrialised countries of the world can have failed to notice the razzmatazz accompanying last month's launch of Microsoft's Windows 95. Neither will they have failed to notice the claims from a myriad commentators that it will revolutionise personal computing and the PC in particular. But what few will have noticed is the enormous impact that this piece of software will have upon the electronics industry.

For a start, this very complex and, without a doubt, very powerful operating system and user interface needs a system with an equally powerful processor. This means at the very least a high speed 486, and better still a Pentium. But what it also means is an enormous potential demand for the next generation of Intel processors, the 150MHz plus P6, the first production samples of which will be distributed in November at a price rumoured to be about $4000 per chip.

Windows 95 will thus provide a very powerful forward driving force for both Intel and the makers of Intel processor clones. Users of low speed 486s and 386s will find themselves unable to use new generations of software and will be forced to buy new systems - an estimated 65million machines world-wide in the next 12 months alone.

It is hardly surprising, therefore, that Intel and its followers are gearing themselves up for bumper sails of processor chips. But, there is a large cloud on the horizon which could wipe out much of this optimism - the fact that Windows 95 will also undoubtedly lead to a world-wide shortage of DRAM chips.

The reason for this is a combination of two things. The first is that Windows 95 needs a lot of RAM in a system - 8 megabytes to work properly and ideally 16 megabytes. With large numbers of existing machines having far less than the minimum 8 meg, there will be a colossal demand for upgraded RAM. It is estimated that some 15million PCs will be upgraded within the next twelve months, in other words probably about 300million standard 4Mbit DRAM chips.

Added to this will be the increased need for memory chips in the 65million new PCs which will be produced and the majority of which will be sold with Windows 95 installed and sufficient memory to run it, at least another 1.2billion chips. This means that these two areas of demand will between them consume virtually the entire world production of 4megabit DRAMs, which is expected to peak over the next couple of years at 1.5billion chips per annum, plus many of the 300million or so 16megabit DRAMs which are now being made.

The result is an impending world shortage of DRAM chips, a shortage which is compounded by failures to invest in new plants in Japan, a result of the high yen and the deep recession there. However, investment in new factories is taking place in other parts of the world, a total of 14 new plants are being built at a cost of over $1billion each (e.g. Siemens and Fujitsu in the UK).

This represents a big increase in production but, even so, analysts are predicting that the consequent 46% increase in production within the next year will be insufficient to meet demand and that another four factories are needed immediately. It should be noted that already DRAMs account for about one quarter of the semiconductor industry's annual sales of $138billion.

The second problem is that the 16 megabit 4-pin DRAMS produced by the Japanese companies have proved unpopular with PC designers; they prefer the US designed 16-pin chips which give them more flexibility since they can be packaged into either 4 or 16 megabyte modules. The result is an unexpected continuation in demand for 4 megabit chips as opposed to 16 megabit chips, despite the fact that the larger chip is now cheaper per bit than the smaller one.

With this impending shortage and inevitable rising chip prices, it is hardly surprising that theft of RAM chips, and the counterfeiting of chips is becoming a problem. Some of the largest chip manufacturers in the Far East have suffered from theft of out of spec chips - these are normally destroyed - which have then resurfaced in the marketplace as full spec branded devices.

Next Month...

In the December 1995 issue of Electronics Today International we will be continuing our ETI Basic programmable micro controller. Dr Pei An shows how to build a bar code reader that will allow a PC to remotely control any device with a range of several hundred yards. From David Geary there is a timed isolator system which should prevent mains equipment being accidentally left on. Whilst if you need some peace and quiet, Robert Penfold shows how to build a simple noise masker. Bart Trepak concludes his practical look at designing a project around the PIC micro controller.
In the main feature article next month, ETI will be taking a look at some interesting ideas for a shape changing robot and reviewing some PC software of interest to readers.

ELECTRONICS TODAY INTERNATIONAL

EDITORIAL
Editor *Nick Hampshire*
Sub Editor *Eamonn Percival*
Editorial Assistant *Lynn Bugden*

CREATIVE
Designer *Nadia Ahdout*
Technical Illustration *John Puczynski*
Photography *Manny Cefai*

ADVERTISEMENT SALES
Display Sales
Alison Wetherill
Advertisement Copy Control
Marie Quilter
Classified Sales
Jim Gale
Group advertising manager
Diane Farnham

MANAGEMENT
Divisional Director
Terry Pattisson
Production Administrator
Theresa Davis
Business Manager
Claire Jenkinson
Marketing Manager
Jason Doran
Copy Sales Manager
David Pegendam

ABC CONSUMER PRESS
ISSN 0142-7229

Subscription rates-UK £25.80 Europe £34.70 Sterling Overseas £36.20 US Dollars Overseas $54.00

Published by Nexus Special Interests, Nexus House, Boundary Way, Hemel Hempstead HP2 7ST. Telephone (01442) 66551. UK newstrade distribution by SM Distribution Ltd, 6 Leigham Court Road, London SW16 2PG. Telephone 0181-667 5111. Overseas and non-newstrade sales by Magazine Sales Department, Argus House, Boundary Way, Hemel Hempstead, HP2 7ST. Telephone (01442) 66551. Subscriptions by Nexus Subscription Services, ETI, Queensway House, 2 Queensway, Redhill, Surrey RH1 1QS. Telephone (01737) 768611.
US subscriptions by Wise Owl Worldwide Publications, 4314 West 238th Street, Torrance, CA90505 USA. For Visa/Mastercard orders in USA - Telephone (310) 375 6258 Fax (310) 375 0548. Pacific Time: 9am-9pm Weekdays. 10am-6pm Weekends.
Typesetting and origination by Ebony, Liskeard, Cornwall. Printed by Wiltshire Ltd. Bristol.

NEXUS Special Interests

Nexus House, Boundary Way,
Hemel Hempstead HP2 7ST
Telephone (01442) 66551
Fax (01442) 66998