Using a PCAT keyboard with a TICkit 63/74

Submitted by:

Glenn Clark - Protean Logic Inc.

Introduction

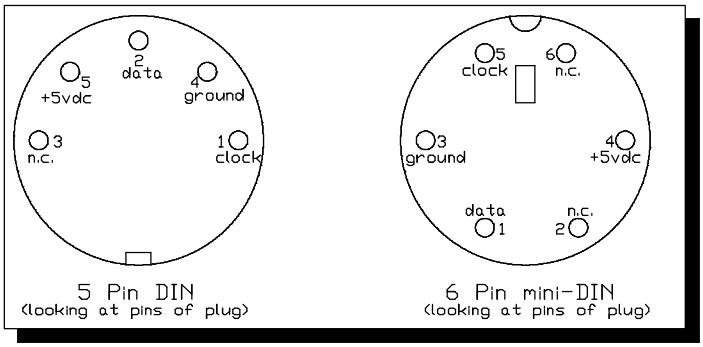
Almost every type of project is becomming more sophisticated these days. Often, this sophistication takes the form of programmability or user interface. Fortunately full featured keyboards are inexpensive and readily available in the form of the PCAT keyboard. These keyboards come with either a larger 5 pin circular DIN connector or a 6 pin smaller circular DIN connector. Regardless of the connector, the same electrical and logical interface is used. Other devices also use the PCAT interface standard like bar code scanners, as well.

The application note details the rather simple process of using an PCAT keyboard for alphanumeric input on a TICkit 63/74 processor. An added bonus is that the PCAT keyboards can buffer keystrokes and delay transmission until the interface signals ready. This means that the TICkit processor can perform other tasks while the keyboard buffers keystrokes and then deal with the data when the TICkit is ready. Effectively, this allows the keyboard or scanner to be utilized in background behind the main TICkit process. The two routines embedded into the TICkit internal library are very simple to use and transfer 8 bit scan codes or commands between the TICkit and the keyboard.

The most difficult part of this application, is to convert the scan codes generated by the PCAT keyboard into usable ASCII codes. First, however, lets examine the electrical nature of the interface.

The Electrical and Logical Interface

Even though the connectors have 5 or 6 pins, only 4 are used by the keyboard. These pins are labeled as follows:



Data	The open collector (drain) signal line for transmitting data bits (must be pulled high)
Clock	The open collector (drain) signal line for clocking data over the cable (must be pulled high), keyboard generates the clock but the TICkit can halt clocking by holding this line low
+5 vdc	The 5 volt power supply for the keyboard
Gnd	The power and signal ground for the keyboard

The two lines used for data transfer are the data and clock lines. For the most part, the keyboard controls the clocking of data to and from the host (TICkit). The host can halt clocking of data or prevent it by holding the clock line low. This is how the host signals that it is not ready to receive data. While the clock line is held low, the keyboard will simply buffer key strokes (scan codes) for later transmission.

The data line also has a special meaning besides sending bits. If the data line is held low prior to the keyboard clocking a scan code, the keyboard assumes it will be receiving a command. Thus, when the clock line is clocked by the keyboard, it is clocking bits FROM the host, not to it. This is how basic elements such as reset, scan code set, or LED control are sent to the keyboard.

The Simple View of Reading the Keyboard

The easiest way to read the keyboard is to use the keybd_input() function contained in the keybd.lib library. This function has a single parameter which is either 0 for false or 255 for true. When the parameter is any value other than zero, the function will wait until a keypress is detected. It will then convert the scan code to an 8 bit value which is either the ASCII code for the key or a special value for the non-character key. When the parameter is zero, the function will return after only one attemp to read the keyboard. If no key presses were stored in the keyboard, then hex FF (255b) is returned. If a key press is available, then its code is converted to an 8 bit value like before.

The following is a simple example which echoes whatever is typed to the console screen of debug:

```
; KEYBD.BAS - Program to convert keyboard input to console output
; Primary Source File

;12-10-2000 - Initial writing

; End Revision History

DEF tic63_i
LIB fbasic.lib

DEF pcat_clock pin_a2
DEF pcat_data pin_a3
DEF pcat_wait 1000w

INC keybd.inc
LIB keybd.lib

; The four standard interrupt handling functions follow
FUNC none irq    ; Executes when the int line goes low
BEGIN
ENDFUN

FUNC none global_int    ; Internal Global Interrupt
BEGIN
ENDFUN

FUNC none timer_int    ; Executes when timer interval expires
BEGIN
ENDFUN

FUNC none stack_overflow    ; Executes when stack overflows
BEGIN
    reset()    ; only sure way to recover
ENDFUN

FUNC none main    ; First Function to execute on startup
BEGIN
    =( _keybd_status, 0b )
    pin_low( pcat_clock )

    rs_conparam_set( debug_pin )
    REP
        con_out_char( keybd_input( True ))
    LOOP
ENDFUN
```

Most of the code shown below is just standard stuff for an FBasic program on the TICkit. However, notice that three values are assigned immediately before referencing the keybd.lib file. These values, keybd_clodk, keybd_data, and keybd_wait, are required by the keybd_input routine to know which pins are used for clock and data. The wait parameter is just a loop count and determines the number of times the internal clock generator looks for a data signal. The reasonable value for wait depends entirely upon the keyboard used. 1000 seems to be a very reliable value for keybd_wait.

The Scan Codes and Their Interpretation

As mentioned earlier, the PCAT keyboards do not send ASCII or another character based equivelent. Instead they send scan codes, which are arbitrary 8 bit codes assigned by IBM. These codes have become an ad-hoc standard are are very consistent from manufacturer to manufacturer as you would expect. When a key is pressed, the keyboard sends the unique scancode for that key, called the "make" scan code. If the key is held, the make scan code is repeated according to the typematic rate set in the keyboard. When the key is released, the "break" codes are sent which are the hex code F0 followed by the scan code for the key. Thus, the host can know when the key is pressed and when it is released. This can be useful information for situations like games or interactive user interfaces where the timing of the keypress is important.

The code below is the lookup table used to convert scan codes to ASCII equivelents and is contained in the file keybd.inc:

```
; Include File for PCAT scan code to ASCII conversion

      ; Secondary Source File

      ; End Revision History

      ALLOC byte pcat_low_scans[128]
      INIT pcat_low_scans[0x00w] 0b ; illegal scan code can mean buffer overrun
      INIT pcat_low_scans[0x01w] 0x89b ; Function F9
      INIT pcat_low_scans[0x02w] 0b ;
      INIT pcat_low_scans[0x03w] 0x85b ; Function F5
      INIT pcat_low_scans[0x04w] 0x83b ; Function F3
      INIT pcat_low_scans[0x05w] 0x81b ; Function F1
      INIT pcat_low_scans[0x06w] 0x82b ; Function F2
      INIT pcat_low_scans[0x07w] 0x8Cb ; Function F12
      INIT pcat_low_scans[0x08w] 0b ;
      INIT pcat_low_scans[0x09w] 0x8Ab ; Function F10
      INIT pcat_low_scans[0x0Aw] 0x88b ; Function F8
      INIT pcat_low_scans[0x0Bw] 0x86b ; Function F6
      INIT pcat_low_scans[0x0Cw] 0x84b ; Function F4
      INIT pcat_low_scans[0x0Dw] 0x09b ; Tab
      INIT pcat_low_scans[0x0Ew] 0x60b ; Main keypad `/~\~
      INIT pcat_low_scans[0x0Fw] 0b ;

      INIT pcat_low_scans[0x10w] 0b ;
      INIT pcat_low_scans[0x11w] 0b ; Left Alt
      INIT pcat_low_scans[0x12w] 0b ; Left Shift
      INIT pcat_low_scans[0x13w] 0b ;
      INIT pcat_low_scans[0x14w] 0b ; 14 77 is the pause keycode**
      INIT pcat_low_scans[0x15w] 0x71b ; Letter Q
      INIT pcat_low_scans[0x16w] 0x31b ; Number 1
      INIT pcat_low_scans[0x17w] 0b ;
      INIT pcat_low_scans[0x18w] 0b ;
      INIT pcat_low_scans[0x19w] 0b ;
      INIT pcat_low_scans[0x1Aw] 0x7Ab ; Letter Z
      INIT pcat_low_scans[0x1Bw] 0x73b ; Letter S
      INIT pcat_low_scans[0x1Cw] 0x61b ; Letter A
      INIT pcat_low_scans[0x1Dw] 0x77b ; Letter W
      INIT pcat_low_scans[0x1Ew] 0x32b ; Number 2
      INIT pcat_low_scans[0x1Fw] 0b ;
```

```
INIT pcat_low_scans[0x20w] 0b ;
INIT pcat_low_scans[0x21w] 0x63b ; Letter C
INIT pcat_low_scans[0x22w] 0x78b ; Letter X
INIT pcat_low_scans[0x23w] 0x64b ; Letter D
INIT pcat_low_scans[0x24w] 0x65b ; Letter E
INIT pcat_low_scans[0x25w] 0x34b ; Number 4
INIT pcat_low_scans[0x26w] 0x33b ; Number 3
INIT pcat_low_scans[0x27w] 0b ;
INIT pcat_low_scans[0x28w] 0b ;
INIT pcat_low_scans[0x29w] 0x20b ; Space
INIT pcat_low_scans[0x2Aw] 0x76b ; Letter V
INIT pcat_low_scans[0x2Bw] 0x66b ; Letter F
INIT pcat_low_scans[0x2Cw] 0x74b ; Letter T
INIT pcat_low_scans[0x2Dw] 0x72b ; Letter R
INIT pcat_low_scans[0x2Ew] 0x35b ; Number 5
INIT pcat_low_scans[0x2Fw] 0b ;

INIT pcat_low_scans[0x30w] 0b ;
INIT pcat_low_scans[0x31w] 0x6Eb ; Letter N
INIT pcat_low_scans[0x32w] 0x62b ; Letter B
INIT pcat_low_scans[0x33w] 0x68b ; Letter H
INIT pcat_low_scans[0x34w] 0x67b ; Letter G
INIT pcat_low_scans[0x35w] 0x79b ; Letter Y
INIT pcat_low_scans[0x36w] 0x36b ; Number 6
INIT pcat_low_scans[0x37w] 0b ;
INIT pcat_low_scans[0x38w] 0b ;
INIT pcat_low_scans[0x39w] 0b ;
INIT pcat_low_scans[0x3Aw] 0x6Db ; Letter M
INIT pcat_low_scans[0x3Bw] 0x6Ab ; Letter J
INIT pcat_low_scans[0x3Cw] 0x75b ; Letter U
INIT pcat_low_scans[0x3Dw] 0x37b ; Number 7
INIT pcat_low_scans[0x3Ew] 0x38b ; Number 8
INIT pcat_low_scans[0x3Fw] 0b ;

INIT pcat_low_scans[0x40w] 0b ;
INIT pcat_low_scans[0x41w] 0x2Cb ; Main keypad ,/<
INIT pcat_low_scans[0x42w] 0x6Bb ; Letter K
INIT pcat_low_scans[0x43w] 0x69b ; Letter I
INIT pcat_low_scans[0x44w] 0x6Fb ; Letter O
INIT pcat_low_scans[0x45w] 0x30b ; Number 0
INIT pcat_low_scans[0x46w] 0x39b ; Number 9
INIT pcat_low_scans[0x47w] 0b ;
INIT pcat_low_scans[0x48w] 0b ;
INIT pcat_low_scans[0x49w] 0x2Eb ; Main keypad ./>
INIT pcat_low_scans[0x4Aw] 0x2Fb ; Main keypad ?//
INIT pcat_low_scans[0x4Bw] 0x6Cb ; Letter L
INIT pcat_low_scans[0x4Cw] 0x3Bb ; Main keypad ;/:
INIT pcat_low_scans[0x4Dw] 0x70b ; Letter P
INIT pcat_low_scans[0x4Ew] 0x2Db ; Main keypad -/_
INIT pcat_low_scans[0x4Fw] 0b ;
```

```
INIT pcat_low_scans[0x50w] 0b ;
INIT pcat_low_scans[0x51w] 0b ;
INIT pcat_low_scans[0x52w] 0x27b ; Main keypad '/"
INIT pcat_low_scans[0x53w] 0b ;
INIT pcat_low_scans[0x54w] 0x5Bb ; Main keypad [/{
INIT pcat_low_scans[0x55w] 0x3Db ; Main keypad =/+
INIT pcat_low_scans[0x56w] 0b ;
INIT pcat_low_scans[0x57w] 0b ;
INIT pcat_low_scans[0x58w] 0b ; Caps lock
INIT pcat_low_scans[0x59w] 0b ; Right shift
INIT pcat_low_scans[0x5Aw] 0x0Db ; Main enter
INIT pcat_low_scans[0x5Bw] 0x5Db ; Main keypad ]/}
INIT pcat_low_scans[0x5Cw] 0b ;
INIT pcat_low_scans[0x5Dw] 0x5Cb ; Main keypad |/\
INIT pcat_low_scans[0x5Ew] 0b ;
INIT pcat_low_scans[0x5Fw] 0b ;

INIT pcat_low_scans[0x60w] 0b ;
INIT pcat_low_scans[0x61w] 0x60b ; Main keypad `/
INIT pcat_low_scans[0x62w] 0b ;
INIT pcat_low_scans[0x63w] 0b ;
INIT pcat_low_scans[0x64w] 0b ;
INIT pcat_low_scans[0x65w] 0b ;
INIT pcat_low_scans[0x66w] 0x08b ; Back space
INIT pcat_low_scans[0x67w] 0b ;
INIT pcat_low_scans[0x68w] 0b ;
INIT pcat_low_scans[0x69w] 0x31b ; Numpad 1
INIT pcat_low_scans[0x6Aw] 0b ;
INIT pcat_low_scans[0x6Bw] 0x34b ; Numpad 4
INIT pcat_low_scans[0x6Cw] 0x37b ; Numpad 7
INIT pcat_low_scans[0x6Dw] 0b ;
INIT pcat_low_scans[0x6Ew] 0b ;
INIT pcat_low_scans[0x6Fw] 0b ;

INIT pcat_low_scans[0x70w] 0x30b ; Numpad 0
INIT pcat_low_scans[0x71w] 0x2Eb ; Numpad DEL
INIT pcat_low_scans[0x72w] 0x32b ; Numpad 2
INIT pcat_low_scans[0x73w] 0x35b ; Numpad 5
INIT pcat_low_scans[0x74w] 0x36b ; Numpad 6
INIT pcat_low_scans[0x75w] 0x38b ; Numpad 8
INIT pcat_low_scans[0x76w] 0x1Bb ; Escape
INIT pcat_low_scans[0x77w] 0b ; Num lock
INIT pcat_low_scans[0x78w] 0x8Bb ; Function F11
INIT pcat_low_scans[0x79w] 0x2Bb ; Numpad +
INIT pcat_low_scans[0x7Aw] 0x33b ; Numpad 3
INIT pcat_low_scans[0x7Bw] 0x2Db ; Numpad -
INIT pcat_low_scans[0x7Cw] 0x2Ab ; Numpad * ; Shifted is print screen**
INIT pcat_low_scans[0x7Dw] 0x39b ; Numpad 9
INIT pcat_low_scans[0x7Ew] 0b ; Scroll lock
INIT pcat_low_scans[0x7Fw] 0b ;
```

```
ALLOC byte pcat_shift_scans[128]
INIT pcat_shift_scans[0x00w] 0b ; illegal scan code can mean buffer overrun
INIT pcat_shift_scans[0x01w] 0xA9b ; Function F9
INIT pcat_shift_scans[0x02w] 0b ;
INIT pcat_shift_scans[0x03w] 0xA5b ; Function F5
INIT pcat_shift_scans[0x04w] 0xA3b ; Function F3
INIT pcat_shift_scans[0x05w] 0xA1b ; Function F1
INIT pcat_shift_scans[0x06w] 0xA2b ; Function F2
INIT pcat_shift_scans[0x07w] 0xACb ; Function F12
INIT pcat_shift_scans[0x08w] 0b ;
INIT pcat_shift_scans[0x09w] 0xAAb ; Function F10
INIT pcat_shift_scans[0x0Aw] 0xA8b ; Function F8
INIT pcat_shift_scans[0x0Bw] 0xA6b ; Function F6
INIT pcat_shift_scans[0x0Cw] 0xA4b ; Function F4
INIT pcat_shift_scans[0x0Dw] 0x0Bb ; Tab
INIT pcat_shift_scans[0x0Ew] 0x7Eb ; Main keypad `/~\~
INIT pcat_shift_scans[0x0Fw] 0b ;

INIT pcat_shift_scans[0x10w] 0b ;
INIT pcat_shift_scans[0x11w] 0b ; Left Alt
INIT pcat_shift_scans[0x12w] 0b ; Left Shift
INIT pcat_shift_scans[0x13w] 0b ;
INIT pcat_shift_scans[0x14w] 0b ; 14 77 is the pause keycode**
INIT pcat_shift_scans[0x15w] 0x51b ; Letter Q
INIT pcat_shift_scans[0x16w] 0x21b ; Number 1
INIT pcat_shift_scans[0x17w] 0b ;
INIT pcat_shift_scans[0x18w] 0b ;
INIT pcat_shift_scans[0x19w] 0b ;
INIT pcat_shift_scans[0x1Aw] 0x5Ab ; Letter Z
INIT pcat_shift_scans[0x1Bw] 0x53b ; Letter S
INIT pcat_shift_scans[0x1Cw] 0x41b ; Letter A
INIT pcat_shift_scans[0x1Dw] 0x57b ; Letter W
INIT pcat_shift_scans[0x1Ew] 0x40b ; Number 2
INIT pcat_shift_scans[0x1Fw] 0b ;

INIT pcat_shift_scans[0x20w] 0b ;
INIT pcat_shift_scans[0x21w] 0x43b ; Letter C
INIT pcat_shift_scans[0x22w] 0x58b ; Letter X
INIT pcat_shift_scans[0x23w] 0x44b ; Letter D
INIT pcat_shift_scans[0x24w] 0x45b ; Letter E
INIT pcat_shift_scans[0x25w] 0x24b ; Number 4
INIT pcat_shift_scans[0x26w] 0x23b ; Number 3
INIT pcat_shift_scans[0x27w] 0b ;
INIT pcat_shift_scans[0x28w] 0b ;
INIT pcat_shift_scans[0x29w] 0x20b ; Space
INIT pcat_shift_scans[0x2Aw] 0x56b ; Letter V
INIT pcat_shift_scans[0x2Bw] 0x46b ; Letter F
INIT pcat_shift_scans[0x2Cw] 0x54b ; Letter T
INIT pcat_shift_scans[0x2Dw] 0x52b ; Letter R
INIT pcat_shift_scans[0x2Ew] 0x25b ; Number 5
INIT pcat_shift_scans[0x2Fw] 0b ;
```

```
INIT pcat_shift_scans[0x30w] 0b ;
INIT pcat_shift_scans[0x31w] 0x4Eb ; Letter N
INIT pcat_shift_scans[0x32w] 0x42b ; Letter B
INIT pcat_shift_scans[0x33w] 0x48b ; Letter H
INIT pcat_shift_scans[0x34w] 0x47b ; Letter G
INIT pcat_shift_scans[0x35w] 0x59b ; Letter Y
INIT pcat_shift_scans[0x36w] 0x5Eb ; Number 6
INIT pcat_shift_scans[0x37w] 0b ;
INIT pcat_shift_scans[0x38w] 0b ;
INIT pcat_shift_scans[0x39w] 0b ;
INIT pcat_shift_scans[0x3Aw] 0x4Db ; Letter M
INIT pcat_shift_scans[0x3Bw] 0x4Ab ; Letter J
INIT pcat_shift_scans[0x3Cw] 0x55b ; Letter U
INIT pcat_shift_scans[0x3Dw] 0x26b ; Number 7
INIT pcat_shift_scans[0x3Ew] 0x2Ab ; Number 8
INIT pcat_shift_scans[0x3Fw] 0b ;

INIT pcat_shift_scans[0x40w] 0b ;
INIT pcat_shift_scans[0x41w] 0x3Cb ; Main keypad ,/<
INIT pcat_shift_scans[0x42w] 0x4Bb ; Letter K
INIT pcat_shift_scans[0x43w] 0x49b ; Letter I
INIT pcat_shift_scans[0x44w] 0x4Fb ; Letter O
INIT pcat_shift_scans[0x45w] 0x29b ; Number 0
INIT pcat_shift_scans[0x46w] 0x28b ; Number 9
INIT pcat_shift_scans[0x47w] 0b ;
INIT pcat_shift_scans[0x48w] 0b ;
INIT pcat_shift_scans[0x49w] 0x3Eb ; Main keypad ./>
INIT pcat_shift_scans[0x4Aw] 0x3Fb ; Main keypad ?//
INIT pcat_shift_scans[0x4Bw] 0x4Cb ; Letter L
INIT pcat_shift_scans[0x4Cw] 0x3Ab ; Main keypad ;/:
INIT pcat_shift_scans[0x4Dw] 0x50b ; Letter P
INIT pcat_shift_scans[0x4Ew] 0x5Fb ; Main keypad -/_
INIT pcat_shift_scans[0x4Fw] 0b ;

INIT pcat_shift_scans[0x50w] 0b ;
INIT pcat_shift_scans[0x51w] 0b ;
INIT pcat_shift_scans[0x52w] 0x22b ; Main keypad '/"
INIT pcat_shift_scans[0x53w] 0b ;
INIT pcat_shift_scans[0x54w] 0x7Bb ; Main keypad [/{
INIT pcat_shift_scans[0x55w] 0x2Bb ; Main keypad =/+
INIT pcat_shift_scans[0x56w] 0b ;
INIT pcat_shift_scans[0x57w] 0b ;
INIT pcat_shift_scans[0x58w] 0b ; Caps lock
INIT pcat_shift_scans[0x59w] 0b ; Right shift
INIT pcat_shift_scans[0x5Aw] 0x0Db ; Main enter
INIT pcat_shift_scans[0x5Bw] 0x7Db ; Main keypad ]/}
INIT pcat_shift_scans[0x5Cw] 0b ;
INIT pcat_shift_scans[0x5Dw] 0x7Cb ; Main keypad |/\
INIT pcat_shift_scans[0x5Ew] 0b ;
INIT pcat_shift_scans[0x5Fw] 0b ;
```

```
INIT pcat_shift_scans[0x60w] 0b ;
INIT pcat_shift_scans[0x61w] 0x7Eb ; Main keypad '/~ 102key
INIT pcat_shift_scans[0x62w] 0b ;
INIT pcat_shift_scans[0x63w] 0b ;
INIT pcat_shift_scans[0x64w] 0b ;
INIT pcat_shift_scans[0x65w] 0b ;
INIT pcat_shift_scans[0x66w] 0x08b ; Back space
INIT pcat_shift_scans[0x67w] 0b ;
INIT pcat_shift_scans[0x68w] 0b ;
INIT pcat_shift_scans[0x69w] 0x91b ; Numpad 1 ; end
INIT pcat_shift_scans[0x6Aw] 0b ;
INIT pcat_shift_scans[0x6Bw] 0x94b ; Numpad 4 ; arrow left
INIT pcat_shift_scans[0x6Cw] 0x97b ; Numpad 7 ; home
INIT pcat_shift_scans[0x6Dw] 0b ;
INIT pcat_shift_scans[0x6Ew] 0b ;
INIT pcat_shift_scans[0x6Fw] 0b ;

INIT pcat_shift_scans[0x70w] 0x90b ; Numpad 0 ; insert
INIT pcat_shift_scans[0x71w] 0x7Fb ; Numpad DEL ; period '.'
INIT pcat_shift_scans[0x72w] 0x92b ; Numpad 2 ; arrow down
INIT pcat_shift_scans[0x73w] 0x95b ; Numpad 5 ; nothing
INIT pcat_shift_scans[0x74w] 0x96b ; Numpad 6 ; arrow right
INIT pcat_shift_scans[0x75w] 0x98b ; Numpad 8 ; arrow up
INIT pcat_shift_scans[0x76w] 0x1Bb ; Escape
INIT pcat_shift_scans[0x77w] 0b ; Num lock
INIT pcat_shift_scans[0x78w] 0xABb ; Function F11
INIT pcat_shift_scans[0x79w] 0x2Bb ; Numpad + ;
INIT pcat_shift_scans[0x7Aw] 0x93b ; Numpad 3 ; pg down
INIT pcat_shift_scans[0x7Bw] 0x2Db ; Numpad - ;
INIT pcat_shift_scans[0x7Cw] 0x2Ab ; Numpad * ; Shifted is print screen**
INIT pcat_shift_scans[0x7Dw] 0x99b ; Numpad 9 ; pg up
INIT pcat_shift_scans[0x7Ew] 0b ; Scroll lock
INIT pcat_shift_scans[0x7Fw] 0b ;

; scan code 83 is Function F7
; Print Screen produces E0 12 E0 7C
; Pause produces 14 77 F0 14 F0 77
; Numpad Enter produces E0 5A
; Numpad / produces E0 4A
; Right Alt produces E0 11
; Right Ctrl produces E0 14

; Cursor Insert produces E0 52
; Cursor Delete produces E0 53
; Cursor Left Arrow produces E0 4B
; Cursor Home produces E0 47
; Cursor End produces E0 4F
; Cursor Up Arrow produces E0 48
; Cursor Down Arrow produces E0 50
; Cursor Page Up produces E0 49
; Cursor Page Down produces E0 51
; Cursor Right Arrow produces E0 4D
```

You will not really need to concern yourself with the scan codes, however. Just reference this keybd.inc in an INCLUDE statement at the beginning of your program and all the code above is automatically included in your program.

The next section of code comes from the keybd.lib file. This executable code handles all the key buffering and sorts out the meanings of the shift, caps lock, alt, and control keys. It generates standard ASCII except for the case of the alt and

shifted alt keys where is returns the ASCII codes with the upper bit set. This same method is used for the cursor control keys.

```
; KEYBD.LIB Library routine to handle scan codes from PCAT keyboard
; Secondary Source File

; requires the followins symbols to be defined:

    ; PCAT_CLOCK
    ; PCAT_DATA
    ; PCAT_WAIT

; End Revision History

GLOBAL byte _keybd_status

FUNC none keybd_leds
    PARAM byte led_pattern
BEGIN
    keybdport_out( pcat_clock, pcat_data, pcat_wait, 0xEDb )
    IF ==( keybdport_in( pcat_clock, pcat_data, pcat_wait ), 0xFAb )
        keybdport_out( pcat_clock, pcat_data, pcat_wait, ~
         ~ b_and(led_pattern, 0y00000111b ))
        IF ==( keybdport_in( pcat_clock, pcat_data, pcat_wait ), 0xFAb )
        ENDIF
    ENDIF
ENDFUN

FUNC byte keybd_input
    PARAM byte wait_flag

    DEF keybd_scrolllock 0y00000001b
    DEF keybd_numlock    0y00000010b
    DEF keybd_capslock   0y00000100b
    DEF keybd_shift      0y00001000b
    DEF keybd_ctrl       0y00010000b
    DEF keybd_alt        0y00100000b
    DEF keybd_ignore     0y01000000b
    DEF keybd_extended   0y10000000b
BEGIN
    =( exit_value, 0b )
    REP
        =( exit_value, keybdport_in( pcat_clock, pcat_data, pcat_wait ))
        IF b_test( _keybd_status, keybd_extended )
            IF ==( exit_value, 0xF0b )
                b_set( _keybd_status, keybd_ignore )
                =( exit_value, 0b )

            ELSEIF ==( exit_value, 0x5Ab )
                ; numpad enter
                IF b_test( _keybd_status, keybd_ignore )
                    b_clear( _keybd_status, keybd_ignore )
                    =( exit_value, 0b )
                ELSE
                    =( exit_value, 13b )
                ENDIF
```

```
        ELSEIF ==( exit_value, 0x4Ab )
            ; numpad /
            IF b_test( _keybd_status, keybd_ignore )
                b_clear( _keybd_status, keybd_ignore )
                =( exit_value, 0b )
            ELSE
                =( exit_value, 0x2Fb )
            ENDIF

        ELSEIF ==( exit_value, 0x11b ) ; right alt ; E0
            IF b_test( _keybd_status, keybd_ignore )
                b_clear( _keybd_status, keybd_alt | keybd_ignore )
            ELSE
                b_set( _keybd_status, keybd_alt )
            ENDIF

            =( exit_value, 0b )

        ELSEIF ==( exit_value, 0x14b ) ; right ctrl ; E0
            IF b_test( _keybd_status, keybd_ignore )
                b_clear( _keybd_status, keybd_ctrl | keybd_ignore )
            ELSE
                b_set( _keybd_status, keybd_ctrl )
            ENDIF

            =( exit_value, 0b )

        ELSEIF ==( exit_value, 0x12b )
            ; Print Scrn/SysRq
            IF b_test( _keybd_status, keybd_ignore )
                b_clear( _keybd_status, keybd_ignore )
                =( exit_value, 0b )
            ELSE
                =( exit_value, 0x9Fb )
            ENDIF

        ELSEIF ==( exit_value, 0xFFb )
            ; no key data
            =( exit_value, 0b )

        ELSEIF >( exit_value, 0x68b )
            ; cursor control keys similar to num keypad

            =( exit_value, ee_read( ~
             ~ pcat_shift_scans[to_word(exit_value )]))

        ELSE
            ; unknown keypress
            =( exit_value, 0b )
            b_clear( _keybd_status, keybd_ignore | keybd_extended )
        ENDIF

        b_clear( _keybd_status, keybd_extended )
    ELSE
        IF ==( exit_value, 0xF0b )
            ; key is being released so ignore
            b_set( _keybd_status, keybd_ignore )
            =( exit_value, 0b )
```

```
ELSEIF ==( exit_value, 0xE0b )
    ; key code is for num pad or something
    b_set( _keybd_status, keybd_extended )

    =( exit_value, 0b )

ELSEIF ==( exit_value, 0x12b ) ; left shift
    IF b_test( _keybd_status, keybd_ignore )
        b_clear( _keybd_status, keybd_shift | keybd_ignore )
    ELSE
        b_set( _keybd_status, keybd_shift )
    ENDIF

    =( exit_value, 0b )

ELSEIF ==( exit_value, 0x59b ) ; right shift
    IF b_test( _keybd_status, keybd_ignore )
        b_clear( _keybd_status, keybd_shift | keybd_ignore )
    ELSE
        b_set( _keybd_status, keybd_shift )
    ENDIF

    =( exit_value, 0b )

ELSEIF ==( exit_value, 0x58b ) ; shift lock
    IF b_test( _keybd_status, keybd_ignore )
        b_clear( _keybd_status, keybd_ignore )
    ELSE
        IF b_test( _keybd_status, keybd_capslock )
            b_clear( _keybd_status, keybd_capslock )
        ELSE
            b_set( _keybd_status, keybd_capslock )
        ENDIF
    ENDIF

    ; should turn on caps lock
    keybd_leds( _keybd_status )
    =( exit_value, 0b )
```

```
ELSEIF ==( exit_value, 0x77b ) ; num lock
    IF b_test( _keybd_status, keybd_ctrl )
        ; This is a Pause keypress
        IF b_test( _keybd_status, keybd_ignore )
            b_clear( _keybd_status, keybd_ignore )
            =( exit_value, 0b )
        ELSE
            =( exit_value, 0x9Eb )
        ENDIF
    ELSE
        IF b_test( _keybd_status, keybd_ignore )
            b_clear( _keybd_status, keybd_ignore )
        ELSE
            IF b_test( _keybd_status, keybd_numlock )
                b_clear( _keybd_status, keybd_numlock )
            ELSE
                b_set( _keybd_status, keybd_numlock )
            ENDIF
        ENDIF

        ; should turn on num lock
        keybd_leds( _keybd_status )
        =( exit_value, 0b )
    ENDIF

ELSEIF ==( exit_value, 0x7eb ) ; scroll lock
    IF b_test( _keybd_status, keybd_ignore )
        b_clear( _keybd_status, keybd_ignore )
    ELSE
        IF b_test( _keybd_status, keybd_scrolllock )
            b_clear( _keybd_status, keybd_scrolllock )
        ELSE
            b_set( _keybd_status, keybd_scrolllock )
        ENDIF
    ENDIF

    ; should turn on scroll lock
    keybd_leds( _keybd_status )
    =( exit_value, 0b )

ELSEIF ==( exit_value, 0x11b ) ; left alt
    IF b_test( _keybd_status, keybd_ignore )
        b_clear( _keybd_status, keybd_alt | keybd_ignore )
    ELSE
        b_set( _keybd_status, keybd_alt )
    ENDIF

    =( exit_value, 0b )

ELSEIF ==( exit_value, 0x14b ) ; left ctrl
    IF b_test( _keybd_status, keybd_ignore )
        b_clear( _keybd_status, keybd_ctrl | keybd_ignore )
    ELSE
        b_set( _keybd_status, keybd_ctrl )
    ENDIF

    =( exit_value, 0b )
```

```
ELSEIF <( exit_value, 0x7fb )
    IF b_test( _keybd_status, keybd_ignore )
        =( exit_value, 0b )
    ELSE
        ; Here is where we adjust for caps and num lock
        IF and( exit_value, <( exit_value, 0x68b ))
            ; This is a QWERTY key
            IF b_test( _keybd_status, keybd_shift )
                ; Key is Shifted
                =( exit_value, ee_read( ~
                ~ pcat_shift_scans[ to_word( exit_value )]))
                IF b_test( _keybd_status, keybd_capslock )
                    ; caps lock is on so lower characters
                    IF and( >( exit_value, 0x40b ), ~
                     ~ <( exit_value, 0x5Bb ))
                        =( exit_value, +( exit_value, 0x20b ))
                    ENDIF
                ENDIF
            ELSE
                ; Key is not shifted
                =( exit_value, ee_read( ~
                ~ pcat_low_scans[ to_word( exit_value )]))
                IF b_test( _keybd_status, keybd_capslock )
                    ; caps lock is on so upper characters
                    IF and( >( exit_value, 0x60b ), ~
                     ~ <(exit_value, 0x7Bb ))
                        =( exit_value, -( exit_value, 0x20b ))
                    ENDIF
                ENDIF
            ENDIF
```

```
                    IF b_test( _keybd_status, keybd_ctrl )
                        IF and( >( exit_value, 0x40b ), ~
                         ~ <( exit_value, 0x80b ))
                            =( exit_value, ~
                             ~ b_and( exit_value, 0y00011111b ))
                        ENDIF

                    ELSEIF b_test( _keybd_status, keybd_alt )
                        IF and( >( exit_value, 0x40b ), ~
                         ~ <( exit_value, 0x80b ))
                            =( exit_value, ~
                             ~ b_and( exit_value, 0y00011111b ))
                            =( exit_value, ~
                             ~ b_or( exit_value, 0y11000000b ))
                        ELSEIF >=( exit_value, 0x80b )
                            =( exit_value, ~
                             ~ b_and( exit_value, 0y00011111b ))
                            =( exit_value, ~
                             ~ b_or( exit_value, 0y11000000b ))
                        ENDIF
                    ENDIF
                ELSE
                    ; This is a numpad key
                    IF b_test( _keybd_status, keybd_shift )
                        ; Key is Shifted
                        IF b_test( _keybd_status, keybd_numlock )
                            =( exit_value, ee_read( ~
                             ~ pcat_shift_scans[ ~
                             ~ to_word( exit_value )]))
                        ELSE
                            =( exit_value, ee_read( ~
                             ~ pcat_low_scans[ to_word( exit_value )]))
                        ENDIF
                    ELSE
                        ; Key is not shifted
                        IF b_test( _keybd_status, keybd_numlock )
                            =( exit_value, ee_read( ~
                             ~ pcat_low_scans[ to_word( exit_value )]))
                        ELSE
                            =( exit_value, ee_read( ~
                             ~ pcat_shift_scans[ ~
                             ~  to_word( exit_value )]))
                        ENDIF
                    ENDIF
                ENDIF
            ENDIF

            b_clear( _keybd_status, keybd_ignore )

        ELSEIF ==( exit_value, 0xFFb )                ; no key data
            IF wait_flag
                =( exit_value, 0b )
            ENDIF
```

```
                ELSE
                    ; unknown keypress
                    b_clear( _keybd_status, keybd_ignore | keybd_extended )
                    =( exit_value, 0b )
                ENDIF
            ENDIF
        UNTIL exit_value
    ENDFUN
```

Once again, you need not concern yourself with the workings of the routine above. Just reference the keybd.lib file at the top of your program in a LIBRARY statement. The code above has many different execution paths, but executes very quickly. Simply calling this routine occasionally from some repetative loop in the body of your program should be sufficient to capture very fast typing without loosing key presses (keyboard buffer overrun).

The program below is typical of what your program will look like. Notice that the two I/O pins are defined and the "wait" time is defined before the keybd.lib file is referenced. The Wait time is actually just a count of the number of times the processor should sample the clock line looking for activity before assuming no data is available. This value is not precise and can vary between keyboard manufacturers. I have found that 1000 is always sufficient. Notice also that a variable called _keybd_status is referenced in the main function. This variable is defined and used within the keybd.lib file. It simply keeps track of shift keys and such.

```
; KEYBD.BAS - Program to convert keyboard input to console output
; Primary Source File

;12-10-2000 - Initial writing

; End Revision History

DEF tic63_i
LIB fbasic.lib

DEF pcat_clock pin_a2
DEF pcat_data pin_a3
DEF pcat_wait 1000w

INC keybd.inc
LIB keybd.lib

; The four standard interrupt handling functions follow
FUNC none irq    ; Executes when the int line goes low
BEGIN
ENDFUN

FUNC none global_int    ; Internal Global Interrupt
BEGIN
ENDFUN

FUNC none timer_int    ; Executes when timer interval expires
BEGIN
ENDFUN

FUNC none stack_overflow    ; Executes when stack overflows
BEGIN
    reset()    ; only sure way to recover
ENDFUN
```

```
FUNC none main    ; First Function to execute on startup
BEGIN
    =( _keybd_status, 0b )
    pin_low( pcat_clock )

    rs_conparam_set( debug_pin )
    REP
        con_out_char( keybd_input( True ))
    LOOP
ENDFUN
```

The sample program only has one line contained within the "big loop". this line outputs whatever character is received from the keyboard. A more likely situation with a real-life program would be to use the keybd_input(False) function which returns even if no key presses are received. Upon this basis, your program would either continue its primary operation, or interpret the keypress then resume the primary operation.