MICROCHIP

Microchip products are available from:

Farnell COMPONENTS

Farnell Components Ltd.
Sales, Marketing and Administration Centre,
Canal Road, Leeds, LS12 2TU

SALES: Tel: 0113 2636311 (24 Hrs)
Fax: 0113 2633411 (24 Hrs)
TECHNICAL SUPPORT: Tel: 0113 2799123
DATALINE: Tel: 0113 2310160 (24 Hrs)
(Data Sheets Only)

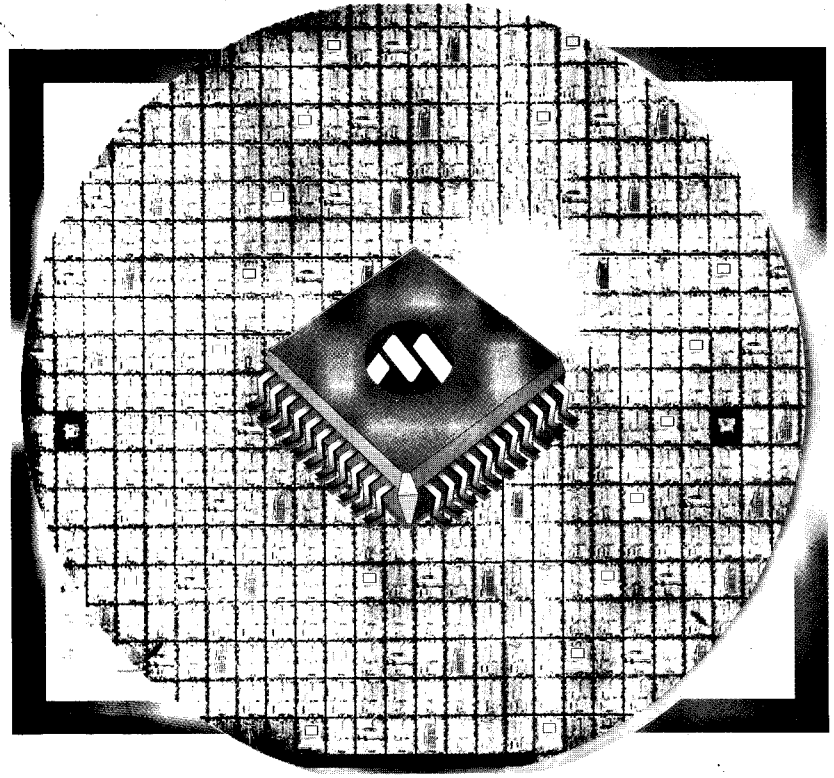
ISBN 1-899013-02-4

9 781899 013029

PIC COOKBOOK

Volume 1

A collection of Working Application Ideas



Farnell COMPONENTS

654991

BY NIGEL GARDNER & PETER BIRNIE

Typeset and printed in the UK by: Character Press Limited, Icknield Way, Baldock, Herts. Tel: 01462 896500.

Circuit diagrams produced with Labcentre Isis Illustrator. Flowcharts produced with Corel Flow.

Introduction

Following on from the Beginners Guide to the Microchip PIC, we felt there was a need for a collection of working applications which could form the building blocks for a number of projects. The idea for this book started sometime in 1994 and has finally evolved into this publication. Articles in magazines are a useful source of examples for those starting out in the great PIC race, but source code is usually provided as a listing for you to type in or only as a pre-programmed device. For those reasons, we have enclosed a diskette with the source code for all the applications discussed.

To aid clarity on circuit diagrams, the oscillator and reset circuitry has been omitted - it's all in the data sheet. You will need to select the final components in this area based on the product requirements but as guidance, most of the designs are based on a 4MHz resonator - see source code listings for any variations.

The applications contained in this publication are not public domain - you have paid for this book, so let others who want the software to do the same. If you develop products based on the our designs and make vast fortunes, please remember where you got the idea and mention us as the source and let us know your successes.

Work is currently under way on Cookbook Volume 2 with a new set of applications.

Happy PICing
Nigel Gardner

Contents

PIC Interface to the DDM4 LCD Display Module

Program - DDM4.ASM **16Cxx**

The Lascar DDM4 is a 4 digit x 7 segment LCD display module which uses a Microchip AY0438 to handle the segment driving. Only 3 pins on a PIC are needed to interface to this module - they are the clock, data in and load lines. This program can be used with any of the PIC family and does not use interrupts.

This example of the DDM4 interface converts a voltage from the A0 analog input to a value from 0-5.1 volts on the display in 20mV resolution steps. The total program takes 124 words of program space, but this includes an 8x8 multiply and a 16 bit to 5 digit BCD routine. A simpler program displaying only the A/D result as 0-255 can take as little as 90 words of program space.

The controlling software is made up of a number of subroutines. The two most important of these being the lookup table to convert a BCD character into the DDM4 bit format and the display output routine to send the data from the character registers to the display.

In the lookup table, the value transferred to the W register is added to the program counter to create an offset. As the value in W will be in the range 0-9, no checking is provided to see if a value >9 has been transferred. The table can be extended to allow conversion of the letters a - f for use in hex display applications.

```
Lookup  addwf   pcl,f
                ; g f a b c d e .       this is the pattern for the
                                        output bits within the DDM4 -
                                        see data sheet
        retlw   b'01111110'  ;0
        retlw   b'00011000'  ;1
        retlw   b'10110110'  ;2
        retlw   b'10111100'  ;3
        retlw   b'11011000'  ;4
        retlw   b'11101100'  ;5
        retlw   b'11101110'  ;6
        retlw   b'00111000'  ;7
```

```
        retlw   b'11111110'  ;8
        retlw   b'11111100'  ;9    the table can be extended to
                                   include additional HEX
                                   characters
```

The display routine outputs 32 bits of information from registers digit1 to digit4 to the data line. Data is latched into the DDM4 on a falling edge of the clock pulse. After the 32 bits are outputted, the load line is pulsed to latch the information into the driver chip. The timings of data, clock and load signals fall within the data sheet specifications for the AY0438 with the PIC running at 4MHz.

```
Display movlw   .32          ; preload register with 32 bits of data
        movwf   digits
        rrf     digit4,f     ; send data serially
        rrf     digit3,f
        rrf     digit2,f
        rrf     digit1,f     ; bit 0 was first data in
        bsf     clock
        btfss   status,carry ; test for 1 or 0 in carry
        goto    zero_bit
        bsf     data         ; set data line high
        goto    n_bit
zero_bit bcf    data         ; set data low
n_bit   bcf     clock        ; latch data into register
        decfsz  digits,f     ; count down digits to be sent
        goto    Display+2    ; loop till all digits outputted
        bsf     load         ; latch data to display
        nop                  ; wait 1uS
        bcf     load
        return
```

The design can be expanded for additional displays on the same set of control wires This would require connection of the lower display DO line to the upper displays DI line and commoning of the CLK and LOAD lines. The software needs the number of digit registers to be increased within the Display routine by 4 for each module added. This enables a number of displays to be chained together without adding to the I/O lines available on the PIC, and all displays are updated at the same time - see figure 1b.

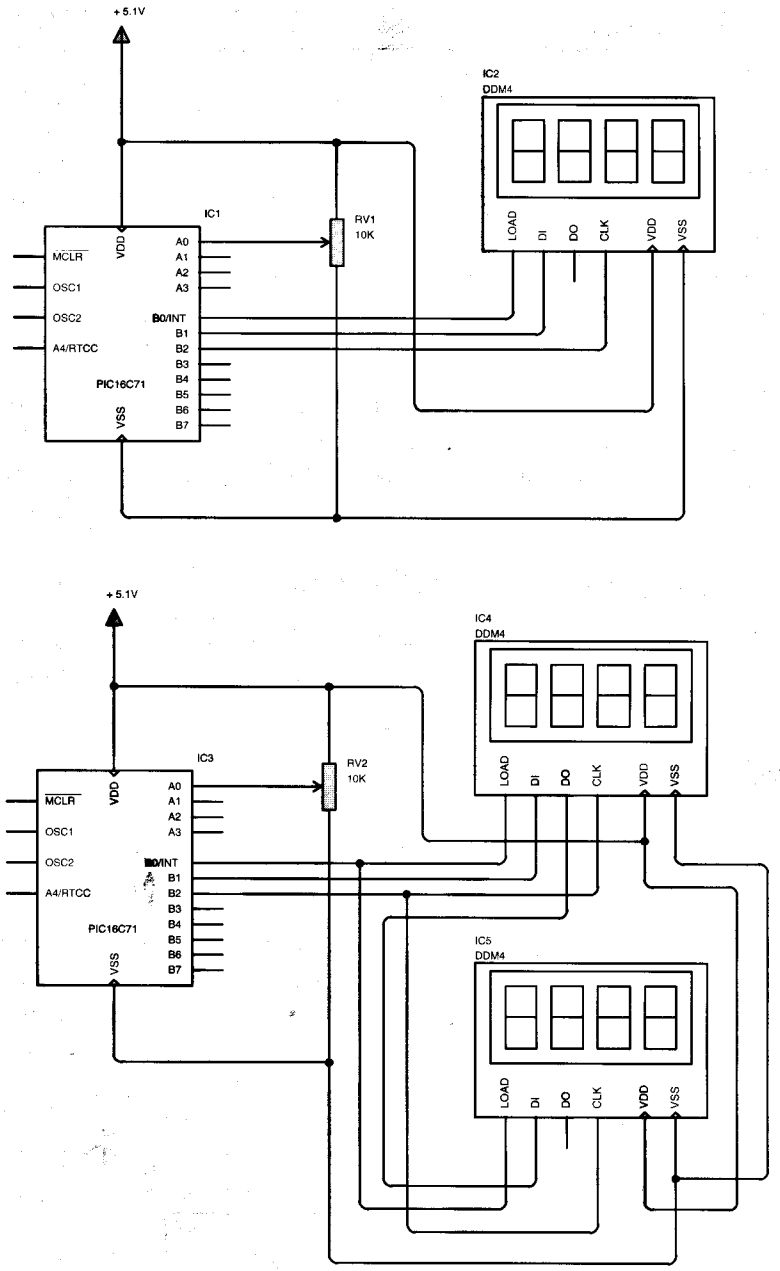


FIGURE 1a + FIGURE 1b

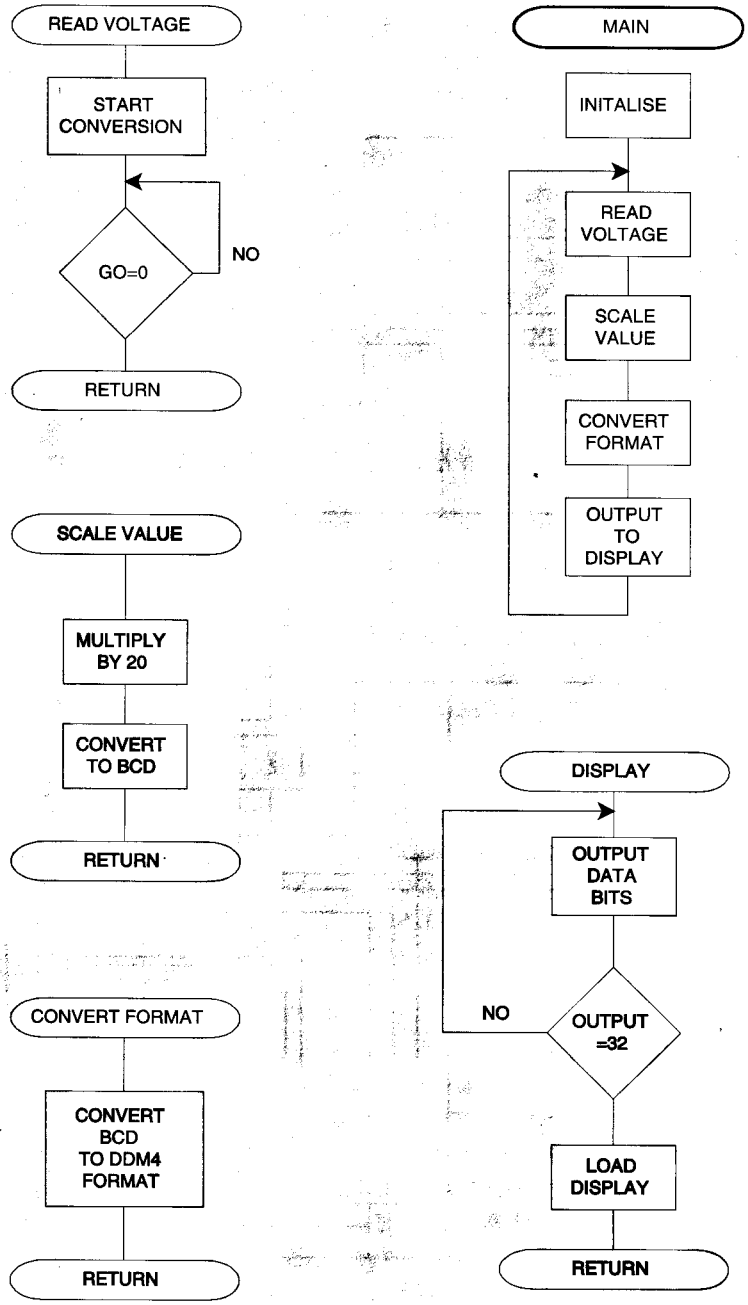


FIGURE 2

Piezo Wakeup

Program - PIEZO.ASM **16C5x**

This routine can be used to wake up a PIC from its sleep state by the use of a piezo sensor. When struck, or if attached to an item which is struck (a bell for example), the piezo sensor generates a voltage pulse which is AC coupled and clamped to the MCLR pin. A negative transient on the pin will force a reset and hence wake the PIC from a sleep state.

On reset / power up, the PIC needs to know if it is waking for the first time. The power down and time-out bits do not have a known state on initial power up so can't reliably be used to direct the software accordingly. A simpler solution is to place a resistor and capacitor on a spare pin. On initial power up, the pin will be at 0V. Subsequent resets or wakeup from sleep conditions will see the pin at Vcc.

A wakeup from sleep on a 16C5x will cause the software to jump to the reset vector.

```
portb   equ     6               ;
#define RC      portb,0         ; rc time constant for testing first time
                                ; power up

        org     00

; ****** initialise ******

init    clrf    portb
        movlw   b'00001111'     ; portb lower 4 bits as inputs
        tris    portb

        btfss   RC              ; test for first pass
        sleep                   ; yes first time

main                            ; program continues here on after
                                ; a wakeup from sleep as opposed
                                ; to the first time through.
```

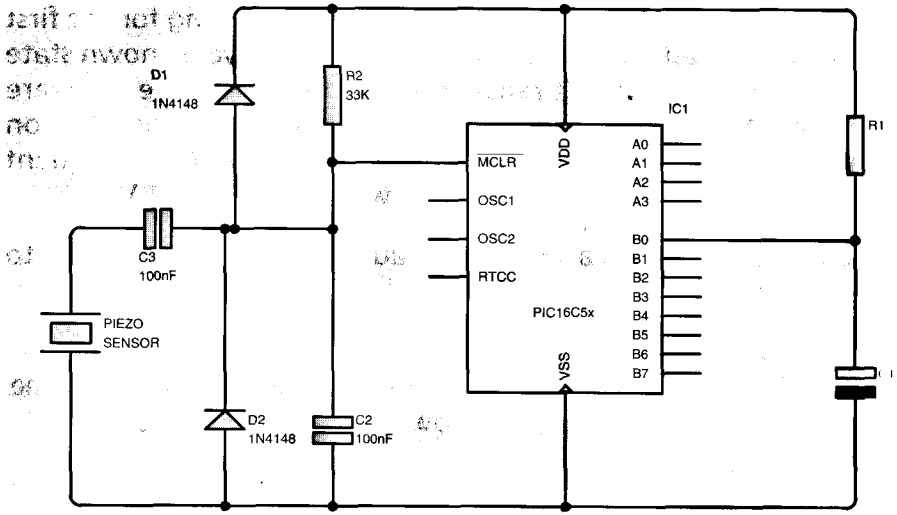


FIGURE 3

PWM on a 16C71

Program - PWM.ASM **16C71**

This example generates a PWM signal at a fixed frequency and can be used for DC light dimming or motor speed control. PWM waveforms are used to control DC power by chopping the supply to the load at a fixed frequency, but varying mark space ratio within this period. The average current is thus controlled and hence the overall brightness or speed.

The PWM control is via an analog input on port A or an 8 bit word on port B. Selection is made between analog or digital on A2 with an over-ride facility - full power - via A3. The PWM signal is presented to the outside world via A4. As A4 is an open collector output, a pull up resistor is needed to generate the PWM waveform. The A1 pin is spare and could be used to monitor the resulting voltage to form a closed loop system.

If the PWM signal is being used in a safety critical system e.g. - engine speed control, then the output should be AC coupled to prevent problems with a stuck bit. If the output does latch in a logic 1 condition, the voltage on the other side of the capacitor will naturally fall to zero.

The voltage converted from A0 or the value on port B will be in the range 0 - 255. This value is then subtracted from 255 to obtain the Mark and Space values with a test to check if a 0 is transferred into the delay routine. In this example, the base frequency for the PWM generation is set by the oscillator and RTCC prescaller.

```
read_dig
    movfw   portb       ; get portb reading
    movwf   on_time     ; save it

pwm_out
    sublw   0ffh        ; get diff
```

```
        bcf     porta,out       ; inverted logic for open collector output
        call    delay
        bsf     porta,out
        movfw   on_time
        call    delay
```

See Figure 5 for flow chart.

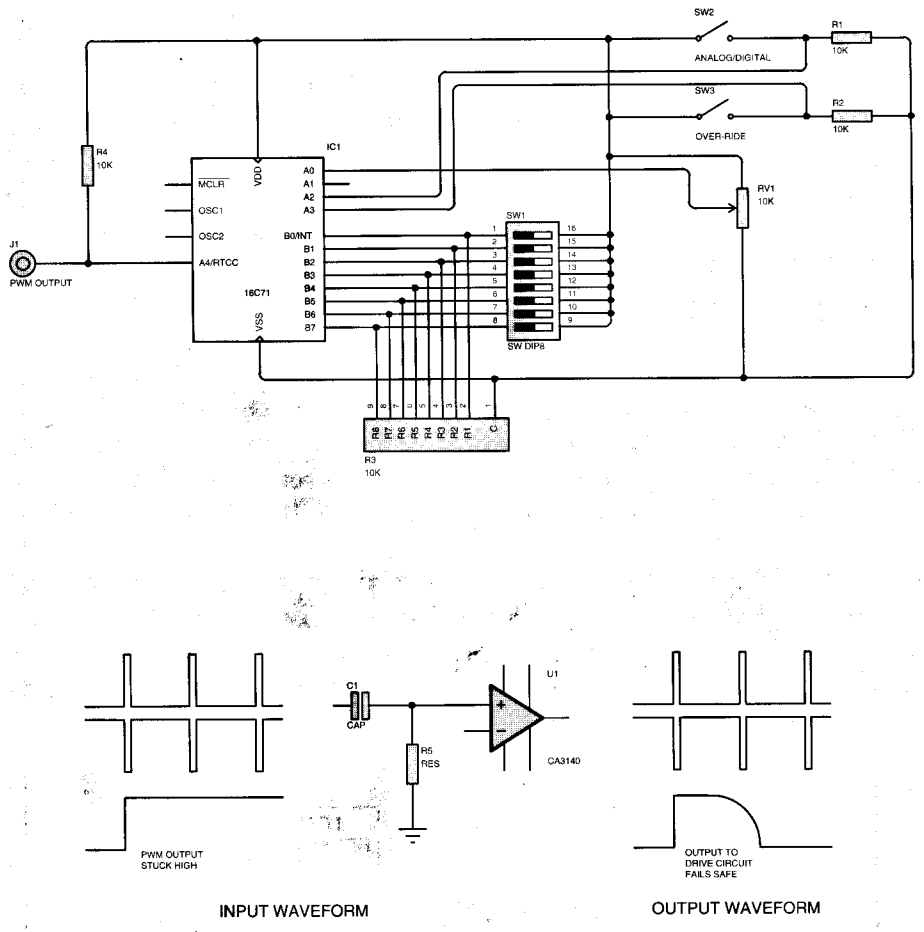


FIGURE 4

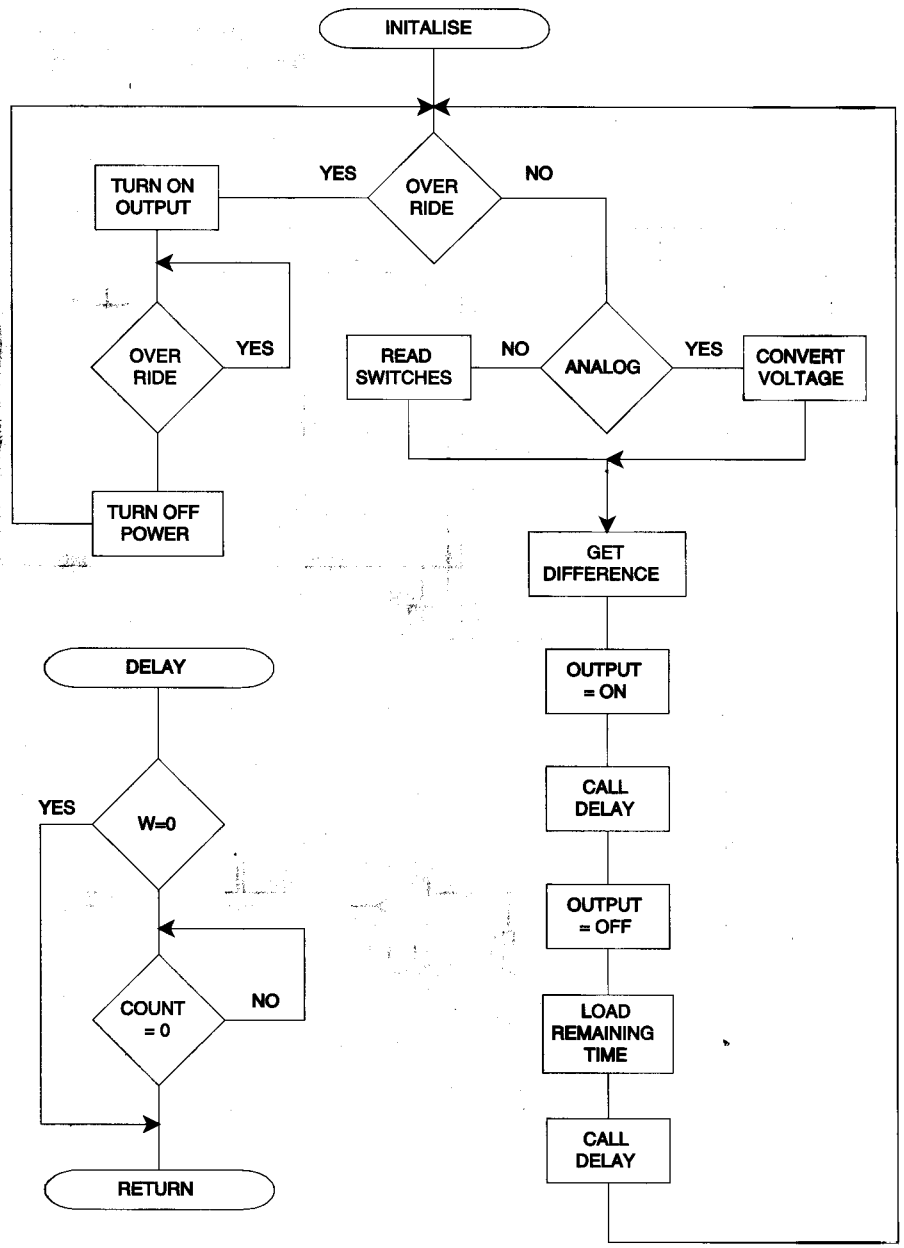


FIGURE 5

Traffic Light Sequencer

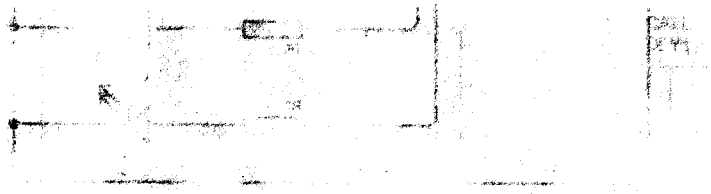
Program - TRAF1.ASM **16Cxx**

This traffic light sequencer could be used as part of a model railway set or with slight modification be scaled up for roadside use.

The basic unit steps through its sequence either manually or at a fixed speed. For home use, the speed of 5 seconds between steps is probably adequate. In the auto mode, the lights continually sequence. The manual mode changes the lights from one direction of traffic flow to the other enabling the standard 'road work' operation or a 4 way junction.

To enable the design to be used commercially, a variable time control could be added to change the duration between changeover sequences. The design would then run on a 16C71, 73 or 74 and the various time settings in an analog form could be read and converted to actual times within the software.

The light sequence is set in the software and follows the standard :- Red - Yellow - Green - Green & Yellow - Red pattern. Modification of the software for 3 way junctions can easily be made by increasing the number of blocks of light patterns with their associated delays.



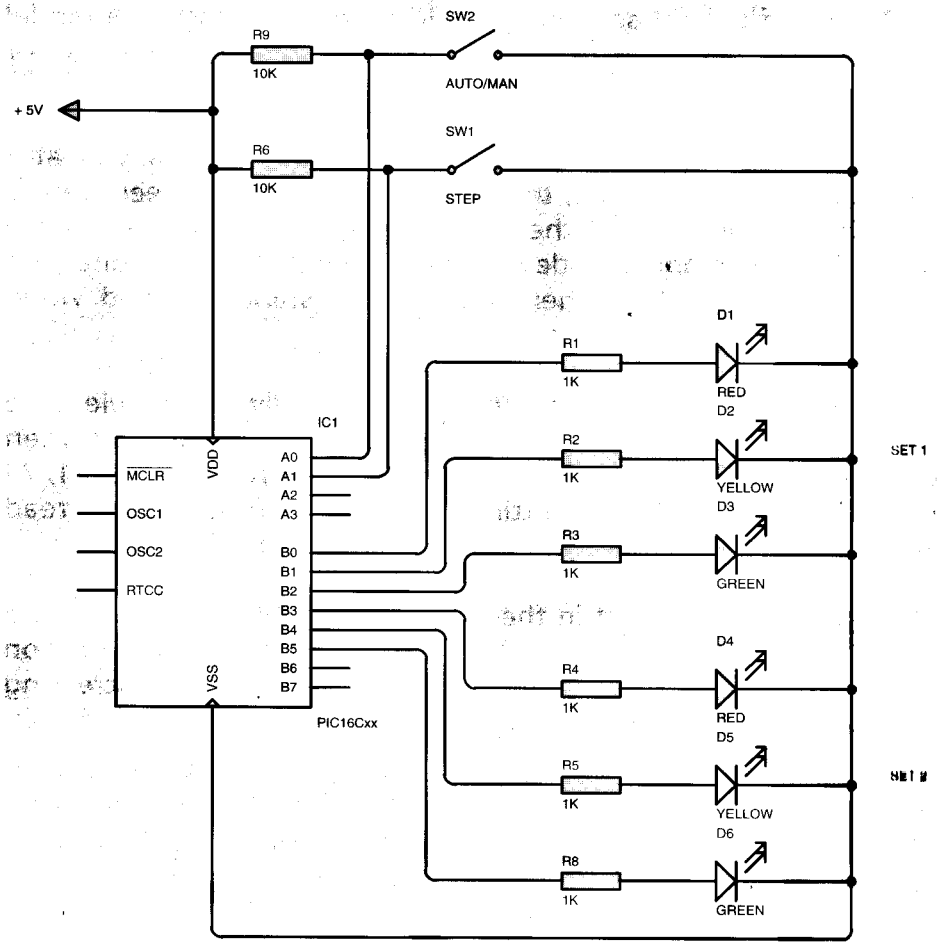


FIGURE 6

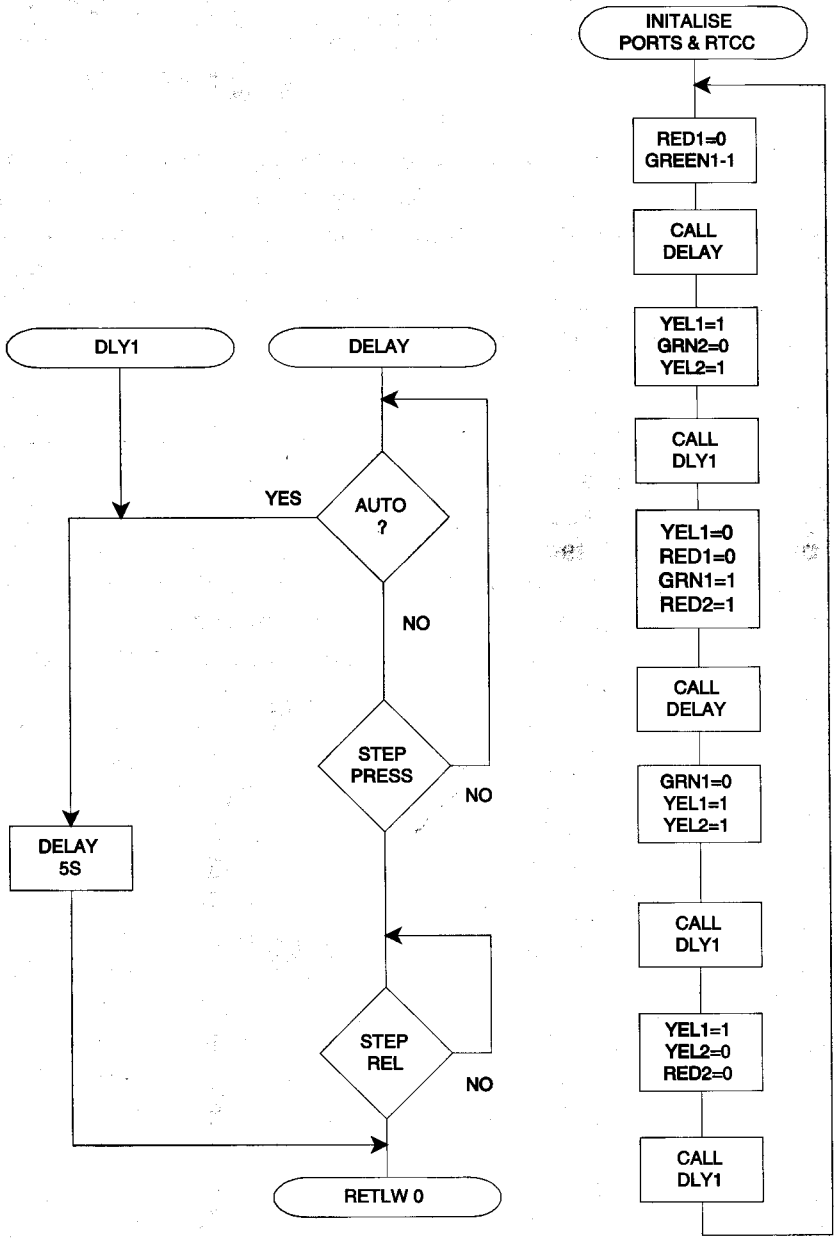


FIGURE 7

Sine Wave Generator

Program - SINE.ASM **16Cxx**

Generation of sine waves can be achieved by a PIC in a number of ways, the commonest being PWM or resistor ladder. This program uses the resistor ladder method with a lookup table to generate the sine waves.

The routine takes the form of two looped blocks, one forming a large time delay, the other a lookup table to convert a number from 0 - 15 to an 8 bit value. This value is then sent to an R/2R ladder network, forming the D/A conversion. If no lookup table is used and a value from 0 - 255 is sent to the output, a triangle wave form is produced. The lookup table produces values which will result in a sine wave being generated. Other shapes can be created by changing the values in the table. A 10nF capacitor is placed across the output of the network to help smooth the transition steps from one level to another as the output changes.

```
Lookup  addwf   pcl,f
        retlw   .128
        retlw   .176
        retlw   .218
        retlw   .244
        retlw   .255
        retlw   .244
        retlw   .218
        retlw   .176
        retlw   .128
        retlw   .78
        retlw   .38
        retlw   .11
        retlw   .0
        retlw   .11
        retlw   .38
        retlw   .78
```

```
; ********************** MAIN PROGRAM ******************

Main
        movlw   freq1           ; load frequency
        addwf   acc1,f          ; add to temp storage & save
        btfsc   _c              ; test if overflow from addition
        incf    phase1,f        ; increment table pointer
        movfw   phase1          ; load it into w
        andlw   0fh             ; strip off >15
        movwf   phase1          ; save it for next time
        call    Lookup          ; get value from lookup table
        movwf   portb           ; output it to the R/2R ladder
        goto    Main            ; repeat the process
```

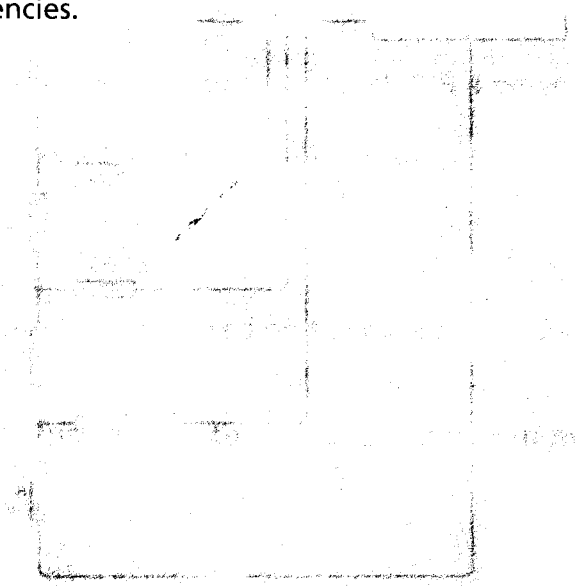
See figure 8 for circuit.

DTMF Signal Source

Program - DTMF.ASM **16Cxx**

This is an adaptation of the sine wave generator, but uses two loops to generate the dual tone multi frequency (DTMF). This design could possibly be used to generate tones to signal over the telephone network - see appropriate telecom specifications for further information.

The two delay loops run with the input value determining the frequency. The lookup table is used to convert the input counter value into a sine wave. The two sine wave values are then added together to produce the DTMF signal. The lookup table has a lower set of values than the sine wave generator as the summation of the two maximum table values must not exceed 255. This must be remembered if the software is modified to produce 3 or 4 simultaneous frequencies.



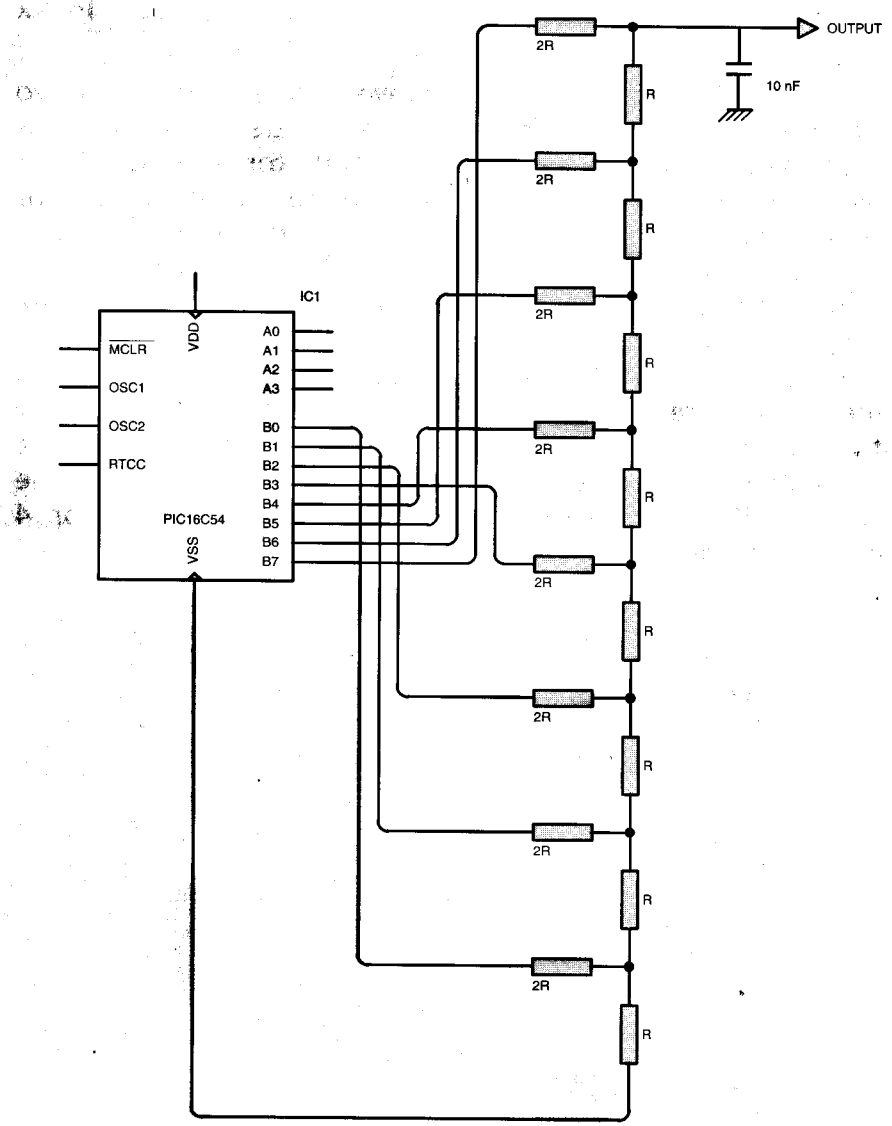


FIGURE 8

BCD Switch Multiplexer

Program - BCD.ASM **16Cxx**

This routine reads the input from 4 BCD edge switches, multiplexed into one port. The delay routine is necessary to enable the diode OR to turn on correctly and settling of the port input voltage. The software was written for a 16C64 and uses port E to drive the common lines of the switches. This can be modified to run on any other PIC or add more switches.

```
main    clrf    porte       ; start with a clear output
        bsf     switch13    ; turn on switch 1 & 3 power
        call    delay       ; allow time for voltages to settle
        movfw   portb       ; get switch settings
        movwf   R0          ; store in R0
        bcf     switch13
        nop
        bsf     switch24    ; turn on switch 2 & 4 power
        call    delay
        movfw   portb       ; get switch settings
        movwf   R1          ; Store in R1
        bcf     switch24
        nop

        ; now do any routines based on the data received in R0 -
          R1 registers
```

The routine can be expanded up to 6 or 8 switches with minimal modification to either software or hardware.

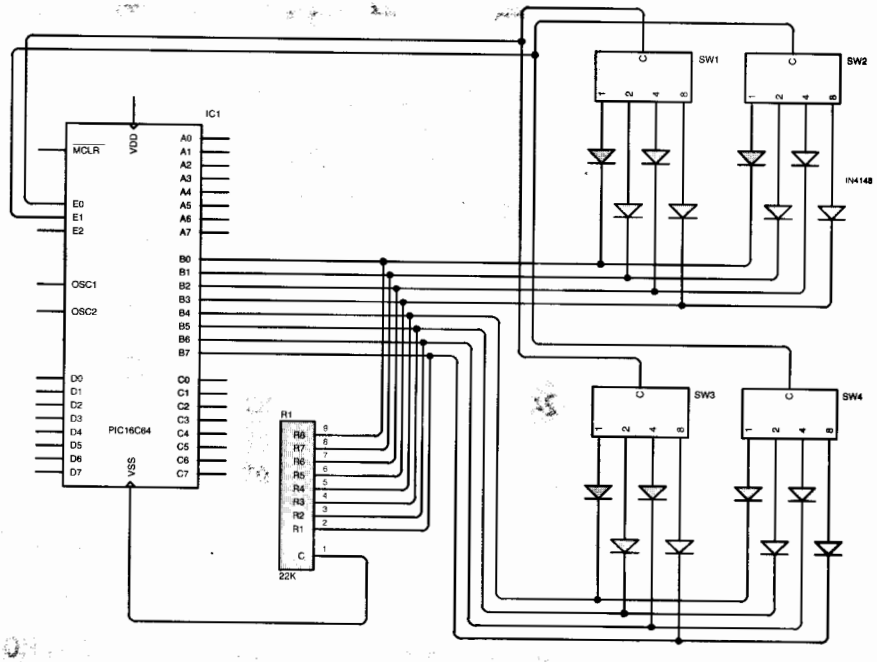


FIGURE 9

Car Left/Right/Hazard Timer

Program - INDI.ASM **16C54**

This is a simple timing application to flash the indicators on a car. An acoustic feedback is employed to inform the driver that the indicators are operating. A hazard warning is also incorporated to comply with vehicle legislation. The flash rate complies with standard automotive specifications for visual signalling systems - EG Guidelines 76/756/EWG (90 pulses per minute 50:50 mark space).

The main part of the program polls the three inputs from the left, right and hazard switches and branches off to the time delay routines after turning on the appropriate lights. The first delay called (DELAY2) toggles the buzzer line to provide audible indication. The lights are then turned off and the second delay (DELAY) completes the on/off mark space requirement with no sound output. This sequence continues until the switch is turned off.

The code could easily be modified to provide audible output only during a hazard warning or if the turn indicators are left on for more than 30 seconds. If the code is converted to run on a 16C71, then the analog inputs could be used to measure the current flowing in each bulb and detect lamp failure. This in turn could either double the flash rate to alert the driver or send the appropriate information to the vehicle diagnostics.

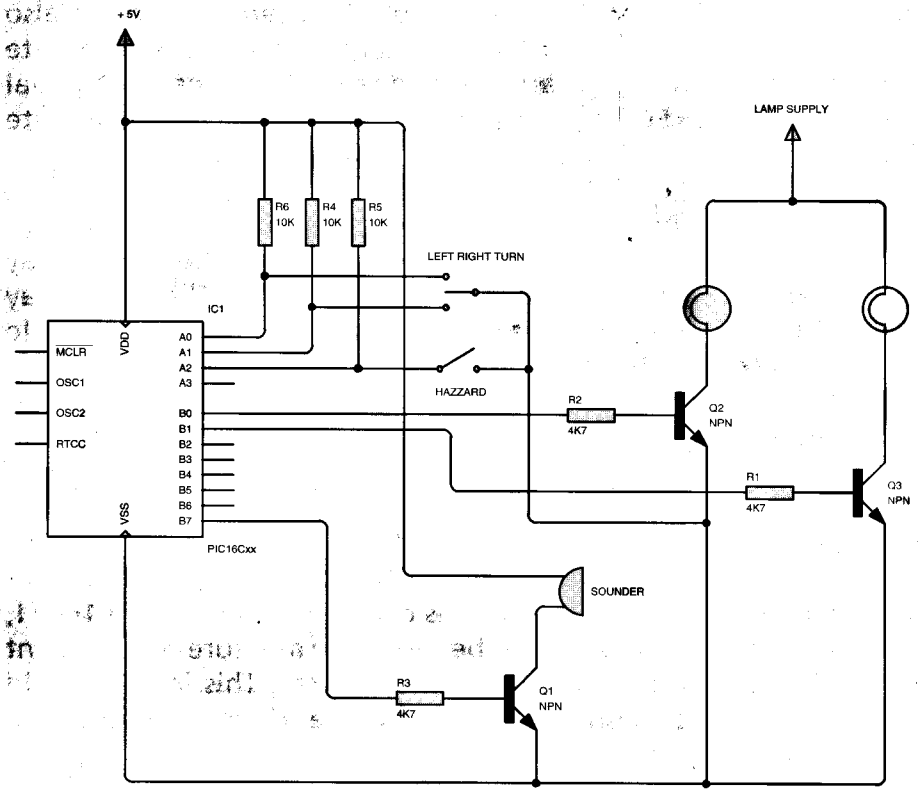


FIGURE 10

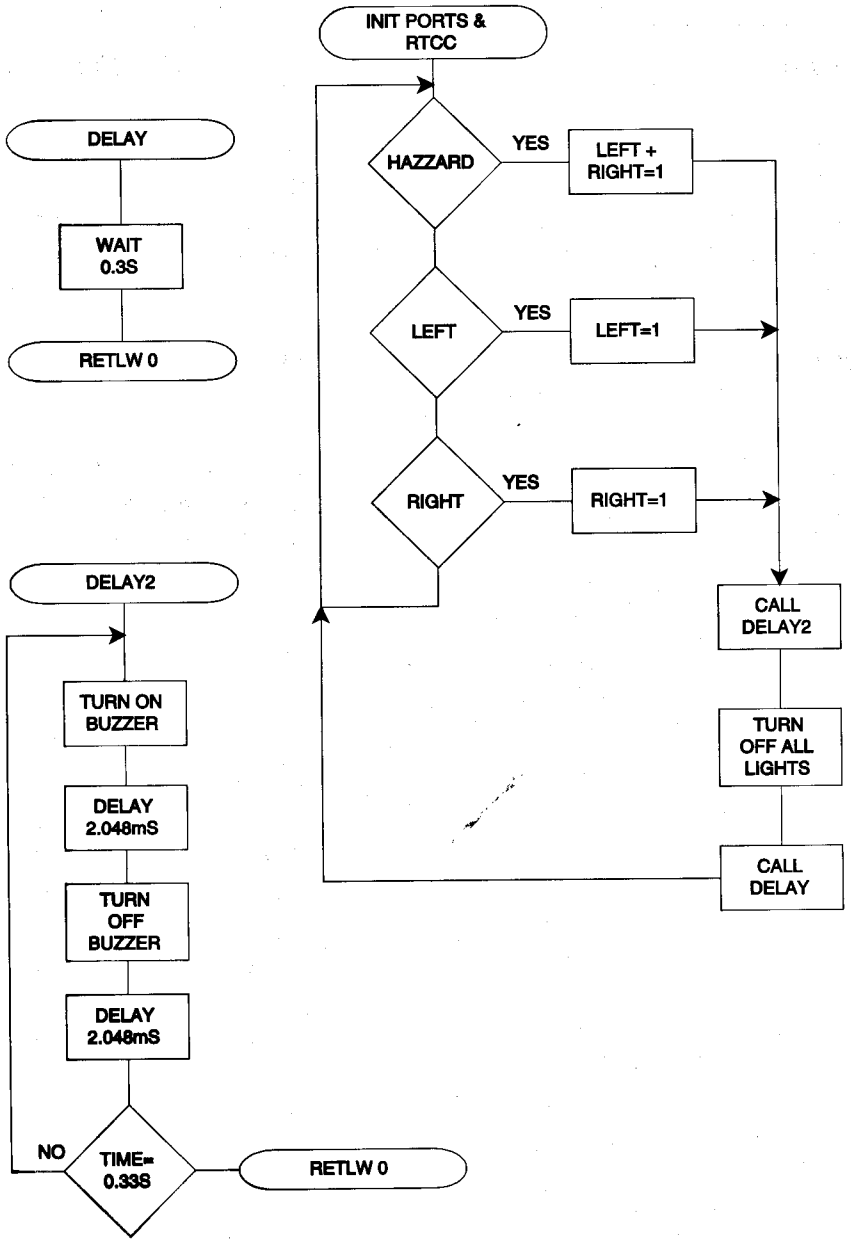


FIGURE 11

Pedestrian Crossing Simulator

Program - PED.ASM **16Cxx**

This code is similar to the traffic light sequencer in that it follows the sequence of change from one set of lights to another. However, the addition of sound and flash operation enables this design to become a fully working product with minimal change.

The sounder shown in the diagram is a small loudspeaker. The warning tone frequency is set within the software.

Modifications to the design could be to include a delay between cycles to allow sensible traffic flow or the provision of a vehicle sensor to allow faster response times when no vehicles are present.

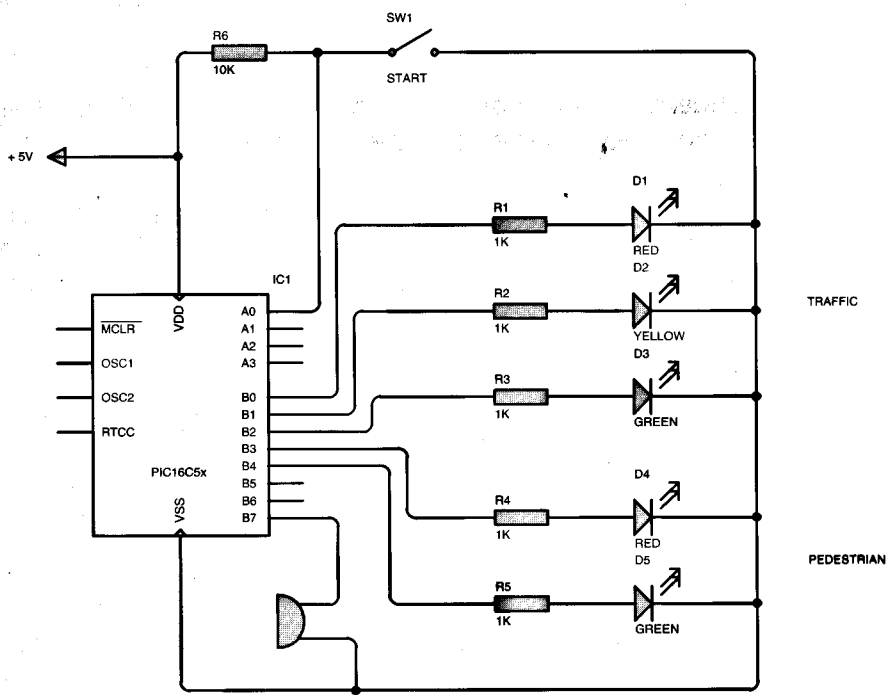


FIGURE 12

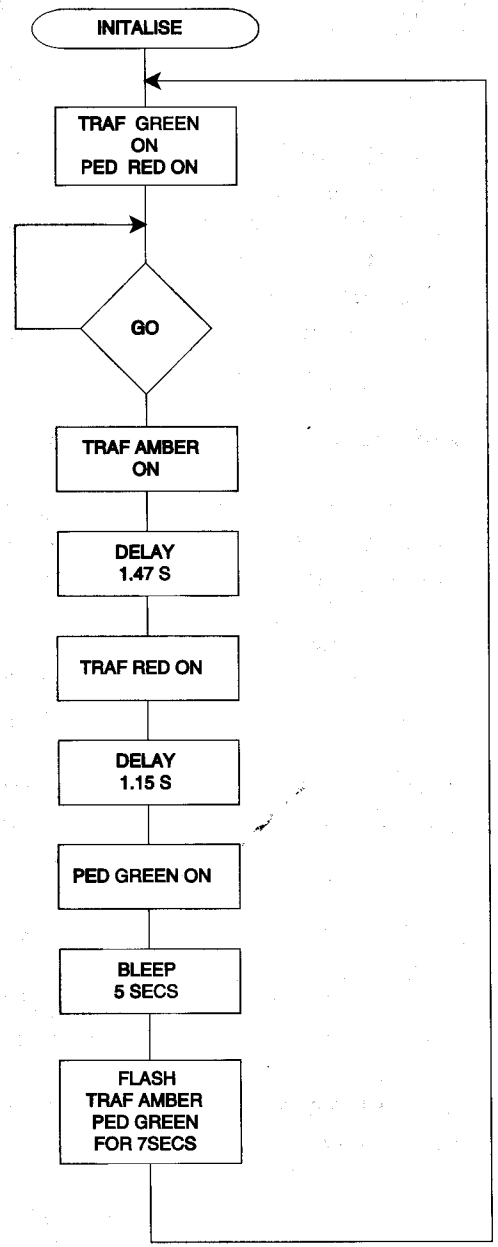


FIGURE 13

RS232 Routines for 16C73/74

Program - RS232.ASM **16C73/74**

This example is intended to show basic asynchronous transmission and reception of data via the USART. There are a number of registers associated with RS232 communications to set up the baud rate and indicate the send/receive status. This application uses the AD232A or MAX232A chip as the interface to the outside world and needs 5 x 100nF external capacitors to operate.

The registers associated with the comms side of the C73/74 are:-

TXREG	to hold the data to be transmitted
RCREG	to hold the data received from the sending end
TXSTA	a status and control register for transmission
RCSTA	a status and control register for reception
SPBRG	the register holding the baud rate
PIR1	interrupt flag register
PIE1	interrupt enable register

The number to be stored in the SPBRG register to represent a set baud rate needs to be calculated from the master oscillator frequency thus:-

For low baud rates - set TXSTA, BRGH = 0

$$\text{SPBRG value} = \left\{ \frac{\text{clock frequency}}{\text{baud rate x 64}} \right\} - 1$$

For high baud rates or if the ratio from the above calculation produces a value greater than 256 - set TXSTA, BRGH = 1

$$\text{SPBRG value} = \left\{ \frac{\text{clock frequency}}{\text{baud rate x 16}} \right\} - 1$$

For example - a 4MHz oscillator and a required baud rate of 4800

needs a value of 12 or 51 to be loaded into the SPBRG register, not forgetting to set the BRGH bit accordingly.

To receive a character after the appropriate registers have been configured, simply test for a logic 1 on the RCIF flag in the PIR1 register and then read the data in the RCREG.

To transmit a character after setting the registers, load the character to be sent into the TXREG. This automatically sends the character out at the required baud rate and the end of transmission is indicated by the TRMT bit being set in the TXSTA register.

Setting of non standard baud rates and bit formats is easily undertaken and information on this can be found in the data sheet.

The code example RS232.ASM simply looks for a # symbol received from a terminal and echoes back the number of times it is received. When the 10 th # is received, a BELL character is returned and the count register is reset. The program has no practical application other than to illustrate and test the comms function.

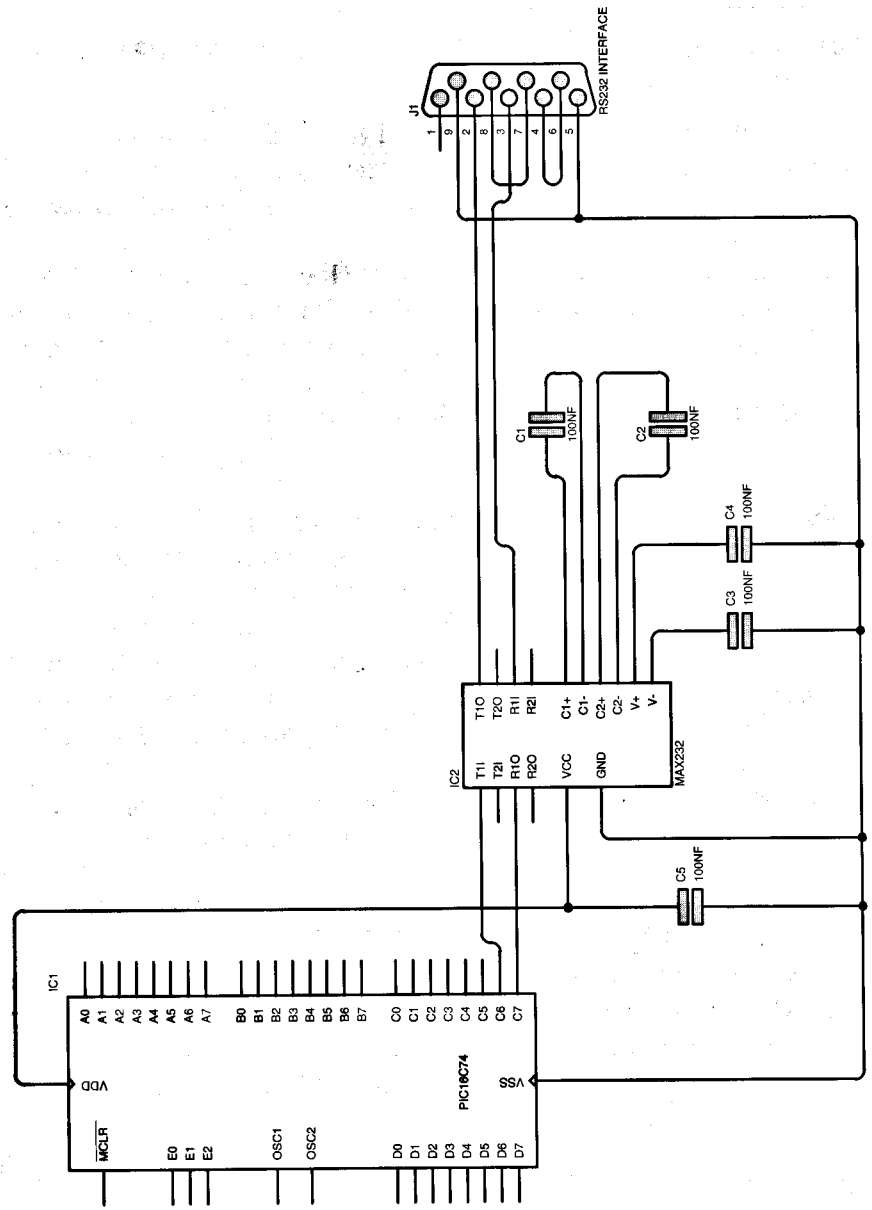
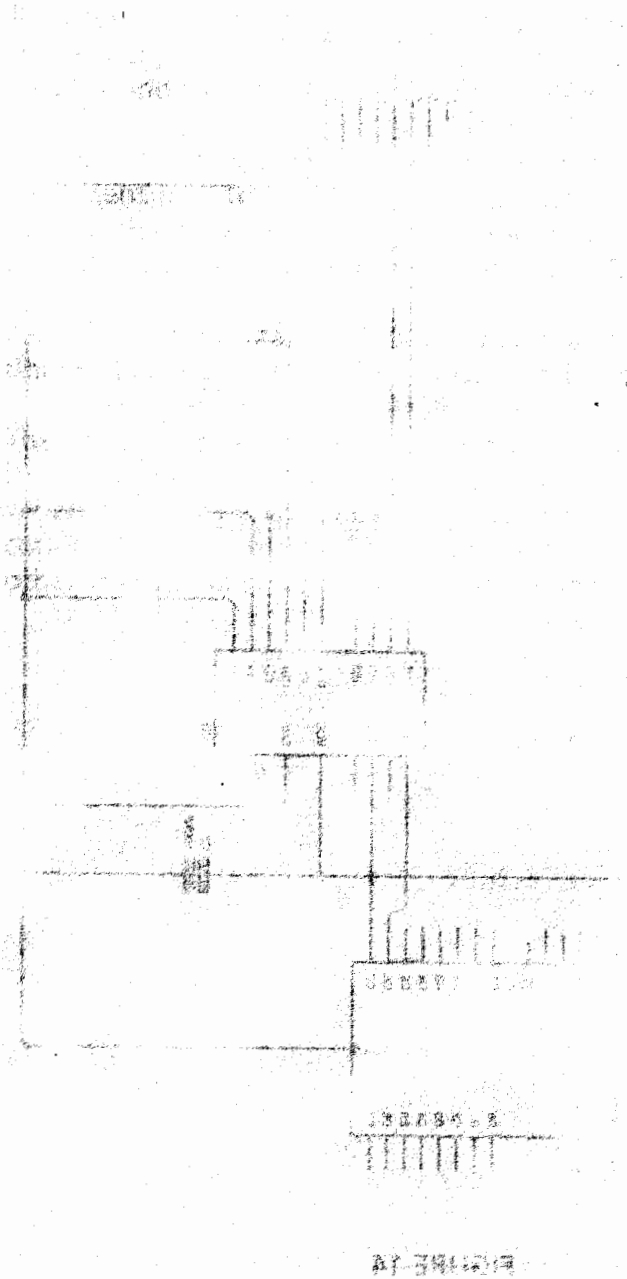


FIGURE 14



TSM4000 LED Display Driver

Program - TSM4000.ASM **16Cxx**

The TSM4000A display is ideal for connection to the PIC. The 3 wire interface saves vast amounts of time and program area within the processor in comparison to multiplexed types.

The PIC interface to the TSM4000A uses clock, data and enable lines and requires a total of 36 clock pulses to latch data to the display. The first clock bit starts off the internal circuitry, followed by 32 data bits and 3 further clock pulses to latch the data out to the led segments, or in the case of the 5480, the output bits are set accordingly.

Data to be output is stored in registers Digit1-4 and a conversion table (Lookup) is used to change the BCD data to a suitable 7 segment format. Decimal points can be set in software prior to outputting the data as can the two external leds by setting the relevant bits in the associated Digit register. The external leds could be used in an alarm clock application to indicate the alarm set function without having to use another 2 pins on the PIC.

This application demonstrates a simple up counter which rolls over at 255. A possible application for this code - with the additional use of the RTCC would be for a timer or clock. The flowchart shows the function of this code example and also a possible count down timer.

The serial data transfer principal can be applied to any 2 or 3 wire interface circuits such as shift registers for I/O expansion (74HC595, 74HC165) or display drivers for LCD's, Vacuum Fluorescent and LED (Micrel MIC8030, MIC80937 and MM5450).

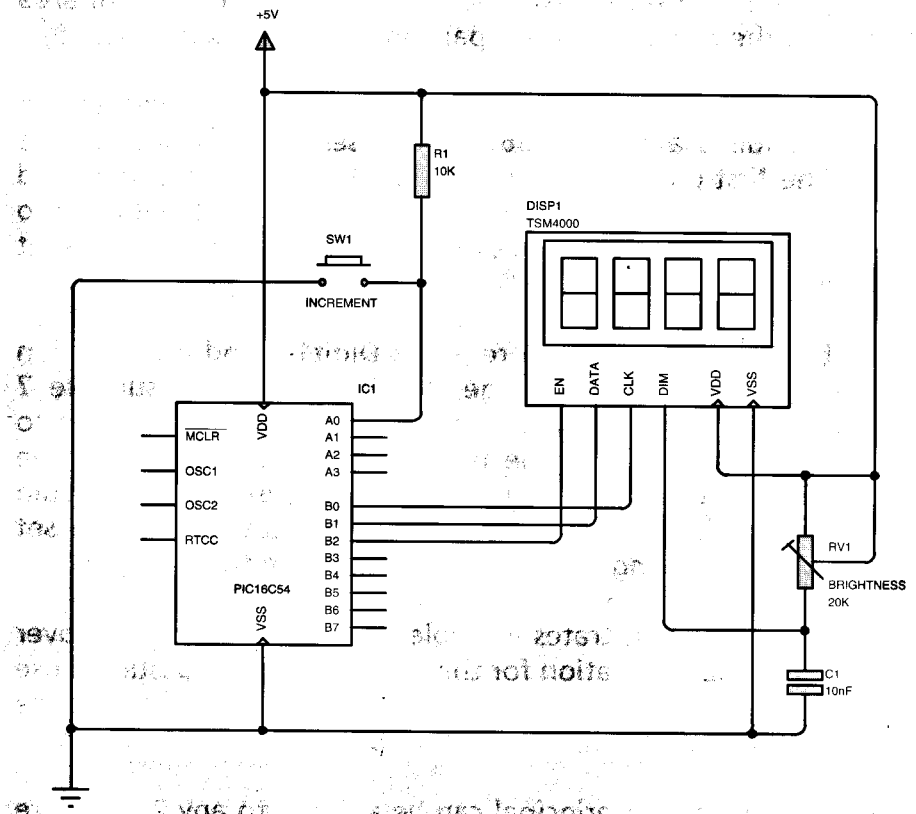
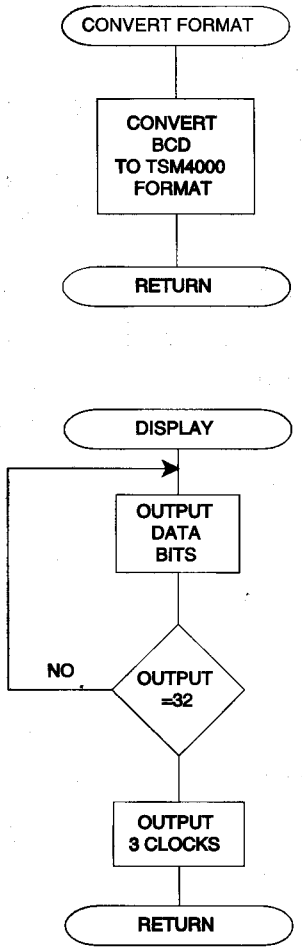


FIGURE 15



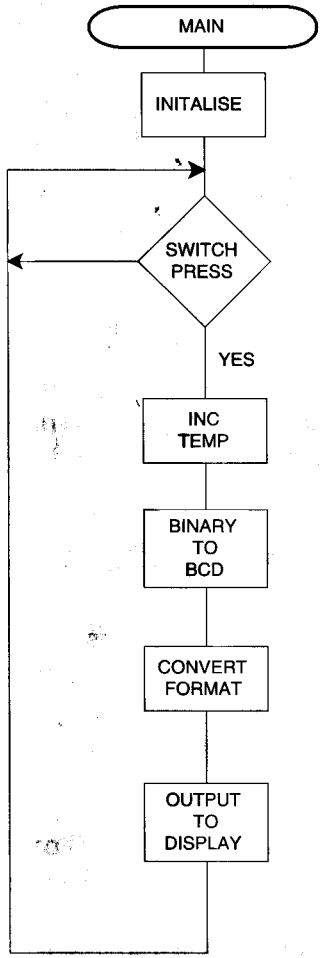


FIGURE 16

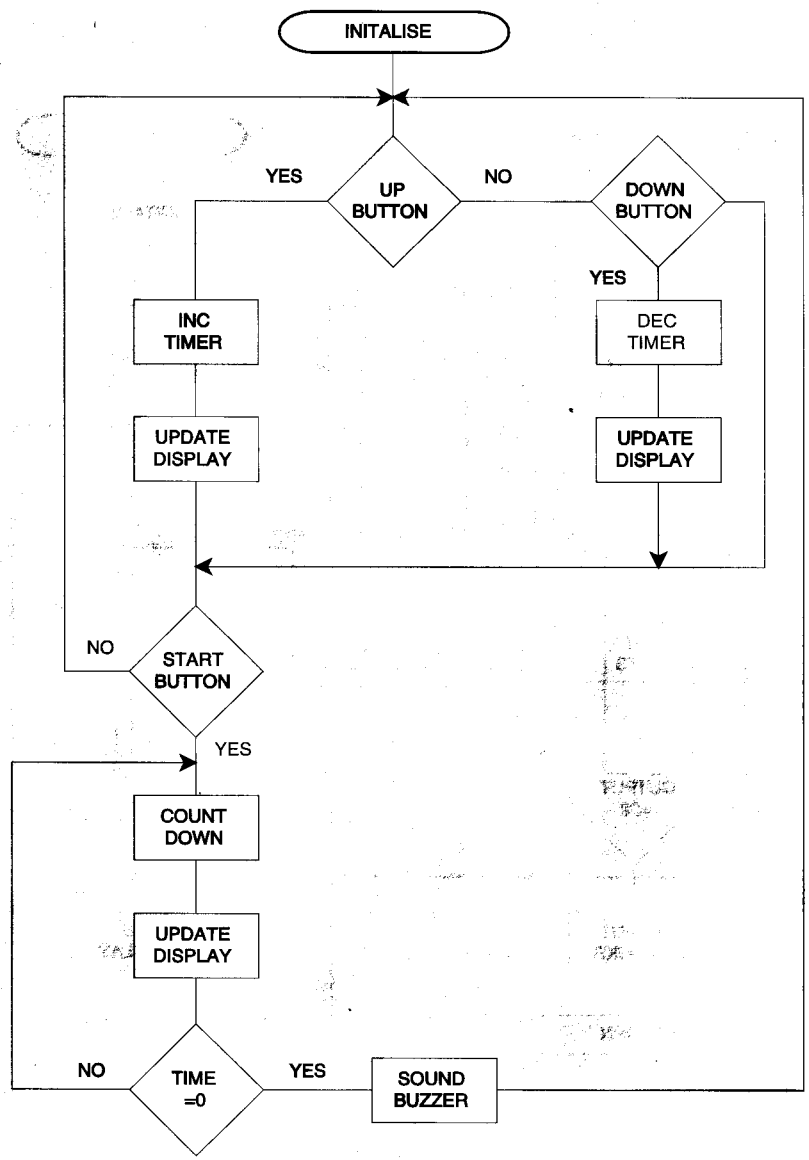


FIGURE 17

PIR Security Centre

Program - SEC_CENT.ASM **16Cxx (28 pin)**

This security centre is a control panel for external PIR sensors warning of a possible intruder outside a building, and at night turn on outside lights. The unit is not intended or designed as a burglar alarm panel to give indication of an actual break in, but merely to indicate there is something moving outside the premises.

There are 4 control zones which can be individually locked out, a manual over ride for the lights and a buzzer disable facility. The design is modular and comprises control box, daylight sensor and light switching relay.

Care should be taken when placing PIR detectors as false triggering can occur during daylight if trees are in the detection range - especially on sunny days. See the PIR manufacturers information for further information.

Hardware

The circuit is shown in fig 18.

Input protection is used on the sensor inputs as the original PIR sensors switched 15v into the security centre when activated. This circuitry should be modified for other voltages or switching modes (contact closure would need some form of pull up on the PIC input).

The keyboard contacts pull the PIC inputs to 0v and have an external 10k resistor package pulling the lines to the 5v rail.

The day/night sensor input can be fed from any light sensor circuitry with lux level setting and inbuilt hysteresis. This circuitry should also have a slow response time to prevent un-nescesary day/night changeover. A possibility for an enhancement would be to use the 16C73 and feed a LDR into one of the analog pins to eliminate the external interface circuitry.

A standard regulated power supply based on 7805 and 7815 regulators was used on the original design, or a battery backup circuit could be incorporated. The PIC and its associated circuitry need minimal current, but take into consideration the load requirements for the PIR sensors and the relays. The grouping of inputs and outputs on the PIC was determined after the PCB was laid out and is one of the advantages of a PIC based design. If required, rejig the I/O to suit your PCB layout and modify the source code accordingly.

The DIP switch selects an on time for the sounder or night lights of 1, 2, 4 or 8 minutes - the basic timing can be modified in software if required.

Software

The program runs with the watchdog enabled to allow safe recovery in the event of a latchup. The watchdog bit is tested on reset to determine a warm or cold reset. If a warm reset, then the program continues in the main loop. If a cold start, then a delay of 45 seconds is initiated to allow time for the PIR sensor output to settle. During this time, the alarm led flashes to indicate the 'warm up' stage. The dip switches are then read to obtain the alarm / light on time.

To add security to the software, the I/O port direction registers are refreshed on each pass through the scan routine. The main loop starts by testing the zone lockout switches, alarm enable, manual over-ride and zone alarm inputs. If a zone is activated or locked out, the appropriate leds are set or cleared followed by the buzzer / light output.

The software follows a logical and structured path and is both easy to follow and modify.

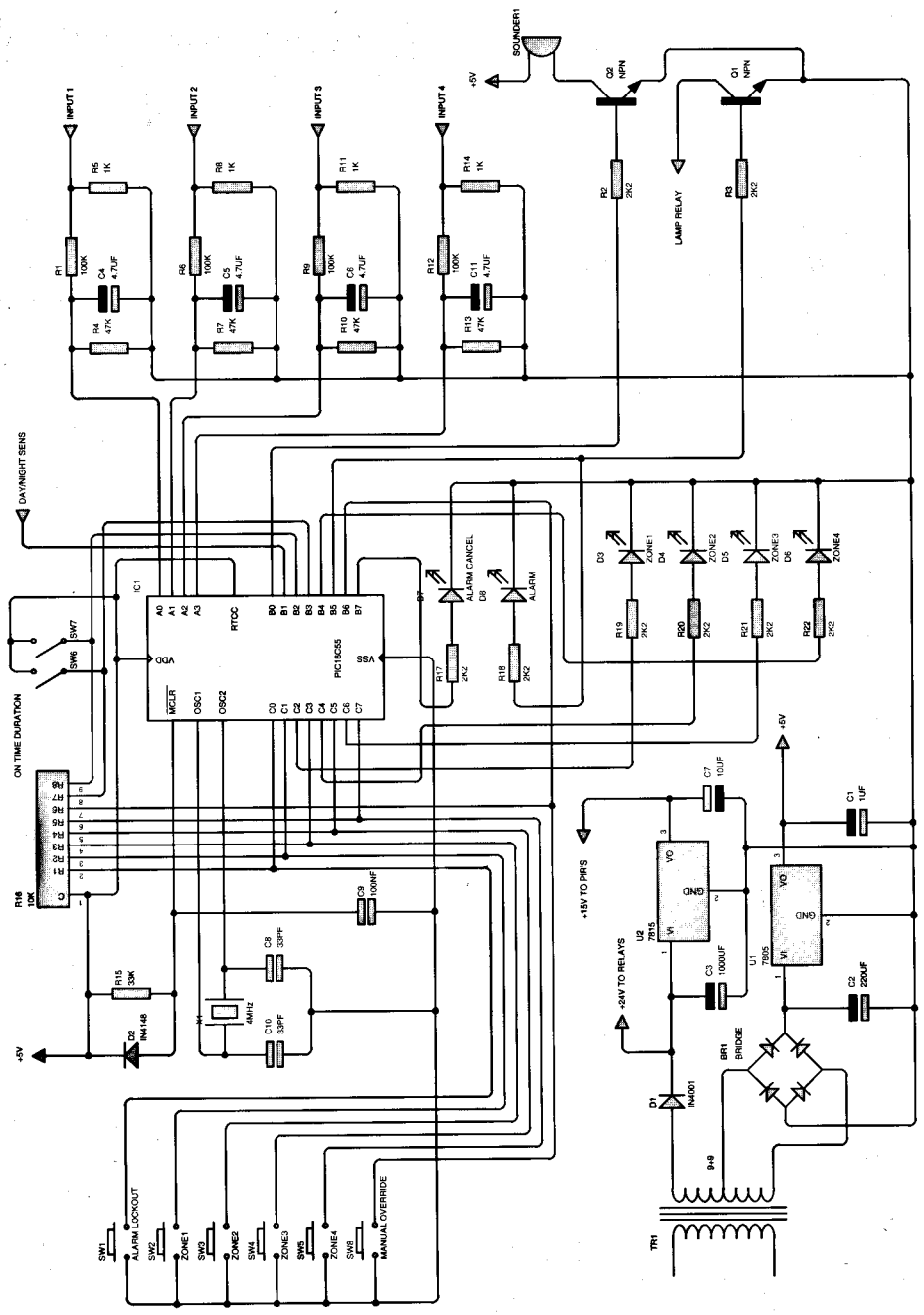


FIGURE 18

Square Wave Tone Generator

Program - SQUARE.ASM **16Cxx**

This square wave generator is one of the simplest designs as an alternative to a 555 timer. How many times have you needed a simple yet accurate frequency source but found the 555 data sheet calculations daunting.

This program is based around a simple time delay subroutine which is controlled by a value in the W register. A test is made on the 3rd bit of the RTCC, which gives a (4x32uS) 128uS time period. In this example, a min to max frequency of 163Hz to 3.9KHz is achieved. By changing the test bit within the RTCC, different frequency ranges can be produced.

If a 16C7x device is used, an analog input can be used as the value to load into the time delay loop and hence produce a variable frequency as opposed to one set by dip switches.

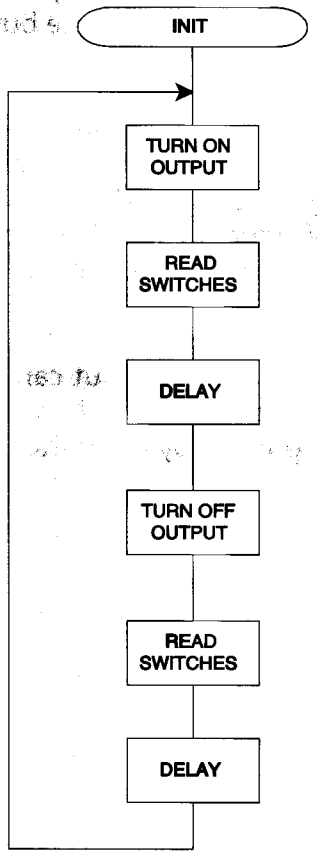


FIGURE 19

Radio Control System

Program - RADIO_TX.ASM & RADIO_RX.ASM **16Cxx**

This radio release system has been successfully used for Clay Pigeon Trap release up to 200 metres range. The transmitter and receiver modules are manufactured by a number of companies including Radiometrix and have been DTI approved. This reduces development costs to an affordable level. This software version is for a 3 channel system and can easily be modified to operate on a single or 8 channel system.

Transmitter

The transmitter is powered by a 9 volt battery and the PIC is placed in the sleep condition to save power. This is further assisted by using the Holtek HT1050 regulator which consumes micro amps of quiescent current. The PIC is woken up by closure of a single switch contact, which in turn pulls the reset line low.

The reset condition wakes up the PIC where a test is made to find out which switch has been depressed. The appropriate code is then transmitted via the transistor level shifter. The level shift is necessary to ensure maximum modulation as the receiver needs all it can get. A transmission burst of 30 blocks of 8 bit code are sent to enable the receiver to lock, decode and check the incoming signal. If after this burst the button is still depressed further bursts are sent.

When the button is released, the PIC places the ports in the correct condition for wakeup and then enters the sleep state. The watchdog is disabled to allow the PIC to remain in a sleep condition until a keypress occurs.

Receiver

The Receiver takes the output from the radio module and attempts to decode the incoming data. As the receiver output is random noise in the absence of a carrier, the software keeps looking for a pattern match. The watchdog is enabled to ensure no latchup of

driven outputs. Upon reception of a valid transmission ie a minimum of 4 consecutive pattern matches, the valid code is then compared with the preset function codes. If a match is found, the appropriate output is switched. The software can be modified to provide latched or momentary output functions.

Another possible modification to the software is to use the carrier detect line from the receiver to activate the decode software. This has been tried and works, however range is reduced as valid data is present from the receiver output before the signal strength is high enough to raise the carrier detect line. This modification could be used for short range - low quiescent current applications with PIC's having interrupt capability.

The release system has performed well over a 3 year period. The PIC software and hardware has remained faultless but care should be made in the choice of the radio modules and their positioning within their enclosure as they can drift in frequency for no apparent reason, so check the manufacturers specifications before, during and after construction.

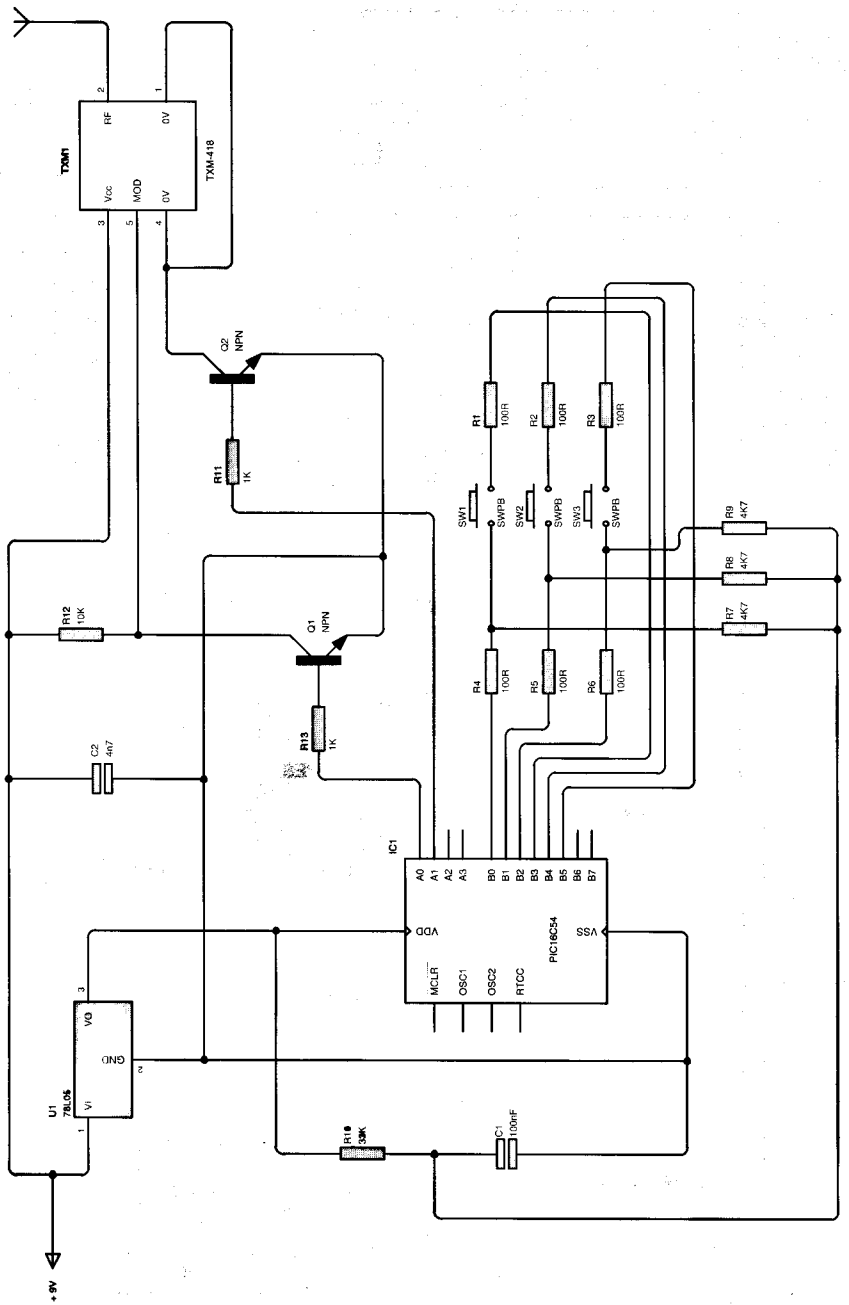


FIGURE 20

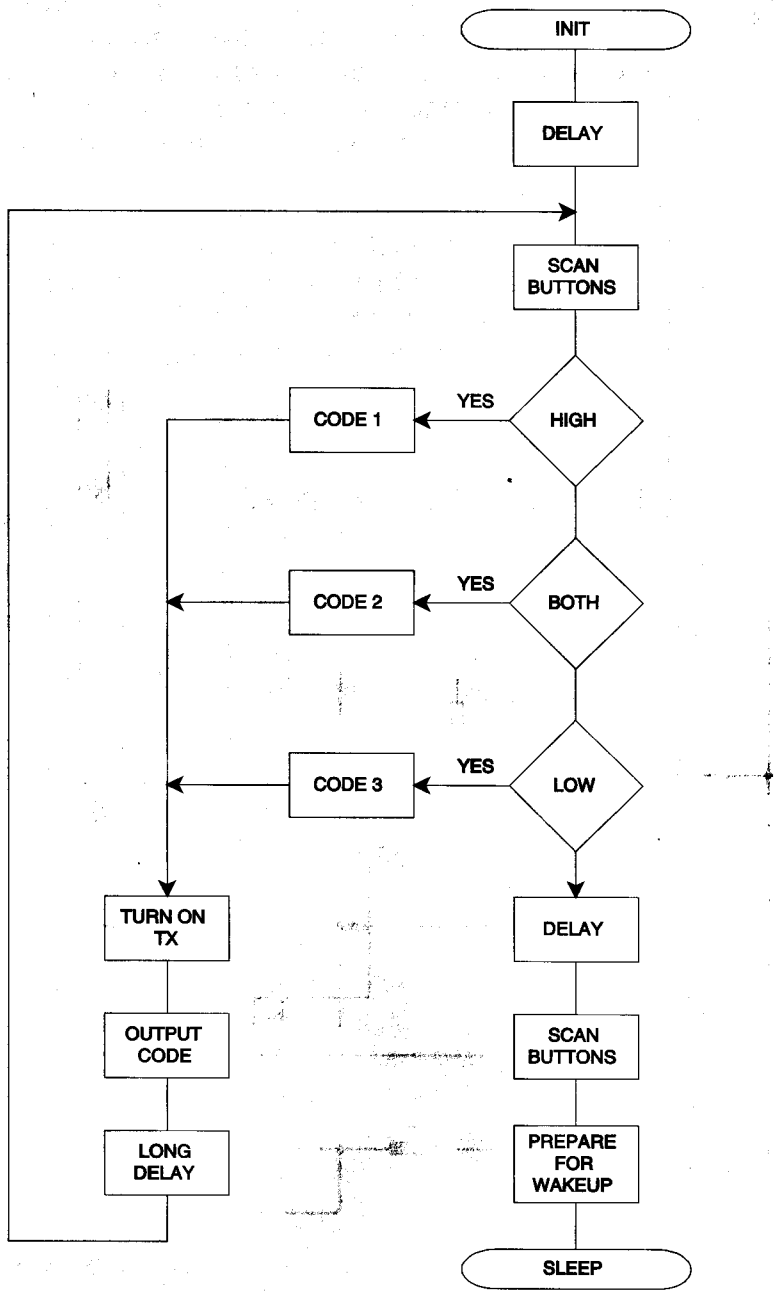


FIGURE 21

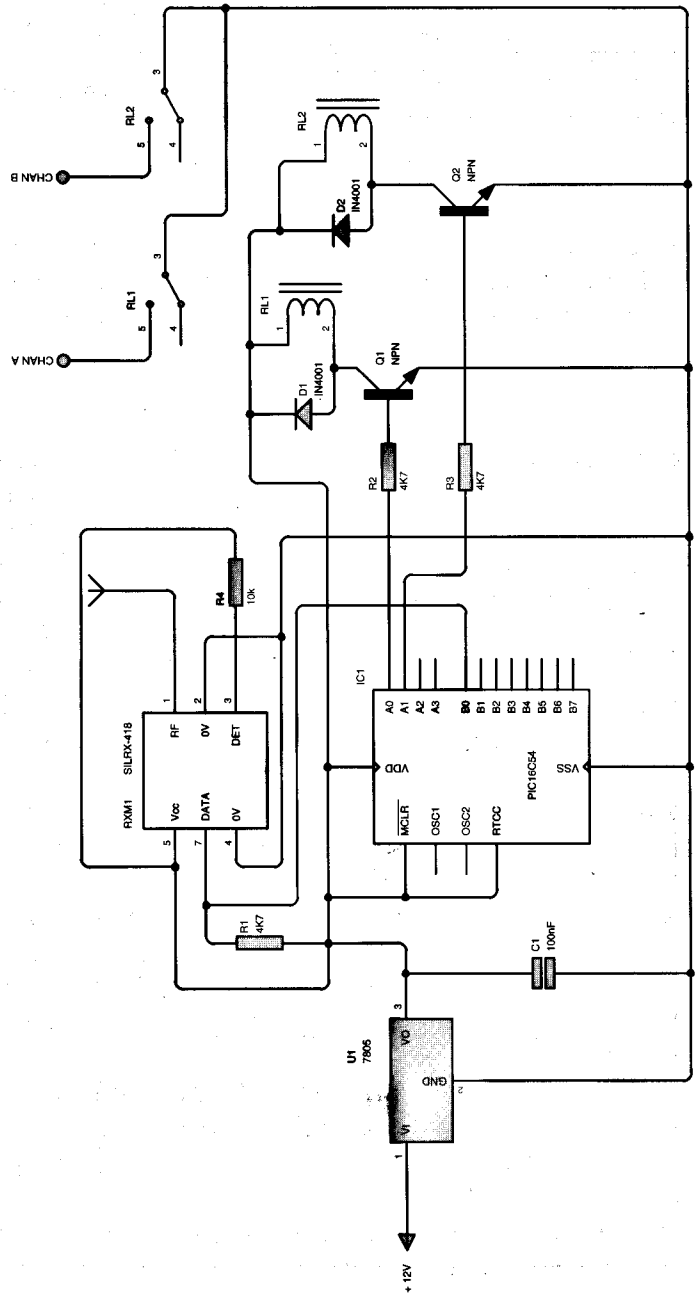


FIGURE 22

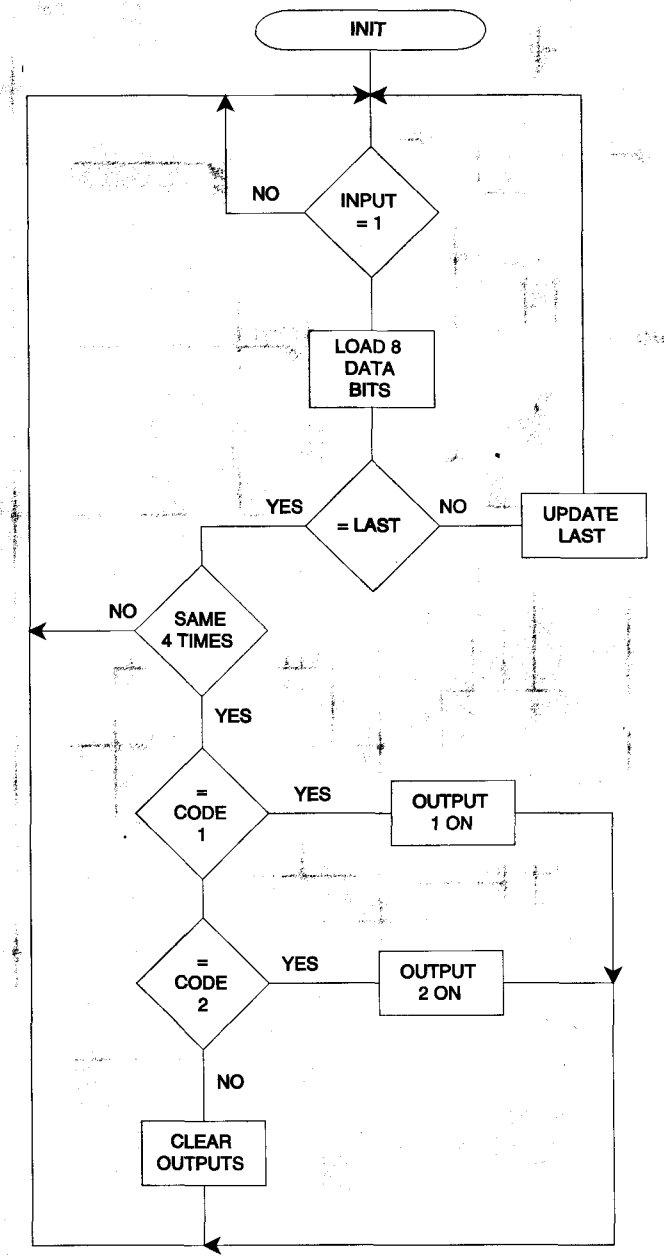


FIGURE 23

Seat Positioner

Program - SEAT1.ASM **16Cxx**

This seat positioner is intended for use with a child's car seat which has a manual rotary position adjuster. It will allow the driver or passenger to place the child in the correct position for awake or sleep without endangering road safety in the process and without stopping the vehicle. For those with back problems it also prevents excessive strain and twisting to the body whilst changing the seat angle.

On the prototype, the motor and control module were located to the rear base of the seat with an extension bar from the motor / gearbox to the existing threaded drive. The motor chosen was a stepper with a 50:1 gearbox to give torque while keeping the motor noise to a minimum. This gives an upright to recline time of circa 2 minutes. This time means that there is no noise or rapid movement to disturb the child. A magnet was fitted to the slider and Hall effect limit sensors to the end support blocks to detect end of position. On reaching the end of track, the motor stops.

Care must be made to ensure that the addition of the motorised module does not effected the safety of the seat.

The unit connects via a 4 way cable and connector to the vehicles 12 volt electrical system - 2 wires for the power and the other 2 for the up / down switch - a single pole switch which can be dashboard mounted.

An enhancement to the existing design could be a bicolour indicator to let the driver know the direction of travel and the current position.

The main program reads the inputs from the end of track sensors and the forward/reverse switch and uses a lookup table to determine which operation should be taken. The alternative is to test each input line and act accordingly. This latter method needs

care in writing as it is possible to miss certain combinations of inputs. The lookup table method is useful but note that the table length doubles for each input line added - 2 input 4 line, 3 input 8 line ..... 8 input 256 line!

If a servo motor is used, the software needs modification to remove the stepper motor drive sequence. Dynamic braking (short circuiting the motor) when the end of travel is reached could be employed to prevent over-run as could PWM control of the motor to give a ramp up and down of the motor speed. This design could equally be used for satellite dish control, greenhouse vent opening or central heating valve control.

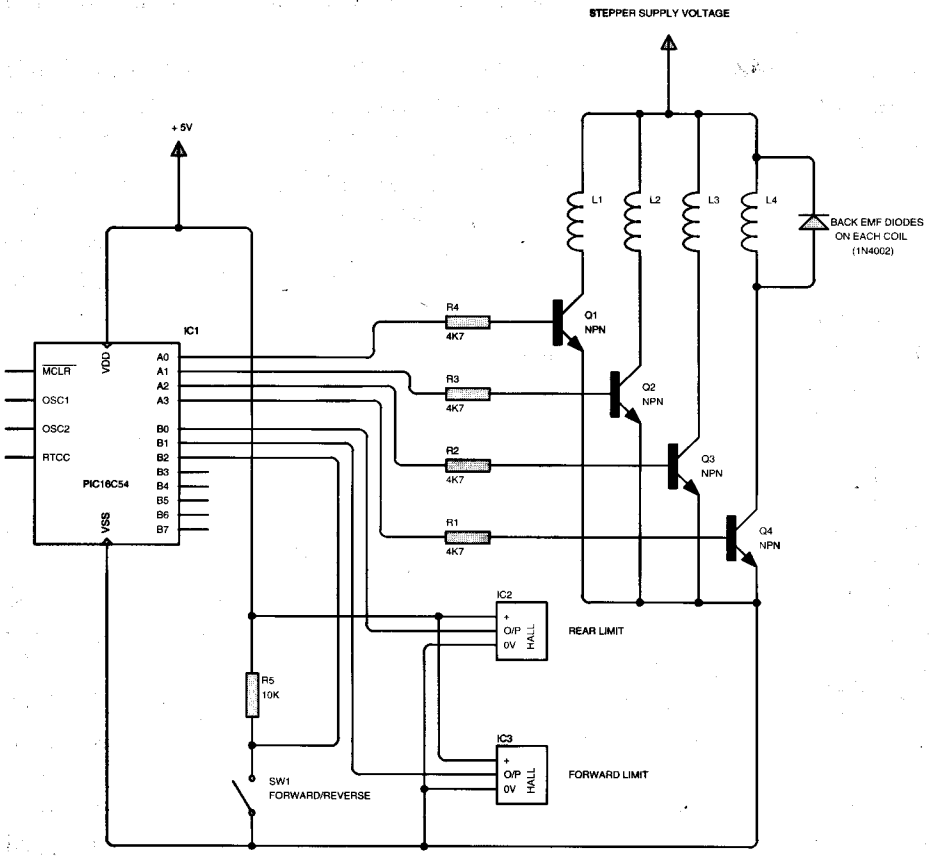


FIGURE 24

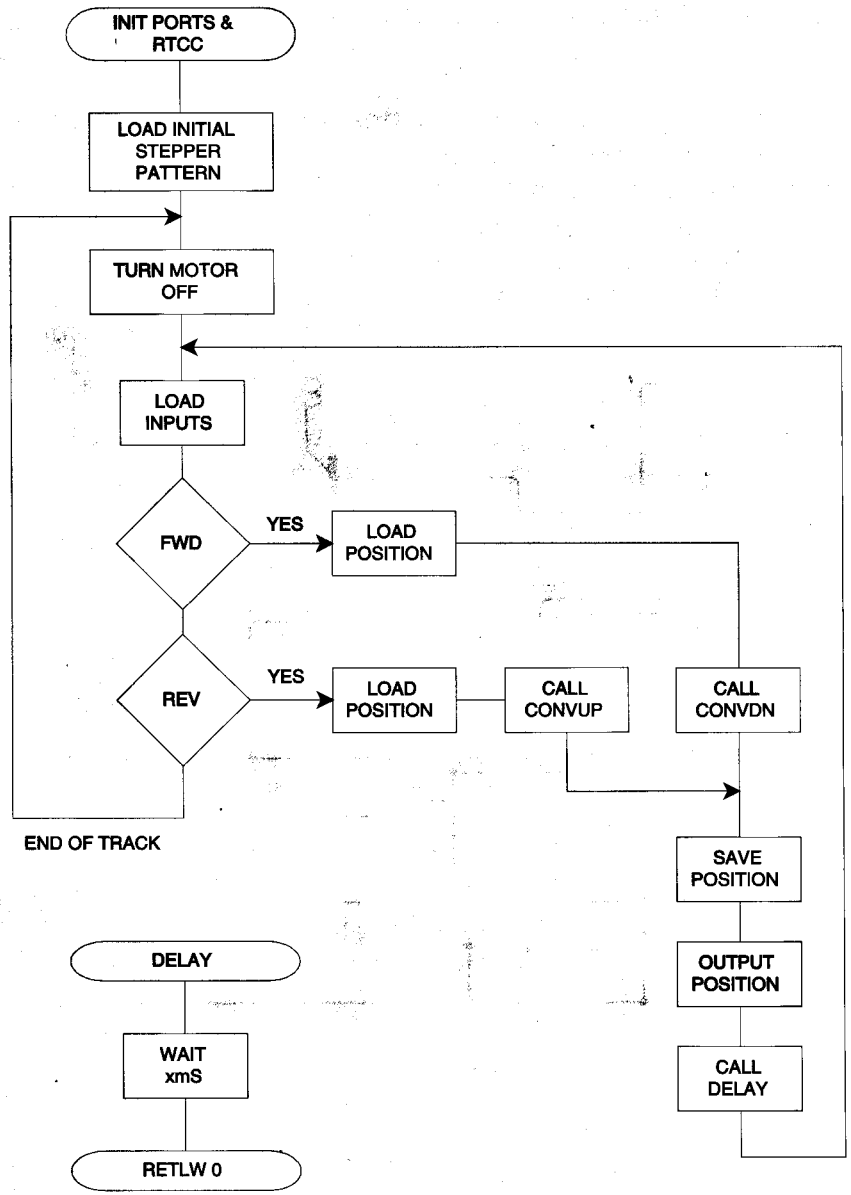


FIGURE 25

Radar Speed Sensor Interface

Program - SPEED1.ASM **16Cxx**

Radar speed sensors rely on a Doppler frequency being produced when a vehicle approaches the sensor.

The Doppler frequency effect is best observed when standing on a railway station. The sound of the train as it passes by is higher as it approaches than when is going away. If both frequencies are measured and the difference divided by 2, the resulting value is proportional to the train speed. With a microwave system the reference frequency is stationary. The Doppler frequency is formed as a result of the vehicle speeding up the reflected frequency. Again the actual speed is proportional to half the Doppler frequency. The output Doppler signal from the microwave module is processed via some analog circuitry (not shown here) and presented to the PIC. In this design, any PIC with interrupt capability can be used.

At 10GHz, a frequency of 32Hz per mph is generated by the Doppler unit and this signal is fed into the A4 pin of the PIC. This pin has a Schmitt input and cleans up the received signal. The RTCC is preloaded with a value such that when it overflows, is equal to a slice of time. At a time period of 62.892 ms, 2 pulses in the mph_new register equal 1 mph. This value is then converted into an actual speed, in this example by simply dividing the value by 2.

The main program monitors the input from the sensor and increments the mph_new register on each positive edge. When the RTCC interrupt occurs, the speed conversion is performed and the value is then compared against a set limit. If exceeded, a warning output is given on the B0 pin .

Expansion of this could be to display the vehicle speed on large 7 segment display, activate a camera above a set speed or general purpose monitoring of traffic flow.

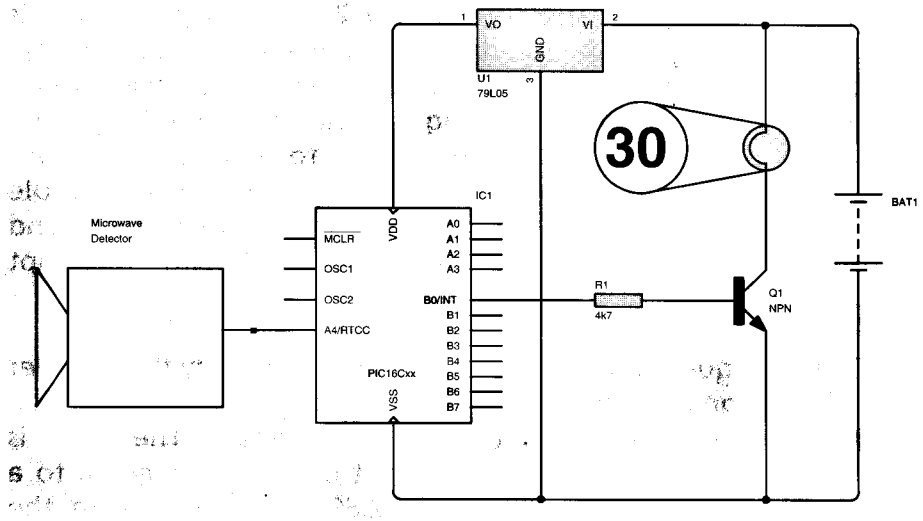
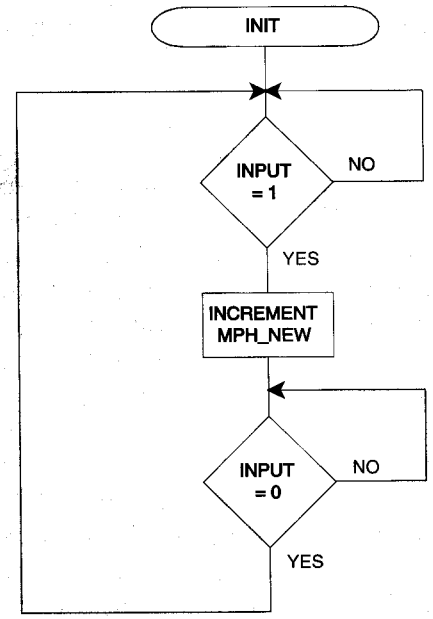


FIGURE 26



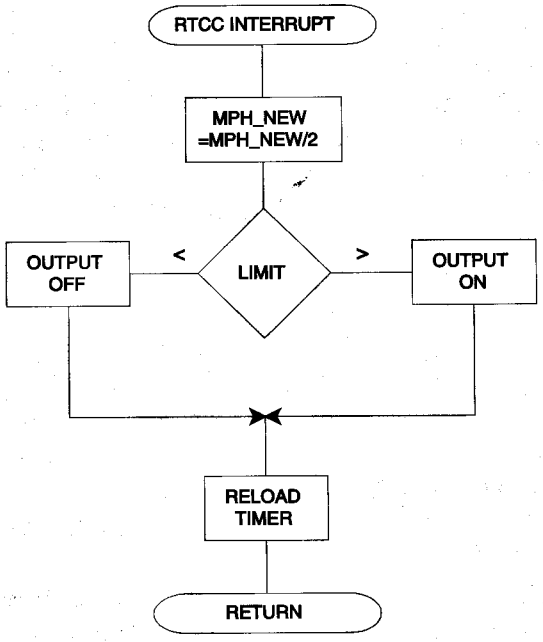


FIGURE 27

Rockwell GPS to PIC Interface

Program - GPS1.ASM **16C74**

Global Position Satellite receiver systems have progressed since their introduction in the 1980's. Some receiver designs as recent as 1993 used the Transputer with DSP software algorithms to extract the information received from the RF front end. The main turning point of miniaturisation and cost reduction appears to be around the Gulf War when it was discovered that the commercial GPS receivers were performing equally well alongside the military units, and at a fraction of the cost. This design is intended for those looking for a low cost interface to the Rockwell MicroTracker LP GPS Engine and uses a PIC16C74 to take the GPS data and present it in a format for the user to view. The PIC handles all the serial comms, sorting out and formatting of information and display driving functions and needs a minimal number of external components to function.

The MicroTracker LP has two data output modes - binary and NMEA. This design is based on the industry standard NMEA format at 4800 bps and interfaced to the serial port of the 16C74. If data manipulation is required on the GPS information, then the binary format is the better one to choose as there is a software overhead converting to and from the NMEA ASCII format. If the interface is only used to display and store data, the NMEA format is easier to work with. This application is intended purely to display information and does no calculation on information received.

Hardware

The heart of the design is the 16C74 microcontroller running at 4MHz (fig 28). An RS232 link interfaces to the GPS serial interface module and uses an Analog Devices AD232A or Maxim MAX232A driver chip to level shift to the PIC.

Other items needed for the PIC are a 78L05 regulator, a 4MHz resonator and a few passive components for decoupling and reset. Power is drawn from the GPS serial interface board via a connection

to the active antenna supply and is in the region of 15mA @12V. This is broken down into MAX232A - 12mA, LCD display - 2mA, PIC16C74 - 1mA. If the RS232 interface is removed to give a direct TTL connections between the MicroTracker and the PIC, a saving can be made on current and components.

A reset switch was added to the PIC to assist in software development but could be eliminated on the final design. If the GPS serial interface module from TDC is used in conjunction with this design, then set the dip switches on their board to 1-5 on, 6-8 off , this then sets NMEA mode at 4800 bps on power up.

The LCD module is the Hitachi LM041L or LM044L having 4 lines of 16 or 20 characters respectively and are interfaced in the 8 bit mode with full handshaking. This method of interfacing provides the fastest display update times but uses 11 I/O . If this design is transported to a 16C73 (28 pin), then the 4 bit write only approach can be taken with the display using only 6 I/O lines.

Software

The flowchart for this program (fig 29) shows the basic operation.

Following initalisation of the ports and other registers, the display is cleared and a sign on message sent. The default message formats of GGA and VTG are turned off and RMC turned on. The RMC message format includes Latitude, Longitude, Time, Date, Speed, Heading, Magnetic Variation and Magnetic Heading. Other message formats can be enabled easily and the relevant information extracted from the data string.

The RMC message contains the following data:- Start field, UTC Time, Data Valid, Latitude, Lat Dir, Longitude, Lon Dir, Speed, Heading, Date, Mag Var, Magnetic Direction, Checksum <CR><LF>. Each field is comma delimited and will typically look like:-

$GPRMC,234215.24,A,3339.686,N,11751.667,W,0.620,293.8,180595,14.0,E*79

Other message formats include information on altitude, number of

satellites in use, track made good and ground speed and can be enabled and set to broadcast at increments of 1 second - see Designers Guide on the Rockwell MicroTracker LP for full information.

The character input from the RS232 port is examined to look for the '$' (24h) symbol, signifying the start of the message string.

```
get_char
        btfss   pir1,rcif       ; test for incoming data
        goto    $-1
        movfw   rcreg
        xorlw   24h             ; test if $ symbol
        btfss   _z
        goto    get_char
```

Upon receipt of the '$', the characters are stored in the PIC's RAM. On receipt of an end of message (Carriage Return - 0Dh) , the software branches into the display mode. The information from the MicroTracker arrives at a minimum of 1 second intervals providing enough time for the display to be updated.

The display section of the program looks at the characters in memory and when a comma (2Ch) is identified - field delimiter, the message type is then determined from a lookup table and sent to the LCD display. Additional text messages are sent prior to the received information to clarify to the user what it is being displayed. At the end of the information processing, the program jumps back to look for the next '$' symbol.

Baud rate calculation for the PIC serial comms port is straight forward after transposing the formula in the data sheet and for low baud rates is:-

$$\text{Divide Ratio} = \left\{ \frac{\text{Clock Frequency}}{\text{Baud rate x 64}} \right\} - 1$$

which works out at 12 for 4800 bps and a 4MHz clock.

This value is loaded into the SPBRG register and is common for both transmit and receive.

```
movlw   .12             ; 4800 baud
movwf   spbrg
movlw   b'00100000'     ; async tx 8 bit
movwf   txsta
bcf     status,rp0      ; return to page 0
movlw   b'10010000'     ; async rx 8 bit
movwf   rcsta
```

Software Modifications

The current version of software does not look for a fix before starting to display information on the display. A modification could be made to examine the data from the MicroTracker and look for the valid signal character before passing the sign on message stage. The information received from the MicroTracker could be examined for a '.' symbol in addition to the ',' to truncate the data displayed at the decimal thus reducing the information presented to the end user. Additional messages could be enabled and stored in alternative locations for extracting information not found in a specific message.

Where next

If an external EEPROM is added to the PIC and connected via the inbuilt I^2C interface, data could be logged in the E^2 upon closure of a switch thus providing a positional logging and ident system. The use of an external E^2 could also be used to update the MicroTracker LP on power up with the last positional information, thus speeding up the TTFF (time to first fix).

As the 16C74 has 8 analog inputs, one could be utilised for monitoring the battery voltage via a simple potential divider and allerting the user to time remaining before total loss of supply.

Sources of additional information

Rockwell MicroTracker LP Designers Handbook - TDC 01 256 332800

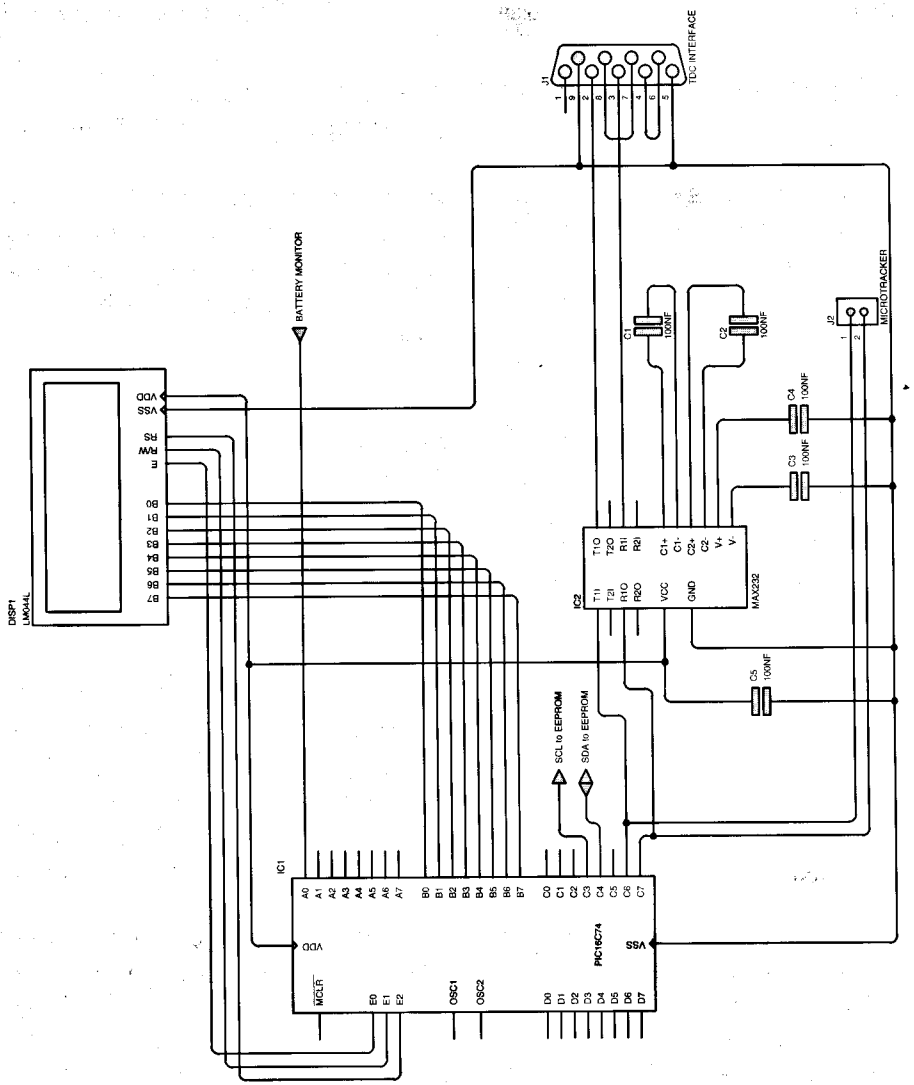


FIGURE 28

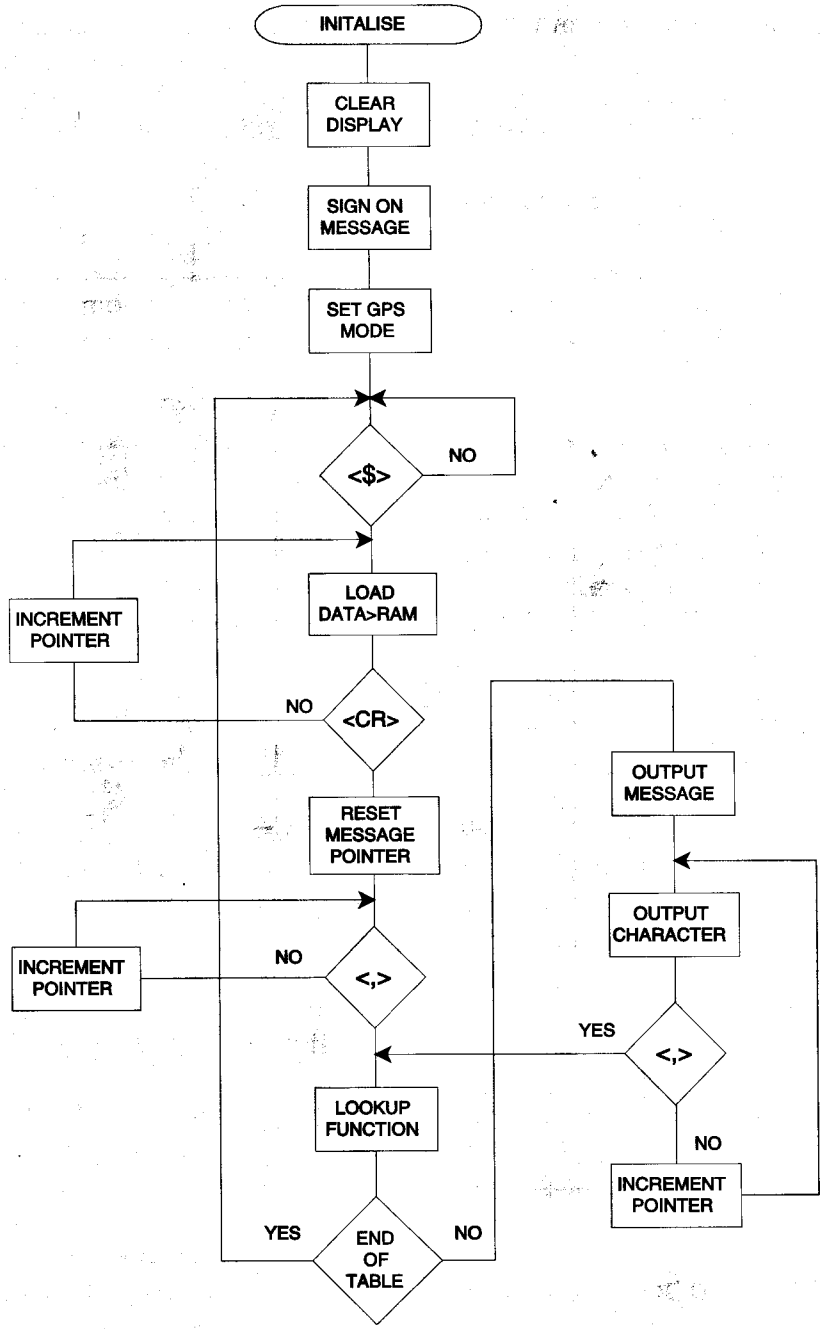


FIGURE 29

Watchdog Timer Code Example

Program - WDT.ASM **16Cxx**

This code example illustrates the use of the Watchdog Timer (WDT) within a program to prevent latchup.

The program begins with the WDT having a divide by 128 postscallar assigned to it. This gives a time-out period of approximately 2.3S (18mS x 128). The main loop of the program clears the WDT then tests for a button press and if valid, turns on both leds and loops around ad-infinitum. If the button is momentarily pressed, led 0 is extinguished and the PIC is placed in a loop without the WDT being cleared. After a period of 2.3S, a reset is forced and the PIC resets. This is then evident as both leds turn on.

This application is intended as a demonstration of the use of the WDT to recover the PIC in latchup conditions. The use of the WDT is a brute force method for recovery and two options are open for a more elegant solution. The first is simply to test the PD and TO bits in the Status register on entering the reset stage of a program and bypassing the initalisation stage if a WDT time-out has occurred. The use of the CLRWDT instruction should be limited to the main program loop wherever possible and then only used once. The only exception being delay subroutines where the delay time is greater than the watchdog time-out period.

For larger programs, the second solution is based on you knowing where you are within a program at any time. If the program is of the modular nature, then assign a number to each block of code. As the program moves from one section to another, load a spare register with that block number. When a WDT time-out occurs, the reset/initalisation code can be made to examine the block pointer register and branch directly into that or the previous block of code. This can reduce the reset and re-initalisation times on large programs.

RTCC/Interupt Based LED Flasher

Program - RTCC.ASM **16C6x/7x/8x**

The use of the PIC to flash a led is a bit of an overkill, but this example shows the use of the RTCC interrupt for timing loops.

Following initalisation of the ports, RTCC prescallar set to 64 and RTCC & Global interrupts enabled, the main program tests the contents of the ticker register to see if its value is 10 (100mS). If true then the led is toggled.

While this main loop is running, the RTCC is counting and forces an interrupt when an overflow occurs - 0ffh to 00h. The program jumps to the interrupt vector at 004h and the interrupt handler is invoked. This code initially tests the RTCC interrupt flag to ensure a valid interrupt and if so, then increments the ticker register ie 10mS has elapsed since the last interrupt. Following this, the RTCC register is pre-loaded with 100. This determines the time duration before the next interrupt

ie 256 - 100 = 156
1uS x 64 x 156 = 9.984mS

Finally the RTCC interrupt flag is cleared and the RETFIE returns the software to its previous operation.

If longer delays are needed, then use multiple registers to count the time period.
e.g. with a basic 10mS tick, an 8 bit register will give 2.55 seconds. Two registers (16 bits) will give 10.8 minutes and three registers (24 bits) gives 2763.6 hours (16.4 weeks)!

UTP Cable Tester

Program - CABLE2.ASM **16C54**

This tester design has been used commercially for a number of years to test UTP (Unshielded Twisted Pair) cables for open, short and crossed pairs. UTP cables are used extensively in computer networks to link terminals and PC's to the main File Server using low cost cabling and connectors. The 8 wires in a UTP cable are formed into 4 pairs which are crimped into an RJ45 connector - similar to the telephone handset connector. If the cable is incorrectly crimped, damaged by a sharp object or disconnected at a junction box between the terminal and the file server, time and expertise is needed to rectify the fault.

This design is very simple and lends itself perfectly for a PIC solution where individual pairs can be tested in isolation for open, short and reversal, together with an adjacent wire shorting test. The sender end uses an RC oscillator running at approximately 300Hz. This eliminates the need for delay loops in the software. Each test re-defines the port direction register to prevent invalid test conditions - unused lines are set to inputs. The sender indicates short circuits between adjacent RJ45 connector pins (lack of a flashing led), while the receiver shows continuity, reversal (bicolour led) and if the pairs have been crossed - led pattern not in sequence.

This tester can save time and callout charges for computer support staff as a first pass method of fault finding and elimination.

The design can form the basic core for number of cable testers from simple audio lead continuity, through data comms right up to a production cable harness tester with an LCD display for fault reporting. As the number of conductors increases, so the number of I/O lines needs to be enlarged. This is possibly best achieved using external shift registers.

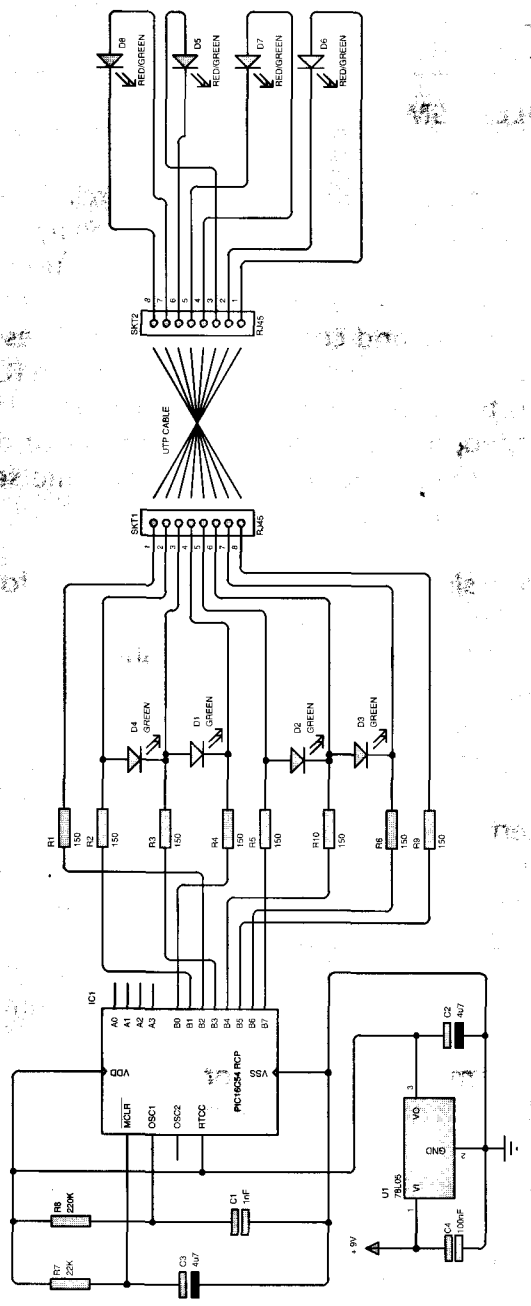


FIGURE 30

Software Hysteresis for Analogue Input on a Digital Port Pin

Program - none **16Cxx**

Digital port pins programmed as inputs are frequently used to monitor the condition of analogue input signals but one of the problems which arises is that when the analogue voltage lies in the region where the input level is undefined in digital terms then it is possible to read a logic '1' or a '0' on successive tests of the port pin. In the case of slow moving input levels this can cause serious problems since the software will respond in an unpredictable manner. A typical example is shown in the intruder alarm circuit of Figure 31A where the output of a Passive Infra Red (PIR) movement sensor is fed to an integrator Cint, Rint and is then applied to digital port pin A0 on a PIC processor. The input level on A0 is normally in the logic '0' area but when motion is sensed the voltage rises slowly towards logic '1' and in most alarm systems this change in level would have to be followed by a return to logic '0' followed by another transition to logic '1' within a given time before an alarm is given. This 'pulse counting' technique means that an alarm is sounded if the input level does say two '0' to '1' transitions in 10 seconds and obviously if the software reading the input pin cannot clearly differentiate between '0' and '1' then false alarms will be generated. What is needed is some feedback from the software back to the hardware to remove the 'uncertain' area of analogue signal input and this is provided as shown in figure 31B.

The analogue input signal from the integrator is connected via R1 to the sensing input A0 and port pin A1 is programmed as an output and is connected to A0 via R2: typical values for R1 and R2 would be 10K and 100K respectively as shown. The software which reads the state of the input pin (A0) has now to perform an additional task - this is that when a '0' is seen on A0, then A1 is set to a logic '0' which causes A0 to be 'pulled' towards '0' via R2. When at some later time A0 is seen to be a logic '1', A1 is then set to a logic '1' which causes A0 to be 'pulled' towards logic '1' via R2. A subsequent reading of

logic '0' on A0 causes A1 to become a '0' an so on and this feedback between the software and hardware places a bias on the input level towards the last digital reading which was taken and provides a simple method of generating analogue hysteresis on the input pin. Figure 32 shows the flow chart of a routine which could be used to implement the function.

The amount by which the input level on A0 is 'pulled' by A1 is called the hysteresis of the circuit and it can be shown that this is approximately (5xR1)/(R1+R2). Assuming a value for R1 of 10K Ohms, the table below shows the hysteresis for a number of values of R2. Note that in practice that the impedance of the driving circuit (the integrator in this case) must be much less than the value of R1 or the hysteresis circuit will feedback into the analogue circuit and cause the accuracy of the overall circuit to be upset.

R2 Ohms	Hysteresis with 5 volt supply Volts
100K	0.5
47K	0.8
22K	1.6
10K	2.5

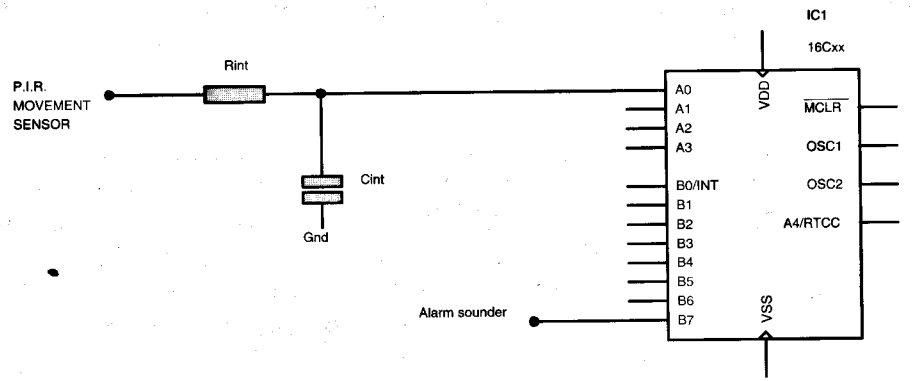


FIGURE 31A

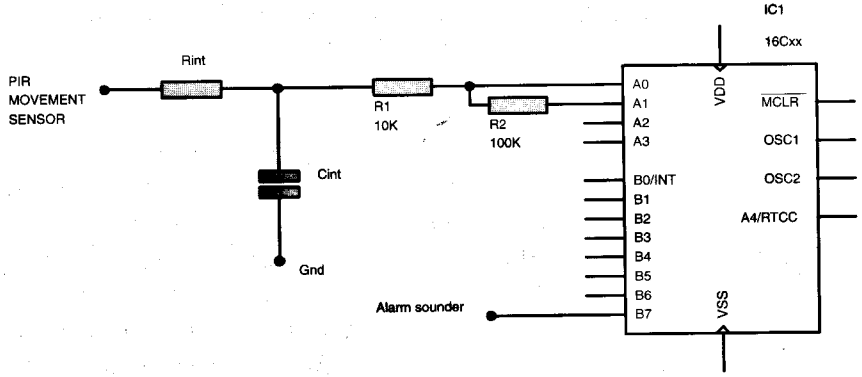


FIGURE 31B

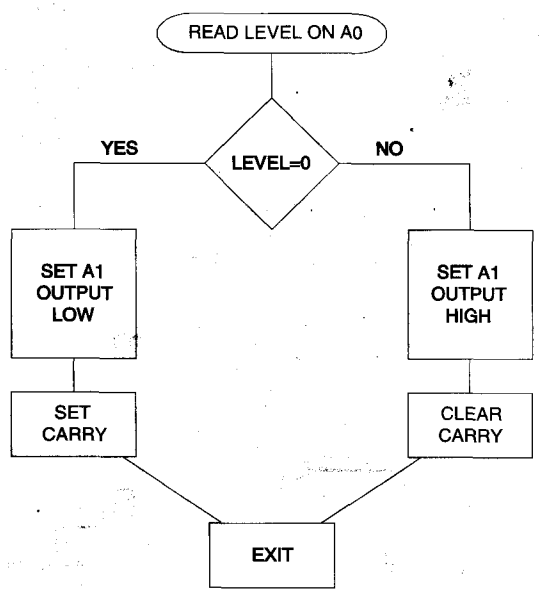


FIGURE 32

Long Term CR Based Timer

Program - CRTIME.ASM **16Cxx**

PIC users who wish to maintain a background timer with a time constant of say one second cannot easily fulfil this function using the internal timer when running at the normal clock rate of 4 MHz as the internal TMR0 divider cannot be programmed to give such a time-out. A simple hardware/software solution exists however in the form of a capacitor/resistor circuit which gives the necessary time constant and a small subroutine which tests the state of the CR network and when the capacitor has charged - discharges it and thus resets the timer. The circuit is shown in Fig 33 and the flow diagram in figure 34.

The PIC is programmed so that A0 is initially an input and capacitor C1 then charges through resistor R1 and the input level falls towards logic '0'. When the level on the port pin has reached logic '0' this can be sensed in software and the timer reset by turning the port pin into an output set to a logic '1' - and allowing the capacitor to discharge. Note that it is good practice to include series resistor R2 in the port pin path to limit the discharge current, particularly if the capacitor is of a high value.

An example of how the software for such a timer could be implemented is given in the program included with this book. In the set-up routine the port pin which is used for the timer is programmed as an input. Subroutine 'CHKTIME' is called during program flow and tests for a logic '0' on the port input and when this is seen A0 is programmed as an output set to logic '1' for a few 10's of microseconds - discharging the capacitor. A0 is then programmed as an input and the process is repeated. In CHKTIME when discharge has taken place the carry is cleared on exit from the subroutine as a flag that one unit of time has passed. In a typical application the circuit values shown give a time out of about 1 second and a register 'LONGTIM' is preset with the time out in seconds for a particular program loop. The main program loop could involve checking switches and other inputs and might occupy

a few milli seconds at most and each time it is completed, subroutine CHKTIME is called and if on return the carry is clear, LONGTIM is decremented until when it is zero - an exit from the loop takes place and the led state is reversed. The accuracy of the timer is only about +- 5% but in most applications this is good enough to solve the problem of implementing a long term time out. In applications where the closed loop capacitor discharge time is excessive then with suitable software changes a number of port pins could be connected in parallel to give the necessary low impedance discharge path.

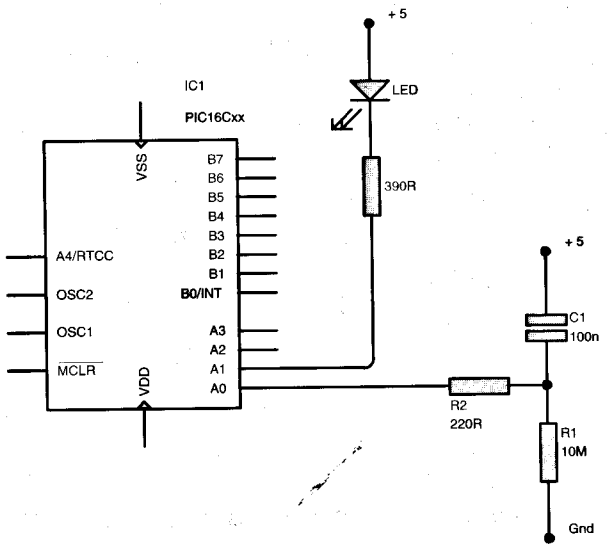


FIGURE 33

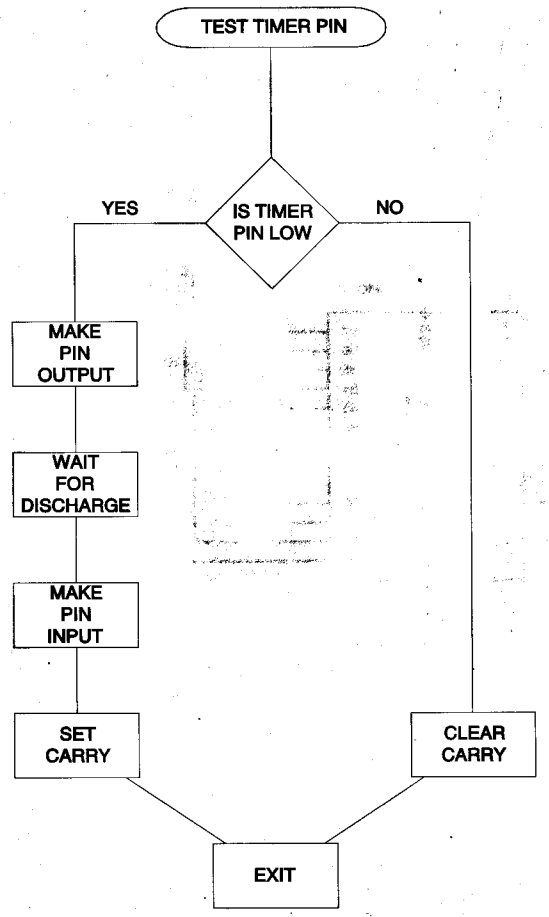


FIGURE 34

Ram Extension Using External Ram Chip

Program - RAM.ASM **16C57**

Applications which require large quantities of random access memory (ram) can use the peripheral port lines of PIC devices to provide read/write access to standard ram devices. Figure 35 shows the circuit arrangement which couples a standard low cost ram device to the PIC, using port B to drive the address lines and port C to read and write the ram data lines with port A providing read/write control. Such a circuit allows access to 256 - 8 bit data bytes which will cover most programmers requirements. The ram is read and written using routines given in the program which is included with this book and which is flow charted in figure 36.

Data is written to the ram by placing the data in location DATAIO and then with the data address in the W register, calling WRITE. The data address in the W register is written to and the byte in DATAIO is written to port C. With the ram address and data set up the read/write line is pulsed low and the chip select line is then pulsed low which writes the byte of data into the ram at the required address.

Data is read from the ram by setting the data address in the W register and calling WRITE. The data address in the W register is written to port B, the data i/o lines on port C are set to be inputs and with the read/write line set high to read, the chip select line is pulsed and the required data byte can then be read via port C and placed in location DATAIO. Note that the data i/o lines on port C are restored to the output mode after a read.

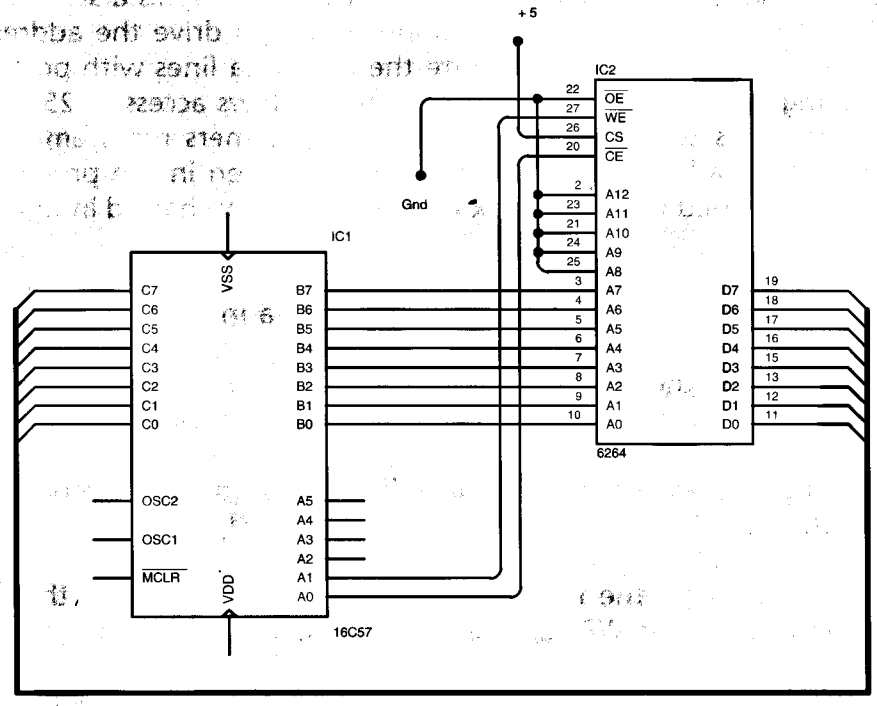


FIGURE 35

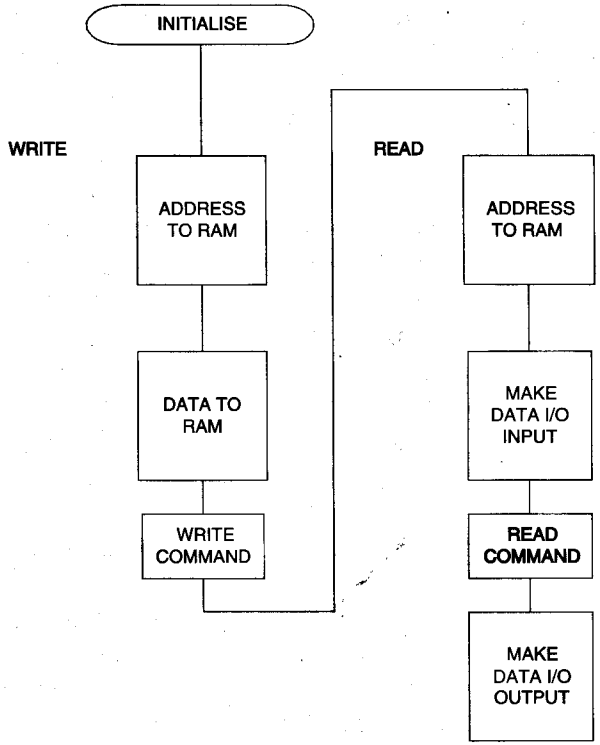


FIGURE 36

Mains Frequency Indicator

Program - MAINS.ASM **16C54**

CAUTION - the circuit described in this application does not provide **ANY** isolation from the live mains supply and thus **EXTREME CARE** must be taken when constructing and using the unit. It is recommended that the completed unit be housed in a sealed box and that it should not be used in damp areas such as kitchens or bathrooms.

The frequency of the alternating current mains supply is taken for granted by most users to be 50 Hz (60 Hz in the USA) but in fact the electricity supply companies are permitted to allow the frequency to drift above or below this figure dependant on loading and other factors. Some equipment's use the frequency of the mains to ensure the correct speed of for instance ac motors and it is often useful to be able to see at a glance what the frequency of the mains supply actually is. This project demonstrates both the hardware and software required to implement a point led display of mains frequency and tables are included to allow the unit to be operated on either 50 Hz or 60 Hz mains systems. The circuit shown in figure 37 displays the frequency of the mains supply from 48 Hz to 52 Hz in steps of 0.5 Hz on an array of 9 LED's and also demonstrates the use of a simple capacitive dropper type power supply for powering low current devices from the mains supply without the need for a transformer or a heat dissipating resistive dropper. The 16C54 micro controller has port pin A3 connected to the mains live supply via two 1 M Ohm resistors which allows the PIC to monitor the sine wave of the mains supply and the software is then able to precisely time the interval between successive zero crossings of the waveform. A look up table is then used to decide which led to light thus providing a clear indication of the current mains frequency. Figure 38 shows a flow diagram for the software.

Measurement of the mains frequency is carried out by waiting for a logic '1' to '0' zero crossing of the voltage waveform on A1 and then timing for a reoccurrence of this event. In practice the timing is

carried out in two parts - first a fixed period of time is allowed to elapse after a '1' to '0' crossing is seen. This period is such that when the frequency being measured is at its highest then the fixed period will be just shorter than the cycle time at this frequency. The timer then starts to time until the next '1' to '0' crossing at which point the time 'count' gives an indication of the mains frequency. Thus for 50 Hz operation the initial period is just less than 19 ms and the period of the highest frequency to be measured is 19.23 ms (52 Hz). The second timer value will then be approximately (19.23 - 19) which gives a result in practice of 35 and at the lowest frequency the result is 236. The advantage of this mode of timing is that the result which has to be used to turn on the appropriate led is an 8 bit value which allows simple testing to be carried out. If the overall cycle time was measured then to gain the necessary accuracy to allow discrimination between adjacent frequencies, a 16 bit or greater timer register would have to be used thus greatly complicating the calculations.

TMR0 in the PIC uses the internal crystal controlled clock as its timing source and the time 'count' from TMR0 is thus highly accurate. The display shows frequency in steps of 0.5 Hz and in the software, a series of equated values LEV0, LEV1 etc. is defined at the start of the program and these values give the highest TMR0 values for each level of the display. Thus after TMR0 has been read a series of test subtracts is carried out (starting at MAINF in the program) the results of which are used to update the display. The circuitry for the unit is extremely simple but a couple of points are worth of mention.

Direct coupling of the ac voltage to the PIC via a HIGH VALUE resistor is a common means of sensing zero crossing of the line voltage, with the internal clamp diodes within the PIC providing 'free' clamping of the incoming signal between ground and the +ve supply. Note that as a precaution against failure of the resistor which will have catastrophic effects, two resistors should be used in series. LED1 - LED9 all share a common current limiting resistor since the software will (should!) ensure that only one led is on at any one time and this is important since the capacitive dropper power supply will only source a few milli Amps and will not allow a number

of leds to be driven at the same time. The power supply for the unit consists of C1 which MUST be an X2 rated 250 volts ac device and which acts as a voltage dropper in conjunction with D1, D2 and C2 provides a 5 volt dc supply for the unit. R3 is included to reduce the current surge to the unit when it is first turned on with C1 discharged and R4 ensures that C1 discharges smartly after the unit is unplugged from the mains as otherwise a nasty shock can be had from the mains plug terminals!

The program has two tables which are used to determine the frequency and these take the form of a series of equates DIV1, LEV0, LEV1, LEV2 — LEV8. The first of these tables is enabled in the program provided and allows operation on 50 Hz mains supplies. If it is required to operate the system on 60 Hz mains supplies then this first table should be commented out with ';' marks at the start of each line and the second table enabled by removing the ';' marks at the start of each line.

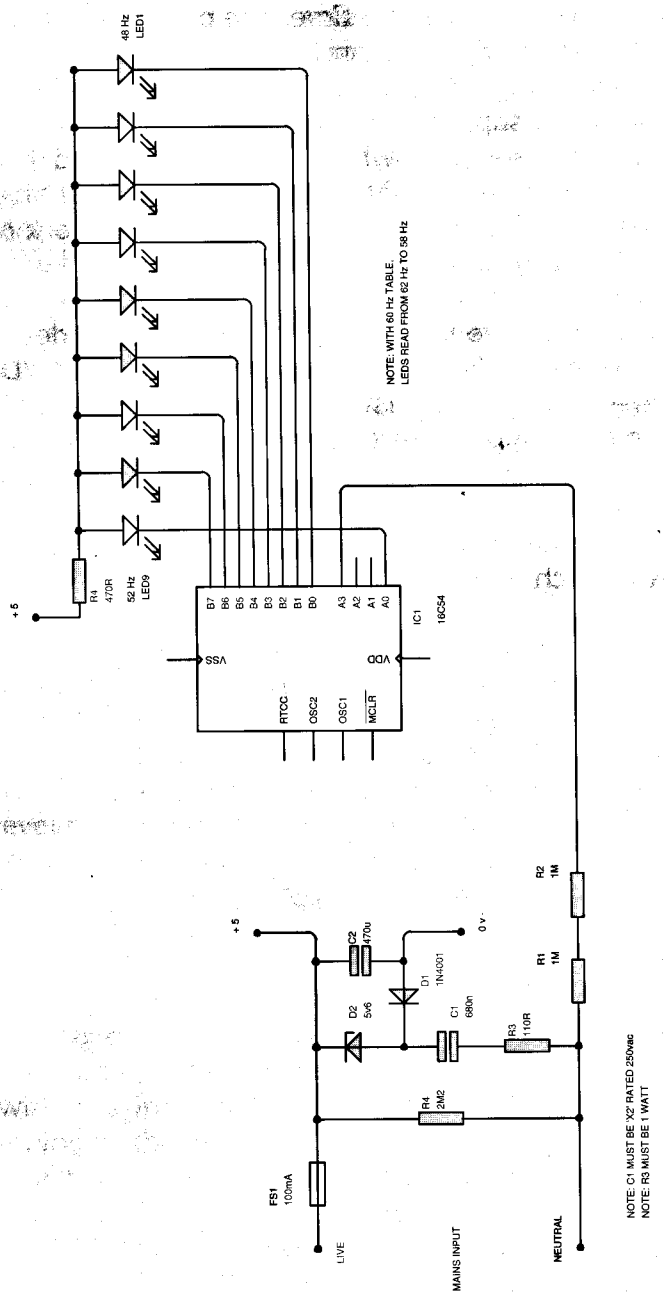


FIGURE 37

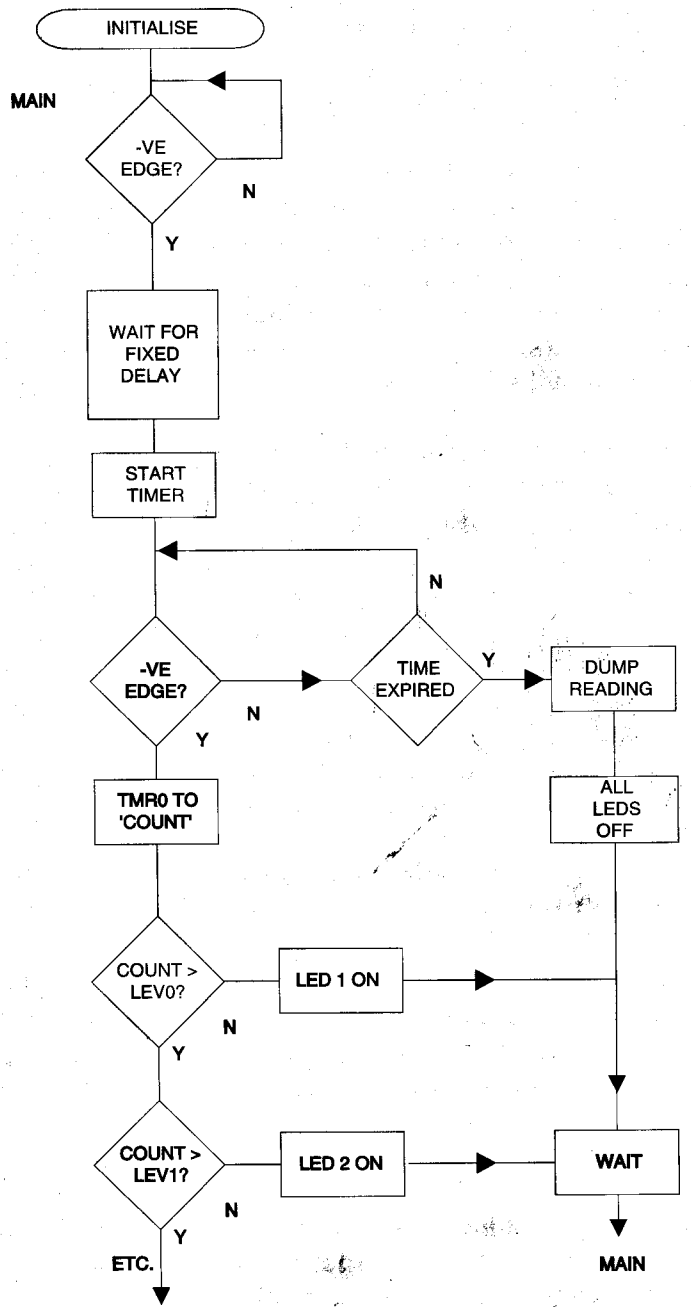


FIGURE 38

Triac Control of AC Load Power

Program - TRIAC.ASM **16C71**

CAUTION - the circuit described in this application does not provide **ANY** isolation from the live mains supply and thus **EXTREME CARE** must be taken when constructing and using the unit. It is recommended that the completed unit be housed in a sealed box and that it should not be used in damp areas such as kitchens or bathrooms. Special care should be taken to ensure that the user control is effectively isolated from the live mains supply.

The control of both large and small loads connected to the ac mains supply can most easily be carried out using a micro controller as a sensing, timing and firing controller for a triac power switch. Triacs are capable of switching ac load currents of from only a few hundred mA up to 15 or 20 Amps by the application of a small trigger current pulse which can normally be generated directly by the port pins of a PIC micro controller. The principles of ac load control using triac's are simple to understand but it is important that a number of points are clear before practical circuits are examined in detail and the following brief description of triac operation should be read and understood before proceeding.

The triac is a three terminal devices which is designed to normally be in the non conducting or blocking mode and thus in figure 39 the triac will not conduct and no load current will flow until a trigger current is applied to the gate. The trigger current pulse can typically be generated from the port pin of a micro controller running on a 5 volt supply if a triac with a sensitive gate is selected. The trigger current - which ranges in value from a few milli Amps to several hundred milli Amps dependant on the device type is applied as a firing pulse to the gate for a period which is long enough to ensure that the triac fires and that load current is flowing, after which time it may be removed. Current will now continue to flow through the triac until the voltage across the load falls to a point where the triac current is below the triac's holding current value. The triac then switches off and will have to be re triggered when

the supply voltage has started to rise again. Note that once the triac has triggered that the gate has no further control over conduction in the device and that once the device is triggered and a current greater than the defined holding current is flowing then the gate current can be removed, thus cutting the amount of power which is required to control the device. In practical circuits it is desirable to limit the generation of electrical noise and it is thus preferable that the triac is turned on when the supply voltage is as near zero as possible thus reducing the current surges which might otherwise occur - particularly when the triac is used to control highly capacitive or inductive loads. In AC control applications, due to the fact that the triac turns itself off when the current falls below the holding value, it is necessary to apply a trigger pulse to the gate at the start of each half cycle.

Where simple on/off control of ac load current is required, then when power is to be applied to the load, the triac is triggered every half cycle of the mains supply, thus allowing load current to flow for nearly 100% of the time.

If proportional control of load power is required then two possible modes of operation can be employed:

burst firing - where the triac is fired at the voltage zero crossing point for say 20 half cycles and then NOT fired for the next 20 half cycles which would gives a 50% reduction in load power. This mode of operation results in the minimum amount of EMC radiation since load current switching occurs at near zero current.

phase control - where the triac is triggered not at a point just after the zero crossing but at some time later in each half cycle such that the load current is reduced by some factor which depends on the time after zero crossing at which the device is fired. This mode of operation results in the generation of significant EMC emissions which will have to be countered by filtering the connections to the mains and also by shielding of the micro controller and its associated circuitry.

A typical practical burst firing triac control circuit with its associated micro controller and power supply is shown in figure 40. This circuit allows the user to set the power to be supplied to the load using 4

switches which can select one of 15 power levels and 'off'. The power supply for the microcontroller is generated by a capacitive dropper/regulator consisting C1 which MUST be an X2 rated 250 volts ac device and which acts as a voltage dropper in conjunction with D1, D2 and C2 provides a 5 volt dc supply for the unit. R3 is included to reduce the current surge to the unit when it is first turned on with C1 discharged and R4 ensures that C1 discharges smartly after the unit is unplugged from the mains as otherwise a nasty shock can be had from the mains plug terminals! Note that this circuit can easily power any PIC microcontroller but that it is only capable of delivering a short trigger pulse to the triac since the pulse may be a current of several 10's of milli Amps. The triac is configured such that anode 1 is connected to mains live/+5 volts with anode 2 connected via the load to mains neutral. In this configuration the micro controller can trigger the triac by placing a logic '0' on B4 and the triac will then conduct and when the current through the device is greater than the defined holding current, B4 can be restored to a logic '1'. The mains neutral phase input is connected via two series connected current limiting resistors to the A4 port pin and relies on the clamp diodes in the PIC to clamp the input level between 0 and +5 volts. This input is used to detect the mains voltage zero crossing point at which time the trigger pulse is generated.

The software is flow charted in figure 41 and performs all aspects of zero crossing, triac firing and user input selection and is based around a 'frame' of 15 cycles of the mains. If full power is to be delivered to the triac load then all 15 cycles are fired (a trigger pulse has to be generated on both the +ve and -ve going zero crossings) If less power is required then only say 8 of the cycles will be fired thus cutting the power fed to the load. This is achieved by reading the 4 switches connected to B0-3 and placing the value - in the range 0 to 15 - in counter OUTCNT. Over each of the next 15 mains cycles the counter is examined and if it is not zero, then the triac is fired (on both half cycles), the counter is then decremented towards zero and if it reaches zero then triac firing does not take place. Thus if the switch reading was 4 then for the first 4 of 15 mains cycles, firing would take place and for the remaining 11 cycles firing would not occur and in this way the user has direct control over load power.

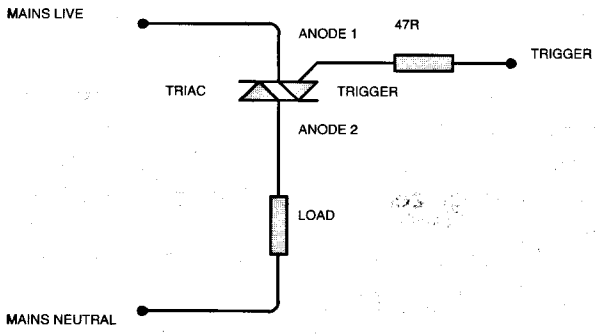


FIGURE 39

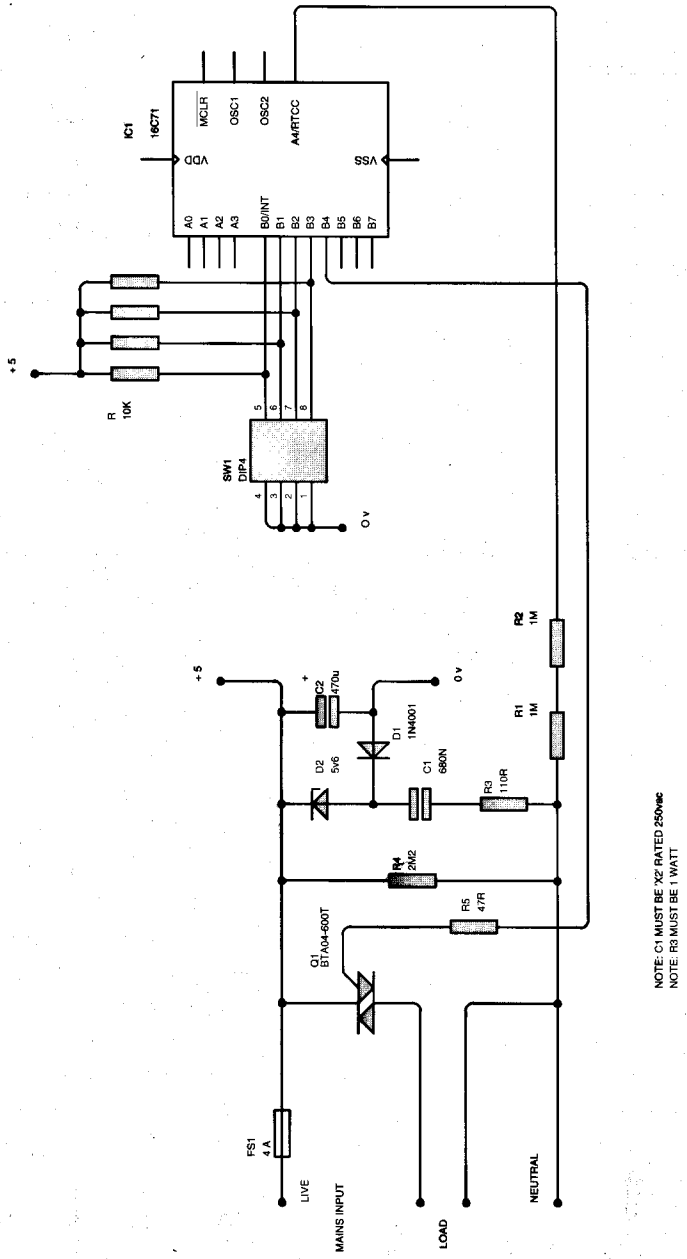


FIGURE 40

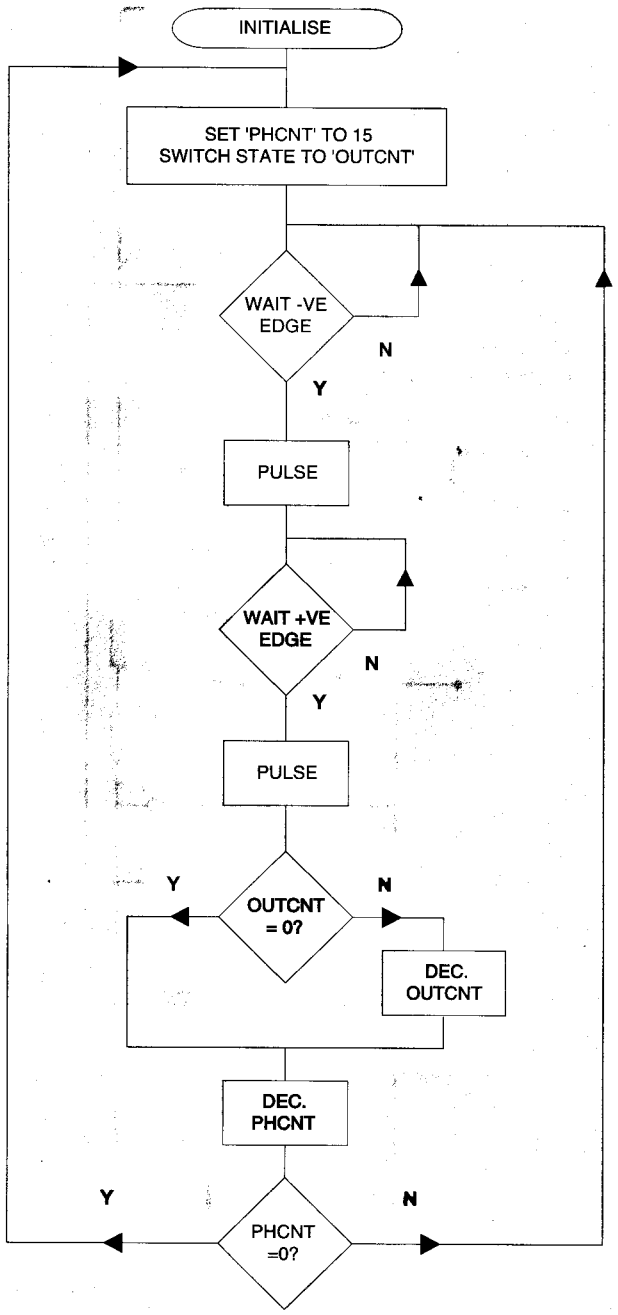


FIGURE 41

EEPROM Routines

Program - EEROM.ASM and EERDWR.ASM **16C57**

Electrically erasable programmable read only memories (EPROM's) are frequently used to provide systems with a means of storage for data and parameters which have to be retained even when the power is removed from the circuit. Micro controller systems almost always use serial EEPROM's since these operate over a 2 or 3 wire interface thus not using up too many precious port pins. The design given here allows the connection of a 2 wire serial EEPROM such as the Arizona Microchip 24LC01B to a 16C57 device and provides simple read and write control over the memory in 8 byte blocks. This EEPROM uses the popular I^2C protocol and can thus be used with the majority of other EEPROM devices on the market.

The circuit is shown in figure 42. The 24LC01B is connected to the 16C57 via the Serial Data / Address input/output or SDA line and the Serial Clock or SCL line. Note that an additional control line may be used to protect against false writes to the EEPROM: the Write Protect or WP line could be connected to another port pin and held normally high to allow read only access and be taken low to permit write access. This line is not handled in this application and is held permanently in the write enabled (low) state. The EEPROM itself has a further 3 control lines, A0, A1 and A2 which are used to address the memory device. In this application the lines are held low which gives a device address of 000 but if it is required to have a number of EEPROM devices connected to the common SDA and SCL port pins, then these would be wired such that each device has its own unique address. In the program it would then be necessary to change the value EECONT (in EEROM.ASM) when it is required to access a device other than that considered here.

The protocol used to read and write to the EEPROM is relatively complex and is comprehensively described in device manufacturers documentation so only a very brief description will be given here. The interface to the EEPROM is idle when the SDA and SCL lines are outputs from the processor and commands and data are sent to the

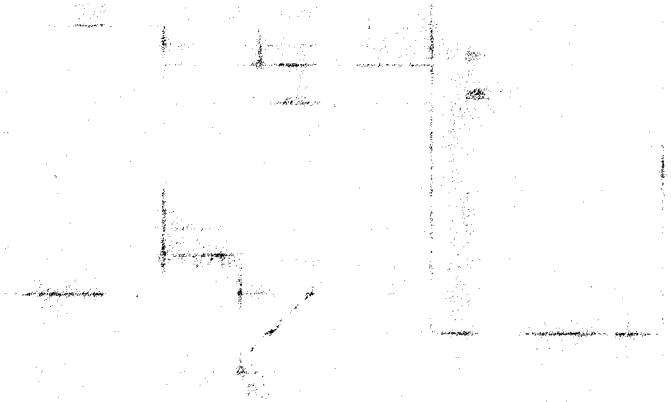
device by placing the require bit state on the SDA line and pulsing the SCL line low and high. Note that many parts are specified as operating at 400 KHz so that is necessary to 'pad out' some of the bit setting/clearing operations with NOP instructions to ensure that the timing requirements are met. All communications start with the sending of a control byte which contains the device type code and its address as set by the hard wired state of the A0, 1 and 2 lines and also a bit which is set to denote a read command and cleared to denote a write command. To write a byte, the address is next sent and then clock pulses on SCL are used to clock the data out to the EEPROM. To read a byte, the address is sent followed by a further transmission of the control byte and then the SDA port pin is turned to an input and clock pulses on SCL are used to clock the data in from the EEPROM.

The programs provide a simple means of using serial EEPROM in almost any PIC application. The routines have been written so that they can run in any device from a 16C54 to a 17Cxx part and are presented here on a format suitable for use in a 16C57 design. Notice that the level of subroutine calls within the EEPROM code itself has been kept to one by the use of macros and that this does expand the code size over that which could be achieved with the use of subroutines. Where a device with a deeper subroutine stack than the 16C57 is used then it should be possible to convert some of the macro calls into subroutines if code space is tight.

The reading and writing of the EEPROM is carried out on the basis of transfer of 8 byte blocks of data and these are specified in a table in EEROM.ASM. The block names all have the form BLKnn and allow the selection of different blocks on different device pages within the EEPROM and a full description of the coding is given in the program comments. The software for the actual EEPROM reading and writing is provided as an include file EERDWR.ASM which could be used to provide EEPROM capability in other programs by including it in the new program along with the required set-up routines which are given in the main program. Using the routines given in the include file is extremely simple as is shown in the demonstration program starting at IDLE in the main program. The ram area in the PIC which is used as a read/write transfer area -

starting at EEFSR, is first loaded with an incrementing pattern (actually the FSR contents) and then EEPROM page identity BLK01 is loaded into the 'W' register. CALL WRITEE is performed which writes the 8 byte ram area starting at EEFSR onto block 1, page 0 of the EEPROM. The PIC ram read/write transfer area is then cleared and with the same EEPROM page/block address in the 'W' register, CALL READEE is performed. On return the ram area has been restored to the original pre write pattern. With a Picmaster ICE and breakpoints set at TEST1, TEST2 and TEST3 it will be possible to see the ram area with the pattern before writing, cleared after writing and re written after reading.

A detailed description of the read and write routines is not given here but flow charts are given in figures 43 and 44 which clearly demonstrate the operation of WRITEE and READEE.



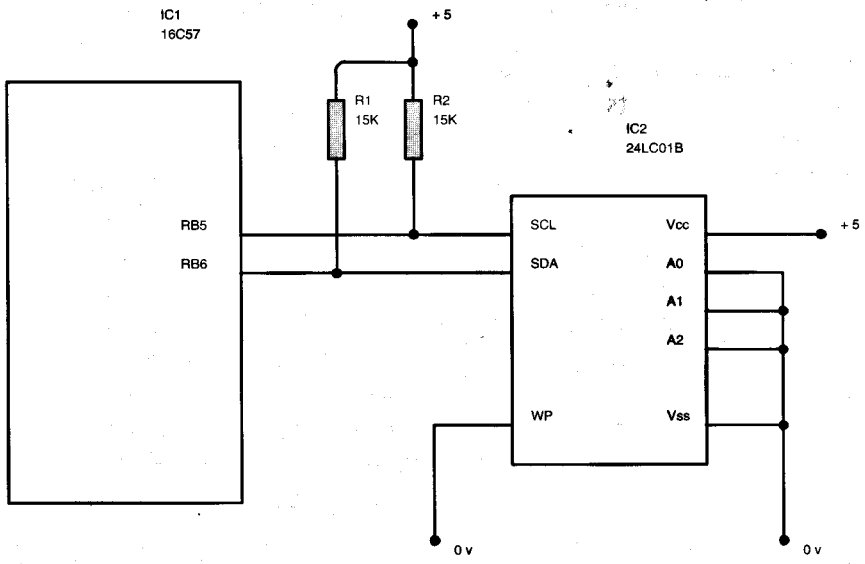


FIGURE 42

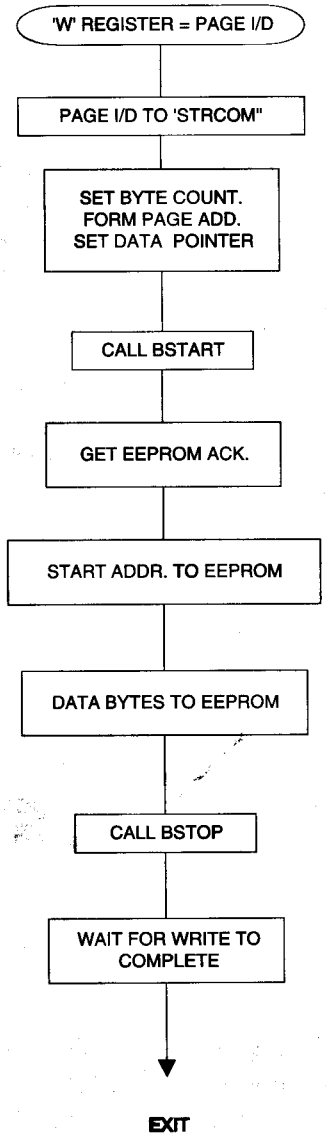


FIGURE 43

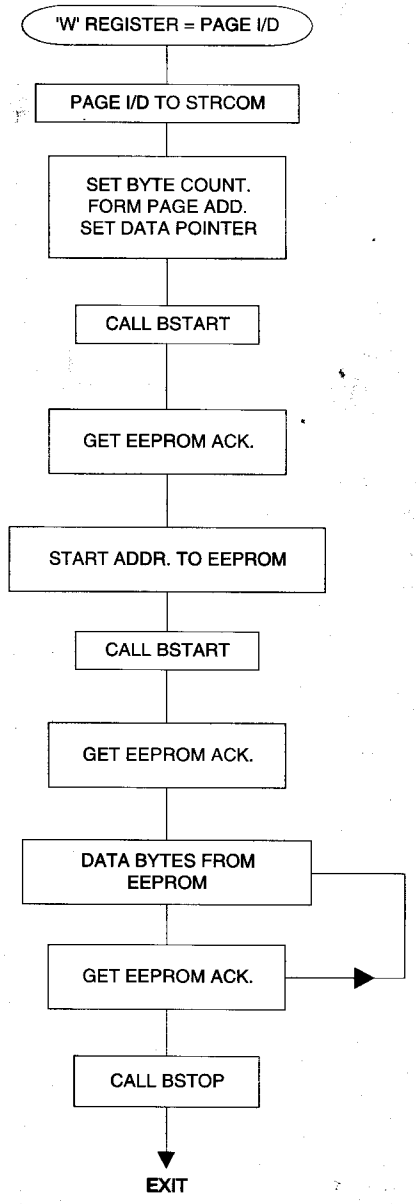


FIGURE 44

Magnetic Swipe Card Reader

Software - SWIPE.ASM **16C54**

The use of magnetic stripe coded cards has now become so widespread that there can be no reader of this book who does not have at least one such card in their wallet. Reading the data from such cards is relatively simple using an industry standard swipe reader head along with the software which is supplied with this book. This project describes the interface to an industry standard swipe reader and also how the data is coded on the card and is designed to allow the software module and hardware design to be used as a starting point for development of a more complex system. The system will operate correctly with most swipe cards such as those issued by banks and credit card companies and will run in a 16C54.

The data stored on a card generally mirrors that which is embossed on the card front and consists of a string of 4 bit BCD digits each of which has an additional 'odd' parity bit. The data string starts with a standard start byte and ends with a standard end byte and in between is stored the data. An overall longitudinal redundancy check (lrc for short) is added to the block to give a second level of protection against errors but this level of checking is not implemented in the program which is included with this book.
The reader head can be obtained from any one of a number of sources and should be a single track, track 2 type which will suit the majority of bank and credit cards (e.g.. NEURON CORPORATION MCR-570N-1R-0201). The head interface is shown in figure 45 and consists of 3 signals at TTL levels:

CARD_IN	- taken low when a card is in the reader mechanism.
STROBE	- clock pulses read from the magnetic stripe.
DATA	- data bits read from the magnetic stripe in synchronism with STROBE.

The head signals are connected to PORT B of the micro controller as shown in the circuit diagram and in addition a bi-colour led is driven

by a further two port pins B0 and B1. This led is used to signal either a good read when it shows green or a bad read when it shows red.

The software is relatively simple and operates around a loop centred on label MAIN. The CARD_IN signal from the head is continuously monitored and when it is seen to go low, a jump occurs to SWIPE which starts the block of code which handles the card interface. The first action is to monitor the STROBE signal from the head and read in data bits to SERIAL - which is used as a shift register for received data bits. The code continuously checks the received data pattern for CRDSTRT and if it is seen then flow drops down to receive the data itself. Note that at all times during reading the STROBE and DATA lines that it is necessary to check that the card is still inserted in the head otherwise the code could stall! This checking is carried out by macro CARDOUT which tests for the card out state and if it is seen - causes a jump to SWIPEY - which is the error exit from the routine. Data is received round a loop starting at SWIPEE and when each block of 5 bits has been clocked into SERIAL, a parity check is performed and if it fails - exit to SWIPEY takes place. Data is packed in upper/lower nibbles in ram starting at CARDATA and the 10 bytes of data are read and stored in ram in the PIC with the data byte count in SWIPCNT - note that this is the count of the BCD bytes which are packed during storage in the PIC.
Exiting from SWIPE code block is either carried out by lighting the green led if the swipe was successful (SWIPEK) or by lighting the red led if the swipe was not successful (SWIPEY) . In either case a jump then occurs to MAINA where a delay is allowed to give the user about 1 second to see if the swipe was successful before turning off the led and looping to main. Those readers equipped with a Picmaster can break the program at label MAINB and examine the memory where the card contents will be available if a good swipe has taken place. This label is also the place where an application program would be able to pick up the data. The software flow for the SWIPE routine is shown in figure 46.

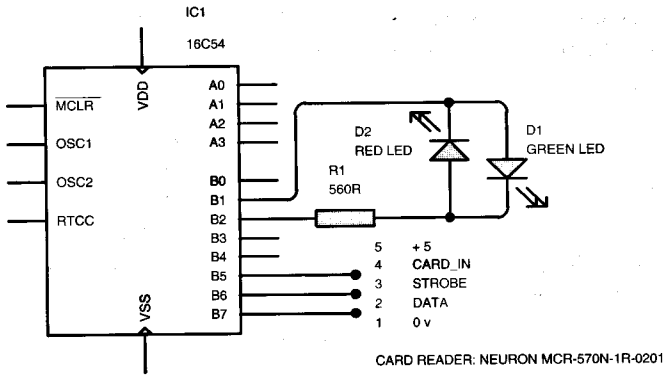


FIGURE 45

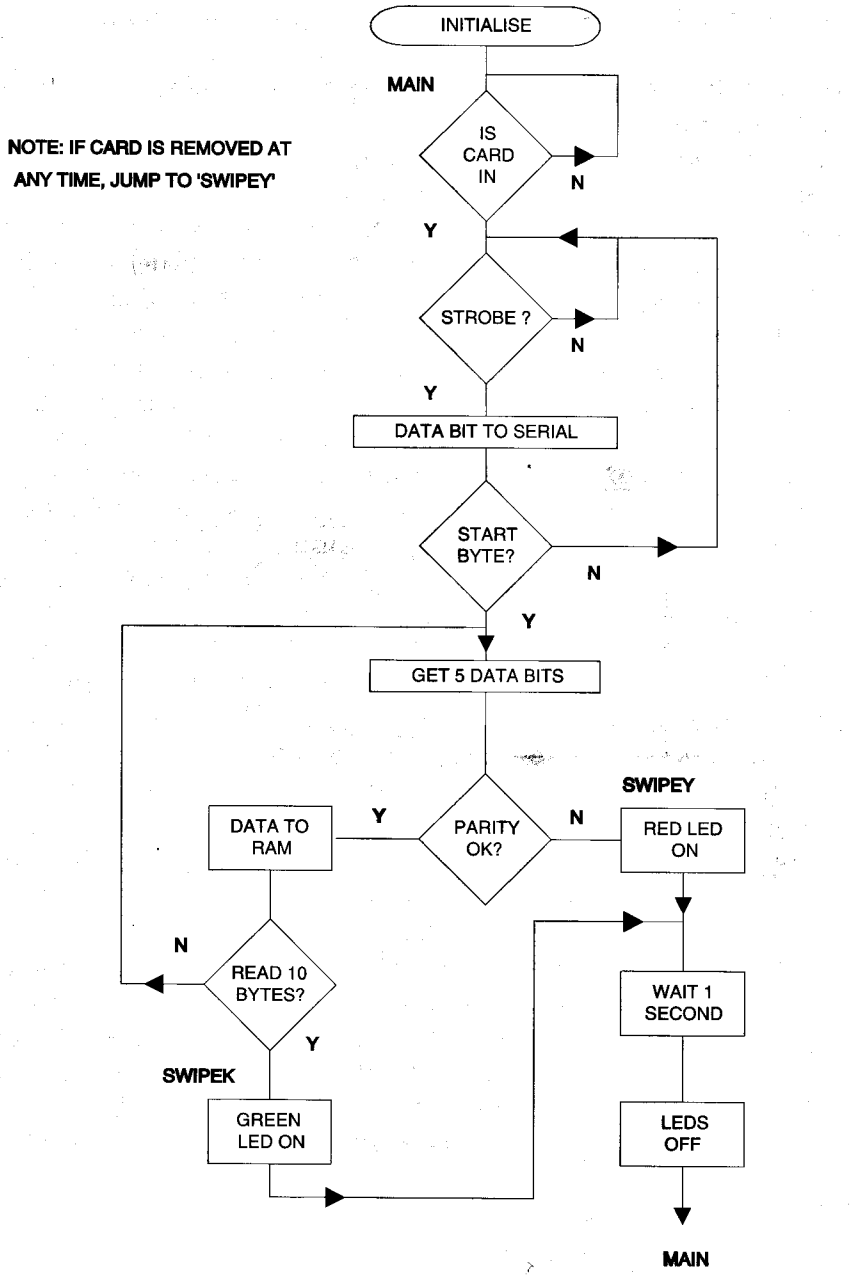


FIGURE 46

Software Protection Dongle

Program - DONGLE.ASM, DONGLE.EXE, DONGLE.C **16C54**

Producers of commercially valuable software such as computer aided design packages often resort to supplying a copy protection device with the package which consists of a 'smart' plug which has to be connected to the serial or parallel port of the computer when the software is in use. These 'plugs' are commonly termed 'dongles' and are an extremely effective method of preventing the copying of software since the package will not operate without the dongle plugged into the computer port - thus preventing simple duplication of the software by copying the master discs. Dongles normally contain a processor which when it receives a message from the PC sends back a coded response and this check is carried out every couple of minutes to ensure that the user has the dongle in place. The processors in commercial dongles are custom programmed for the software company to prevent unauthorised duplication of the dongle and hence copying of the software and subsequent loss of revenue to the software company. PIC micro controllers such as the 16C54 are ideal for supplying the necessary intelligence for a dongle and figure 47 shows the circuit diagram of such a device.

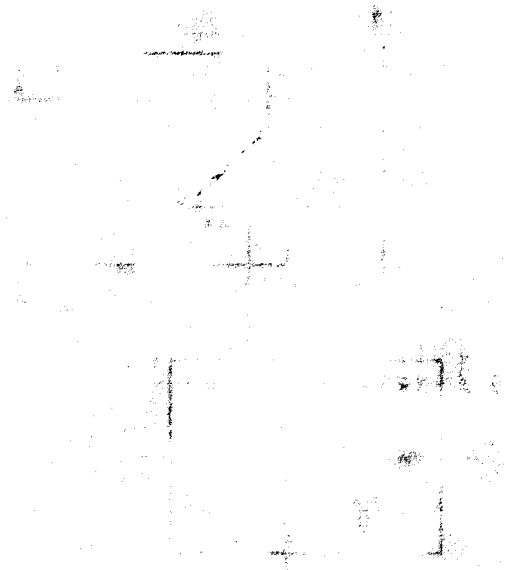
The hardware for the dongle is designed to be plugged into, and powered from the PC's serial port and consists of the processor itself which is fitted with software to perform serial data reception and transmission and message encoding and also simple interface circuitry to allow the processor to communicate at V28 levels. The dongle requires both +ve and -ve voltage rails for correct operation and these are provided by drawing a small amount of current from the computer serial port data and handshake lines which will be under the control of the PC based program while the dongle is in use. The DTR line from the PC is held at V28 high providing approximately +12 volts and this rail is regulated using a low standby current 5 volt regulator (IC3) to provide a stable +5 volt supply for the 16C54 processor (IC1). The RTS line from the PC is held at V28 high low providing approximately -12 volts. Note that the TX

data line from the PC can provide both a +ve and a -ve supply as it switches between mark and space conditions and that these three lines from the PC are connected via low forward voltage drop Schottky diodes to the positive and negative supply reservoir capacitors C2 and C1. Operational amplifier IC2 is used to translate the logic level serial data output from the processor into the V28 compatible levels which are required to drive the receive data input to the computer. The received data stream from the PC consists of a series of pulses of + - 12 volts and these are passed through R1 to limit the current flow with the internal clamp diodes in the PIC ensuring that the voltage swing is within the device specification limits.

Program DONGLE.ASM contains complete demonstration software which can be used as a starting point for a 'real' dongle application and examination of figure 48 will show that the program is simple and easy to understand. DONGLE.EXE is a program which can be run in a PC to test correct operation of the dongle hardware and software and the source code for this program is also included as DONGLE.C and can be used as a starting point for further development (this work will however require the use of Borland Turbo 'C' compiler software) The dongle operates by communicating with the PC in serial data format at 1200 baud's (bits per second) with a data byte format of 1 start bit, 8 data bits and 1 stop bit. A message is transmitted from the PC consisting of the character string 'DONGLE' and when this is received by the 16C54 in the dongle, the six bytes are stored and then after a short delay are transmitted back to the PC except that the byte positions are reversed so that the PC receives the string 'ELGNOD'. The software in the PC knows the scrambling algorithm which the dongle imparts on the data it receives and can thus check that the dongle is in place and functioning correctly. This very simple scrambling of the data bytes would in practice be an easy code to crack and the generation of the response message is normally carried out using a much more complex algorithm than that used here. The DONGLE.EXE program which is provided is meant to be used as a test program to verify correct operation of the hardware and software of the dongle and in practice the code to communicate with the dongle would be incorporated into the application program running in the PC such

that at regular intervals the presence of the dongle is checked for and if it is found not to be plugged into the serial port - the user will be denied access to the program. To test the operation of prototype dongle, load DONGLE.EXE into the current directory of a PC and at the DOS prompt type DONGLE followed by a 'return'. Connect the dongle to the serial port of the PC according to figure 47 and follow the instructions given on the screen of the PC for a demonstration of the simple principles of operation of the dongle.

This particular version of a dongle occupies the serial port of the PC and will prevent connection of for instance printers or modems to this port of the PC while the dongle dependant program is in use. Variations on the theme allow over plugging of the dongle with a standard serial connector and in this case the serial communications may take place over say the ring indicator or other seldom used lines which are included in the serial port. Alternately the dongle may be connected to the parallel port in which case as before measures may have to be taken to allow the user the use of the parallel port for connection to his printer while the dongle remains in place.



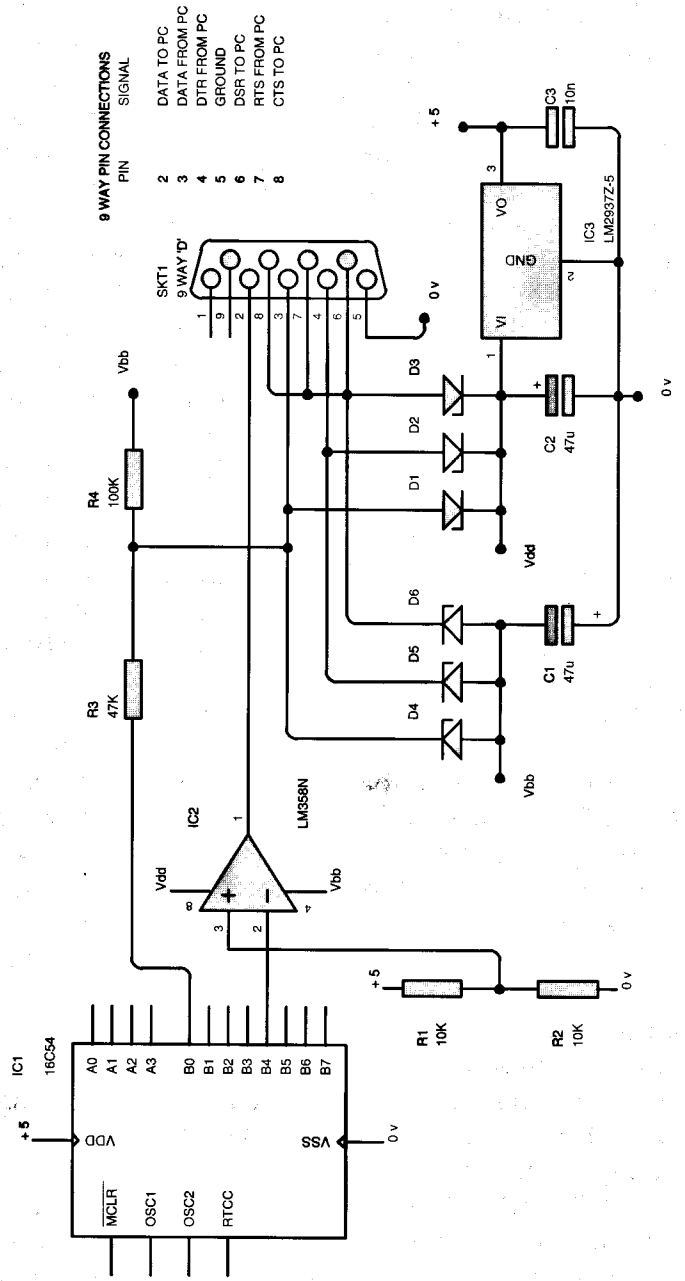


FIGURE 47

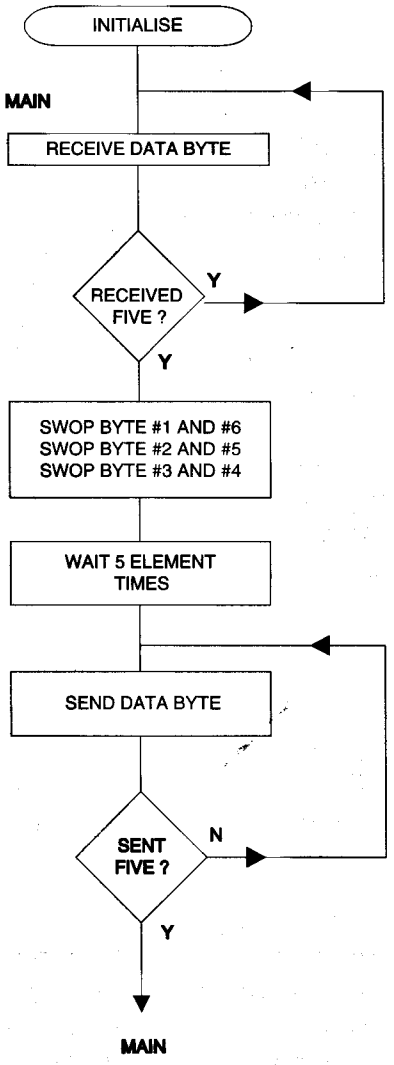


FIGURE 48

Melody Player

Program - TUNE.ASM **16C54**

Applications which require audio output such as door chimes, children's toys etc. can easily generate the necessary drive signals directly from micro controller software by means of look up tables and can produce sounds ranging from beeps of tone to complete tunes. The high speed of instruction execution of PIC processors makes them ideal for these applications and this article describes the hardware and software of a simple low cost melody playing system. Programmers should note that all of the look up tables and timings given in this book are based on a 4 MHz crystal or oscillator source which gives a 1 MHz instruction clock rate.

The principles of simple tone generation are that the option register is set for internal clock source with a given pre scalar divisor ratio which defines the input clock rate to the TMR0 register. To generate a square wave signal with an even mark to space ratio the tone output pin is set low, TMR0 is written with the number of input clock cycles which are required to give a delay equal to one half cycle and then when TMR0 overflows the tone output pin is set high and the process repeated - thus generating one cycle of a square wave with an even mark space ratio. If this process is repeated a continuous square waveform is produced which can be filtered if required before being used to drive a sounder. Note that TMR0 counts UP from the preset value and so if we require say decimal 40 counts per half cycle of the tone, TMR0 has to be loaded with 256 - 40 = 216 and NOT 40! In practical applications a counter will be set up to count the total number of cycles of tone which are produced thus allowing the program to determine both the frequency and the duration of the tone. Program TUNE.ASM generates the necessary bursts of tone to play the introduction to 'jingle bells' and allows experimentation with different tone frequencies and tone pulse duration's by simply altering a table. This ready made melody playing program could be extended if required to form the basis of a child's tune playing toy or a multi tune doorbell since the frequency and duration of each note is held in a look up table which

gives plenty of flexibility as to the notes and tunes which can be played. The flow charts for the program are given in figures 50 and 51 and the extremely simple circuit for application is shown in figure 49. The sounder can be almost any piezo ceramic resonator of the type which is normally sold as a low volume alarm sounder and is connected to the four pins of PORT A to ensure that sufficient drive is available.

The looking up of data tables in PIC processors is complicated by the fact that they share a common feature in that it is not possible for the processor to directly read program memory and where it is necessary to store look up tables then these must be stored as an array of RETLW instructions. When the RETLW instruction is executed, the W register is loaded with the pre defined value and thus execution of the instruction:

```
RETLW   .7
```

causes a return from subroutine to be executed with the value decimal 7 in the W register. By creating a table of 'RETLW' instructions each with a given return value is possible to set up a table of note frequencies. Thus the values for TMR0 which generate notes of a particular frequency are calculated, they can be defined in the program as:

```
TONEA   EQU nn      ; TMR0 count for frequency A
TONEB   EQU nn      ; TMR0 count for frequency B
TONEC   EQU nn      ; TMR0 count for frequency C
```

A routine can then be written which sounds a note for a given time and which accesses a table of values to effectively play a melody. Thus to play up and down the first three notes of the scale, a table would be created as:

```
TABLE   RETLW   TONEA
        RETLW   TONEB
        RETLW   TONEC
        RETLW   TONEB
        RETLW   TONEA
        RETLW   .0
```


The note sounding routine would then retrieve each note in turn, sound it for a fixed period and thus play the simple melody. Note that as is the case with most tables that a special 'end of table' entry has to be included to allow the software which retrieves the table values to determine when the end of the table has been reached.

By extending this principle it is possible to create large tables which contain long melodies. In practice, notes will not always have to be sounded for the same time and so a refinement can be added where each note is defined as a pair of table values with the first value giving the frequency and the second the duration of the note. Thus the above table with the addition of note duration values might look like the following:

```
RETLW   TONEA
RETLW   .100
RETLW   TONEB
RETLW   .200
RETLW   TONEC
RETLW   .200
RETLW   TONEB
RETLW   .100
RETLW   TONEA
RETLW   .100
```

The routine which sounds the notes now has to call the table twice for each note and then use the retrieved data to set the tone frequency and duration and this technique is used in program TUNE.ASM. Note that the duration's are expressed as a cycle count of the current note and so will require some experimentation if the tables given in the program are altered.

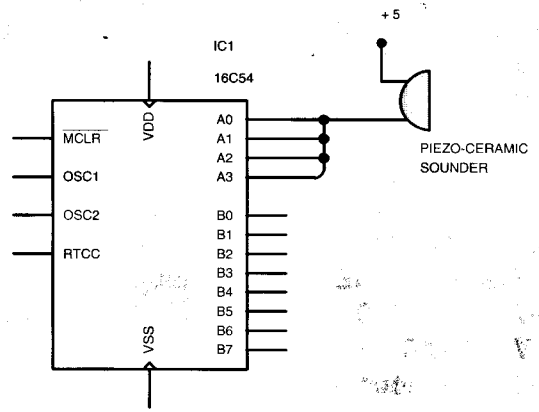


FIGURE 49

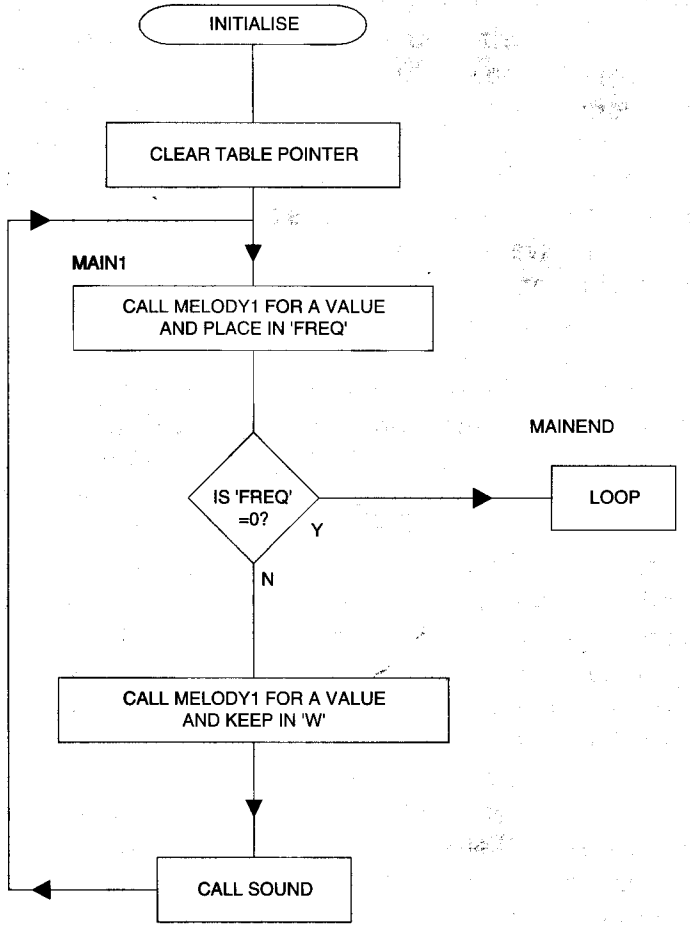


FIGURE 50

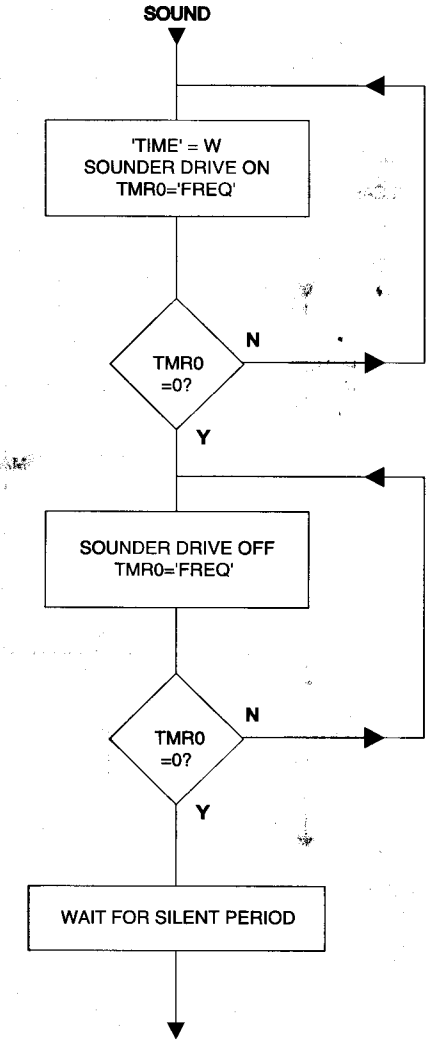


FIGURE 51

Infra Red Data Link

Program - IRLINK.ASM **16C54**

Infra red (IR) remote control is most often found in television remote controls which allow the user the luxury of channel selection, volume control etc. without the need to move from the comfort of the living room settee. The hand set normally contains a custom integrated circuit which scans the keypad and when a key is pressed, this chip sends a data signal to the television set by modulating an infra red light emitting diode. An infra red detector diode mounted on the front panel of the television set is used to receive the infra red light from the transmitter and a further custom integrated circuit is used to decode the data and turn this into a series of signals which can be used to control the television set. The remote control system described here is intended to introduce some of the principles of IR transmission and reception and will allow the remote control of up to three loads under the control of the 16C54 micro controller.

The system operates by transmitting a waveform which has a known mark/space ratio which can be measured by the receiver. When a stream of pulses with the correct mark/space ratio is seen, the receiver will step to the next output state - turning off the current load and turning on the next one. Four modes are offered, load 1 on, load 2 on, load 3 on or all loads off. As with all IR data transmissions, 'carrier' modulation principles are used to send the data so that instead of turning on the IR transmitter diode for say a logic '1' and off for a logic '0', when a logic '1' is to be sent the IR diode is rapidly turned on and off or 'modulated' at a rate of 38 KHz and the detector at the receiver can then be designed to receive the 38 KHz frequency and output a logic '1' only when this frequency is received. This technique of carrier modulation is widely used in data transmission and gives a high level of noise immunity since only signals with the correct carrier frequency will be recognised by the receiver. The transmitter sends the data waveform all the time the button on the transmitter is depressed and reception of the waveform for a short period of time is used to 'step' the receiver

function to its next state. In the case of a remotely controlled light switch this could mean that successive operations of the transmitter button would control the states of 3 lamps. Figure 52 shows the remote control transmitter which consists of two 555 timers, one of which is connected to as to act as a low frequency data signal generator and the other to act as the high frequency modulator driver for the IR transmitter diode. The frequency of the data signal is not critical since the receiver software is looking for a defined mark to space ratio and this is set by 1% resistors R1 and R2 with the frequency being set by C1. This technique is easier to implement than one which relies on a defined data signal frequency since this would have to be set up in each transmitter using a preset resistor and would be subject to drift caused by changes in temperature, supply voltage etc. The 'data' output from the first 555 timer (IC1 pin 3) is used to turn off transistor TR1 which normally holds the second of the two 555 timers in the reset position. The output pulse on IC1 pin 3 thus allows the second 555 timer to free run at its frequency of 38 KHz and the output on IC1 pin 3 is used to drive TR2 which modulates the current through the IR transmitter diode D1. Note that the range of the system is dependant almost exclusively on the current passed through this diode and that a range of up to 15 metres can be easily achieved if R8 is reduced to 47 ohms or less. Another more expensive but more efficient method of increasing the range is to drive 2, 3 or even 4 GL360 diodes in series with a current limiting resistor of a few 10's of Ohms.

The receiver circuit is shown in figure 53 and consists of a low cost IR receiver integrated circuit IC2 which receives the modulated data stream from the transmitter using a lens moulded into the device package to concentrate the incoming signal. The output from IC2 is a logic '0 when the 38 KHz modulated stream is seen and this data signal is connected to the A0 input of the 16C54 (IC1). The receiver software is flowcharted in figure 54 and continuously scans the IR data input and monitors for the appearance of carrier which is read as a logic '0'. When this is seen, macro TIME measures the length of time for which the data signal is '0' and then the time for the logic '1' state, allowing the mark space ratio of the waveform to be calculated. When a number of consecutive pulses with the correct mark to space ratio have been counted, the next output state is

entered - that is the current load drive is turned off and the next load drive is turned on. The software now monitors for non reception of the correct waveform when the transmitter button is released. A short dead period is then allowed before the scanning process is restarted. This sequence of testing and checking ensures that the load drive state is not cycled if for instance the transmitter button is kept continuously depressed. Loads can be controlled from OUT1, 2 and 3 which are active high and in order to aid diagnostics three led drives are provided on B3, 4 and 5 such that these LED's mirror the state of the main load drives on B0, 1 and 2.

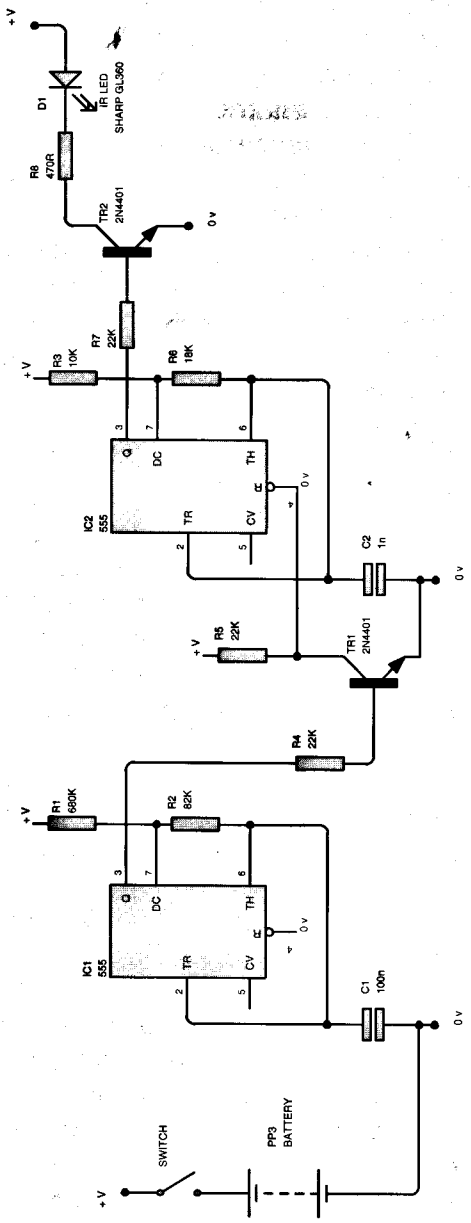


FIGURE 52

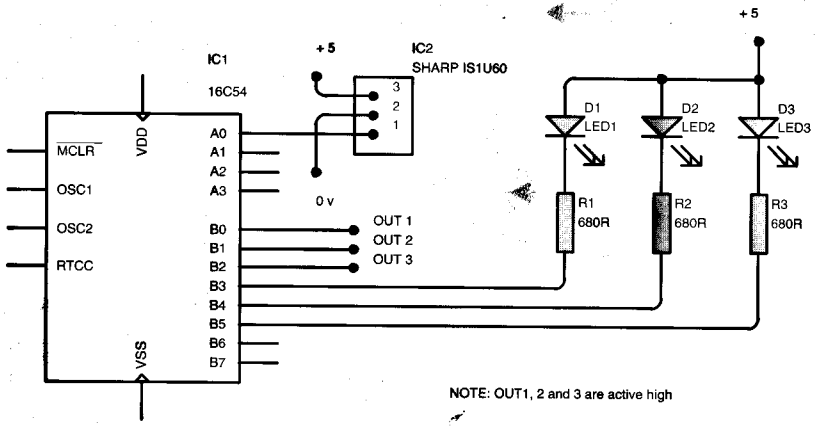


FIGURE 53

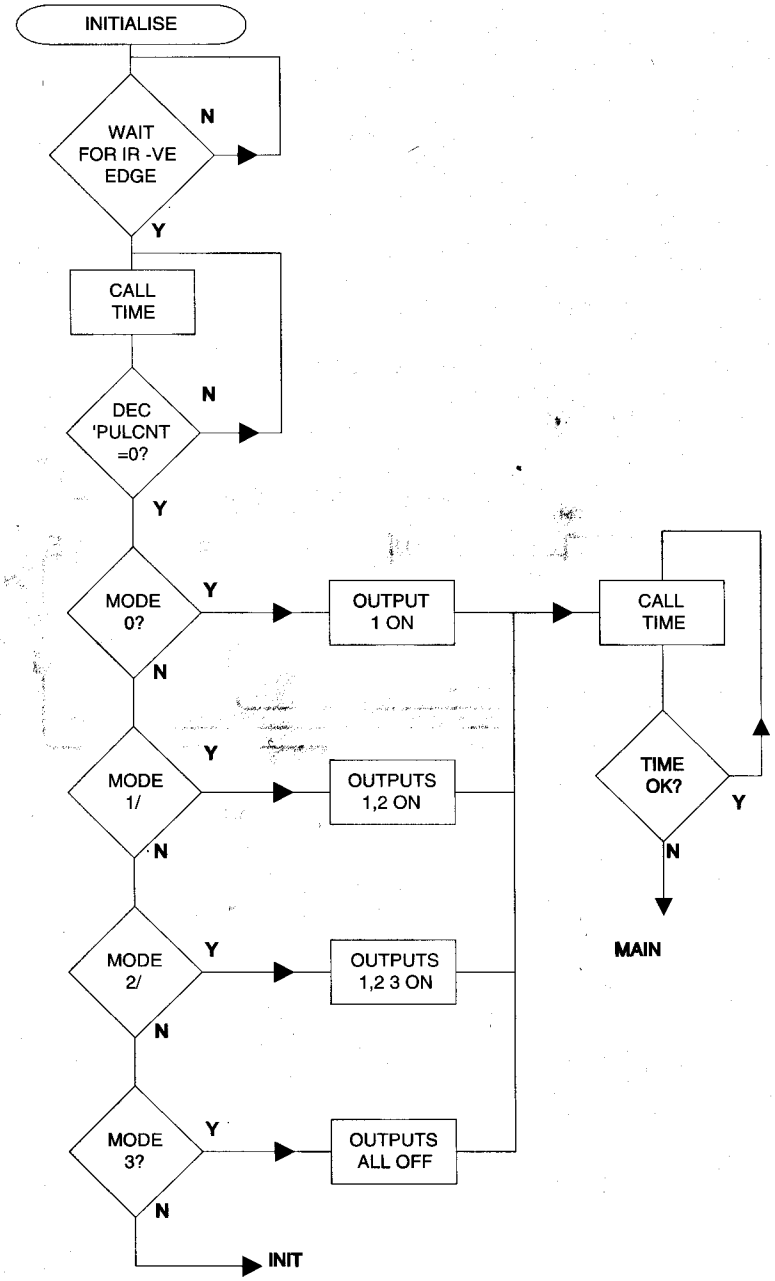


FIGURE 54a

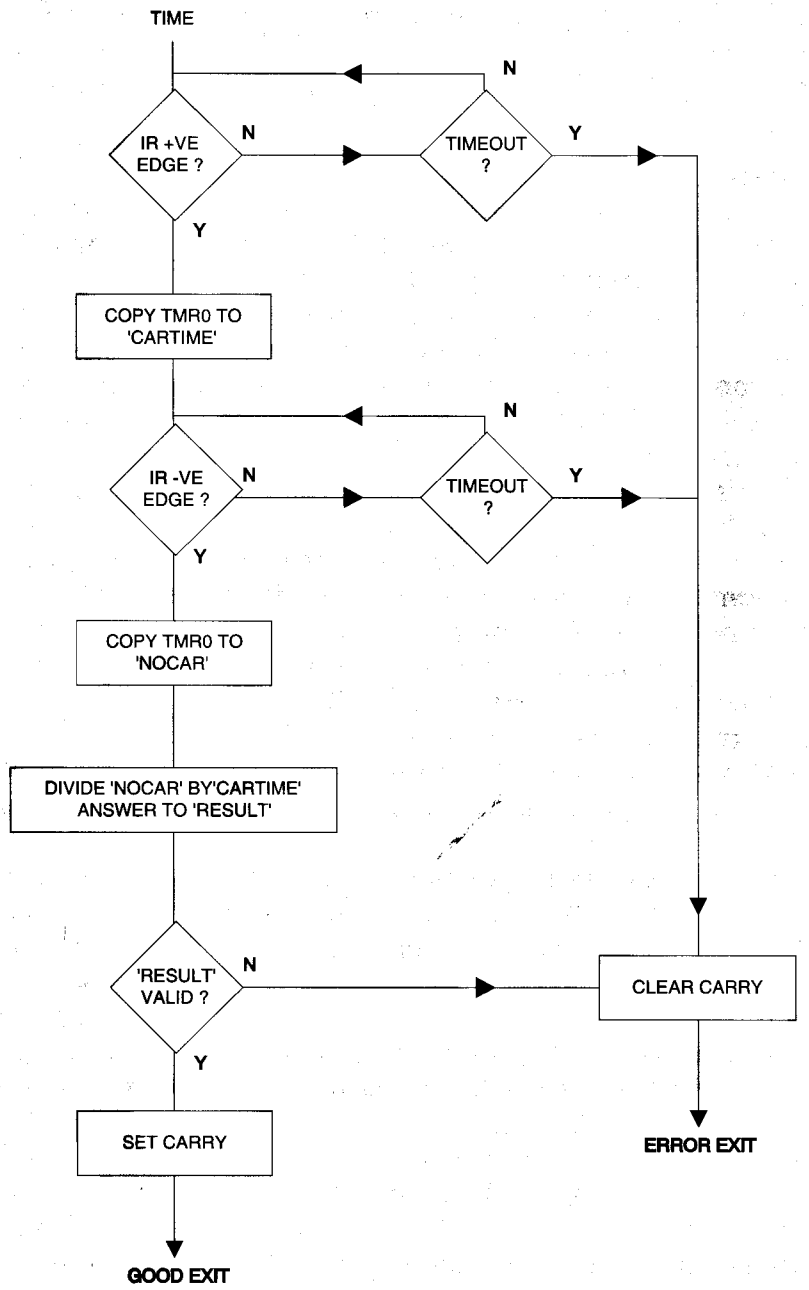


FIGURE 54b

Boat/Caravan Intruder Alarm

Program - ALARM.ASM **16C54**

Owners of boats and caravans frequently have to leave their prized and expensive possessions unattended for long periods of time and the risk of intruders breaking in and causing damage is always high. This project presents a simple yet effective intruder alarm which needs no external keys or switches for setting and yet provides a good level of security for an extended period of time. The alarm system is based on one door contact (more can be added in series if required) which is monitored by a 16C54 micro controller which controls an alarm sounder and also two status LED's. The unit is built into a small plastic case which is secreted within the boat (or caravan) where it is unlikely to be found by an intruder. The unit case is fitted with a single pole toggle switch to turn the unit on and off and also a monitor led which shows the current state of the system. The door contact in the prototype system consisted of a reed switch and magnet with the switch glued to the inside of the door and the magnet to the door itself such that when the door was closed, the reed contact was made. A led is fitted outside the door to show the user - and any prospective intruder which mode the system is in.

The system operates in the following way. Prior to leaving the boat, the unit is turned on at which point both the monitor and door LED's flash on and off every second - indicating that the unit is ready to arm itself when the exit door is closed. When the door is closed, after a short delay the monitor led is tuned off and the door led changes its on/off flashing to being mainly off with a very short 'on' flash every second. This indicates to the user - who is now safely outside the boat, that the system has armed. When the user returns to the boat and opens the door, a silent delay of a few seconds is allowed for the unit to be switched off. After this time the alarm sounder is beeped every second for 15 seconds as a warning that the system has been activated after which time the alarm sounds for a period of 2 minutes.

The circuit of the alarm is shown in figure 55 and the software flow chart in figure 56. The hardware itself is very simple since all aspects of timing, debouncing etc. are performed in software running in the 16C54. The door switch input to the PIC is normally held high by R5, and C1 is provided to kill any noise spikes which might be induced in the wiring and which could cause damage to PIC input pin RB2. The door contact is open when the door is open and closed when the door is closed. The micro controller drives two LED's, the monitor led fitted to the unit case and the door led fitted outside the door. The alarm sounder is driven by TR1 and should be a 'self oscillating' type suitable for 12 volt dc operation. The power supply is relatively simple but note that it is advisable to use a low standby current regulator for the 5 volt supply to save discharging the 12 volt battery if the alarm is left armed for any appreciable period of time. The software is very straight forward and should be easy to understand when it is read in conjunction with the flow diagram. One unusual feature is the use of a double byte code to attempt to prevent the sounder being enabled if for instance a lightning strike causes the processor to run wild and this operates as follows. Just before the alarm is sounded, two 'security' bytes are written into ram and in the alarm sounder time-out loop these bytes are constantly checked and if they are not correct then the alarm is turned off.

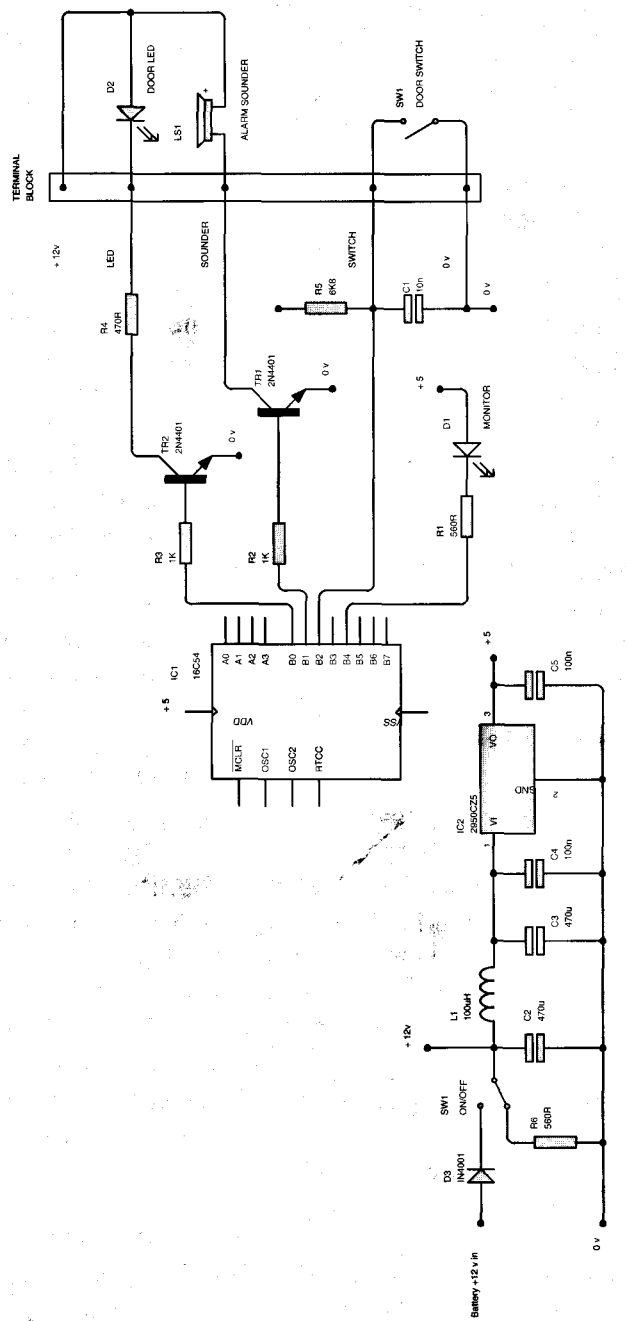


FIGURE 55

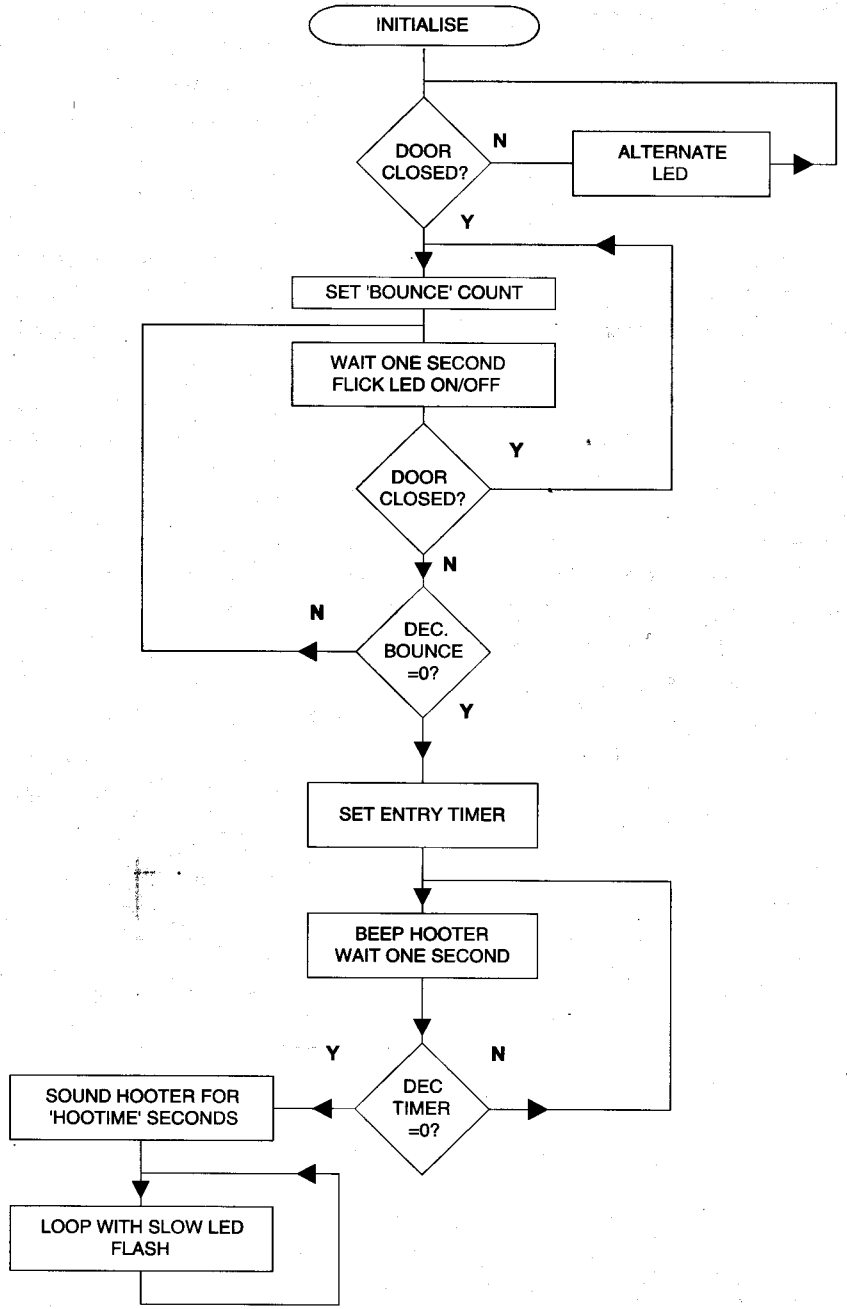


FIGURE 56

Macros

Program - none

Many programmers, both beginners and experts, never use macros and yet their use simplifies the writing and subsequent understanding of many low level functions which are implemented in micro controllers. Put in simple terms a macro is a line or a number of lines of program which can be called by a user selected name whenever that segment of program is required. Unlike subroutine calls the macro code is inserted (by the assembler) into the program at every point where the macro call is made and as we will see later this has a number of benefits and drawbacks. Macro declaration for the PIC assembler is similar to that of most assemblers in that the name of the macro is declared followed by the lines of code which make up the macro followed by a statement which ends the macro declaration. As an example the following macro consists of a single line of code which turns on an LED connected to bit 0 of port B.

```
LEDON   MACRO
        BCF     PORTB,0     ; Bit 0 of PORT B low - turns on the LED
        ENDM
```

With this macro defined at the start of the program then when the programmer needs to turn on the LED he inserts the line:

```
LEDON
```

and the assembler automatically inserts the contents of the macro, in this case BCF PORTB,0. It is obviously much easier to remember that when the LED is to be turned on the LEDON should be entered rather than BCF PORTB,0 and this is one of the advantages of using macros in assembler programming. If a second macro definition is made as:

```
LEDOFF      MACRO
            BSF  PORTB,0 ; Bit 0 of PORT B high - turns off
                           the LED
            ENDM
```

then the led can be turned on and off using simple 'high level' commands LEDON and LEDOFF. One further advantage of handling input/output in this way is that the macro definitions can be made in a block at the start of the program and if the location of the I/O pin has to be subsequently changed it can be done by one simple change of the macro definition without having to search through the code for every occasion on which the relevant pin is manipulated. (this function can of course also be achieved by equating all the i/o values rather than by using 'raw' numbers in the assembler instructions) Another advantage of using macros rather than subroutines is that the execution time is faster since the 'JUMP TO SUB ROUTINE' and 'RETURN FROM SUB ROUTINE' functions do not have to be performed every time the segment of code is to be executed. To summarise, the advantages of using macros in assembler programming are:

Simplifies calling up of commonly used segments of code using 'high level' titles.
Simplifies changing of commonly used segments of code as these are written once when the macro is defined.
Speeds execution of program segment.
Simplifies the understanding of code by the use of meaningful macro labels.

The declaration of macros will in many cases have to be done on a program by program basis since the macros will often be specific to one particular program or hardware situation. The exception is where macros perform often used functions on the processor registers themselves and in this case it is convenient to have a library of standard macros which apply to say the 16C74 processor and which can be included in any program by the use of the 'INCLUDE' statement. Two examples of macros which could be placed in such a library are given below and these are used for register bank selection and for register push and pull during interrupt.

In programs destined for PIC devices such as the 16C74 it is frequently necessary to change between register banks 0 to 1 by manipulating the RP0 bit in the status register. (bit 5) and macros can be used to simplify such operations by reducing them to simple 'English language' commands. Thus to access BANK 1, macro

'BANK1' can be defined and the code called up each time it is required by simply placing 'BANK1' in line with the rest of the program code.

```
;     BANK0 and BANK1 select register bank 0 or 1 in the 16C74

BANK0   MACRO
        BCF     STATUS,5        ; Select register bank 0
        ENDM

BANK1   MACRO
        BSF     STATUS,5        ; Select register bank 1
        ENDM
```

Micro controllers which have an interrupt structure often need a mechanism for saving and restoring the internal control registers while interrupt processing takes place and macros PUSH and PULL do just this. Note that it is necessary to have a WCOPY location in ram on both memory banks since on entering the interrupt routine the current bank status is not known and it is convenient to make these locations at the top of the ram register banks - for example at 0x7F and 0xFF in a 16C74. It is only necessary to define one of the WCOPY locations as shown, and to not use the other location.

```
;     Equate the variables at the program start - note that location
;     0xFF on BANK1 must be reserved as BANK 1 WCOPY.

      SCOPY     EQU 0x7D    ; Ram location for save of STATUS
                            register
      PCOPY     EQU 0x7E    ; Ram location for save of PCLATH
                            register
      WCOPY     EQU 0x7F    ; Ram location for save of W register

;     PUSH/PULL save/restore the W, STATUS, PCLATH registers
      during interrupt

PUSH  MACRO
      MOVWF     WCOPY       ; Save the W register on current
                            BANK
```

```
        SWAPF   STATUS,W    ; Get the status register without
                            modifying the status!
        BANK0               ; Select bank 0
        MOVWF   SCOPY       ; Save the STATUS register
        MOVFW   PCLATH
        MOVWF   PCOPY       ; Save the PCLATH register

        ENDM

PULL    MACRO
        BANK0               ; Select BANK0
        MOVFW   PCOPY
        MOVWF   PCLATH      ; Restore the PCLATH

        SWAPF   SCOPY,W     ; Get the status register
        MOVWF   STATUS      ; Restore the STATUS register

        SWAPF   WCOPY,F     ; Get W from pre PUSH bank and
                            leave Z bit as is !
        SWAPF   WCOPY,W     ; Restore the W register
        ENDM
```

There is no limit to the number of lines of code which can be placed in a macro definition but care has to be taken when labels are used within the macro since these have to be defined in a particular way (see the PIC assembler reference manual) since each time the macro is called labels have to be given different values since by definition, labels can normally only be used in one location in a given program. The down side of macros is not apparent until macros with more than a couple of lines of code are written and the macro is subsequently called several times. Each time the macro is called the code defined for the macro is inserted by the assembler into the program and if for instance the macro has 10 lines of code and is called 5 times, then each time the macro is called by a single line of assembler - 10 lines of code are inserted into the program and in this case 50 lines of code are generated. In many microprocessor programs this code expansion will not be significant but where code space is limited problems may well occur and in this case it is obviously better to define a subroutine and call this when the function is required.

Include Files

Program - none

In the same way that macros are largely ignored by many programmers- the facility of 'include' files is also greatly under used and yet the advantages which this assembler function offers should not be ignored. The principle of the include file is that a section of code can be removed from a given program and placed in a separate file which will be 'included' in the main program by a statement as:

```
INCLUDE   "C:\SOFT\TEST.ASM"
```

This statement causes the assembler to look in directory C:\SOFT\ for the file 'TEST.ASM' and it then effectively places the file in line with the rest of the program code. The procedure may at first seem to be almost too simple but in practice it can be extremely useful to the more experienced programmer who has accumulated a number of routines which he commonly uses in his programs to perform such functions as serial data communications, EEPROM reading and writing and so on. Once these routines have been thoroughly tested it is extremely useful to place them in an include file where they will reside safe from any unintentional editing or other disturbances - to be called up by an INCLUDE statement in the main program. This not only protects the precious code from unintentional changes which might occur if they were to be incorporated in the main program but it also reduces the size of the file which has to be edited when the main program is being debugged or updated. A further advantage is that a number of programs can effectively share the same segments of code since there is no limit to the number of different programs which can use the same include file within their code. As an example in the case where an application calls for serial data communications between two micro controllers running different software. The serial data communications routines are written as a file which is included in the programs which control both 'ends' of the application and then if any of the communications parameters such as baud rate have to be altered

then this is carried out once in the include file of communications routines with immediate and identical effect on both main programs. The set of routines which are included in this book for the reading and writing of EEPROM devices are presented as an include file along with a program which calls up the include file and this should be studied as an example of one way to ensure that the include file fits correctly into a given application. Programmers who write include files should remember that the file is effectively placed by the assembler into the calling file at the point of the INCLUDE statement and thus all code, labels, memory reservations etc. must comply with the normal rules and regulations which govern these aspects of programming.

The general structure of large programs can be greatly simplified if the program is broken down into segments - each of which is handled as an include file. In the case of a program with an identity of P407.ASM for instance, examination of the program listing for P407 could reveal that the file starts with the include statement which refers to the Microchip generated processor register include file P16Cxx.inc. This is then immediately followed by definitions of constants and variables used in the program and then include statements for P407MAC.ASM, P407SUB.ASM and P407TAB.ASM. These three files contain the macros, subroutines and tables for P408.ASM and partitioning these chunks of code out of the main program makes for a simpler and more secure development environment. The main body of program code is placed after the final include statement.

Serial Data Transmission and Reception - UART Software

Program - see DONGLE.ASM

Serial data communications are frequently required in micro controller systems and a general lack of understanding of the basic principles often leads to major problems when programmers attempt to write their first set of routines. In the case of micro controllers such as the 16C74 which have an in built 'Universal Asynchronous Receiver and Transmitter' or UART then the programming of registers is all that is required to allow data communications to be implemented but where a software UART has to be created then a good understanding of the principles of serial communications is vital. This project explains these principles and also describes in detail the operation of the transmit section of a software UART.

Asynchronous serial data transmission as a means of inter connecting computers with peripheral devices such as printers and modems is a means of communication that has in fact been in use in one form or another since the early telegraph systems evolved over 100 years ago. These early systems used high voltage levels to transmit data as a series of coded pulses but operated in exactly the same way as today's fast serial communications systems which operate at lower voltage levels. The world standards for today's serial data transmission systems are EIA-232-D and CCITT V24/28 and these standards define similar voltage levels as being acceptable for signalling. The receiver input voltage must be greater than +- 3 volts and in general signalling is carried out by the use of +- 12 volt levels driven from integrated circuits specifically designed for the purpose.

In order to explain how serial data transmission and reception 'works' imagine a simple communications link with a transmitter and a receiver connected by a length of wire. Figure 57 shows the voltage waveform at the transmitter during data transmission. In the idle state the transmitter outputs a MARK condition as a -ve

voltage and the system is termed a negative MARK system. When it is required to send a data byte, the output voltage is made +ve for a period which we call the element time and a START bit is sent to line (the +ve state is termed a MARK). At the receiving end of the link this change of voltage is sensed and a timer is started which has a period of 1.5 element times. Once the transmitter has sent the start bit for the correct length of time it changes the line voltage to reflect the data state of the first bit in the data byte to be sent and holds this level for one element time. The receiver samples the line when the timer expires thus reading the first data bit state at the centre point of the element and it also resets its timer to 1 element time. This process is repeated until all of the 8 data bits have been sent and received at which point the transmitter sets the line to a -ve voltage for one element time and transmits a STOP bit. Clearly if the element timers at both ends of the link are set for the same period and if the receiving circuitry knows how many data bits are to be sent in each byte then data can be simply transferred as described. Note that the inverse of the element time is the 'baud' rate, named in memory of the inventor of serial data communications systems - Baudot. Thus if the element time is 1 ms, the baud rate is 1000 baud's or bits per second. This simple principle applies on today's serial data links operating at 57600 bits per second in just the same way as it did in 1880 when data rates were 50 bits per second! In practice the content of the data byte, number of stop bits and other parameters can vary between applications and Figure 58 shows the wide range of formats which can be found. The baud rate must of course be fixed in a given system and is normally set to reflect the maximum speed which the signalling path will support.

Start bits	always 1 bit
Stop bits	1 or 2 bits, always 1.5 for 5 bit Baudot code as used in Telex systems
Data bits	5 bits for Baudot systems (ie. Telex) 6 bits for tele type setting systems 7, 8 bits for inter computer systems
Parity bits	none if no error detection is required odd or even if error detection is required

Figure 58 - common communications formats

In many applications which require serial data communications a purpose made UART device is used to perform all data communications functions and the controlling processor simply treats this as a peripheral which will deliver received data stripped of the asynchronous protocol and send transmit data complete with its encapsulation of start, stop and perhaps parity bits. As an alternative it is now common to find that a software UART has been designed into a product thus saving on the cost of a separate UART device and with the added advantage that non standard signalling formats and baud rates can be generated.

Software arts are not complex to design but the receiver requires some careful thought as to the means of timing and synchronisation since at high baud rates this can have a significant effect on the overall communications link performance. The assembler code for a soft UART transmitter routine XMIT is given in figure 59 and consists of timed 'bit bashing' of the required data out of a parallel i/o port. Figure 60 shows the flow chart for the code which is now described in detail and which is in fact used in the DONGLE.ASM program which is included with this book The data format is 1 start bit, 8 data bits and 1 stop bit and the code is designed to run in any of the PIC family of micro controllers.

Initialisation of the OPTION register is required before sending data through the software UART such that TMR0 can be used to generate exactly the correct period for each signalling element and at 1200 baud's this period is 833 us. Before calling XMIT, the W register is loaded with the data byte which is to be transmitted to line and in XMIT the data is transferred to SERIAL which is used to shift the data to the output port pin. First the number of data bits is set up - in this case 8, plus 1 for the start bit and the transmit output is then set to the start level. Note that the output from the PIC is connected to an RS232 driver which inverts the signal so that to send the standard +ve start condition we have to take the PIC port pin low. The program then jumps to XMITC and calls WAITEM macro which causes a delay of one element time to take place using TMR0. On exit from the timer loop if BITCNT is zero then the routine is exited but if non zero then BITCNT is decremented and if it is non zero a jump to XMITA takes place. On entry to XMIT the data byte

was held in register SERIAL and at XMITA this register is shifted to the right setting the carry with the state of the next data bit which is in turn moved to the transmit data i/o pin. Program flow now arrives at XMITC where as before the clock is preset with the element time and the bit is 'sent' to the transmit data i/o pin. After all 8 data bits have been sent BITCNT will decremented to zero and the last data bit is immediately followed by the stop bit which is similarly timed. Note that during the process the micro controller is totally dedicated to acting as a UART and cannot easily be used to perform other processing functions.

The instruction cycle time of the 16C57 is 1us with a 4 MHz resonator and the small timing errors introduced into the data elements have little effect at speeds up to 4800 baud's but at 9600 baud's and higher speeds it may be necessary to increase the processor clock speed to achieve an acceptable distortion level.

```
;       XMIT sends the 8 bit (or less) byte in W register to the transmit
        data port as an
;       asynchronous data byte with 1 start and 1 stop bit with the
        least significant bit first.

XMIT    MOVWF   SERIAL      ; Data shifter
        MOVLW   NUMBIT+1    ; Get the bit count + 1 for
                            start bit
        MOVWF   BITCNT      ; Preset data bit (down) counter
        TXLOW               ; Set start bit level - then send
                            start bit
        GOTO    XMITC       ; Wait for start element to go

;       Data transmit loop - set the transmit data level from the carry
        and wait for an element

XMITA   RRF     SERIAL      ; Clock shift register RIGHT
                            through carry
        SKPNC               ; If data (carry) is '0', skip
        GOTO    XMITB
        TXLOW               ; Data is '0'
        GOTO    XMITC
```



```
XMITB   TXHI                ; Data is '1'

XMITC   WAITEM              ; Wait for the element to go

;       Count the elements as they are sent

        TSTF    BITCNT      ; Zero if just sent the stop bit
        SKPNZ               ; Skip next if bit count is not zero
        RETURN              ; Exit from XMIT

        DECFSZ  BITCNT      ; Dec. bit count, skip if zero
        GOTO    XMITA       ; Loop until all bits are sent

;       Bit count has zeroed, send the stop bit

        TXHI                ; Set stop bit
        GOTO    XMITC       ; Wait for the stop bit to go
```

Figure 59 - assembler code for software transmit UART subroutine

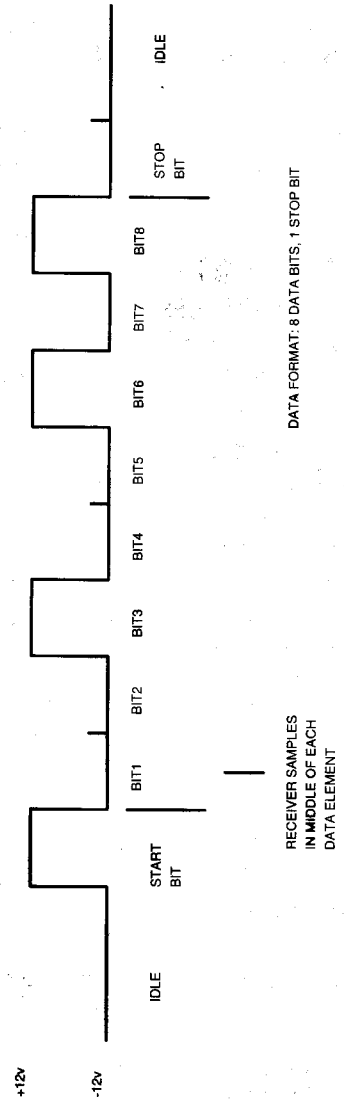


FIGURE 57

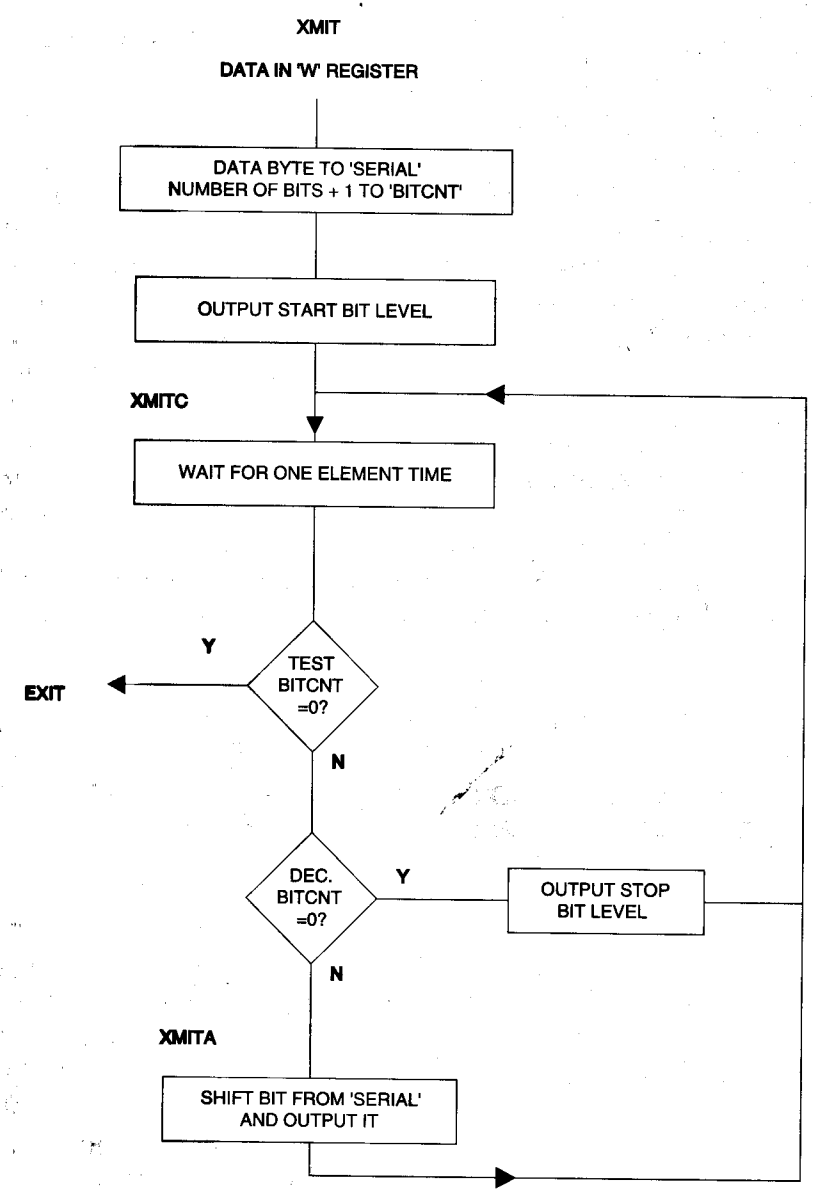


FIGURE 60

String Lookup and Display

Program - see DOTLCD.ASM

The architecture of PIC micro controllers does not allow the direct reading of program memory and the storage of text strings is achieved by placing a series of RETLW instructions in such a way that by adding an offset to the program counter - the value appended to the RETLW instruction can be retrieved in the W register. This simple function is however complicated by the memory paging structure of the PIC and this project describes the a program to run in a 16C74 which allows unlimited access to strings of data or text. The code also allows two instruction line calling of a routine to output a string to a dot matrix lcd display and should be studied in association with the LCD dot matrix drive project since they share a common program.

The following code demonstrates table entry retrieval and when CALL TABLE is executed will cause the ASCII value for 'C' to be returned at label JILL

```
FRED    MOVLW   .2
        MOVWF   OFFSET
        CALL    TABLE       ; Get a table value
JILL    NOP                 ; W register has ASCII 'C'
.
.
.
TABLE
        MOVFW   OFFSET
        ADDWF   PC
        RETLW   'A'
        RETLW   'B'
        RETLW   'C'
        RETLW   'D'
```

If the value OFFSET is saved and incremented before each call to TABLE then it is simple to step down the table of values and retrieve

the string which could then be fed to say an lcd display. In practice unless some careful steps are taken to look after the PCLATH register - the return from the TABLE call will be at best unpredictable. The code provided allows for the simplest possible calling for string display on the lcd - the following two lines of code being all that is required in the main body of code - to cause the required string to be displayed.

```
MOVLW   .2      ; select the string -
CALL    STRING  ; - and display it.
```

Looking now closely at the code, note that STRING is a subroutine on the same ROM page as the main code body which stores the string number in STRNUM, resets the offset register TABOFF (in macro TABSET) and then after selecting ROM page 1, calls DOSTR.. This subroutine first resets the PCLATH since it is looped through every time a string character is to be retrieved and if the PCLATH is not set up the program will crash. The W register is loaded with the string identity and a call is made to CALLSTR which is a simple jump table to the actual strings themselves. Thus if W is 2 on entry, a jump to MSG2 will take place.

At the start of each message string - MSG1 MSG2 etc. a macro is called which adjusts the PCLATH according to the ROM area on which the table is placed, loads the table offset value and adds it to PCL (program counter low). This causes flow to jump to the next location in the string and the resultant RETLW will return a value in the W register to subroutine DOSTR where a call to CHALCD puts the character to the lcd display. Note that in the message strings that the last entry has the value LAST (0x80) added to it to set the top bit. This allows a simple test of the state of the top bit to be used to identify the end of the string. When this occurs - the loop through DOSTR is terminated by a RETURN command which returns flow to STRING where the PCLATH is adjusted and exit takes place back to the main code block.

Dot Matrix LCD Driver

Program - DOTLCD.ASM **16C74**

A large number of imbedded controller designs make use of standard dot matrix lcd displays since these provide a relatively simple way of displaying information to the user. The implementation of the interface to these displays is however not simple since it is necessary to pass control codes to the display in addition to the data which is to be displayed. The accompanying code offers a full set of routines for controlling and displaying data on a 2 line x 16 character lcd display and will work with any of the standard dot matrix displays which use the Hitachi lcd controller chip family. Typical examples from Hitachi are the LM016XMBL or LM016L or the equivalent from Vikay.

The interface connections to the lcd panel consist of a parallel data path and 3 control lines. The parallel data path can be of either 4 or 8 lines and in the case of DOTLCD.ASM the interface is operated in the 4 line mode since this gives the most economical use of port pins at the expense of only a few lines of code. To operate in this mode, display data bit lines DB0-3 are connected to +5 volts and data is transferred to the display using DB4-7 only.

The control signals are as follows:

RS - register select, held HIGH for data transfer and LOW for command transfer.
R/W - read write control, HIGH for read and low for write.
EN - enable which is a clock signal to the display.

In practice the R/W line is only needed if it is necessary to read back from the display's internal character generator memory or check the 'busy' state and in this implementation the R/W line is held low at all times.

The interface connections are shown in figure 61 and as can be seen port A of a 16C74 is used to control the display using only 6 port

pins. The software for controlling the display can be found on disk as DOTLCD.ASM and the following description relates to that program. A flowchart for the program is given in figure 62.

Interface control - in general terms the interface is operated by setting the desired control or data byte on the data lines and setting the RS line according to whether a command or data transfer is to take place. The EN (or clock) line is then pulsed high and low which clocks the byte on the data lines into the display and in the macro section of the program CLKLCD provides code to carry out this simple clock function. Since the display is operating in 4 line mode the 8 bit control/data bits are transmitted in two 4 bit nibbles with the top nibble being sent first.

Subroutines are provided which take a value in register LCDCH and then write this to the lcd:

CHALCD writes the character value in the W register to the display.

COMLCD writes the command value in the W register to the display.

These routines operate in a similar fashion and the operation of CHALCD will now be described. The first action is to move the top nibble of CHALCD into the bottom position of the W register and to 'and' the value to effectively strip off the top bit of the byte. This is done since when display characters are stored as part of a string the last character has the top bit set to flag that this IS the last character in the string. (see article on string storage). The nibble is then stored on the port A interface to the lcd and the RS bit is set high to signal that a data byte is being sent. The lcd is then clocked using the CLKLCD macro and a similar process takes place to clock the lower nibble to the lcd. Note that after the two nibbles have been transferred - that the RS line is left in the 'control' state. A time delay must now be carried out before the lcd is ready to accept the next control or data byte and this is achieved by jumping to LCD64 which gives a 64 us delay before exiting from the CHALCD subroutine. Note that display characters must be in ASCII format

which suits the mode of operation of most assemblers. Thus to load the character 'Z' into the W register the instruction MOVLW 'Z' will cause MPASM to load the ASCII code for 'Z' (0x5A) into the W register.

Initialisation - when the display powers up it has to be initialised as to the number of display lines, number of characters, number of data lines and a host of other parameters. The display is initially reset to have an 8 line interface and it is necessary in this application to first set the interface to 4 lines and this is done following the INITA label. The device data sheet calls for particular commands to be sent several times with delays between each transmission and this is achieved by in-line code starting at INITA. The code which follows down to IDLE then sends the various commands to the display which set up the display size, cursor control etc. and if it is required to change these settings then it will be necessary to consult the lcd manufacturers data sheets.

Application - subroutines CHALCD and COMLCD have already been described and in simple terms these are all that would be required in order to use the lcd. In practice it is easier to have a small set of subroutines which perform display reset, cursor control etc. and these are included as follows;

PUTLCD - allows an ASCII character to be loaded into the W register and when this subroutine is called - the character will be transferred to the display.

LCDCLR - is called with no parameters and clears the lcd, leaving the cursor (not displayed) at the left hand end of the top line of the display.

CUR1 - allows the position of the next display character on line 1 to be selected. The W register is loaded with a value between 0 and 15 which specifies the position on line 1 and the subroutine is called.

CUR2 - allows the position of the next display character on line 2 to be selected and functions in a similar way to CUR1.

The test code shown relies on a set of strings which are stored as part of this program and the article on string storage and retrieval explains the format used. As can be seen - with the assistance of the macros and subroutines provided, using the lcd is extremely simple. Reset of the display is achieved by a single line command - CALL LCDCLR, cursor positioning is achieved by 2 lines of code and string output again by only 2 lines of code.

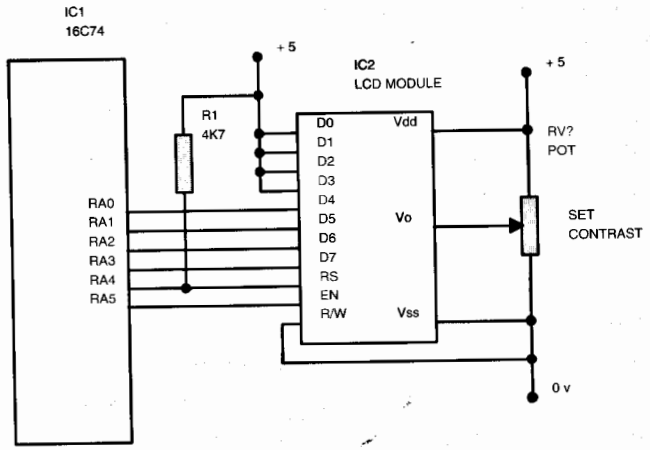


FIGURE 61

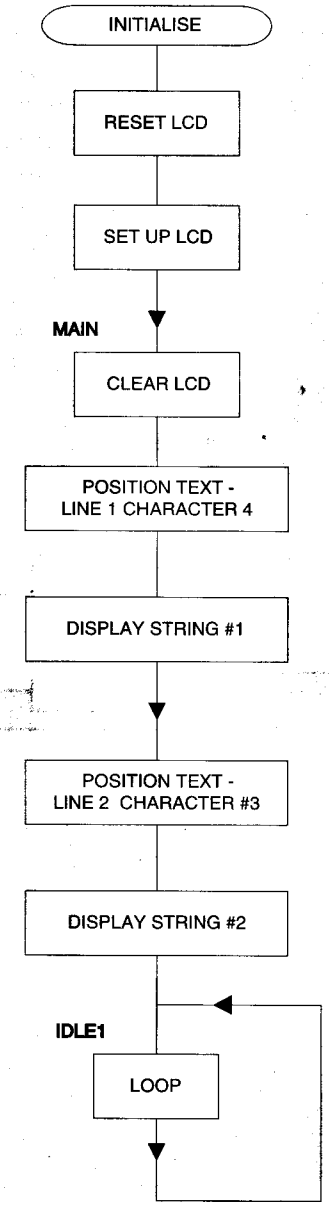


FIGURE 62

Real Time Clock Interface

Program - CLOCK.ASM **16C74**

Many designs which utilise a PIC processor also require a real time clock and whilst this function can be achieved in software within the processor there are often good reasons why an external device is a better option. In terms of ease of use and low cost one of the current 8 pin serial devices is best suited to the job but many designers are wary of embarking on the design due to the need to write serial communications software to 'talk' to the real time clock chip over its 2 or 3 wire bus. The description which follows and the accompanying source code fully covers the design, use and implementation of a serial real time clock interface to the Dallas Semiconductor Corporation DS1302 device. Figure 63 shows the circuit of the real time clock chip and its connection to the 16C74 while figures 64 and 65 contain the flowcharts for reading and writing the clock chip.

The DS1302 is a fully featured part which performs all the usual clock and calendar functions including seconds, minutes, hours, months and years along with date and day of the week registers. It can be programmed to operate in 12 or 24 hour mode and runs from either the system 5 volt supply or if this fails from a backup battery and there is even provision for software selection of a charging circuit for the backup battery should this be required! In addition to the clock functions the device has on board 31 bytes of 8 bit ram which - like the clock is non volatile as long as either the 5 volt rail or a backup battery are connected.

Connection to the host PIC device (in this case a 16C74) is by means of three lines:

SCLK - is a signal which is generated by the PIC and is used to clock commands and data to the clock chip and also to clock data out of the clock chip.

I/O - is the data line which is used to transfer data from the PIC into

the clock chip and from the clock chip to the PIC. Note that this line is bi-directional so the software must be able to 'turn round' the PIC i/o port pin during the read operation.

RST/ - is a control line which is used to initiate and terminate transfers to and from the clock chip.

In order to communicate with the device a logic level is applied on the PIC i/o pin and the clock line is taken high then low. This clocks the data bit into the device and by repeating the process an 8 bit word can be transferred. Thus when it is desired to write to the hours register, the command:

b'1 0 0 0 0 1 0 0'

is sent. Bit 0 set to zero signifies that a write is required and bits 1 to 6 specify that this is to the hours register. (bit 7 is always set). The command is followed by clocking from the micro controller to the rtc chip the 8 bit data byte which is to be written - in this case to the hours register. If a read is required from the hours register then the same command is sent but with bit zero set and in this case before the 8 clock pulses are sent to clock the data byte from the clock chip, the i/o line on the PIC must be turned into an input. This allows the PIC to read the data bits as each clock pulse is sent to the clock chip. The other registers in the device are accessed in a similar manner and the DS1302 data sheet has a table showing the commands which have to be used in each case. Note that data is stored in the clock chip in BCD format with 2 BCD bytes per 8 bit word and that in some cases bits may be used to flag a particular mode of operation - for example bit 7 in the hours register is set for 12 hour operation and clear for 24 hour operation. Thus if the time is 12:47 the values in the hours and minutes clock registers will be 0x12 and 0x47.

The accompanying software sets up registers in the PIC for days, hours, minutes and seconds and allows the settings in these registers to be transferred to the clock chip and for it to be restarted (with the seconds zeroed). Code is also provided to allow the appropriate registers in the clock chip to be read back to the PIC registers. Two further routines are provided which allow 31 bytes of data to be read from registers in the PIC and to be stored in the

clock chip and for these bytes to subsequently be read back into the PIC (NVREAD and NVWRITE): these are fully functional but they will not be described in detail here as they operate in a similar fashion to the real time clock subroutines.

Real time clock routine initialisation.

Some of the more important parts of the initialisation code may be broken down and analysed as follows:

Equates - these specify the port pin allocations and also the commands which are loaded into the clock chip. If it is required to access other registers in the device then it will be necessary to consult the DS1302 data sheet for details.

Memory equates - these specify the registers which are to be used to mirror the real time clock registers.

Macros - these are used to simplify writing and understanding of the code since is obviously much easier to remember that 'RTCDLO' sets the RTC Data line low than to remember the assembler command 'BCF RTCPORT,RTCIO'.

Real time clock write routine RTCPUT transfers the following registers into the corresponding registers in the clock chip:

RTCD - day setting as a digit in the range 1 to 7.
RTCH - hours setting as two BCD bytes in the range 0 to 2 and 0 to 3
RTCM - minutes setting as two BCD bytes in the range 0 to 5 and 0 to 9
RTCS - seconds setting. Note that in this application that when the clock is written that the seconds are zeroed to give a clock start at the preset hours/minutes etc.

To call the routine it is only necessary to set up these registers as required and CALL RTCPUT. The clock will then start with the preset days, hours and minutes and with the seconds set to zero.

Looking in detail at the routine it can be seen that on entry the

clock is stopped using subroutine CLKSTOP which uses a general purpose subroutine RTCTX to transmit the appropriate command to the clock chip. The command DAYWR is then loaded into register SERIAL which is used as a general purpose shift register to allow data to be moved out of and into the PIC i/o pin. Data value RTCD is then loaded into a temporary holding register GP1 and subroutine RTCWR called to cause the command and data bytes to be transferred to the clock chip. This process of loading the command to SERIAL and the data to GP1 is repeated and RTCWR called until the required data has been transferred to the clock chip. Note that the final command - SECWR has its flag bits set to cause the clock chip to be restarted with the seconds set to zero and when this command has been sent the transfer of data is complete and the clock chip will start to run from the required time.

Real time clock read routine RTCGET transfers the clock chip registers discussed above from the clock chip to the corresponding registers in the PIC processor. To call the routine it is only necessary to CALL RTCGET and transfer of the registers from the clock chip to the PIC will occur.

Looking in detail at the routine it can be seen that command SECRD is loaded into register SERIAL and subroutine RTCRD is called. In RTCRD the command is sent to the clock chip using RTCTX and then the i/o line is turned to an input and subroutine RTXRX is used to receive the contents of the clock chip seconds register into SERIAL. After storing the data in RTCS the process is repeated in a similar fashion until the minutes, hours and days have been transferred.

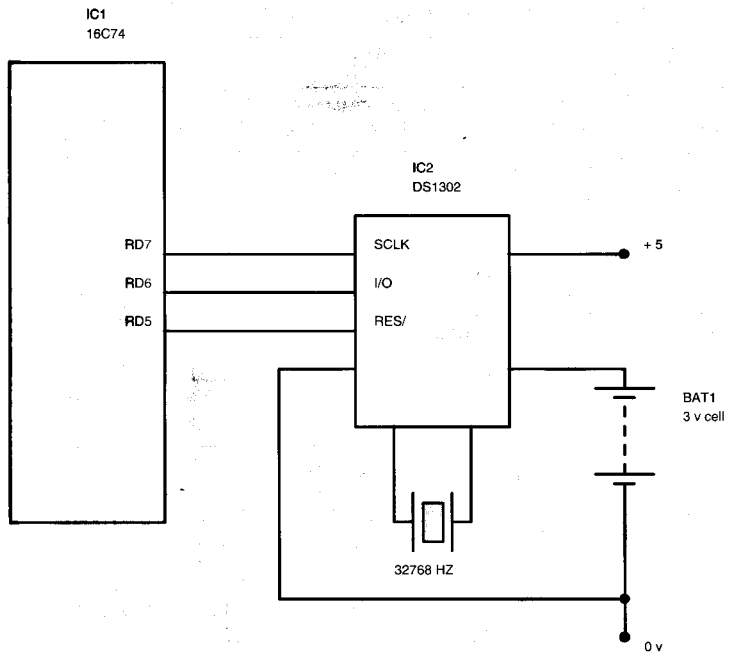


FIGURE 63

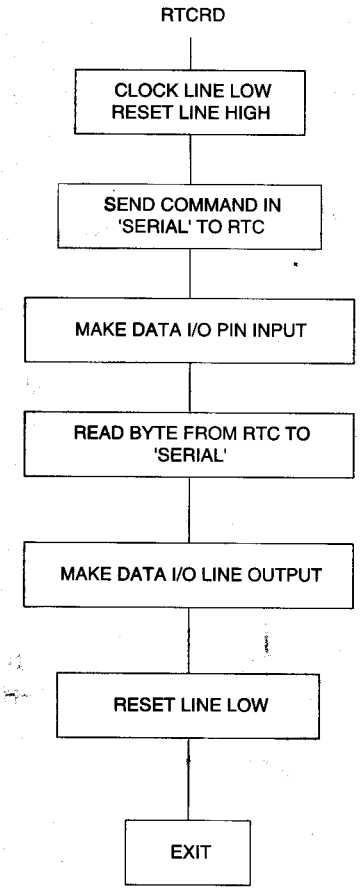


FIGURE 64 REAL TIME CLOCK WRITE

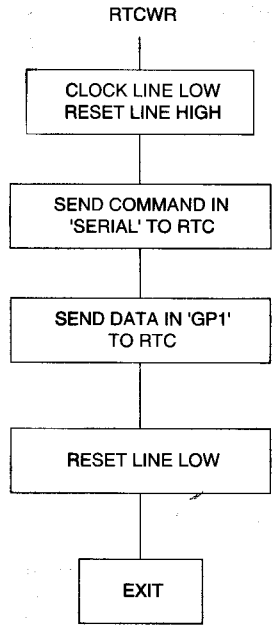


FIGURE 65 REAL TIME CLOCK READ

Interrupt Driven Keypad

Program - KEYPAD.ASM **16C74**

Many micro controller applications require a key pad and it is useful to be able to 'hide' the code in an interrupt routine such that the foreground program has simply to test a flag to find out if a character is waiting to be serviced and if the flag is set then it can pick the character up from a known location. The program given in this project runs in a 16C74 and handles a 16 key pad, returning the key hit in coded form for ease of parsing in the foreground program.

The keypad circuit is shown in figure 66 and consists of a standard 4x4 matrix with the four rows connected to port pins B4 - 7 and the four columns connected to B0 - 3. The port pins driving the columns are configured as outputs and the row pins as inputs with the 'weak' pull up resistors enabled on these latter inputs by setting bit 7 in the OPTION register and writing logic '1' to port bits B4 - 7. The interrupt control register INTCON is configured to respond to any change on port pins B0 - 3 and the global interrupt is enabled. Timers TMR0 and TMR2 are both used in this application: TMR0 is used to time a sounder which gives key 'feedback' and TMR2 is used to time the key debounce.

In the idle mode the column lines (outputs) are all held low and the row lines (inputs) are held in the high state by their internal pullups. When a key is pressed one of the row inputs will be taken low by the cross connected column line and this 'change on PORT B' will cause an interrupt. The processor in the mean time will have been performing a fore ground task which will be interrupted by the key hit - causing the program to enter the interrupt routine. In order to ensure that the foreground process is not interfered with it is essential in any system which employs interrupts that the necessary control registers be stored away at the start of the interrupt and restored on exit from the interrupt. Macro PUSH saves the W, STATUS and PCLATH registers on entry to the interrupt and just before the exit 'RETFIE' instruction is executed the PULL macro

restores these registers. It is worth studying the code used in these macros since it is essential that care is taken writing the push and pull code since the status register must not corrupted on exit from the interrupt.. After the PUSH macro has been executed the interrupt flags are examined and if the 'change on port B' interrupt flag is set then the key pad process is entered at label INTA.

The keypad flow chart is shown in figure 67 and key handling starts by running a timer which is designed to allow the key press to 'settle' for a few tens of milli seconds before establishing a firm contact. TMR2 is used for this debounce and at label INTA the debounce time is set in increments of 1/4 of a second to - in this case 40 milli seconds. After the debounce timer has expired each row of the keypad is checked in turn to test if a key is still down and if no key is operated then an exit occurs via label INTX. At INTB1, with a key down the key code starts to be formed by placing a value of 4 in KEYVAL - the register in which the key scan result will be generated. The column levels are now adjusted such that only column 0 is low and columns 1 - 3 are high and the rows are tested until one is found to be low (or none as the case may be). If a given row is found to be low then a jump occurs to either INTB3, 4, 5 or 6 where a value of 0, 4, 8 or 12 is (effectively) added to KEYVAL and the scan is complete. This will result in a key value at INTB3 of 4, 8, 12 or 16. If no key is down then KEYVAL is decremented and the column levels are adjusted such that only column 1 is low and columns 0, 2 and 3 are high and the rows are again tested until one is found to be low: on this occasion if a row is found to be low then the key values at INTB3 will be 3, 7, 11 or 15. This process is continued until a row is found to be low and the result is that at INTB3, KEYVAL has a value of from 1 to 16 dependant on the key which has been pressed. In the event that no row is found to be low due to the key being released then the column scan is set idle by macro KPIDLE and the interrupt routine is exited via INTX.
With a unique value in KEYVAL (at label INTB3) it would be possible to set a flag and exit but to make the job of parsing the key hit easier a lookup table is used to convert the scan output value into a more meaningful form. Thus for instance if there are ten digit keys on the pad it makes sense to have these represented by values which correspond to their key identity and a further enhancement

is to set the top bits of all non numeric keys since these will probably be 'command' keys such as ENTER, SCROLL etc. which require different handling to digit entries. Table KEYCON is used to convert the scan value to the required output value and consists of a set of RETLW instructions which will return the equated values shown. Note that in the case of command keys that the assembler adds COMMND to the equated values - thus setting the top bit as described above. With the key hit value processed and placed in KEYVAL, one last task remains - to set FLAG,KEYHIT which can be picked up by the foreground program. The W, STATUS and PCLATH are restored by the PULL macro and the interrupt is then quit.

In the foreground program when it is required to wait for a key entry it is now only necessary to form a code loop which calls subroutine GETKEY and this process is shown at label IDLE. Subroutine GETKEY tests FLAG,KEYHIT and if a hit is seen, it sounds a key feedback 'beep', clears the flag and exits with the carry SET. On return from GETKEY the code then tests the top bit of KEYVAL and if this is set then this is a command and at IDLEA parsing can commence. If the key is a number then a branch to IDLEB occurs where the number can be handled.

Note that since the interrupt on change on port B occurs on both rising and falling edges that when the key is released, the interrupt will occur again but ***that as the key IS released then*** after the debounce timer has elapsed, the tests ***for a row drive seen low will fall and the interrupt will immediately be quit and a 'no key press' result returned.***

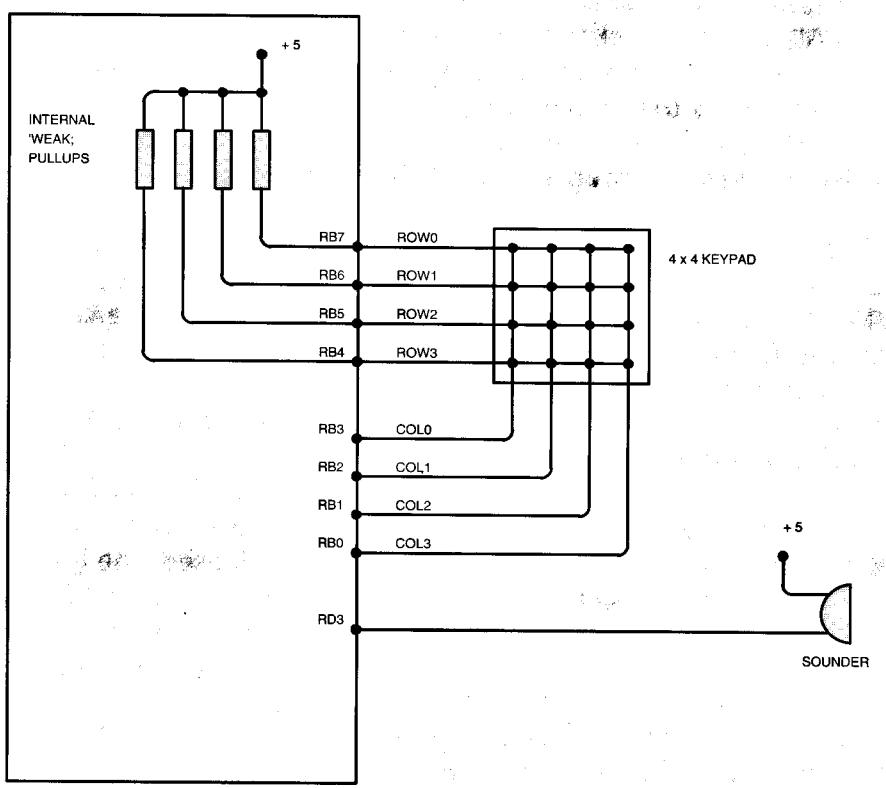


FIGURE 66

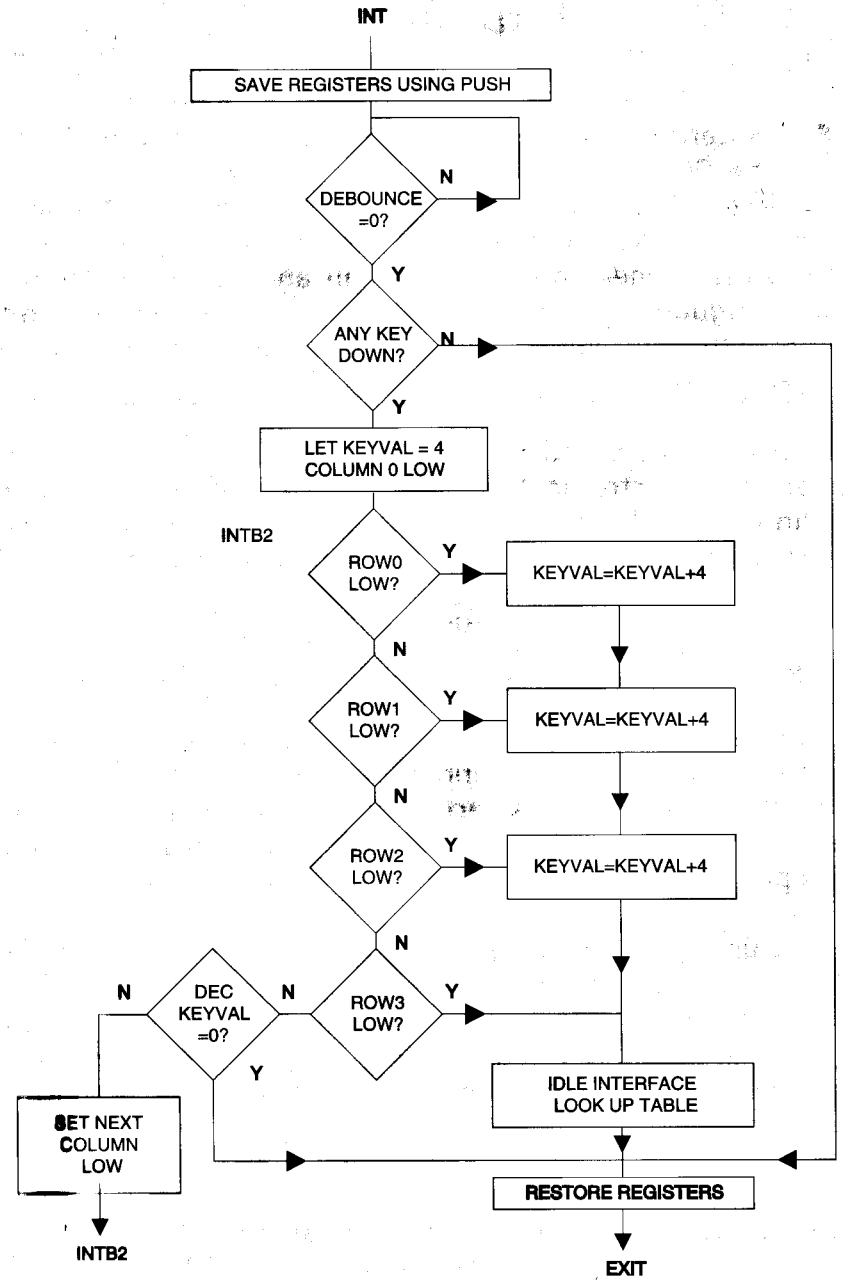


FIGURE 67 INTERRUPT KEYPAD

Authors Information

Nigel Gardner (AMIEE, FInstSMM) is an Electronic Engineer of some 20 years industrial experience. He has worked for Siliconix, National Semiconductor, Zilog and Spectrol Reliance. His design experience covers Analog, Digital, RF, Microprocessor, Pneumatics, PCB Design and Software writing in various languages. He has 2 patent applications accredited together with 13 technical articles published and this is his 3rd book on the PIC Microcontroller.

Nigel currently owns and runs Bluebird Electronics - working as an Independent Electronic Research and Development Engineer providing PIC training and Support, Microprocessor Hardware/Software solutions, and Technical Authorship.

When time and weather permit Nigel enjoys flying his microlight aircraft.

Peter Birnie has been working with microprocessors since their introduction in 1975 beginning with the 6800 family . He has worked for British Telecomm and DCE developing data communication systems for Prestel, together with the Telexbox and Faxbox products.

Peter set up his own business, PB Micro Designs, to develop and market the NewsNet and PictureNet products which are used throughout Europe's newspaper industry. The business currently designs products based on a wide range of Microcontrollers including the Arizona Microchip PIC and many others. Application expertise covers all aspects of hardware design and the writing of software in assembler, C++ and other languages.

Peter unwinds by spending time on his boat.

The following products are available to speed up PIC development but need to be used in conjunction with some form of programmer (PIC START) or In Circuit Emulator (PIC MASTER) if you wish to progress beyond just simulation.

Beginners Guide to the Microchip PIC
A comprehensive introduction to the PIC, how it works and how to use it. Source code examples, simulator, assembler and projects on 3.5" disk.
FEC 489-359

16C54/56/58/61/71/84 Hardware Starter Kit
PCB, 8 leds, regulator, 4MHz crystal, reset components.
FEC 459-884

16C55/57 Hardware Starter Kit
PCB, 8 leds, regulator, 4MHz crystal, reset components.
FEC 459-896

Project Board (18 pin)
PCB, leds, regulator, push buttons, speaker, thermistor, dip switch, PP3 battery holder, 4MHz crystal, reset components.
FEC 527-646

16C64/74 Hardware Starter Kit
PCB, leds, regulator, push buttons, RS232 interface, EEPROM socket, 200 hole patch area, PIC Soft disk (631-000).
FEC 630-639

PIC Soft
Project board software source code, application source code from Microchip Embedded Control Handbook plus assembler, simulator and editor - replaces project software pack listed on project board box with enhancements.
FEC 631-000

The above products are available from:

Farnell Electronic Components Ltd
Canal Road
Leeds LS12 2TU
TEL 0113 263 6311 FAX 0113 263 3411