THE WEST RAND AMATEUR RADIO CLUB

October 2009
Volume 10, Issue 4

ANODE

Inside this issue:

Editor's Comments 1
Keyboard Input for PIC Projects 1

Special points of interest:

- Contact details on back page (corrected & updated)
- Ham-Comp Latest on web site.

Editor's Comments

Volume 10, Issue 4
October 2009

Keyboard Interfacing

This months main article is from Electronics and Wireless World. It shows one way of getting input to a microprocessor circuit. A problem I faced many years ago when you had to build/design your own keypad if you wanted numeric or function input from a human. Nowadays referred to an "HID" (Human Interface Device). The application is for a PIC but it just as easily be applied to another microprocessor. As keyboards are so cheap these days, this could easily solve a problem when you want to "talk" to a system or a radio or a teletype…

Ham-Comp meeting

The recent Ham-Comp meeting showed the members that visited the club house, the ways in which we can now get round the "barriers". When

(continued on page 10)

Keyboard Input for PIC Projects

Wiring a few links to a spare port of the microcontroller that is read on reset, or a few switches that are scanned when the PIC software is running may be sufficient. However, a low cost alternative is to use a standard PC keyboard. These keyboards cost only a few pounds and it is an input device that we are all familiar with. As a bonus there are three LEDs that can be controlled by the PIC program to show program status.

Within the article all data generated by the keyboard is given in hexadecimal 'NN'h form to distinguish between data and key characters such as function key Fl. The PIC keyboard software was written for the 16F877 microcontroller but should work with most PIC microcontrollers, however only the 16F87x and 16C74 family has the built in serial port used for testing.

AT keyboard

All current PCs are supplied with an AT style keyboard that have a PS/2 type connector. The keyboard was designed by IBM to be software configurable so that there is no need to manufacture different keyboards for different countries . Only the key tops need changing between countries not the keyboard circuit. This software flexibility allows keys to be added. For example, recent addition of the Euro currency key (tx), and some keyboards now include dedicated internet browser keys.

Keyboard internals

Internally these low cost AT keyboards consist of the keys sitting on a moulded clear rubber mat, this mat is

(continued on page 2)

Keyboard Input for PIC Projects

(continued from page 1)

placed on top of two plastic sheets with conductive circuit tracks printed on them. This conductive pattern is a 22 by 6 matrix where pressing down a key will make the connection between the two layers at a unique intersection. The keyboard controller continually scans this matrix and determines which key position has been pressed and sends this data to the PC.

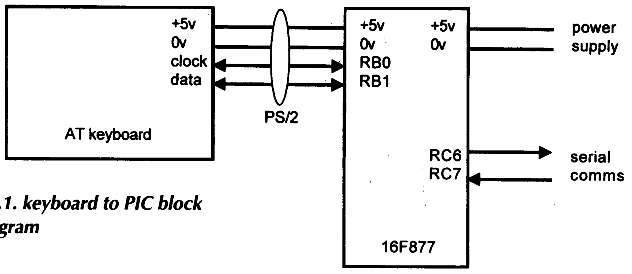


Fig.1. keyboard to PIC block diagram

The keyboard controller board is a small single-sided printed circuit board consisting of a surface mount controller (hidden under black protective coating), a few discrete components, 18 wire links and the three keyboard LEDs. Figure 2 shows the keyboard viewed from underneath, for clarity the two conductive sheets have been removed but they connect to the edge connector at the top of the printed circuit board.

Power supply

The keyboard will work off a 5-volt supply, so the same supply can power both the PIC circuit and keyboard. However the electrical characteristics sticker on the base of my 'Ever Green Touch' keyboard (manufactured in China) states that it requires 5V at 170 mA.

It is hard to imagine that a single customised controller chip requires all this power so I measured the current and found that it was only 8 mA, and with all three LEDs on the keyboard consumed a total of 20 mA. This is many times what the PIC microcontroller consumes, but if you are considering a battery powered application, then the current the keyboard requires will need to be taken into account.

Keyboard controller

The original keyboard design had a single chip microprocessor, but now a customised controller chip is used. This keyboard controller chip takes care of all keyboard matrix scanning, key de-bouncing and communications with the computer, and has an internal buffer if the keystroke data cannot be sent immediately. The PC motherboard decodes the data received from the keyboard via the PS/2 port using interrupt IRQ1.

The one thing that these keyboards do not generate is ASCII values. With a typical AT keyboard having more than 101 keys, a single byte could not store codes for all the individual keys, plus these keys along with shift, control, or alt, etc. Also for some functions there is no ASCII equivalent, for example 'page up', 'page down', 'insert', 'home', etc.

When the keyboard controller finds that a key is being pressed or released it will send this keystroke information, known as scan codes, to the PIC microcontroller. There are two different types of scan codes - make codes and break codes.

make code

A make code is sent whenever a key is pressed or held down. Each key, including 'shift', 'control' and 'alt', sends a specific code when pressed. Cursor control keys, 'delete', 'page up', 'page down', 'ins', 'home' and 'end', send extended make codes. The make code is preceded by 'E0'h to indicate an extended code. The only exception is the 'pause' key that starts with a unique 'E1'h byte.

break code

A break code is sent when a key is released. The break code is the make code preceded by

(Continued on page 3)

Keyboard Input for PIC Projects

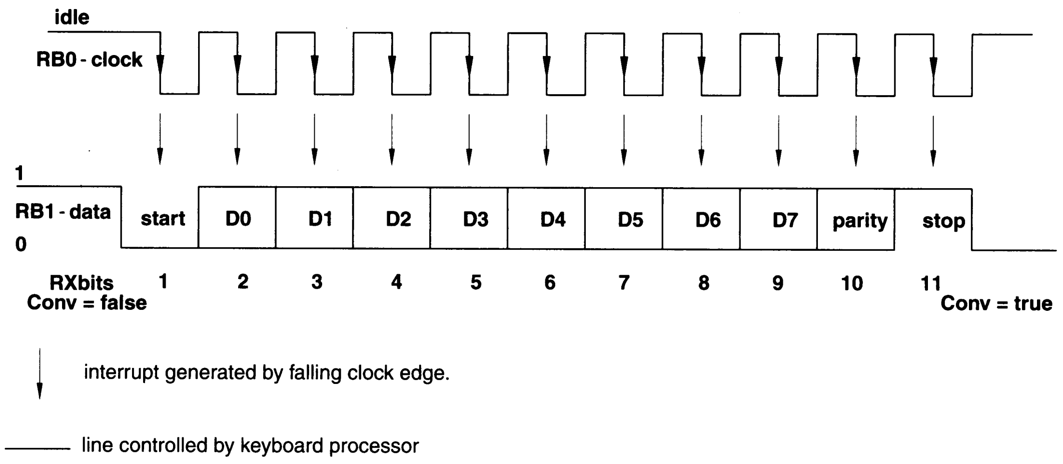


Fig. 7. Serial data sent from keyboard to PIC, data is read on the falling clock edge.

(Continued from page 2)

'F0'h byte. For extended keys the break code has an 'E0'h preceding the 'F0'h and make code value. The only exception is the 'pause' key as it does not have a break code and does not auto-repeat when held down.

key code

Every key is assigned its own unique code so that the host computer processing the information from the keyboard can determine exactly what happened to which key simply by looking at the scan codes received. There is no direct relationship between the scan code generated by a particular key and the character printed on the key top.

The set of make and break codes for each key comprises a scan code set. There are three standard scan code sets numbered 1, 2, and 3 - stored within the keyboard controller. Scan code set 1 is retained for compatibility for older IBM XT computers. Scan set 3 is very similar to the set 2 but the extended codes are different. Scan code set 2 is the default for all AT keyboards and all scan codes discussed here are from this set

scan code

If, for example, you press 'shift' and 'A' then both keys will generate their own scan codes, the 'A' scan code value is not changed if a shift or control key is also pressed. Pressing the letter 'A' generates '1C'h make code and when released the break code is 'F0'h, '1C'h.

Pressing 'shift' and 'A' keys will generate the following scan codes :

The make code for the 'shift' key is sent '12'h.
The make code for the 'A' key is sent '1C'h.
The break code for the 'A' key is sent 'F0'h, '1C'h
The break code for the 'shift' key is sent 'F0'h,'12'h.

If the right shift was pressed then the make code is '59'h and break code is 'F0'h, '59'h.

By analysing these scan codes the PC software can determine which key was pressed. By looking at the shift keystroke the software can distinguish between upper and lower case.

Keyboard commands

The main purpose of the keyboard is to accept typed data and send this information to the host computer, however there are several commands that can be sent to the keyboard controller. Figure 3 shows some of the more common keyboard commands. There are other commands that can be used to change make or

(continued on page 4)

Keyboard Input for PIC Projects

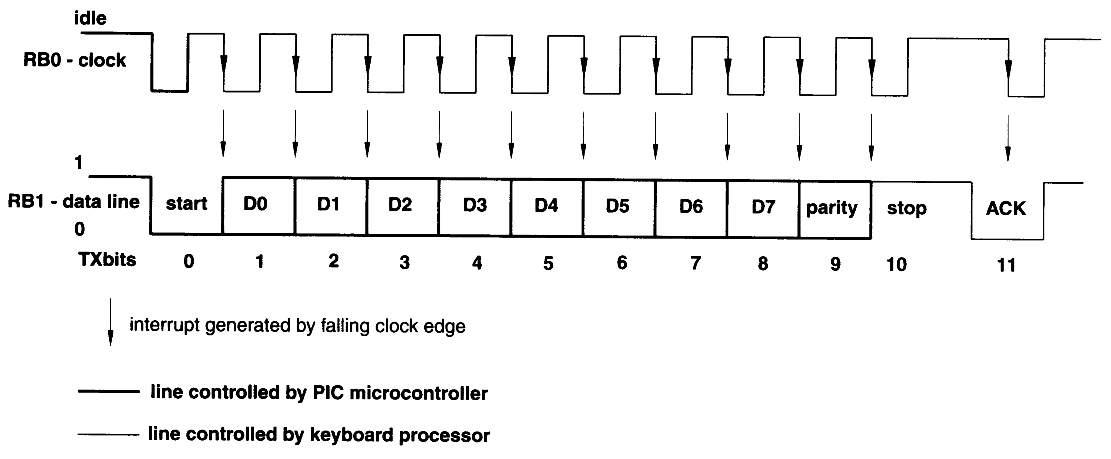

idle
RB0 - clock
1
RB1 - data line
0
start
D0
D1
D2
D3
D4
D5
D6
D7
parity
stop
ACK
TXbits
0
1
2
3
4
5
6
7
8
9
10
11
interrupt generated by falling clock edge
line controlled by PIC microcontroller
line controlled by keyboard processor

Fig. 8. Sending commands to the keyboard, data is set on the falling clock edge and read by the keyboard on the rising clock edge.

(Continued from page 3)
break codes for individual keys, but the commands given here are the most useful. The possible keyboard response to these keyboard commands is given in Fig. 4.

Keyboard self test

When the keyboard is first powered up it runs a self diagnostic test., this test primarily looks for keys that are 'stuck' down. All the LEDs on the keyboard will also briefly switch on and off as part of this self test. When the keyboard is plugged into a PC you may be forgiven for thinking that this was part of the PC start-up sequence as it happens around the same time as the PC is powering up and also running diagnostic tests.

After running the self-test the keyboard processor sends 'AA'h byte if everything is working correctly. If the keyboard processor finds a fault it will send 'FE'h byte. If the keyboard reports a fault then the PC BIOS will display 'Keyboard error or no keyboard present' followed by the less than useful message 'Press F1 to continue' (!).

'ED' keyboard LED command

The keyboard processor does not switch the Num Lock', 'Caps Lock', and 'Scroll Lock' LEDs whenever the appropriate key is pressed. Control of these LEDs is done by the host computer sending LED on/off commands to the keyboard processor. The keyboard LEDs and the corresponding keys are independent of each other.

To tell the keyboard which LED to turn on or off, send command 'ED'h and wait for the keyboard to respond with acknowledge byte ('FA'h). Then send the binary number '00000ABC' where the W bit is the state of the 'Caps Lock' LED, 'B' is the state of the Num Lock' LED, and 'C' is the state of the 'Scroll Lock' LED. Logic 'F is LED on, '0' for LED off. The keyboard will then respond (again) with 'FA'h indicating that it has successfully received the information.

The most significant five bits in the byte containing the LED information must be zero. If any of those bits is set then the keyboard processor will respond with 'FE'h (error) and wait for a properly formatted byte. There are no mechanisms for ‘asking' the keyboard controller the status of these LEDs, if you are using the LEDs and need to know which are on or off then the PIC program will need to store this information.

'EE' echo test

As the name suggests this command echoes

(Continued on page 5)

Keyboard Input for PIC Projects

(Continued from page 4)

back the command value. It can be used as a quick test to make sure that the keyboard is connected and working.

'F0' set scan code command

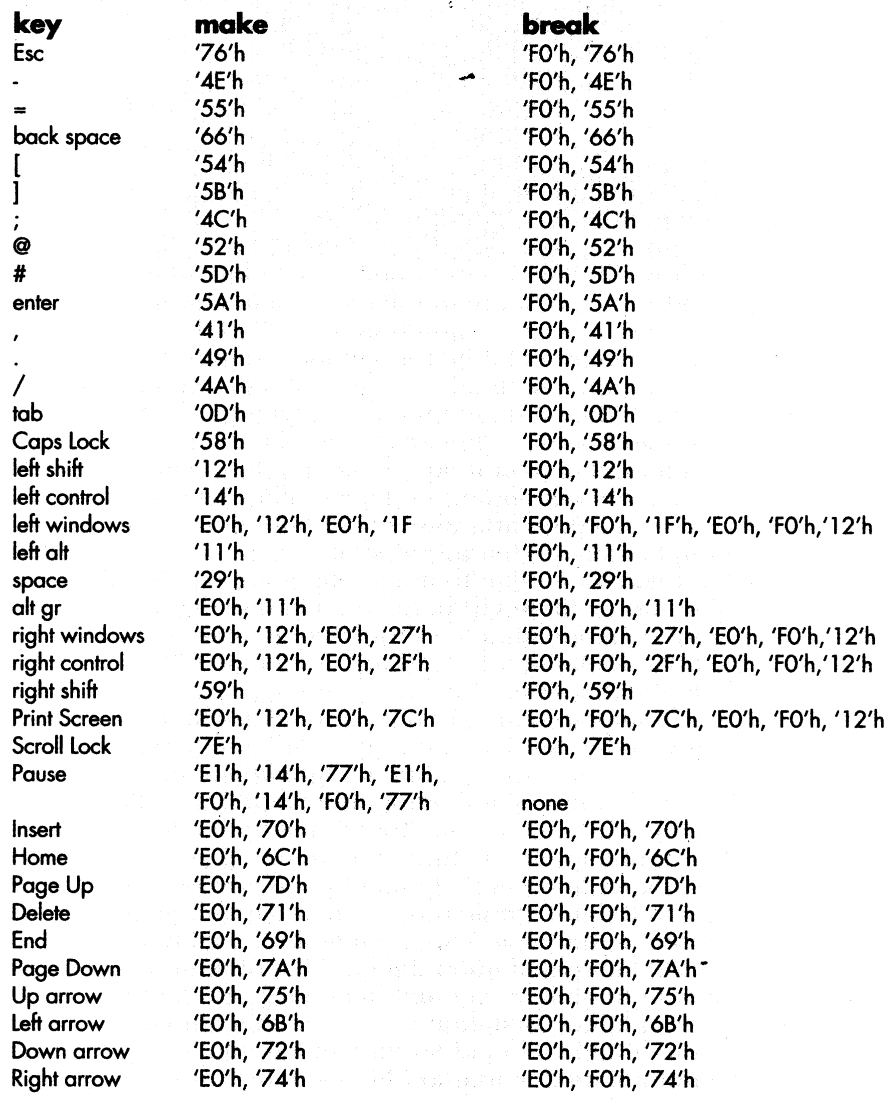
If you want to change to a different scan code set, send 'F0'h command byte to the keyboard. The keyboard processor will respond with 'FA'h (acknowledge). Then send '01'h, "02"h, or '03'h for scan code sets 1. 2, or 3.

When the new scan code is received the keyboard will again reply with 'FA'h.

To find out which scan code set is currently being used by the keyboard send '00'h instead of a new scan code set number. The keyboard will then respond with scan code number '01'h, '02'h (default) or '03'h.

All the scan codes presented here are those actually generated by the keyboard. When the keyboard is plugged into the PC the BIOS may translate some of these scan codes for compatibility reasons. Consequently a PC program may report slightly different scan codes for some keys.

Fig. 16. Keyboard scan codes.



key	make	break
Esc	'76'h	'F0'h, '76'h
-	'4E'h	'F0'h, '4E'h
=	'55'h	'F0'h, '55'h
back space	'66'h	'F0'h, '66'h
[	'54'h	'F0'h, '54'h
]	'5B'h	'F0'h, '5B'h
;	'4C'h	'F0'h, '4C'h
@	'52'h	'F0'h, '52'h
#	'5D'h	'F0'h, '5D'h
enter	'5A'h	'F0'h, '5A'h
,	'41'h	'F0'h, '41'h
.	'49'h	'F0'h, '49'h
/	'4A'h	'F0'h, '4A'h
tab	'0D'h	'F0'h, '0D'h
Caps Lock	'58'h	'F0'h, '58'h
left shift	'12'h	'F0'h, '12'h
left control	'14'h	'F0'h, '14'h
left windows	'E0'h, '12'h, 'E0'h, '1F	'E0'h, 'F0'h, '1F'h, 'E0'h, 'F0'h,'12'h
left alt	'11'h	'F0'h, '11'h
space	'29'h	'F0'h, '29'h
alt gr	'E0'h, '11'h	'E0'h, 'F0'h, '11'h
right windows	'E0'h, '12'h, 'E0'h, '27'h	'E0'h, 'F0'h, '27'h, 'E0'h, 'F0'h,'12'h
right control	'E0'h, '12'h, 'E0'h, '2F'h	'E0'h, 'F0'h, '2F'h, 'E0'h, 'F0'h,'12'h
right shift	'59'h	'F0'h, '59'h
Print Screen	'E0'h, '12'h, 'E0'h, '7C'h	'E0'h, 'F0'h, '7C'h, 'E0'h, 'F0'h, '12'h
Scroll Lock	'7E'h	'F0'h, '7E'h
Pause	'E1'h, '14'h, '77'h, 'E1'h, 'F0'h, '14'h, 'F0'h, '77'h	none
Insert	'E0'h, '70'h	'E0'h, 'F0'h, '70'h
Home	'E0'h, '6C'h	'E0'h, 'F0'h, '6C'h
Page Up	'E0'h, '7D'h	'E0'h, 'F0'h, '7D'h
Delete	'E0'h, '71'h	'E0'h, 'F0'h, '71'h
End	'E0'h, '69'h	'E0'h, 'F0'h, '69'h
Page Down	'E0'h, '7A'h	'E0'h, 'F0'h, '7A'h
Up arrow	'E0'h, '75'h	'E0'h, 'F0'h, '75'h
Left arrow	'E0'h, '6B'h	'E0'h, 'F0'h, '6B'h
Down arrow	'E0'h, '72'h	'E0'h, 'F0'h, '72'h
Right arrow	'E0'h, '74'h	'E0'h, 'F0'h, '74'h

'F2'h device identity command

The keyboard will respond to this command with 'FA'h (acknowledge) followed by the keyboard device type numbers 'AB'h,'83'h. When the keyboard is plugged into a PC the computer needs to know what type of device is connected to which PS/2 port. Other PS/2 devices can also be connected, such as a PS/2 mouse, which will respond with ID number '00'h,'00'h.

'FF'h keyboard test command

If the keyboard is wired to the same 5-volt supply as the PIC, then it is possible that the self test result will appear before the PIC microcontroller has initialised, particularly if the

(continued on page 6)

Keyboard Input for PIC Projects

(continued from page 5)

PIC power up timer is enabled. If the keyboard is already powered then sending command byte 'FF'h will force the keyboard to reset and run the self-test. This command is acknowledged by the keyboard ('FA'h) before the self test is executed. Alternatively use the 'F2'h command to get the keyboard device id number.

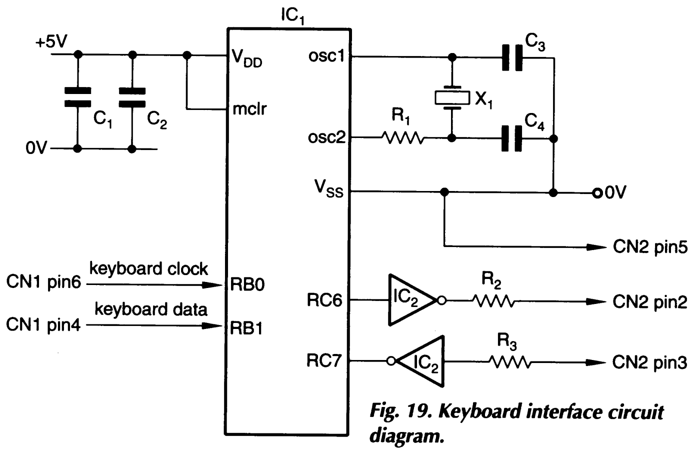


Fig. 19. Keyboard interface circuit diagram.

Typematic

When you press and hold down a key on the keyboard that key becomes typematic. This means the keyboard will keep sending that key's make code until the key is released. The typematic delay is a short delay between the sending of the first and second make scan code. Typematic rate is how many characters per second will appear after this initial typematic delay. The typematic delay can range from 0.25 second to 1 second and the typematic rate can range from 2 characters per second (cps) to 30 cps.

'F3'h set keyboard repeat rate

These typematic values can be changed using the 'F3'h command (set auto repeat rate), send 'F3'h and the keyboard will respond with 'FA'h byte, then the keyboard waits for the data byte that specifies the auto-repeat delay and rate.

With the exception of the 'pause' key, all keys will auto repeat. The default delay is 500 mS and the auto repeat default is 10 characters per second. It is unlikely that these default values will need to be changed, but there may be circumstances where longer delays are needed to allow the PIC to process information between key presses.

Keyboard serial data

The AT keyboard transmission protocol is a serial format, with one line providing the data and the other line providing the clock. The data length is 11 bits with one start bit (logic 0), 8 data bits (LSB first), odd parity bit and a stop bit (logic 1). The clock rate is approximately 10 to 30 kHz and varies from keyboard to keyboard.

The communications protocol is bi-directional, but as there are only two lines the handshaking between keyboard and PIC is more complicated. Unusually the keyboard generates the clock irrespective of the direction of data flow. The keyboard communications protocol is a strange mix with elements of both synchronous (separate data and clock) and asynchronous (start/stop bits) data transmission.

Both the keyboard clock and data lines are open collector outputs and require pull-up resistors to +5V. The PIC microcontroller has internal pull-up resistors on Port B which are enabled in the 'iniPIC' routine, if the keyboard is connected to another port then external pull-up resistors will be needed.

How the code works

The keyboard clock signal is connected to RB0 and used to generate an interrupt on the falling edge. The keyboard data line is connected to

(Continued on page 7)

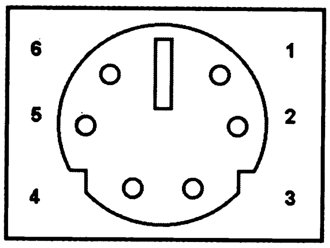
Keyboard Input for PIC Projects

(Continued from page 6)

PIC port RB1. Running the iniPIC routine initialises the various register options, sets the timer prescaler and initialises the variables. In program keybd.asm, the serial communication port is initialised.

Fig. 23. Wiring of the keyboard 6 pin mini-DIN PS/2 socket - viewed from the solder side.

pin 1	- no connection
pin 2	- +5v
pin 3	- no connection
pin 4	- PIC RB1 (data)
pin 5	- 0v
pin 6	- PIC RB0 (clock)



TimerOverflow is the T0IF flag of the 8-bit timer 0, this flag is set whenever the timer has counted up to 255 and starts counting again at 0. This flag is used to indicate a timeout and various counts are then automatically cleared. Without this, if the received data becomes corrupt and the RXbits count is wrong, then all following data will be decoded incorrectly. An alternative method if the timer is being used within the application program is to use the watchdog timer.

Variables TXbits and RXbits are counters indicating which bit in the serial keyboard data is being sent or received. The Conv flag is set whenever the data had been received from the keyboard. ReceiveDataFlag is the serial communication RCIF flag that is set whenever data is received from the PC via the serial port (keybd.asm only). This value is stored in variable TX and the ToKey routine is called.

Receiving data from keyboard

The keyboard will transmit data to the PIC microcontroller as soon as a key is pressed if both the clock and data lines are high, as this indicates idle status. If the clock line is held low by the PIC microcontroller then the keyboard cannot send and the keyboard controller will buffer the keystroke data.

Variable RXbits keeps track of which bit is being received, as RXbits is incremented on each interrupt. Variable keywork stores the bit pattern of the data received from the keyboard. This is achieved by setting the carry flag according to the logic status of the data at port RB1, then using the rotate right PIC instruction to shift the carry bit into the keywork variable. If RXbits = 10 this indicates the PIC is processing the parity bit, however this bit is ignored by the PIC program. On receiving RXbit = 11 (stop bit) the Conv flag is set indicating the end of data. Setting this flag causes the routine FromKey to be called from the main program loop.

FrornKey routine clears the Conv (convert) flag and sends the received keyboard data (contained in variable char) to the PmtHex (print hex) routine in the keydb.asm code.

This PrntHex routine converts the binary data into the ASCII suitable for display. Adding 48 to a binary decimal number converts that number to its ASCII text equivalent, if the number is greater than 9 then adding 55 will convert the hexadecimal number into an ASCII character.

The PrntHex routine then calls the SendPC routine. This routine waits for the TXIF flag to be set, this indicates that the serial communications TXREG (transmitter register) is empty. TXREG register is loaded with the char data and this data is automatically transmitted via the serial port to the PC. These routines are not required in keybd1.asm.

Sending data to the keyboard When the PIC microcontroller needs to send data to the keyboard, the routine ToKey is called. ToKey sets the clock line low for 60 milliseconds using timer 0. Bringing the clock line low prevents the keyboard from transmitting data. While the data line is held low the clock line is set to in-

(continued on page 8)

Keyboard Input for PIC Projects

(continued from page 7)

put and the keyboard will start generating a clock signal.

To make a port pin an output a '0' is sent to the TRISB (data direction register), a '1' sets that relevant port pin to an input. Data to be transmitted is output on the clock interrupt and read by the keyboard on the rising clock edge.

PIC software

Sending the scan codes to the PC is a useful demonstration (and functional test) of the keyboard to PIC connection. It allows specific keyboard scan codes to be verified but it is of very limited application.

(Continued on page 9)

Assembler listings

```
 ; keybd.asm
 ; PIC AT-keyboard reader
 ; Written by Roger Thomas
 ; MPASM 23 January 2002

  __config H'0F02'

 TMR0   EQU H'01' ; timer0
 STATUS EQU H'03' ; register
 C      EQU H'00' ; carry flag
 Z      EQU H'02' ; zero flag
 RP0    EQU H'05' ; page bit
 PORTB  EQU H'06' ; port B
 RB1    EQU H'01' ; keybd data
 INTCON EQU H'0B' ; register
 IRQ_RB0       EQU H'01' ;
interrupt
 T0IF   EQU H'02' ; timer0
 IRQ_EN EQU H'07' ; irq
 OPT_REG       EQU H'01' ;
register
 TRISB  EQU H'06' ; port B
 PIR1   EQU H'0C' ; peripheral
 RCIF   EQU H'05' ; serial comm
 RCSTA  EQU H'18' ; serial comm
 RCREG  EQU H'1A' ; serial comm
 TXSTA  EQU H'18' ; serial comm
 SPBRG  EQU H'19' ; serial comm
 TEMP   EQU H'20' ; irq handler
 IRQW   EQU H'2A' ; irq handler
 IRQS   EQU H'2B' ; irq handler
 IRQSTK EQU H'2C' ; irq handler
 CHAR   EQU H'2D' ; output
 RXBITS EQU H'2E' ; bit count
 TXBITS EQU H'2F' ; bit count
 KEYDATA       EQU H'30' ; keybd
calc
 KEYWORK       EQU H'31' ; keybd
calc
 TX     EQU H'32' ; transmit
 PARITY EQU H'33' ; keyboard
 FLAGS  EQU H'34'
 CONV   EQU H'00'

        ORG 0
        goto MAIN
```

```
        ORG 4     ; interrupt
        MOVWF     IRQW
        SWAPF     STATUS,W
        BCF       STATUS,RP0
        MOVWF     IRQS
        MOVF      TEMP,W
        MOVWF     IRQSTK
        CALL      IRQ
        MOVF      IRQSTK,W
        MOVWF     TEMP
        SWAPF     IRQS,W
        MOVWF     STATUS
        SWAPF     IRQW,F
        SWAPF     IRQW,W
        RETFIE

MAIN    CALL      INIPIC
LOOP    BTFSS     FLAGS,CONV
        GOTO      MAIN1
; if conv = true then
        CALL      FROMKEY
MAIN1   BTFSS     INTCON,T0IF
        GOTO      MAIN2
; if T0IF = true then
        MOVF      TXBITS,W
        SUBLW     D'00'
        BTFSS     STATUS,Z
        GOTO      MAIN2
; if TXBITS = 0 then
        CLRF      RXBITS
        CLRF      KEYDATA
        BCF       INTCON,T0IF

MAIN2   BTFSS     PIR1,RCIF
        GOTO      MAIN3
; if RCIF = true then
        MOVF      RCREG,W
        MOVWF     TX
        CALL      TOKEY
MAIN3   GOTO      LOOP

IRQ     MOVF      TXBITS,W
        BTFSC     STATUS,Z
        GOTO      IRQ6
; if TXBITS = true then
; begin
        MOVF      TXBITS,W
```

```
        SUBLW     D'09'
        BTFSS     STATUS,Z
        GOTO      IRQ1
        MOVF      PARITY,W
        MOVWF     TX
IRQ1    MOVF      TXBITS,W
        SUBLW     D'10'
        BTFSS     STATUS,Z
        GOTO      IRQ2
        MOVLW     D'255'
        BSF       STATUS,RP0
        MOVWF     TRISB
        BCF       STATUS,RP0
        GOTO      IRQ4
IRQ2    RRF       TX,F
        BTFSS     STATUS,C
        GOTO      IRQ3
        BSF       PORTB,RB1
        COMF      PARITY,F
        GOTO      IRQ4
IRQ3    BCF       PORTB,RB1
IRQ4    INCF      TXBITS,F
; end
        MOVF      TXBITS,W
        SUBLW     D'12'
        BTFSS     STATUS,Z
        GOTO      IRQ5
; if TXBITS = 12 then
        CLRF      TXBITS
IRQ5    GOTO      IRQEND
IRQ6    INCF      RXBITS,F
        MOVF      RXBITS,W
        SUBLW     D'11'
        BTFSS     STATUS,Z
        GOTO      IRQ7
; if RXbits = 11 then
; keydata = keywork
        MOVF      KEYWORK,W
        MOVWF     KEYDATA
        BSF       FLAGS,CONV
        CLRF      RXBITS
        CLRF      KEYWORK
; end
        GOTO      IRQEND
IRQ7    MOVF      RXBITS,W
        SUBLW     D'10'
        BTFSS     STATUS,Z
        GOTO      IRQ8
```

Keyboard Input for PIC Projects

(Continued from page 8)

The main function of this software is to use the keyboard as an input device to a PIC microcontroller. Rather than send the scan code to the PC, the scan value should be checked for various scan codes and appropriate data values modified within the PIC application program.

Assembler listing keybd1.asm shows a simple method of reading the keyboard scan codes and if specific keys are pressed, then the keyboard LEDs are turned on or off. The program looks for the letter 'A' (scan code '1C'h), when this is pressed all the LEDs are switched on (variable led determines which LEDs are switched on). When the letter 'B' is pressed (scan code '32'h) all the LEDs are switched off. All other key presses are ignored. These keyboard keys and which LEDs are activated can be changed, or values changed when specific keys are pressed.

Testing the interface

When the PIC is programmed with the keybd. asm code any make and break scan codes will be sent as ASCII characters to the PIC serial port. This requires the 74LS14 and two resistors to be fitted. A suitable three-wire serial cable to connect the PIC to the PC's serial port will need to be made.

The Windows Hilgraeve HyperTerminal (supplied with Windows) program can be used to view these keyboard generated scan codes as they are transmitted by the PIC software as text. The program properties should be set up as follows - direct to com, speed as 57600 baud, 8 bits, no parity, no flow control and one stop bit.

Figure 13 is a HyperTerminal screen showing the self test passed byte followed by the scan codes for letters A (make code = '1C'h, break code = 'F0'h, '1C'h), B (make code = '32'h', break code = 'F0'h, '32'h), C (make code = '21'h, break code = 'F0'h, '21'h).

Followed by the extended scan codes generated when pressing the insert key (make code = 'E0'h, '70'h, break code = 'E0'h, 'F0'h, '70'h) and eight byte extended code when the pause key was pressed (make code = 'E1'h, '14'h, '77'h, 'E1'h, ' F0'h, '14'h, 'F0'h, '77'h, no break code).

Figure 14 shows an interactive Windows program displaying the keyboard response to various commands sent to the keyboard from the PC via the serial communications port. The four buttons (reset, keyboard id, echo, and scan code) when pressed will send that particular command to the keyboard and the keyboard's responses can be seen. The three LEDs can be switched on or off and when the button marked 'LED' is pressed this command is sent to the keyboard and the appropriate LEDs should be lit on the keyboard.

The 'AA'h is the result of the keyboard self test, 'FA'h is the command acknowledgement for the device identity request. The keyboard responds with device type 'AB'h and '83'h'. The two 'FA'h bytes are acknowledgement of the scan code query command and keyboard processor responds with scan set 2. The final two 'FA'h are for the LED command acknowledge. The program will also show any make or break codes if any keys are pressed on the keyboard.

This Windows program (two versions are available, one for Windows 95/98/ME and the other for Windows XP) and the two PIC assembler source code programs (keybd.asm and keybd1.asm) will be available from EW -just email j.lowe@cumulusmedia.co.uk stating which one you'd like.

Construction

The PIC circuit can be built using strip board, the 20 MHz crystal can should be connected to

(continued on page 10)

Keyboard Input for PIC Projects

(continued from page 9)

0 Volts for correct operation. The two inverters and series current limiting resistor are for the optional PC serial communications. They are not necessary for the keyboard connection.

The PIC expects to interface to a serial line driver which in operation would invert the data, as a serial driver IC is not used then the data has to be inverted.

Care is needed when wiring the PS/2 socket - particularly for the power connection. Remember to observe the keyboard self test when the keyboard is plugged into the socket. All the LEDs should briefly flash if the wiring is correct. If not then disconnect the power supply and check the wiring.

Acknowledgements

My thanks to Andrew Thomas for help with the PIC programming. PIC is a registered trademark of Microchip Technology Incorporated, USA. Windows is a registered trademark of Microsoft Corporation.

From Wireless World

Editor's Comments

someone says "it's impossible"! Reply "Not for a Ham"!

When someone says you cannot make an old program work under Windows XP/Vista/7, tell them I showed you how.

Electronic Recycling

Elektor magazine has a Summer and Winter Issue that has a large number of simple electronic projects. All of them are tried and tested by the editorial staff. This means that most can be constructed and will work on switch on.

Unfortunately many of the kits offered over the Internet for money, do not work for a variety of reasons. Building kits without either the basic knowledge or experience is demoralising. Lots of things can go wrong. Poor assembly documentation, circuit explanation and test point voltages and waveforms make it impossible to fault-find.

I recommend locally built and designed kits for the simple reason that the designer can be readily talked to by the constructor. Instead of the other side of the planet and only communicates with email.

One of the most avidly read articles or features of any electronic magazine has always been the "Circuit Ideas". This was scanned for useful ideas and the merits discussed at length with my colleagues. The photocopier was the only way to "capture" this information then. Nowadays you can search the Internet and find a multitude of sites offering "Circuit Ideas". Unfortunately the majority are badly written or not explained using basic principles.

In our next issue I am going to fill it with "Circuit Ideas".

JB 2009-10-11

The West Rand Amateur Radio Club
Established in 1948
KG33XU 26.14122 South - 27.91870 East

P.O. Box 562
Roodepoort
1725

Phone: 082 342 3280 (Chairman)
Email: zs6wr.club@gmail.com
Web page: www.jbcs.co.za/ham_radio

Bulletins (Sundays at …)
11h15 Start of call in of stations
11h30 Main bulletin start

Frequencies
439.000MHz 7.6MHz split
Input: 431.4MHz (West Rand Repeater)
145,625 MHz (West Rand Repeater)
10,135 MHz (HF Relay)

Radio Amateurs do it with more frequency!

Chairman	Joop Hesp	ZS6C	082 342 3280	**zs6wr.club@gmail.com OR joophesp@telkomsa.net**
Vice Chairman	Geoff	ZS6GRL	082 546 5546	glevey@gmail.com
Secretary	Phillip	ZS6PVT	083 267 3835	phillipvt@sse.co.za
Treasurer	Craig Woods	ZS6CRW	082 700 0163	craig.woods@absamail.co.za
Member	Romeo Nardini	ZS6ARQ	082 552 4440	roshelec@global.co.za
Member (Anode)	John Brock	'PieRat'	011 768 1626	brockjk@gmail.com
Member (Technical)	Ron	ZR6RON	082 902 8343	ronnie@calidus.co.za
SARL Liaison	Willem	ZS6WWJ	082 890 6775	marie.w@absamail.co.za

West Rand members - we need your input!

To make this the best ham radio magazine in South Africa we need your input. Please submit articles, comments, suggestions etc.

Please send plain text with no formatting to the email address below.

In July 2003, we re-published an Anode Compendium on CD. It has the issues from July 2000 until June 2005. This included the new Adobe reader. It has been updated, check with the chairman for details.

We need your input! Email us articles, comments and suggestions please. zs6wr.club@gmail.com