

Connecting a PC keyboard to the I²C-bus

AN434

CONNECTING A PC KEYBOARD TO THE I²C BUS

This application note illustrates the use of a low-cost 8-bit microcontroller—the 8XC751—to interface a standard PC/AT keyboard to the I²C bus. The 8XC751 (83C751 = ROM-version, 87C571 = EPROM-version) is ideally suited for the task thanks to its built-in I²C interface, small form-factor (24-pin DIP or 28-pin PLCC) and low power consumption (11mA typical @12 MHz; see Figure 1). The application software easily fits within the 2K bytes code and 64 bytes data memory provided on the 8XC751.

The PC/AT Keyboard

The PC/AT keyboard transmits data in a clocked serial format consisting of a start bit, 8 data bits (LSB first), an odd parity bit and a stop bit as shown in Figure 2. Besides clock and data, the 5-pin connector (Figure 3) also includes power, ground and a no connect. Note that the PS/2 keyboard interface is logically equivalent, though it uses a different connector. (A sixth pin provides an additional no connect).

When a key is pressed, the PC/AT keyboard transmits a 'make' code and, when the key is released, a 'break' code. The make code consists of an 8-bit 'scan' code denoting the key pressed. The 'break' code (key released) consists of the same 8-bit scan code preceded by a special code—0F0H.

A notable difference from a regular ASCII keyboard is the way SHIFT, CTRL, ALT, etc. control keys work. For an ASCII keyboard, the control keys directly modify the code output. For example, a 61H (ASCII code for 'a') is output if the 'A' key is pressed by itself, while a 41H (ASCII code for 'A') is output if the SHIFT and 'A' keys are pressed simultaneously.

The PC/AT keyboard handles such a key combination as two separate key presses, i.e., SHIFT-MAKE, 'A'-MAKE, SHIFT-BREAK, 'A'-BREAK. The 'A' scan code (1CH) is the same for both the shifted and unshifted state. To determine whether the 'A' scan code is interpreted as 'A' or 'a' the PC must keep track of the presence or absence of a prior SHIFT-MAKE.

Keyboard-to-I²C Hardware (Figure 4)

The 8XC751 on-chip I²C interface allows direct connection of the SDA (Serial Data) and SCL (Serial Clock) pins to the corresponding I²C bus lines. Since the I²C bus is open collector (allowing multimasters), 10K resistors are used to pull the lines to the idle state between keypresses.

The PC/AT keyboard interface is equally simple. The CLK output from the keyboard is used to generate an interrupt (INT0). In response, the 8XC751 interrupt service routine samples the keyboard serial DATA connected to port 0 bit 2 (P0.2).

When used with a PC, the keyboard implements a bidirectional communication protocol by exploiting the fact that both the keyboard and PC can drive the open collector CLK and DATA lines. However, bidirectional communication is not required for basic keyboard operation and in this application, the keyboard is treated as an 'input-only' device.

Keyboard-to-I²C Software

The keyboard-to-I²C software performs three major functions:

- Capture the clocked serial data from the keyboard
- Translate the keyboard data to the corresponding ASCII code
- Send the ASCII code as an I²C message.

When a key is pressed, the CLK output from the keyboard generates an interrupt via INT0. The 8XC751 shifts in the DATA from the keyboard on P0.2 (port 0, bit 2) and extracts the 8-bit scan code from the 11-bit packet.

Next, the scan code is interpreted and converted to the corresponding ASCII code using a look-up table. Keyboard multi-code outputs are converted to single ASCII codes by tracking the state (i.e. shifted vs. unshifted) of the keyboard and using separate look-up tables for each. For

example, a keyboard SHIFT-MAKE, 'A'-MAKE, SHIFT-BREAK, 'A'-BREAK sequence is converted to the ASCII code for uppercase 'A' (41H). The flowchart in Figure 5 depicts the keyboard data capture and code conversion process.

The 8XC751 operates as an I²C slave. When the master issues a read command, the 8XC751 returns the converted ASCII character. The seven least significant bits are used for the ASCII code, while the most significant bit is used as a NEW flag (0 = new, 1 = old). The key code remains marked as new until the master issues a write to the 8XC751 at which point it is marked as old and will be overwritten by the next key processed.

The keyboard-to-I²C software is shown immediately following Figure 5. Less than half the code space available on the 8XC751 is used, leaving room for extra features such as parity checking and more complete keyboard control state mapping using additional look-up tables.

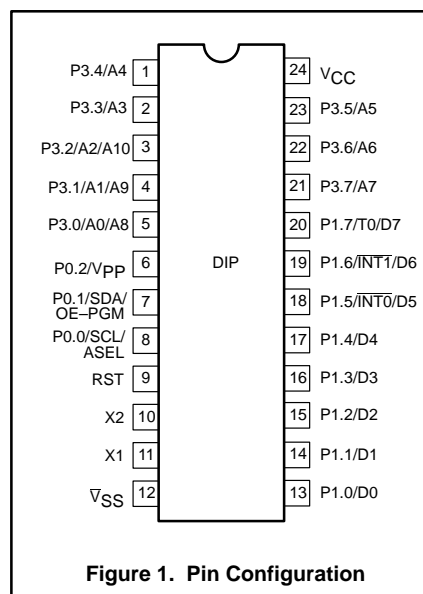


Figure 1. Pin Configuration

Connecting a PC keyboard to the I²C-bus

AN434

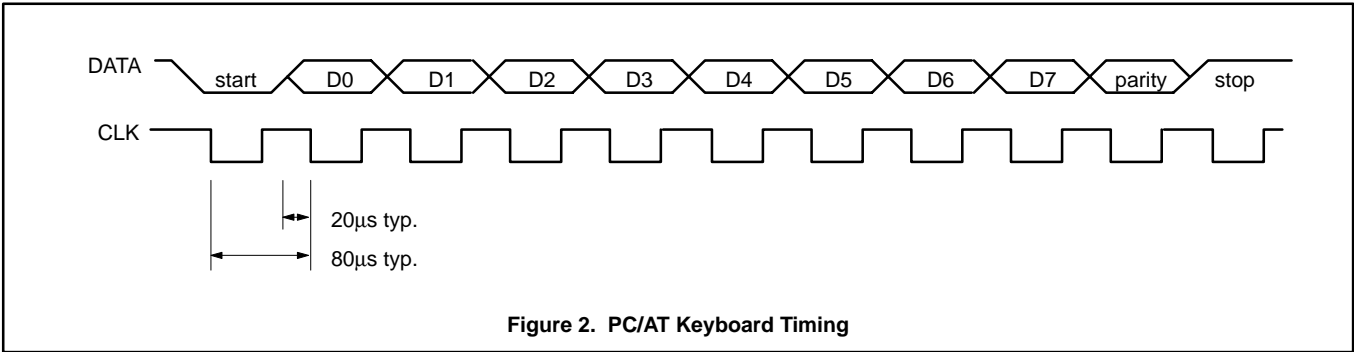


Figure 2. PC/AT Keyboard Timing

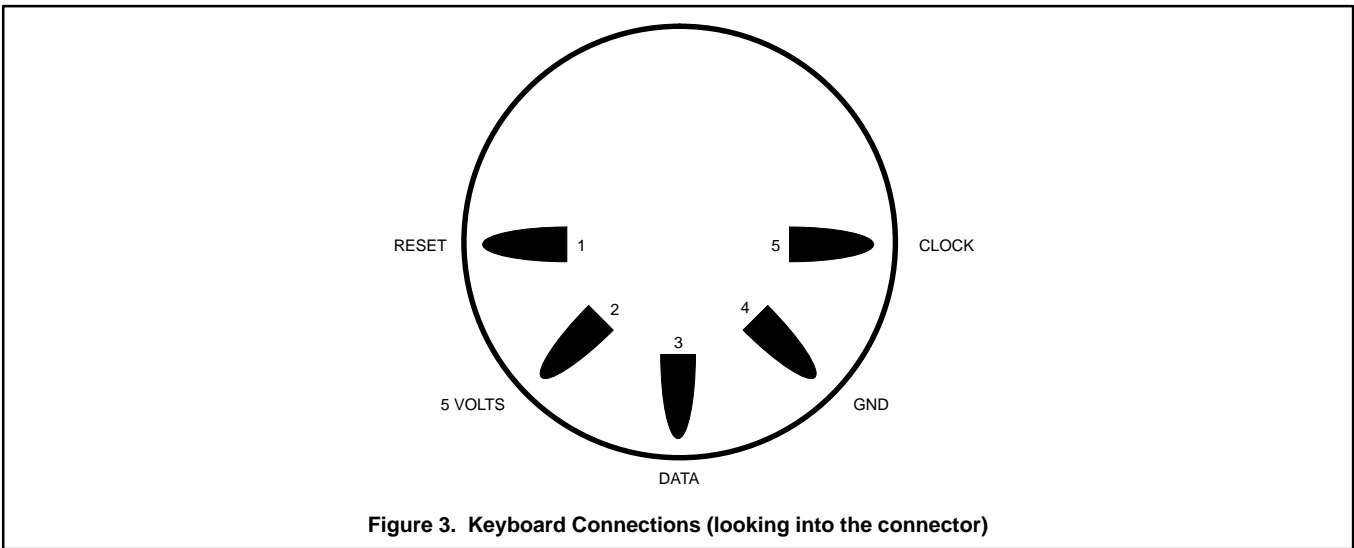


Figure 3. Keyboard Connections (looking into the connector)

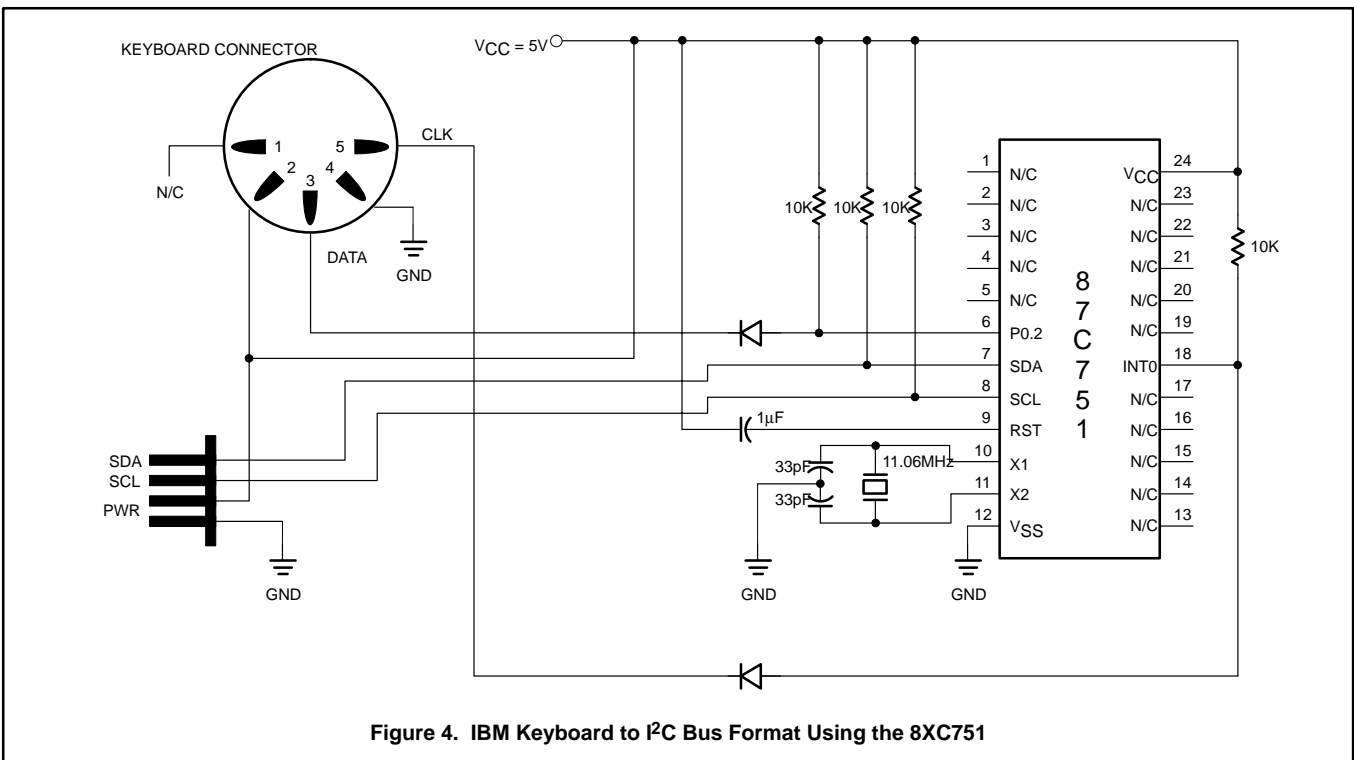


Figure 4. IBM Keyboard to I²C Bus Format Using the 8XC751

Connecting a PC keyboard to the I²C-bus

AN434

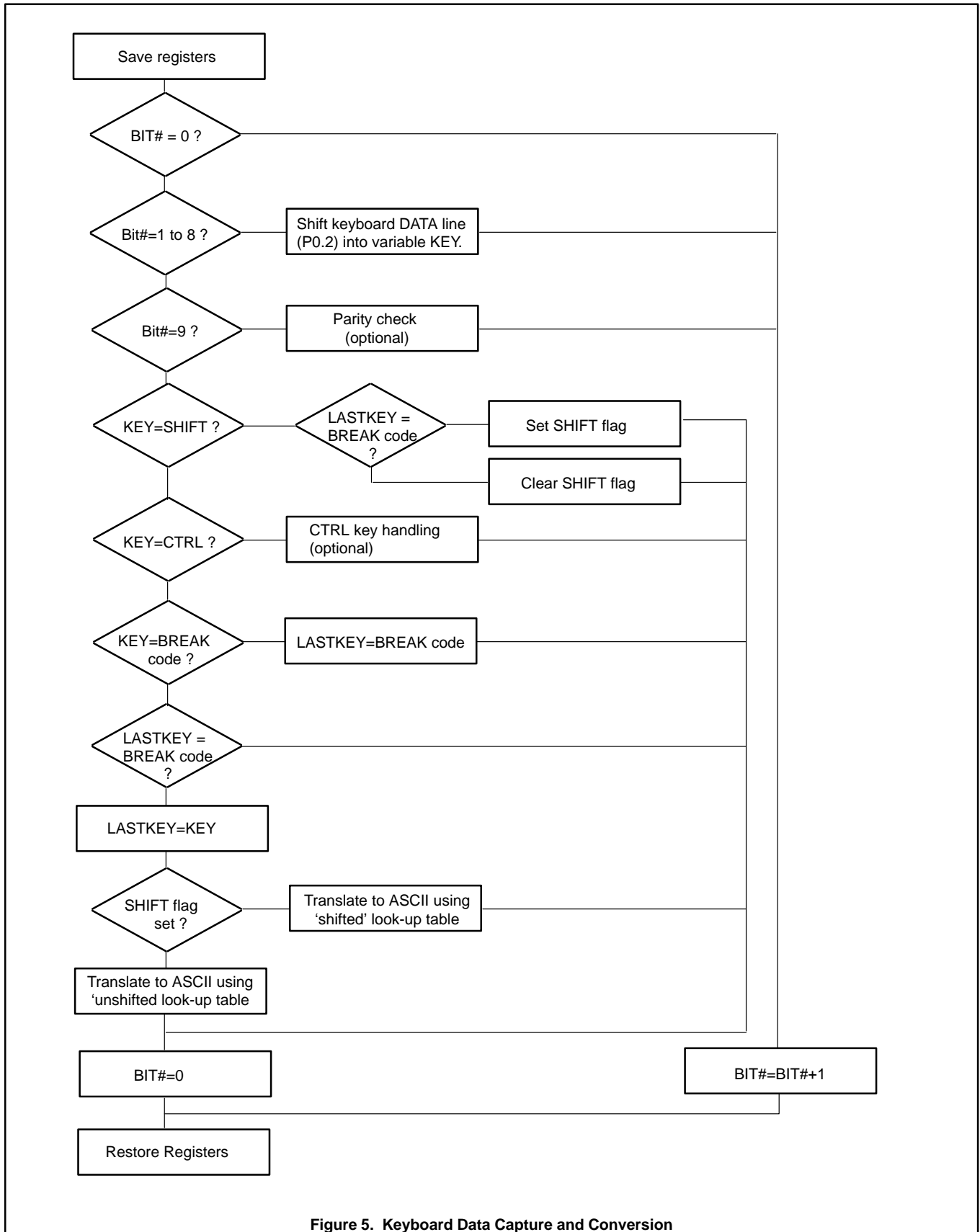


Figure 5. Keyboard Data Capture and Conversion

Connecting a PC keyboard to the I²C-bus

AN434

```

0001 0000 ;*****
0002 0000 ;
0003 0000 ; Copyright Micro AMPS Ltd
0004 0000 ; & Philips Semiconductors
0005 0000 ; Dec 1990
0006 0000 ;
0007 0000 ;*****
0008 0000
0009 0000 ; Read data under interrupt from an IBM keyboard
0010 0000 ; Hardware resources:
0011 0000 ; Kbd clock on interrupt INT0 P1.5
0012 0000 ; Kbd data on pin P0.2
0013 0000
0014 0000
0015 0000 ; This program reads keys in from the keyboard
0016 0000 ; and translates them to ASCII
0017 0000
0018 0000
0019 0000
0020 0000 #include equates.51
0001+ 0000 ; direct addresses for the standard 8051 processor
0002+ 0000
0003+ 0000 p0 .equ 80h ; port 0
0004+ 0000 sp .equ 81h ; stack pointer
0005+ 0000 dpl .equ 82h ; data pointer low
0006+ 0000 dph .equ 83h ; data pointer high
0007+ 0000
0008+ 0000 pcon .equ 87h ; power control
0009+ 0000 tcon .equ 88h ; timer control
0010+ 0000 tmod .equ 89h ; timer mode
0011+ 0000 tl0 .equ 8ah ; timer 0 low
0012+ 0000 th0 .equ 8ch ; timer 0 high
0013+ 0000
0014+ 0000 th1 .equ 8dh ; timer 1 high
0015+ 0000
0016+ 0000 p1 .equ 90h ; port 1
0017+ 0000 scon .equ 98h ; serial control
0018+ 0000 s0con .equ 98h ; serial control
0019+ 0000 s0buf .equ 99h ; serial data
0020+ 0000
0021+ 0000 p2 .equ 0a0h ; port 2
0022+ 0000 p3 .equ 0b0h ; port 3
0023+ 0000 ien0 .equ 0a8h ; interrupt enable
0024+ 0000 ie .equ 0a8h
0025+ 0000
0026+ 0000 psw .equ 0d0h ; program status word
0027+ 0000 acc .equ 0e0h ; accumulator
0028+ 0000 b .equ 0f0h ; b register
0029+ 0000
0030+ 0000 ; bit addressed flags
0031+ 0000
0032+ 0000 it0 .equ 88h ; int 0 edge/level trigger
0033+ 0000 ie0 .equ 89h ; int 0 edge detect
0034+ 0000 it1 .equ 8ah ; int 1 edge/level trigger
0035+ 0000 ie1 .equ 8bh ; int 1 edge detect
0036+ 0000 tr0 .equ 8ch ; timer 0 enable/disable
0037+ 0000 tf0 .equ 8dh ; timer 0 overflow detect
0038+ 0000 tr1 .equ 8eh ; timer 1 enable/disable
0039+ 0000 tf1 .equ 8fh ; timer 1 overflow detect
0040+ 0000
0041+ 0000 ri .equ 98h
0042+ 0000 ti .equ 99h
0043+ 0000
0044+ 0000 ien0.7 .equ 0afh ; global int enable/disable
0045+ 0000
0046+ 0000 p0.0 .equ 080h ; port 0 bit 0
0047+ 0000

```

Connecting a PC keyboard to the I²C-bus

AN434

```

0048+ 0000    b.0      .equ 0f0h          ; b reg bits
0049+ 0000    b.1      .equ 0f1h
0050+ 0000    b.2      .equ 0f2h
0051+ 0000    b.3      .equ 0f3h
0052+ 0000    b.4      .equ 0f4h
0053+ 0000    b.5      .equ 0f5h
0054+ 0000    b.6      .equ 0f6h
0055+ 0000    b.7      .equ 0f7h
0056+ 0000
0057+ 0000    a.0      .equ 0e0h          ; accumulator bits
0058+ 0000    a.1      .equ 0e1h
0059+ 0000    a.2      .equ 0e2h
0060+ 0000    a.3      .equ 0e3h
0061+ 0000    a.4      .equ 0e4h
0062+ 0000    a.5      .equ 0e5h
0063+ 0000    a.6      .equ 0e6h
0064+ 0000    a.7      .equ 0e7h
0065+ 0000
0066+ 0000    rth      .equ 8dh          ; timer 0 reload high
0067+ 0000    rtl      .equ 8bh          ; timer 0 reload low
0068+ 0000
0021  0000    #include kbd.h
0001+ 0000    #define reg .equ
0002+ 0000
0003+ 0000    ;
0004+ 0000    ; 8xc751 special register set
0005+ 0000    ;
0006+ 0000    ; 751 I2C byte registers
0007+ 0000
0008+ 0000    I2CON     .equ 098h          ; I2C control
0009+ 0000    I2CFG     .equ 0d8h          ; I2C configuration
0010+ 0000    I2DAT     .equ 099h          ; I2C data
0011+ 0000    I2STA     .equ 0f8h          ; I2C status
0012+ 0000
0013+ 0000    IE        .equ 0a8h          ; interrupt enable
0014+ 0000
0015+ 0000    TCON     .equ 088h          ; timer/counter control
0016+ 0000
0017+ 0000    TL        .equ 08ah          ; timer 0 low
0018+ 0000    TH        .equ 08ch          ; timer 0 high
0019+ 0000    RTL        .equ 08bh          ; timer reload low
0020+ 0000    RTH        .equ 08dh          ; timer reload high
0021+ 0000
0022+ 0000    ; 751 I2C bit registers
0023+ 0000
0024+ 0000    ;I2CNFG
0025+ 0000
0026+ 0000    SLAVEN     .equ 0dfh
0027+ 0000    MASTRQ     .equ 0deh
0028+ 0000    TIRUN     .equ 0dch
0029+ 0000    CT1       .equ 0d9h
0030+ 0000    CT0       .equ 0d8h
0031+ 0000    CLRTI     .equ 0ddh
0032+ 0000
0033+ 0000    RDAT      .equ 09fh
0034+ 0000    ATN       .equ 09eh
0035+ 0000    DRDY      .equ 09dh
0036+ 0000    ARL       .equ 09ch
0037+ 0000    STR       .equ 09bh
0038+ 0000    STP       .equ 09ah
0039+ 0000    MASTER     .equ 099h
0040+ 0000
0041+ 0000    ; I2CON
0042+ 0000
0043+ 0000    CXA       .equ 09fh
0044+ 0000    IDLE      .equ 09eh
0045+ 0000    CDR       .equ 09dh

```

Connecting a PC keyboard to the I²C-bus

AN434

```

0046+ 0000  CARL      .equ 09ch
0047+ 0000  CSTR      .equ 09bh
0048+ 0000  CSTP      .equ 09ah
0049+ 0000  XSTR      .equ 099h
0050+ 0000  XSTP      .equ 098h
0051+ 0000
0052+ 0000  ;I2STA
0053+ 0000
0054+ 0000  XDATA     .equ 0fdh
0055+ 0000  XACTV     .equ 0fch
0056+ 0000  MAKSTR    .equ 0fbh
0057+ 0000  MAKSTP    .equ 0fah
0058+ 0000
0059+ 0000  ; IE bit registers
0060+ 0000
0061+ 0000  EA        .equ 0afh ; clr to disable all interrupts
0062+ 0000  EI2       .equ 0ach ; set to enable iic interrupt
0063+ 0000  ETI       .equ 0abh ; set to enable timer 1 overflow interrupt
0064+ 0000  EX1       .equ 0aah ; set to enable ext int 1
0065+ 0000  ET0       .equ 0a9h ; set to enable timer 0 overflow interrupt
0066+ 0000  EX0       .equ 0a8h ; set to enable ext int 0
0067+ 0000
0068+ 0000  ; Value definitions.
0069+ 0000
0070+ 0000  CTVAL     .equ 02h ;CT1, CT0 bit values for I2C.
0071+ 0000
0072+ 0000
0073+ 0000  ; Masks for I2CFG bits.
0074+ 0000
0075+ 0000  BTIR      .equ 10h ; mask for TIRUN bit.
0076+ 0000  BMRQ      .equ 40h ; mask for MASTRQ bit.
0077+ 0000
0078+ 0000
0079+ 0000  ; Masks for I2CON bits.
0080+ 0000
0081+ 0000  BCXA      .equ 80h ; mask for CXA bit.
0082+ 0000  BIDLE     .equ 40h ; mask for IDLE bit.
0083+ 0000  BCDR      .equ 20h ; mask for CDR bit.
0084+ 0000  BCARL     .equ 10h ; mask for CARL bit.
0085+ 0000  BCSTR     .equ 08h ; mask for CSTR bit.
0086+ 0000  BCSTP     .equ 04h ; mask for CSTP bit.
0087+ 0000  BXSTR     .equ 02h ; mask for XSTR bit.
0088+ 0000  BXSTP     .equ 01h ; mask for XSTP bit.
0089+ 0000  ;
0090+ 0000
0091+ 0000
0092+ 0000  SCL       .equ p0.0 ; port bit for I2C serial clock line.
0093+ 0000  SDA       .equ p0.1 ; port bit for I2C serial data line.
0094+ 0000
0022 0000
0023 0000  IICADD   .equ 088h ; our I2C slave address
0024 0000  MAXBYTES .equ 1 ; max bytes to recv or trans
0025 0000
0026 0000  rcvdat   .equ 04h ; I2C received data buffer
0027 0000  xmtdat   .equ 06h ; I2C transmitter buffer
0028 0000
0029 0000  STACK    .equ 08h
0030 0000
0031 0000  flags    .equ 020h ; byte used as flags
0032 0000  noack    .equ (flags-20h) ; I2C flags.0, ...1, ...2, etc
0033 0000  recvd    .equ (flags-20h)+1 ;
0034 0000  sent_flag .equ (flags-20h)+2 ;
0035 0000  i2c_busy .equ (flags-20h)+3 ;
0036 0000
0037 0000  Cntrl    .equ (flags-20h)+8 ; control key flag
0038 0000  Shift    .equ (flags-20h)+9 ; shift key flag
0039 0000

```

Connecting a PC keyboard to the I²C-bus

AN434

```

0040 0000 bitcnt .equ flags+2
0041 0000 bytecnt .equ flags+3
0042 0000
0043 0000 adrrcvd .equ flags+4
0044 0000 rwflag .equ (adrrcvd-20h)*8 ; adrrcvd.0
0045 0000
0046 0000 tick .equ 025h ; count 10mS ticks to give 1sec tick
0047 0000 i2ctime .equ 027h ; I2C timeout - used on slow I2C bus
0048 0000
0049 0000
0050 0000 NBits .equ 29h ; # bits read so far
0051 0000 NBytes .equ NBits+1 ; # bytes in buffer
0052 0000 lastkey .equ NBytes+1 ; last key was?
0053 0000 keytemp .equ lastkey+1 ; used to build the key bit by bit
0054 0000 keybuff .equ keytemp+1 ; store the chars here
0055 0000
0056 0000
0057 0000 INMAX .equ 8 ; size of keyboard buffer
0058 0000 KEYCLK .equ p1.5 ; keyboard clock signal on ext int 0
0059 0000 KEYDAT .equ 82h ; keyboard data line
0060 0000
0061 0000 EDGEINT .equ 08ah
0062 0000
0063 0000 ; reset and interrupt vectors.
0064 0000 ;
0065 0000 .org 0 ; reset vector
0066 0000 01 50 ajmp start
0067 0002
0068 0003 .org 0003h ; external interrupt 0
0069 0003 01 B5 ajmp kbd
0070 0005
0071 000B .org 0bh ; counter/timer 0
0072 000B 21 EA ajmp badint
0073 000D
0074 0013 .org 013h ; external interrupt 1
0075 0013 21 EA ajmp badint
0076 0015
0077 001B .org 01bh ; timer 1 - I2C timeout
0078 001B 21 DA ajmp timerI
0079 001D
0080 0023 .org 023h ; I2C interrupt
0081 0023 21 38 ajmp i2cint
0082 0025
0083 0025
0084 0048 .org 48h
0085 0048
0086 0048 done:
0087 0048 01 48 ajmp $ ; main routine waiting for key presses
0088 004A
0089 0050 .org 50h
0090 0050
0091 0050 start:
0092 0050 78 FF mov r0,#0ffh ; power supply settling time
0093 0052 79 FF mov r1,#0ffh
0094 0054 7A 04 mov r2,#04h
0095 0056
0096 0056 D8 FE dly1:djnz r0,$
0097 0058 D9 FC djnz r1,dly1
0098 005A DA FA djnz r2,dly1
0099 005C
0100 005C reset:
0101 005C 75 80 FF mov p0,#0ffh
0102 005F 75 90 FF mov p1,#0ffh
0103 0062 75 B0 FF mov p3,#0ffh
0104 0065
0105 0065 75 81 08 mov sp,#STACK ; initialize stack pointer
0106 0068 D2 8A setb EDGEINT ; make ext int 0 edge activated

```

Connecting a PC keyboard to the I²C-bus

AN434

```

0107 006A
0108 006A C2 09      clr Shift                ; clear keyboard shift flag
0109 006C
0110 006C 78 29      mov r0,#NBits            ; clear the input buffers
0111 006E 79 10      mov r1,#10h
0112 0070 75 E0 00    mov acc,#0
0113 0073
0114 0073 F6         clr rp:mov @r0,a         ; do the clearing
0115 0074 08         inc r0
0116 0075 D9 FC      djnz r1,clr rp
0117 0077
0118 0077 ;          mov xmdat,#'. '          ; transmit buffer filled with
0119 0077 ;          mov xmdat+1,#0ffh        ; $ff when empty
0120 0077
0121 0077 75 20 00    mov flags,#0
0122 007A 75 27 00    mov i2ctime,#0
0123 007D
0124 007D restart:
0125 007D
0126 007D 75 D8 82    mov I2CFG,#80h+CTVAL    ; enable slave functions
0127 0080 75 98 40    mov I2CON,#BIDLE       ; place in idle state
0128 0083
0129 0083 75 A8 91    mov ien0,#91h          ; enable external & IIC interrupts
0130 0086
0131 0086
0132 0086 ;***** Main loop *****
0133 0086
0134 0086 main:
0135 0086 E5 2A      mov a,NBytes            ; if data in keybuff then
0136 0088 60 0C      jz empty               ; copy to I2C xmt buffer
0137 008A
0138 008A 75 A8 00    mov ien0,#0h           ; disable all ints temporarily
0139 008D 85 2D 06    mov xmdat,keybuff
0140 0090 75 2A 00    mov NBytes,#0         ; clear keyboard buffer full flag
0141 0093 75 A8 91    mov ien0,#91h          ; enable external&IIC interrupts
0142 0096
0143 0096 empty:
0144 0096
0145 0096 30 01 0A    jnb recvd,notread     ; recvd flag tells 751 to clear
0146 0099 ; I2C xmt buffer when I2C master
0147 0099 85 06 E0    mov acc,xmdat         ; reads the data from the 751
0148 009C 44 80      orl a,#80h            ; the master writes any data
0149 009E 85 E0 06    mov xmdat,acc         ; back which will set the MSB of
0150 00A1 ; the data buffer. This is reqd.
0151 00A1 ; to sync the two processors.
0152 00A1 C2 01      clr recvd             ; reset I2C received flag
0153 00A3
0154 00A3 notread:
0155 00A3
0156 00A3 E5 2B      mov a,lastkey         ; detect alt key for special
0157 00A5 C3         clr c                 ; for special functions
0158 00A6 94 11      subb a,#11h
0159 00A8 70 01      jnz notalt
0160 00AA
0161 00AA 00         nop                   ; alt code goes here
0162 00AB
0163 00AB notalt:
0164 00AB E5 2A      mov a,NBytes
0165 00AD C3         clr c
0166 00AE 94 08      subb a,#INMAX        ; limit the input buffer to INMAX
0167 00B0 40 01      jc notdone           ; if data is buffered
0168 00B2 ; then buffer overflow
0169 00B2 00         nop                   ; code goes here
0171 00B3
0172 00B3 notdone:
0173 00B3
0174 00B3 80 D1      sjmp main            ; go back to start

```


Connecting a PC keyboard to the I²C-bus

AN434

```

0175 00B5
0176 00B5 ;***** End of Main loop *****
0177 00B5
0178 00B5
0179 00B5 ; ***** External int 0 ISR *****;
0180 00B5 ;
0181 00B5 ; keyboard interrupt service routine ;
0182 00B5 ;
0183 00B5 ; *****;
0184 00B5
0185 00B5
0186 00B5 kbd:
0187 00B5 C0 D0 push psw ; save .equ during ISR
0188 00B7 C0 E0 push acc
0189 00B9
0190 00B9 85 29 E0 mov acc,NBits ; NBits=bit number next expected
0191 00BC ; from the keyboard
0192 00BC B4 00 02 cjne a,#0,bit1_8 ; if not bit 0 then bit 1 to 8
0193 00BF
0194 00BF
0195 00BF
0196 00BF ;***** Keyboard Bit 0 *****
0197 00BF
0198 00BF bit0: ; discard bit 0 - Start bit
0199 00BF 80 70 sjmp bump
0200 00C1
0201 00C1
0202 00C1 ;***** Keyboard Bit 1-8 *****
0203 00C1
0204 00C1 bit1_8:
0205 00C1 B4 09 00 cjne a,#9,$+3 ; CY flag is set if acc < 9
0206 00C4 50 0C jnc bit9
0207 00C6
0208 00C6 A2 82 mov c,KEYDAT ; read data for keyboard data line
0209 00C8 E5 2C mov a,keytemp ; data arrives least sig bit 1st
0210 00CA 03 rr a ; hence old value is rotated and new
0211 00CB 92 E7 mov a.7,c ; bit is or'ed to the msb
0212 00CD 85 E0 2C mov keytemp,acc
0213 00D0 80 5F sjmp bump
0214 00D2
0215 00D2
0216 00D2 ;***** Bit 9 *****
0217 00D2
0218 00D2 bit9:
0219 00D2 B4 09 02 cjne a,#9,bit10
0220 00D5
0221 00D5 80 5A sjmp bump ; parity check code would go here
0222 00D7
0223 00D7
0224 00D7
0225 00D7 ;***** Bit 10 *****
0226 00D7
0227 00D7 ; The stop bit - Key Scan is now complete so convert to ASCII
0228 00D7
0229 00D7 bit10:
0230 00D7 85 2C E0 mov acc,keytemp ; get next key
0231 00DA
0232 00DA B4 12 14 cjne a,#12h,not1s ; is it the left shift char?
0233 00DD
0234 00DD
0235 00DD ;***** Left Shift has Been Pressed *****
0236 00DD
0237 00DD 85 2B E0 mov acc,lastkey ; if last key was
0238 00E0 B4 F0 07 cjne a,#0f0h,makels ; $f0 then shift is released
0239 00E3
0240 00E3 C2 09 clr Shift ; next keys will be unshifted
0241 00E5 75 2B 12 mov lastkey,#12h ; copy left shift key to last key

```

Connecting a PC keyboard to the I²C-bus

AN434

```

0242 00E8 80 3F      sjmp tidy
0243 00EA
0244 00EA      makels:
0245 00EA D2 09      setb Shift                ; next keys will be shifted
0246 00EC 75 2B 12  mov lastkey,#12h        ; copy left shift key to last key
0247 00EF 80 38      sjmp tidy
0248 00F1
0249 00F1      ;***** End of Shift Routine *****
0250 00F1
0251 00F1      notls:
0252 00F1      ;      mov acc,keytemp          ; get next key
0253 00F1 B4 14 03  cjne a,#14h,notctrl    ; is it a control char?
0254 00F4
0255 00F4
0256 00F4      ;***** Control State *****
0257 00F4
0258 00F4 00      nop                      ; control state goes here
0259 00F5 80 32      sjmp tidy
0260 00F7
0261 00F7      ;***** End of Control State *****
0262 00F7
0263 00F7      notctrl:
0264 00F7
0265 00F7 B4 F0 04  cjne a,#0f0h,notbreak ; if current key $f0 then break
0266 00FA
0267 00FA      ;***** Key Break *****
0268 00FA
0269 00FA F5 2B      mov lastkey,a            ; record break code in last key
0270 00FC 80 2B      sjmp tidy                ; but don't store in the buffer
0271 00FE
0272 00FE      notbreak:
0273 00FE
0274 00FE 85 2B E0  mov acc,lastkey          ; if last key was $f0 then
0275 0101 B4 F0 05  cjne a,#0f0h,not_f0    ; ignore the next scan code
0276 0104
0277 0104 75 2B 00  mov lastkey,#0            ; which is a break code
0278 0107 80 20      sjmp tidy
0279 0109
0280 0109      not_f0:
0281 0109
0282 0109
0283 0109      ;***** Normal Key Press *****
0284 0109
0285 0109~      #ifdef buffered
0286 0109~
0287 0109~      ;***** Buffered Code *****
0288 0109~
0289 0109~      ; buffered code
0290 0109~      push 0                    ; r0 used as an indirect pointer
0291 0109~      ; so save it
0292 0109~      mov acc,#keybuff         ; copy data into keyboard
0293 0109~      add a,NBytes
0294 0109~      mov r0,a
0295 0109~
0296 0109~      mov a,keytemp            ; get current key
0297 0109~      mov lastkey,a           ; & copy to lastkey
0298 0109~
0299 0109~      push dph                 ; dp used to point to xlat tables
0300 0109~      push dpl                 ; since in ISR save dp contents
0301 0109~
0302 0109~      jb Shift,shifted        ; if in unshifted state
0303 0109~      mov dptr,#unshift       ; use the unshift table
0304 0109~      sjmp skipl
0305 0109~
0306 0109~      shifted:
0307 0109~      mov dptr,#shift         ; else use the shift table
0308 0109~

```

Connecting a PC keyboard to the I²C-bus

AN434

```

0309 0109~ skip1:
0310 0109~      movc a,@a+dptr          ; translate char in Acc to Ascii
0311 0109~
0312 0109~      pop dpl                ; restore the data pointer
0313 0109~      pop dph
0314 0109~
0315 0109~      cjne a,#0,Not0        ; if data is zero discard
0316 0109~
0317 0109~      ; sjmp NoSave          ; discard code goes here
0318 0109~
0319 0109~      Not0:
0320 0109~      mov r0,#keybuff      ; Save ascii value in buffer
0321 0109~      mov @r0,a           ; buffered keyboard entry
0322 0109~      inc NBytes
0323 0109~
0324 0109~      NoSave:
0325 0109~      pop 0              ; restore r0
0326 0109~
0327 0109~      ;**** End of Buffered Code
0328 0109~
0329 0109      #endif
0330 0109
0331 0109
0332 0109      #define unbuffered 1
0333 0109
0334 0109      #ifndef unbuffered
0335 0109
0336 0109 E5 2C      mov a,keytemp      ; get current key
0337 010B F5 2B      mov lastkey,a     ; & copy to lastkey
0338 010D
0339 010D C0 83      push dph          ; dp used to point to xlat tables
0340 010F C0 82      push dpl          ; since in ISR save dp contents
0341 0111
0342 0111 20 09 05    jb Shift,shifted      ; if in unshifted state
0343 0114 90 01 EB    mov dptr,#unshift     ; use the unshift table
0344 0117 80 03      sjmp skip1
0345 0119
0346 0119      shifted:
0347 0119 90 02 6B    mov dptr,#shift       ; else use the shift table
0348 011C
0349 011C      skip1:
0350 011C 93          movc a,@a+dptr      ; translate char in Acc to Ascii
0351 011D
0352 011D D0 82      pop dpl                ; restore the data pointer
0353 011F D0 83      pop dph
0354 0121
0355 0121 B4 00 00    cjne a,#0,Not0        ; if data is zero discard
0356 0124
0357 0124          ; sjmp tidy          ; discard code goes here
0358 0124
0359 0124      Not0:
0360 0124 F5 2D      mov keybuff,a       ; store in keyboard buffer
0361 0126 75 2A 01    mov NBytes,#1        ; mark byte read
0362 0129
0363 0129      tidy:
0364 0129 75 29 00    mov NBits,#0         ; clear flags ready for next key
0365 012C 75 2C 00    mov keytemp,#0
0366 012F 80 02      sjmp intdone
0367 0131
0368 0131      ;***** End of Keyboard Translation and Save *****
0369 0131
0370 0131
0371 0131      ;***** Normal unfinished key exit *****
0372 0131
0373 0131      bump:
0374 0131 05 29      inc NBits           ; inc number of bits read so far
0375 0133

```

Connecting a PC keyboard to the I²C-bus

AN434

```

0376 0133  intdone:
0377 0133 D0 E0      pop acc
0378 0135 D0 D0      pop psw
0379 0137 32        reti
0380 0138
0381 0138      ;***** End of Ext Int 0 ISR *****
0382 0138
0383 0138
0384 0138
0385 0138
0386 0138      ;***** I2C CODE SLAVE *****
0387 0138
0388 0138      i2cint:                                ; I2C interrupt entry point
0389 0138 D2 03      setb i2c_busy                      ; semaphore on xmtdata buffer
0390 013A
0391 013A C0 D0      push psw                          ; save registers used in ISR
0392 013C C0 E0      push acc
0393 013E C0 00      push 0                            ; R0 no bank switching
0394 0140
0395 0140 C2 AC      clr EI2                          ; make I2C ISR interruptable
0396 0142 31 E9      acall clrint                       ; execute a reti
0397 0144
0398 0144      slave:
0399 0144 75 27 03   mov i2ctime,#3                      ; set up I2C timeout watchdog 30 mS
0400 0147
0401 0147 75 98 9C   mov I2CON,#BCARL+BCSTP+BCSTR+BCXA
0402 014A                                ; clear start status
0403 014A
0404 014A 30 9E FD   jnb ATN,$                          ; wait for next data bit
0405 014D 75 22 07   mov bitcnt,#7
0406 0150
0407 0150 31 C9      acall recvb2                          ; get remainder of slave address
0408 0152 F5 24      mov adrrcvd,a
0409 0154 C2 E0      clr a.0                            ; mask r/w bit to check address
0410 0156 B4 88 3B   cjne a,#IICADD,goidle      ; idle again if not for us
0411 0159
0412 0159 20 20 1F   jb rwflag,read            ; test for read or write
0413 015C
0414 015C
0415 015C
0416 015C      ;***** I2C Receive Code *****
0417 015C
0418 015C 78 04      mov r0,#rcvdat            ; r0 points to data buffer
0419 015E 75 23 01   mov bytecnt,#MAXBYTES
0420 0161
0421 0161      rcvloop:
0422 0161 31 BF      acall sendack                    ; acknowledge the address
0423 0163 31 C6      acall rcvbyte                    ; wait for the next data byte
0424 0165 30 9D 0F   jnb DRDY,exitwr            ; end of frame
0425 0168 F6        mov @r0,a
0426 0169 08        inc r0
0427 016A D5 23 F4   djnz bytecnt,rcvloop
0428 016D
0429 016D      ; no more room
0430 016D
0431 016D 31 BF      acall sendack                    ; ack last byte
0432 016F 31 C6      acall rcvbyte                    ; get but discard next one
0433 0171 75 99 80   mov I2DAT,#80h            ; send neg ack
0434 0174 30 9E FD   jnb ATN,$                  ; wait till gone
0435 0177      exitwr:
0436 0177
0437 0177 D2 01      setb recvd
0438 0179 80 13      sjmp msgend
0439 017B
0440 017B      ;***** End of Receive Routine *****
0441 017B
0442 017B

```

Connecting a PC keyboard to the I²C-bus

AN434

```

0443 017B ;***** I2C Transmit Code *****
0444 017B
0445 017B read:
0446 017B 78 06 mov r0,#xmtdat ; r0 points to data buffer
0447 017D 75 23 01 mov bytecnt,#MAXBYTES
0448 0180 31 BF acall sendack ; acknowledge address
0449 0182
0450 0182 txloop:
0451 0182 E6 mov a,@r0 ; get next data byte
0452 0183 08 inc r0 ; bump buffer pointer
0453 0184 31 A7 acall xmitbyte ; transmit the byte to the I2C
0454 0186 20 00 03 jb noack,exitrd ; if not acknowledged then exit
0455 0189 D5 23 F6 djnz bytecnt,txloop
0456 018C
0457 018C 80 00 exitrd: sjmp msgend
0458 018E
0459 018E ;***** End of I2C transmit *****
0460 018E
0461 018E
0462 018E ;***** Repeated start state *****
0463 018E msgend:
0464 018E 30 9E FD jnb ATN,$ ; wait for stop or repeated start
0465 0191 20 9B B0 jb STR,slave ; if repeat start do again
0466 0194
0467 0194 ; stop so enter idle mode
0468 0194
0469 0194 goidle:
0470 0194 75 27 00 mov i2ctime,#0 ; stop I2C timeout
0471 0197 75 98 F4 mov I2CON,#BCSTP+BCXA+BCDR+BCARL+BIDLE
0472 019A
0473 019A D0 00 pop 0 ; restore state before I2C ISR
0474 019C D0 E0 pop acc
0475 019E D0 D0 pop psw
0476 01A0
0477 01A0 D2 AC setb EI2
0478 01A2 D2 02 setb sent_flag ; flag to say data has been sent
0479 01A4 C2 03 clr i2c_busy ; flag denotes exiting I2C routine
0480 01A6
0481 01A6 22 ret
0482 01A7
0483 01A7
0484 01A7 ;***** General I2C routines *****
0485 01A7
0486 01A7 xmitbyte: ; transmit data in acc to I2C
0487 01A7 75 22 08 mov bitcnt,#8
0488 01AA
0489 01AA xmitbit:
0490 01AA F5 99 mov I2DAT,a
0491 01AC 23 rl a
0492 01AD 30 9E FD jnb ATN,$
0493 01B0 D5 22 F7 djnz bitcnt,xmitbit
0494 01B3 75 98 A0 mov I2CON,#BCDR+BCXA ; switch to rcv mode
0495 01B6 30 9E FD jnb ATN,$ ; wait for ack
0496 01B9 85 99 20 mov flags,I2DAT ; save ack bit
0497 01BC 22 ret
0498 01BD
0499 01BD
0500 01BD
0501 01BD rdack: ; receives data byte then sends ack
0502 01BD 31 C6 acall rcvbyte ; I2C receive, data returned in acc
0503 01BF
0504 01BF sendack:
0505 01BF 75 99 00 mov I2DAT,#0 ; I2C ack = data low and clock high
0506 01C2 30 9E FD jnb ATN,$
0507 01C5 22 ret
0508 01C6
0509 01C6 rcvbyte: ; I2C receive, data returned in acc

```

Connecting a PC keyboard to the I²C-bus

AN434

```

0510 01C6 75 22 08  mov bitcnt,#8
0511 01C9
0512 01C9 E4      recvb2: clr a
0513 01CA
0514 01CA 45 99      rbit:  orl a,I2DAT
0515 01CC 23      rl a
0516 01CD 30 9E FD  jnb ATN,$
0517 01D0 30 9D 06  jnb DRDY,rbex          ; exit if not a data bit
0518 01D3 D5 22 F4  djnz bitcnt,rbit
0519 01D6
0520 01D6 A2 9F      mov c,RDAT          ; get last bit - do not clear ATN
0521 01D8 33      rlc a              ; shift into byte
0522 01D9
0523 01D9      rbex:
0524 01D9 22      ret
0525 01DA
0526 01DA      ; IIC timer interrupt service
0527 01DA      timerI:
0528 01DA 75 A8 00  mov ien0,#0          ; break point address in ICE751
0529 01DD D2 DD      setb CLRTI         ; clear the interrupt
0530 01DF      fixup:
0531 01DF 75 D8 00  mov I2CFG,#0        ; turn off I2C
0532 01E2 75 98 BC  mov I2CON,#BCXA+BCDR+BCARL+BCSTR+BCSTP
0533 01E5          ; reset I2C flags
0534 01E5 31 E9      acall clrrint
0535 01E7 01 5C      ajmp reset         ; restart program
0536 01E9
0537 01E9
0538 01E9      ;***** call here to make code interruptible *****
0539 01E9
0540 01E9      clrrint:
0541 01E9 32      reti
0542 01EA
0543 01EA      ;***** unused interrupts are vectored to here *****
0544 01EA      badint:
0545 01EA 32      reti
0546 01EB
0547 01EB      #include attable.h
0001+ 01EB      unshift          ; scan code
0002+ 01EB 00      .byte 0          ; 0
0003+ 01EC 00      .byte 0          ; 1 - f9
0004+ 01ED 00      .byte 0          ; 2 - f7
0005+ 01EE 00      .byte 0          ; 3 - f5
0006+ 01EF 00      .byte 0          ; 4 - f3
0007+ 01F0 00      .byte 0          ; 5 - f1
0008+ 01F1 00      .byte 0          ; 6 - f2
0009+ 01F2 00      .byte 0          ; 7 - f2
0010+ 01F3 00      .byte 0          ; 8 -
0011+ 01F4 00      .byte 0          ; 9 - f10
0012+ 01F5 00      .byte 0          ; a - f8
0013+ 01F6 00      .byte 0          ; b - f6
0014+ 01F7 00      .byte 0          ; c - f4
0015+ 01F8 09      .byte 09h       ; d - tab
0016+ 01F9 60      .byte ``        ; e - `
0017+ 01FA 00      .byte 0          ; f -
0018+ 01FB
0019+ 01FB 00      .byte 0          ; 10
0020+ 01FC 00      .byte 0          ; 11 - left shift
0021+ 01FD 00      .byte 0          ; 12
0022+ 01FE 00      .byte 0          ; 13
0023+ 01FF 00      .byte 0          ; 14
0024+ 0200 71      .byte 'q'       ; 15
0025+ 0201 31      .byte '1'       ; 16
0026+ 0202 00      .byte 0          ; 17
0027+ 0203 00      .byte 0          ; 18
0028+ 0204 00      .byte 0          ; 19
0029+ 0205 7A      .byte 'z'       ; 1a

```

Connecting a PC keyboard to the I²C-bus

AN434

```

0030+ 0206 73      .byte 's'   ; 1b
0031+ 0207 61      .byte 'a'   ; 1c
0032+ 0208 77      .byte 'w'   ; 1d
0033+ 0209 32      .byte '2'   ; 1e
0034+ 020A 00      .byte 0     ; 1f
0035+ 020B
0036+ 020B 00      .byte 0     ; 20
0037+ 020C 63      .byte 'c'   ; 21
0038+ 020D 78      .byte 'x'   ; 22
0039+ 020E 64      .byte 'd'   ; 23
0040+ 020F 65      .byte 'e'   ; 24
0041+ 0210 34      .byte '4'   ; 25
0042+ 0211 33      .byte '3'   ; 26
0043+ 0212 00      .byte 0     ; 27
0044+ 0213 00      .byte 0     ; 28
0045+ 0214 20      .byte ' '   ; 29
0046+ 0215 76      .byte 'v'   ; 2a
0047+ 0216 66      .byte 'f'   ; 2b
0048+ 0217 74      .byte 't'   ; 2c
0049+ 0218 72      .byte 'r'   ; 2d
0050+ 0219 35      .byte '5'   ; 2e
0051+ 021A 00      .byte 0     ; 2f
0052+ 021B
0053+ 021B 00      .byte 0     ; 30
0054+ 021C 6E      .byte 'n'   ; 31
0055+ 021D 62      .byte 'b'   ; 32
0056+ 021E 68      .byte 'h'   ; 33
0057+ 021F 67      .byte 'g'   ; 34
0058+ 0220 79      .byte 'y'   ; 35
0059+ 0221 36      .byte '6'   ; 36
0060+ 0222 00      .byte 0     ; 37
0061+ 0223 00      .byte 0     ; 38
0062+ 0224 00      .byte 0     ; 39
0063+ 0225 6D      .byte 'm'   ; 3a
0064+ 0226 6A      .byte 'j'   ; 3b
0065+ 0227 75      .byte 'u'   ; 3c
0066+ 0228 37      .byte '7'   ; 3d
0067+ 0229 38      .byte '8'   ; 3e
0068+ 022A 00      .byte 0     ; 3f
0069+ 022B
0070+ 022B 00      .byte 0     ; 40
0071+ 022C 2C      .byte ',,'  ; 41
0072+ 022D 6B      .byte 'k'   ; 42
0073+ 022E 69      .byte 'i'   ; 43
0074+ 022F 6F      .byte 'o'   ; 44
0075+ 0230 30      .byte '0'   ; 45
0076+ 0231 39      .byte '9'   ; 46
0077+ 0232 00      .byte 0     ; 47
0078+ 0233 00      .byte 0     ; 48
0079+ 0234 2E      .byte '.,'  ; 49
0080+ 0235 2F      .byte '/'   ; 4a
0081+ 0236 6C      .byte 'l'   ; 4b
0082+ 0237 00      .byte ';'   ; 4c
0083+ 0238 70      .byte 'p'   ; 4d
0084+ 0239 2D      .byte '-'   ; 4e
0085+ 023A 00      .byte 0     ; 4f
0086+ 023B
0087+ 023B 00      .byte 0     ; 50
0088+ 023C 2C      .byte ',,'  ; 51
0089+ 023D 00      .byte 0     ; 52
0090+ 023E 00      .byte 0     ; 53
0091+ 023F 5B      .byte '['   ; 54
0092+ 0240 3D      .byte '='   ; 55
0093+ 0241 00      .byte 0     ; 56
0094+ 0242 00      .byte 0     ; 57
0095+ 0243 00      .byte 0     ; 58
0096+ 0244 00      .byte 0     ; 59

```

Connecting a PC keyboard to the I²C-bus

AN434

```

0097+ 0245 0D      .byte 13      ; 5a
0098+ 0246 5D      .byte ']'     ; 5b
0099+ 0247 00      .byte 0       ; 5c
0100+ 0248 5C      .byte 92      ; 5d
0101+ 0249 00      .byte 0       ; 5e
0102+ 024A 00      .byte 0       ; 5f
0103+ 024B
0104+ 024B 00      .byte 0       ; 60
0105+ 024C 00      .byte 0       ; 61
0106+ 024D 00      .byte 0       ; 62
0107+ 024E 00      .byte 0       ; 63
0108+ 024F 00      .byte 0       ; 64
0109+ 0250 00      .byte 0       ; 65
0110+ 0251 08      .byte 8       ; 66
0111+ 0252 00      .byte 0       ; 67
0112+ 0253 00      .byte 0       ; 68
0113+ 0254 00      .byte 0       ; 69
0114+ 0255 00      .byte 0       ; 6a
0115+ 0256 00      .byte 0       ; 6b
0116+ 0257 00      .byte 0       ; 6c
0117+ 0258 00      .byte 0       ; 6d
0118+ 0259 00      .byte 0       ; 6e
0119+ 025A 00      .byte 0       ; 6f
0120+ 025B
0121+ 025B 00      .byte 0       ; 70
0122+ 025C 7F      .byte 127     ; 71
0123+ 025D 00      .byte 0       ; 72
0124+ 025E 00      .byte 0       ; 73
0125+ 025F 00      .byte 0       ; 74
0126+ 0260 1B      .byte 27      ; 75
0127+ 0261 00      .byte 0       ; 76
0128+ 0262 00      .byte 0       ; 77
0129+ 0263 00      .byte 0       ; 78
0130+ 0264 2B      .byte '+'     ; 79
0131+ 0265 00      .byte 0       ; 7a
0132+ 0266 2D      .byte '-'     ; 7b
0133+ 0267 2A      .byte '*'     ; 7c
0134+ 0268 00      .byte 0       ; 7d
0135+ 0269 00      .byte 0       ; 7e
0136+ 026A 00      .byte 0       ; 7f
0137+ 026B
0138+ 026B      shift:                ; scan code
0139+ 026B 00      .byte 0       ; 0
0140+ 026C 00      .byte 0       ; 1 - f9
0141+ 026D 00      .byte 0       ; 2 - f7
0142+ 026E 00      .byte 0       ; 3 - f5
0143+ 026F 00      .byte 0       ; 4 - f3
0144+ 0270 00      .byte 0       ; 5 - f1
0145+ 0271 00      .byte 0       ; 6 - f2
0146+ 0272 00      .byte 0       ; 7 - f2
0147+ 0273 00      .byte 0       ; 8 -
0148+ 0274 00      .byte 0       ; 9 - f10
0149+ 0275 00      .byte 0       ; a - f8
0150+ 0276 00      .byte 0       ; b - f6
0151+ 0277 00      .byte 0       ; c - f4
0152+ 0278 00      .byte 0       ; d - tab
0153+ 0279 7E      .byte '~'     ; e - ~
0154+ 027A 00      .byte 0       ; f -
0155+ 027B
0156+ 027B 00      .byte 0       ; 10
0157+ 027C 00      .byte 0       ; 11 -
0158+ 027D 00      .byte 0       ; 12
0159+ 027E 00      .byte 0       ; 13
0160+ 027F 00      .byte 0       ; 14
0161+ 0280 51      .byte 'Q'     ; 15
0162+ 0281 21      .byte '!'     ; 16
0163+ 0282 00      .byte 0       ; 17

```


Connecting a PC keyboard to the I²C-bus

AN434

```

0164+ 0283 00      .byte 0      ; 18
0165+ 0284 00      .byte 0      ; 19
0166+ 0285 5A      .byte 'Z'    ; 1a
0167+ 0286 53      .byte 'S'    ; 1b
0168+ 0287 41      .byte 'A'    ; 1c
0169+ 0288 57      .byte 'W'    ; 1d
0170+ 0289 40      .byte '@'    ; 1e
0171+ 028A 00      .byte 0      ; 1f
0172+ 028B
0173+ 028B 00      .byte 0      ; 20
0174+ 028C 43      .byte 'C'    ; 21
0175+ 028D 58      .byte 'X'    ; 22
0176+ 028E 44      .byte 'D'    ; 23
0177+ 028F 45      .byte 'E'    ; 24
0178+ 0290 24      .byte '$'    ; 25
0179+ 0291 23      .byte '#'    ; 26
0180+ 0292 00      .byte 0      ; 27
0181+ 0293 00      .byte 0      ; 28
0182+ 0294 20      .byte ' '    ; 29
0183+ 0295 56      .byte 'V'    ; 2a
0184+ 0296 46      .byte 'F'    ; 2b
0185+ 0297 54      .byte 'T'    ; 2c
0186+ 0298 52      .byte 'R'    ; 2d
0187+ 0299 25      .byte ''     ; 2e
0188+ 029A 00      .byte 0      ; 2f
0189+ 029B
0190+ 029B 00      .byte 0      ; 30
0191+ 029C 4E      .byte 'N'    ; 31
0192+ 029D 42      .byte 'B'    ; 32
0193+ 029E 48      .byte 'H'    ; 33
0194+ 029F 47      .byte 'G'    ; 34
0195+ 02A0 59      .byte 'Y'    ; 35
0196+ 02A1 5E      .byte '^'    ; 36
0197+ 02A2 00      .byte 0      ; 37
0198+ 02A3 00      .byte 0      ; 38
0199+ 02A4 00      .byte 0      ; 39
0200+ 02A5 4D      .byte 'M'    ; 3a
0201+ 02A6 4A      .byte 'J'    ; 3b
0202+ 02A7 55      .byte 'U'    ; 3c
0203+ 02A8 26      .byte '&'    ; 3d
0204+ 02A9 2A      .byte '*'    ; 3e
0205+ 02AA 00      .byte 0      ; 3f
0206+ 02AB
0207+ 02AB 00      .byte 0      ; 40
0208+ 02AC 3C      .byte '<'    ; 41
0209+ 02AD 4B      .byte 'K'    ; 42
0210+ 02AE 49      .byte 'I'    ; 43
0211+ 02AF 4F      .byte 'O'    ; 44
0212+ 02B0 29      .byte ')'    ; 45
0213+ 02B1 28      .byte '('    ; 46
0214+ 02B2 00      .byte 0      ; 47
0215+ 02B3 00      .byte 0      ; 48
0216+ 02B4 3E      .byte '>'    ; 49
0217+ 02B5 3F      .byte '?'    ; 4a
0218+ 02B6 4C      .byte 'L'    ; 4b
0219+ 02B7 3A      .byte ':'    ; 4c
0220+ 02B8 50      .byte 'P'    ; 4d
0221+ 02B9 5F      .byte '_'    ; 4e
0222+ 02BA 00      .byte 0      ; 4f
0223+ 02BB
0224+ 02BB 00      .byte 0      ; 50
0225+ 02BC 00      .byte 0      ; 51
0226+ 02BD 22      .byte '"'    ; 52
0227+ 02BE 00      .byte 0      ; 53
0228+ 02BF 7B      .byte '{'    ; 54
0229+ 02C0 2B      .byte '+'    ; 55
0230+ 02C1 00      .byte 0      ; 56

```

Connecting a PC keyboard to the I²C-bus

AN434

```
0231+ 02C2 00      .byte 0      ; 57
0232+ 02C3 00      .byte 0      ; 58
0233+ 02C4 00      .byte 0      ; 59
0234+ 02C5 0D      .byte 13     ; 5a
0235+ 02C6 7D      .byte '}'    ; 5b
0236+ 02C7 00      .byte 0      ; 5c
0237+ 02C8 7C      .byte '| '   ; 5d
0238+ 02C9 00      .byte 0      ; 5e
0239+ 02CA 00      .byte 0      ; 5f
0240+ 02CB
0241+ 02CB 00      .byte 0      ; 60
0242+ 02CC 00      .byte 0      ; 61
0243+ 02CD 00      .byte 0      ; 62
0244+ 02CE 00      .byte 0      ; 63
0245+ 02CF 00      .byte 0      ; 64
0246+ 02D0 00      .byte 0      ; 65
0247+ 02D1 08      .byte 8      ; 66
0248+ 02D2 00      .byte 0      ; 67
0249+ 02D3 00      .byte 0      ; 68
0250+ 02D4 31      .byte '1'    ; 69
0251+ 02D5 00      .byte 0      ; 6a
0252+ 02D6 34      .byte '4'    ; 6b
0253+ 02D7 37      .byte '7'    ; 6c
0254+ 02D8 00      .byte 0      ; 6d
0255+ 02D9 00      .byte 0      ; 6e
0256+ 02DA 00      .byte 0      ; 6f
0257+ 02DB
0258+ 02DB 30      .byte '0'    ; 70
0259+ 02DC 2E      .byte '.'    ; 71
0260+ 02DD 32      .byte '2'    ; 72
0261+ 02DE 35      .byte '5'    ; 73
0262+ 02DF 36      .byte '6'    ; 74
0263+ 02E0 38      .byte '8'    ; 75
0264+ 02E1 1B      .byte 27     ; 76
0265+ 02E2 00      .byte 0      ; 77
0266+ 02E3 00      .byte 0      ; 78
0267+ 02E4 2B      .byte '+'    ; 79
0268+ 02E5 33      .byte '3'    ; 7a
0269+ 02E6 2D      .byte '-'    ; 7b
0270+ 02E7 00      .byte 0      ; 7c
0271+ 02E8 39      .byte '9'    ; 7d
0272+ 02E9 00      .byte 0      ; 7e
0273+ 02EA 00      .byte 0      ; 7f
0274+ 02EB
0275+ 02EB
0548 02EB
0549 02EB      .end
tasm: Number of errors = 0
```