MICROCONTROLLER-BASED ACCESS CONTROL SYSTEM

VINAY CHADDHA*

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

System overview

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

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

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

Hardware details

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

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

- Eleven bidirectional input/output (I/O) pins
- 1240 bytes of OTPROM program memory
- 64 bytes of user RAM
- 15-stage multiple-function timer

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

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

This I²C bus-compatible 2048-bit (2-kbit) EEPROM is organised as 256x8 bits. It can retain data for more than ten years. Using just two lines (SCL and SDA) of the memory, the microcontroller can read and write the bytes corresponding to the data required to be stored.

(***Note***. For details of the microcontroller and programming of I²C EEPROM, you may refer to the article 'Caller ID Unit Using Microcontroller' published in April '99 issue of EFY and the article 'Remote-controlled Audio Processor Using Microcontroller' published in Sep. '99 issue of EFY. For additional data on Motorola microcontrollers, refer to EFY-CDs of this year's January and February issues. The information pertaining to I²C EEPROM is available on STMicroelectronics' Website.)

Power supply. The power supply unit provides a constant 5V supply to the entire unit. This is a conventional circuit us-

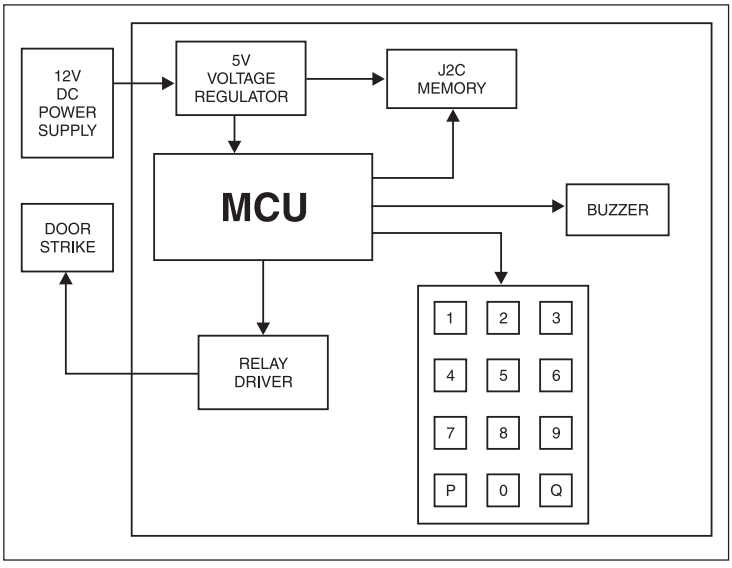

Fig. 1: Block diagram of the access-control system

PARTS LIST

Semiconductors:

IC1(U1)	- MC68HC705KJ1 microcontroller
IC2 (U2)	- ST24C02 I2C EEPROM
IC3 (MN1)	- MN1280 reset stabiliser
IC4 (Reg1)	- 7805 +5V regulator
T1, T2 (Q1, Q2)	- BC547 npn transistor
D1, D2	- 1N4007 rectifier diode
LED1	- Red LED

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

R1-R6	- 10-kilo-ohm
R7-R9	- 1-kilo-ohm

Capacitors:

C1, C2 C3, C4,	- 33pF ceramic disk
C6, C7	- 0.1µF ceramic disk
C5	- 10µF, 10V electrolytic

Miscellaneous:

Xtal (Y1)	- 4MHz quartz crystal
PZ1 (BZ1)	- Ceramic piezo buzzer
Con1	- Power-supply connector
Con2	- 2-pin male/female Berg connectors - 7-pin male/female Berg connectors
SW1-SW12	- Tactile keyboard switch
RL1 (RLY1)	- 1C/O, 12V, 250-ohm miniature relay

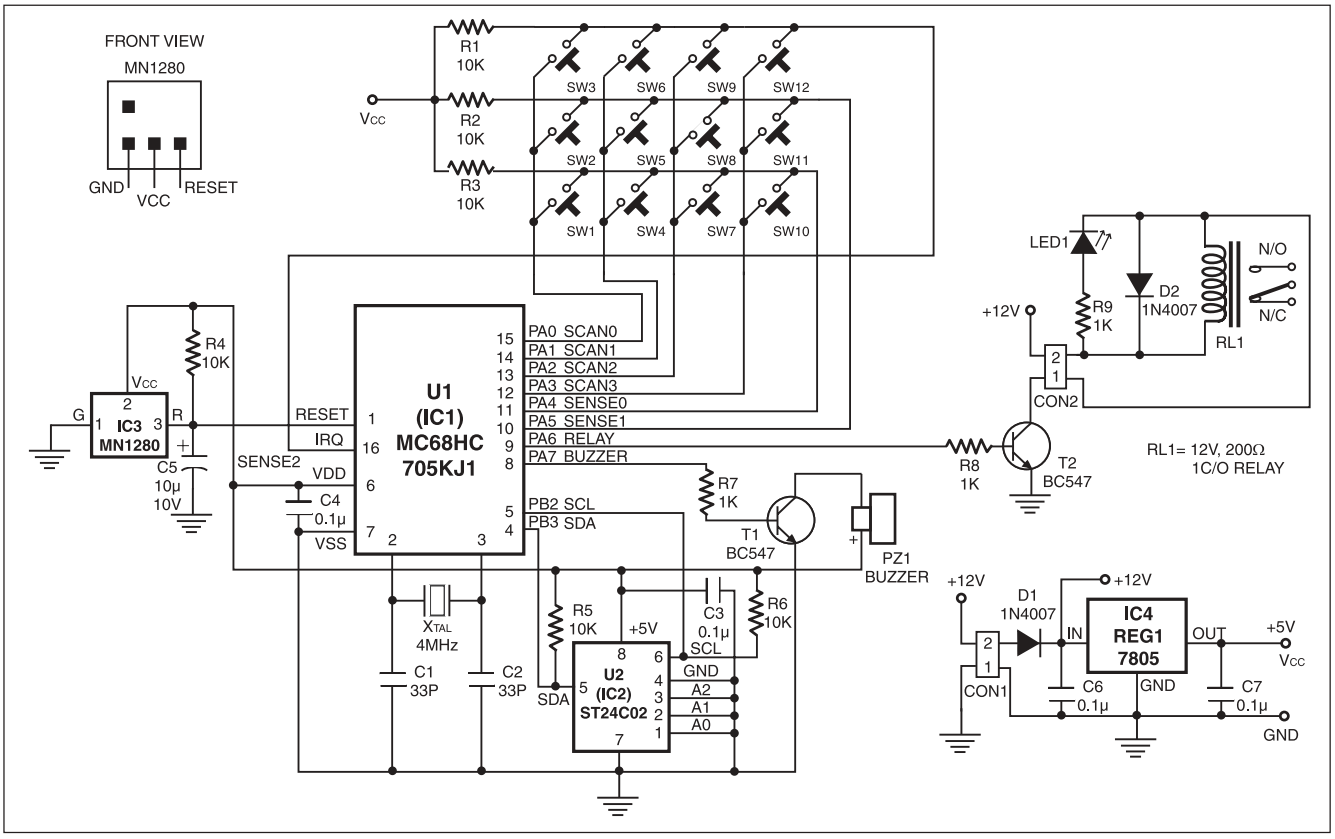

Fig. 2: Schematic diagram of the access-control system

ing external 12V DC adaptor and fixed 3-pin voltage regulator 7805. Diode D1 is used in series with 12V input to avoid damage to the unit in case reverse voltage is applied by mistake.

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

The keypad is arranged in a 3x4 matrix. There are four scan lines/pins, which are set in output mode, and three sense keys, which are used as input lines to the microcontroller.

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

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

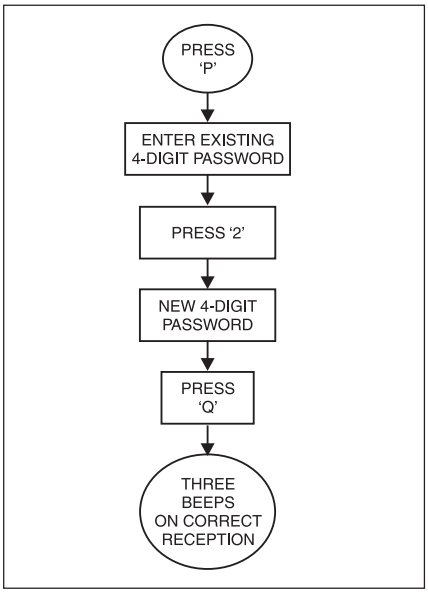

Fig. 3: Flow-chart for changing the password

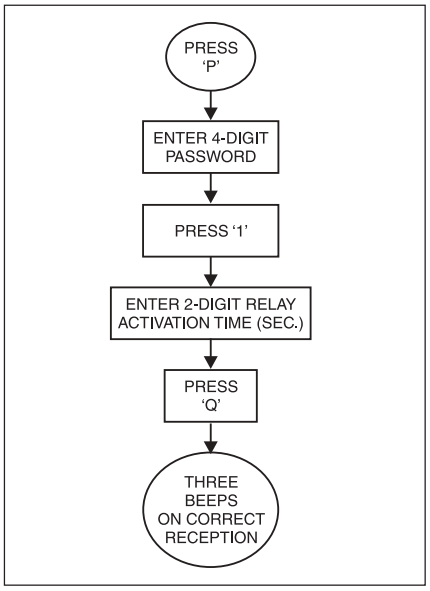

Fig. 4: Flow-chart for changing the relay-activation duration

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

Relay. A single-pole double-throw (SPDT) relay is connected to pin 9 of the microcontroller through a driver transistor. The relay requires 12 volts at a cur-

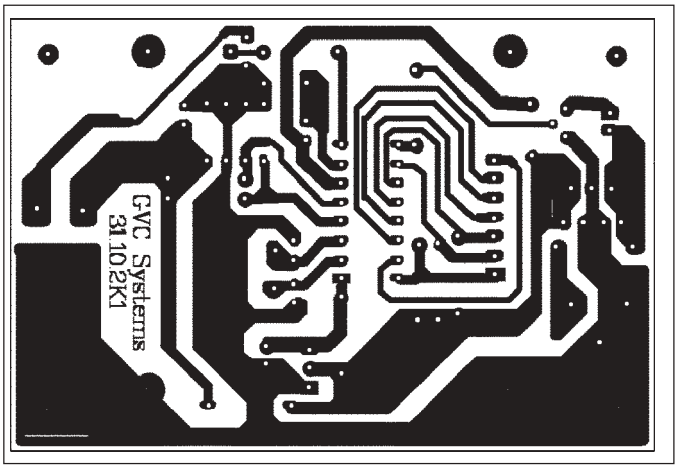

Fig. 5: Actual-size, single-side PCB for the access-control system without keypad (Main PCB)

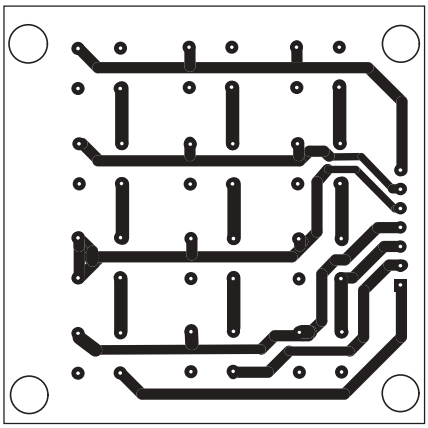

Fig. 6: Actual-size, single-side PCB for the keypad

rent of around 50 mA, which cannot be provided by the microcontroller. So the driver transistor is added. The relay is used to operate the external solenoid forming part of a locking device or for operating any other electrical device. Normally, the relay remains off. As soon as pin 9 of the microcontroller goes high, the relay operates.

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

Operation

The complete design is based on two parameters: the password and the relay-activation time duration. Both these parameters can be changed without making any change in the hardware. The user can change these parameters any number of times using the keypad. The flow-charts for changing the password and relay-activation time duration are shown in Figs 3 and 4, respectively.

Testing

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

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

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

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

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

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

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

Operation

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

Change the password and the relay time. Check the op-

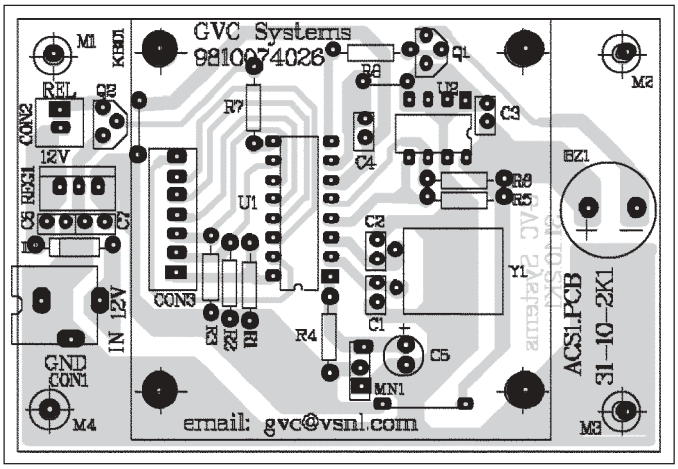

Fig. 7: Component layout for the PCB in Fig. 5

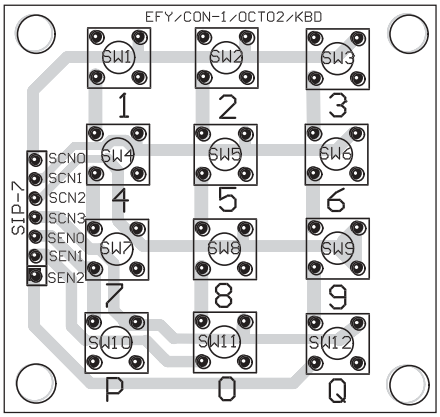

Fig. 8: Component layout for the PCB in Fig. 6

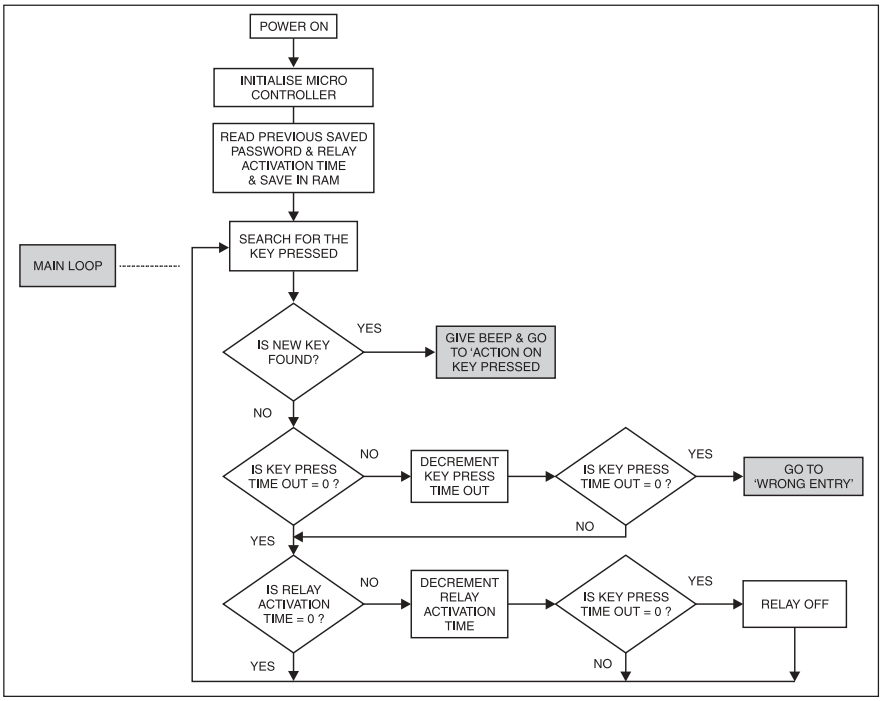

Fig. 9(a): Flow-chart for the access-control system, continued in Figs 9(b) and 9(c)

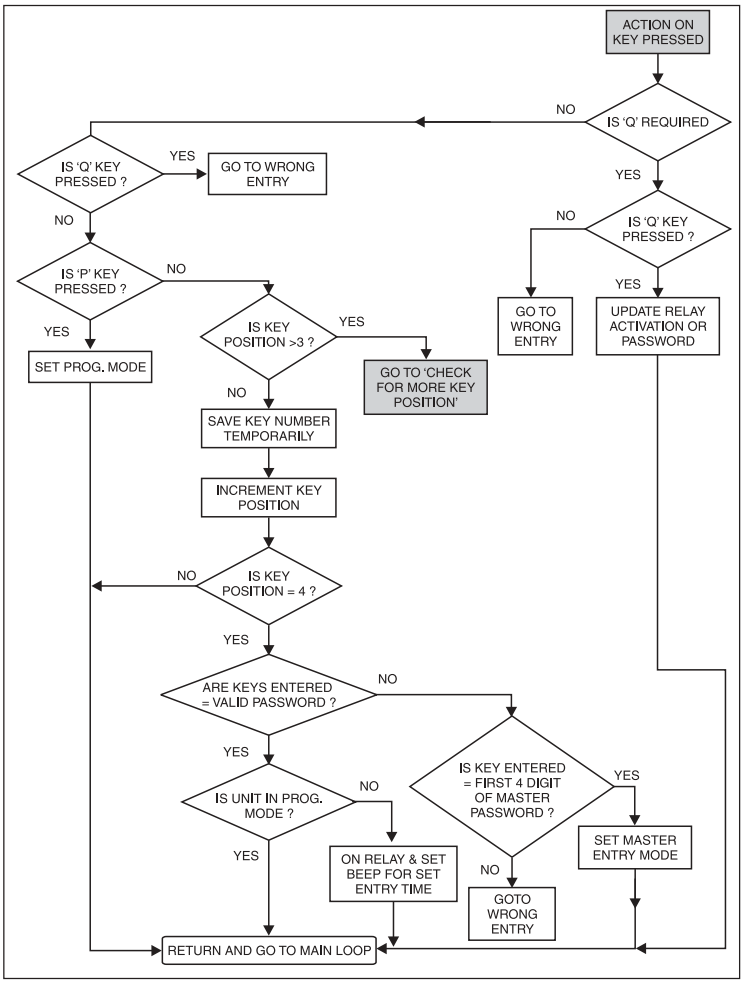

Fig. 9(b): Flow-chart for the access-control system, continued from Fig. 9(a)

eration with new password and relay activation period. Since there is no memory, the new password and relay time entered will be lost as soon as you switch off the unit. The next time you switch on the unit, the password is again set to 1111 and the relay time to 10 seconds as default parameters.

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

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

The software

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

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

(***Note***. The books (in pdf format) and WinIDE software are available free of cost on Motorola's Website and have been reproduced by courtesy of Motorola in EFY-CDs of this year's January and February issues. The mentioned CDs also contain DOS-based programs. The JICS board may be bought from Motorola's authorised distributors.)

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

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

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

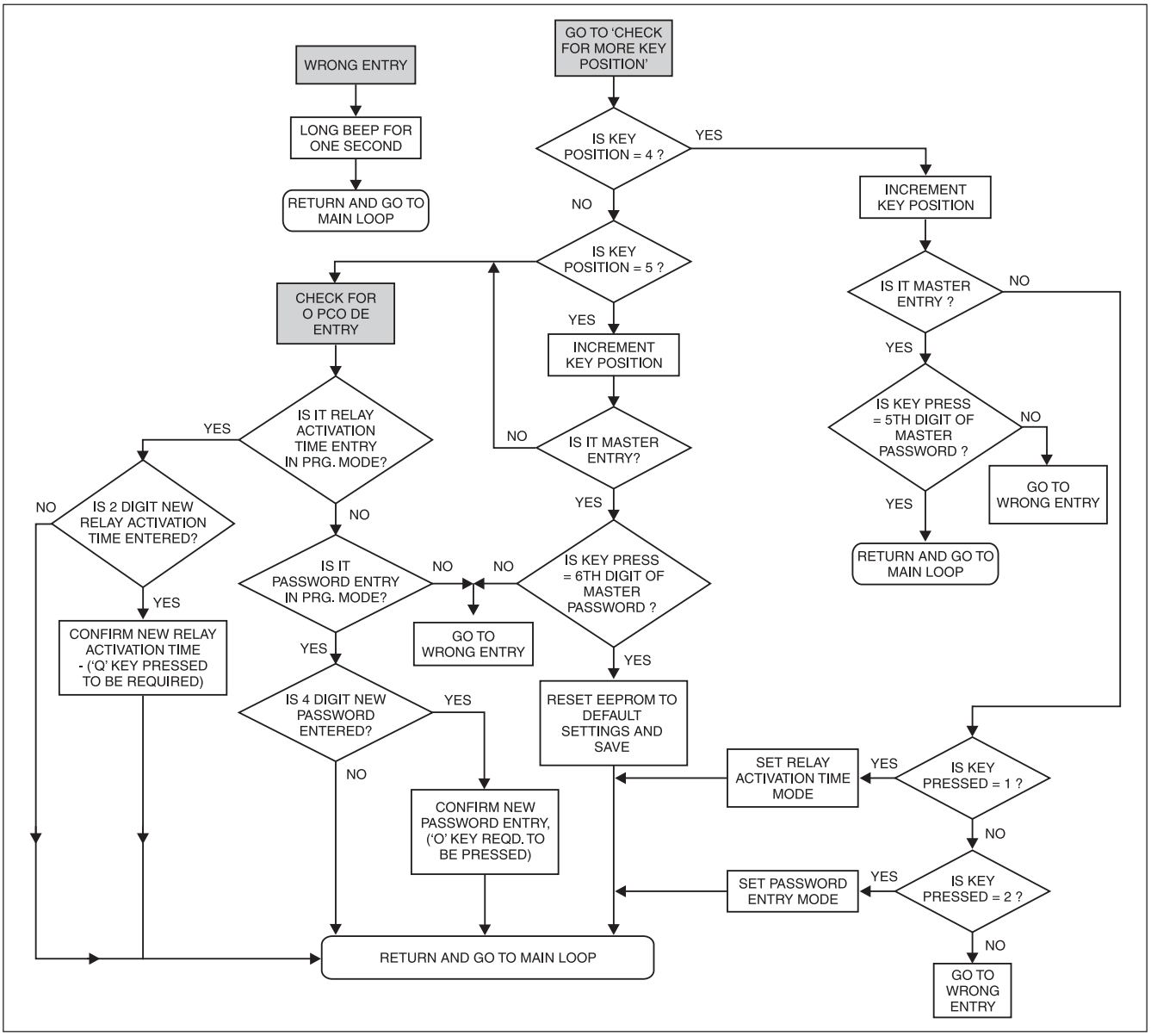

Fig. 9(c): Flow-chart for the access-control system, continued from Fig. 9(b)

Assembly-level program is to be saved in a file with .ASM extension.

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

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

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

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

Possible modifications

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

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

MAIN.ASM

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

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

et_buff     db          2

org         300h

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

start:      rsp

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

   init_port    ddra        ;; initialise port a
   init_port    porta

   init_port    ddrb        ;; initialise port b
   init_port    portb

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

init_timer          ;; initialise timer

chk_mem             ;; check EEPROM

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

brset   bad_mem,status,read_defval

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

read_defval:    jsr     read_def_val

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

chk_hs_over                                 brclr half_sec,tim_status,chk_1_sec
            bclr    half_sec,tim_status
            jsr     chk_set_beep

chk_1_sec                                   brclr one_sec,tim_status,ret_act1sec
            bclr    one_sec,tim_status

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

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

ret_act1sec

; ************************ CHECK FOR KEY ****************************************
; if new key found flag set then goto act kbd else goto main_loop
chkbd                                       brclr new_key_found,status,ret_chkbd  ; if new key found then set
bclr    new_key_found,status        ; flag
jsr     act_kbd                 ; call actkbd
ret_chkbd   jmp     main_loop               ; else goto main loop

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

lda     kbd_pos             ; read kbd pos

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

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

chk4poet:                                   brclr po_entry_time,entry_status,chk_pgm_key
            jmp     wrong_entry

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

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

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

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

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

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

```
  jmp     entry_over              ; call entry_over

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

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

  ;; program comes here when unit is in program-
ming mode and 4 digit password enters
  ;; if 4 digit entered # password then goto wrong
entry
  ;; else return
  xxxx:           lda     kbd_buff        ;
compare kbd_buff with
  cmp     password        ; password
  bne     wrong_entry     ; if # goto wrong entry
  lda     kbd_buff + 1    ; if = compare
kbd_buff + 1 with
  cmp     password + 1            ; password + 1
  bne     wrong_entry             ; if # goto wrong
entry
  ret_actkbd    jmp     quick_beep      ; give
small beep after every
                                  ; key press
  ret_actkbd1:  rts                     ; return


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

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

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

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


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

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

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

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

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

  entry_table     db      5t,2,4,6,8,10t,12t,14t,16t,18t

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


  ;; program comes here when po_entry_time =
0
  ;; program here checks for po_password
  ;; if po_password = 1 then
  ;;    call pack_buff
  ;;     store change password in password vari-
able
  ;;    store in eeprom
  ;;    call entry_over
  ;;    give acc_beep
  ;;    return
  chk4popassword
                                  brclr
po_password,entry_status,chk4more
          bclr    po_password,entry_status
  upd_password    jsr     pack_buff       ; call
pack_buff
  lda     kbd_buff        ; save kbd_buff in
  sta     password        ; password
  lda     kbd_buff + 1    ; save kbd_buff + 1 in
  sta     password + 1    ; password + 1

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

  chk4more        bra     wrong_entry     ;
else give long beep


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



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

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

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

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

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

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

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

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

                tst             buzzer_time_out
                beq             chk_half_sec
                dec             buzzer_time_out

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

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

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

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

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

read_def_val    clrx
rdv_loop:       lda     def_table,x
                sta     password,x
                incx
                cpx     #max_iic_bytes
                bne     rdv_loop
                rts

;; here program pack entry time from et_buff\et_buff + 1
;; first byte is in et_buff
;; second byte is in et_buff + 1
;; output to entry_time var

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

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

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

                org     7cdh
                jmp     start

                org     7f1h
                db      20h

                org     7f8h
                fdb     timint

                org     7fah
                fdb     dummy

                org     7fch
                fdb     dummy

                org     7feh
                fdb     start
```

IIC.ASM

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


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

```
  bsr          delay_small
  bclr         sda,iicport    ;
  rts                         ; leave with sda as input

  delay_small:   nop
            nop
            nop
            nop
            nop
            nop
            rts

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

  restart_tx:

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

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

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

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

            bclr         sda,iicport
            bsr          delay_small
            bclr         scl,iicport
            rts

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

            bsr          gen_start
            lda          #0a0h
  dev_addr:      jsr          byte_iic            ; sda is input on return
            bcs          restart_rx
            lda          e_add
               jsr          byte_iic            ; second byte as mem add
            bcs          restart_rx
            bsr          gen_start
            lda          #0a1h
            jsr          byte_iic          ; sda is input on return
  read_iicbyte:  ldx          #8
  read_iicbit:   bset         scl,iicport          ; set scl high
  ;          jsr          delay_small          ; delay
  ;          bclr         scl,iicport          ; and again low
  brset        sda,iicport,iic_1     ; read data bit
  iic_0        clc
            bra          read_iic
  iic_1        sec
  read_iic      rola
               jsr          delay_small          ; delay
               bclr         scl,iicport          ; and again low
            decx
            bne          read_iicbit
            sta          e_dat
            bra          gen_stop
```

STDJ1.ASM

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

VARIABLE.ASM

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

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

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

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

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

buzzer_time_out db     00
beep_time       equ    10t

entry_time_out  db     00
hooter_time     equ    2
hooter_alarm_tout db   00

e_add           db     00
e_dat           db     00
iic_buff        db     00


kbd_pos         db     00
last_key        db     00
same_key        db     00

def_timer       equ    14h

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

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

key_scan_cntr   db     00
```

READKBD2.ASM

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

;; sense2 line is at irq
  kbd_sense
  sense_line     lda     key_port              ;read key port
            and     #30h
            bil     key_found
            ora     #40h
            cmp     #70h
            bne     key_found          ; no some key pressed
            bra     no_key_found       ; yes no
```

```
key pressed

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

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

  no_key_found   ldx     #0fh                  ;
  key_matched    txa                          ;load
accumulator with 'X'
  cmp     kbd_pos               ;compare it with
kbd pos
  beq     ret_kbs               ;if equal return
  cmp     last_key              ;compare it with last
key
  bne     new_key               ;if equal return
  inc     same_key              ;else goto new key
& inc same
  lda     same_key              ;for max debounce
load same key
  cmp     #max_debounce         ;compare it
with 4
  bne     ret_kbs               ;if not equal goto ret
kbs
  upd_key        lda     last_key              ;load
last key
  sta     kbd_pos               ;store it at kbd pos
  cmp     #0fh                  ;is it key release
  beq     ret_kbs               ;yes-do not set flag

  bset    new_key_found,status  ;set bit of new
key found in
  bra     ret_kbs               ;status and goto ret
kbs

  new_key        sta     last_key
                 clr     same_key
                 bra     kbs_over

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

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

  cs1:           lda     key_scan_port
                 and     #0f0h
                 sta     key_scan_port         ; reset all
scan lines to zero on ports

                 ldx     key_scan_cntr         ; output
scan table to scan port one by one
                 lda     scan_table,x
                 ora     key_scan_port
                 sta     key_scan_port
  ret_sense_line
  kbs_over       rts

  max_keys       equ     12t

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

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

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

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

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

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

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

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

PORTS.ASM

```
  k_program      equ     10t
  k_pgm_ok       equ     11t

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

  key_port       equ     porta

  scan0          equ     0      ; 16
  scan1          equ     1      ; 15
  scan2          equ     2      ; 14

  scan3          equ     3      ; 13
  sense0         equ     4      ; 12
  sense1         equ     5      ; 11
  ;;sense2       equ     irq    ; irq

  led_port       equ     porta
  led_arm        equ     6
  toggle_led     equ     40h

  buzzer_port    equ     porta
  buzzer         equ     7
```

MACRO.ASM

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

  ;; clear memory from 0c0h

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


  ;; intialise timer
  $macro         init_timer
                 lda     #def_timer
                 sta     tscr
                 cli                   ;enable interrupt
  $macroend
  ;; intialise porta , portb

  $macro         init_port       port
                 lda             #def_%1
                 sta             %1
  $macroend
```

EFY note. *All relevant files will be included in Nov. '02 EFY-CD.* ❑

** This project is a result of teamwork by Dimpi Thukral, Harnam Singh, Ruchi Sharma, and Satish Pal Singh led by Vinay Chaddha, proprietor GVC Systems, Noida.*