

A CW Keyboard

You've probably heard of ham radio operators. Hams still use Morse code (among other modes). Yes, some hams still use old-fashioned keys and enjoy that. Some hams use keyboards that send Morse code (CW in ham parlance). In this application note, I'll marry a standard PS/2 keyboard to a Basic Stamp and make it send Morse code. In addition to the code, the program will have several predefined messages and store your callsign and a custom message in the Stamp's EEPROM.

This is all possible because of AWC's PAK-VI chip. This chip converts a standard keyboard to output RS232 suitable for use with a Stamp.

About the PAK-VI

By default, the PAK-VI converts the keyboard's code to ASCII (keyboards use a strange code that doesn't relate to ASCII at all). This is what you usually want. However, if you want to, you can switch the PAK-VI to raw mode and take control yourself. This is useful if you are using the keys for something other than ASCII characters, or you want to read a PS/2 mouse (good for reading an XY position).

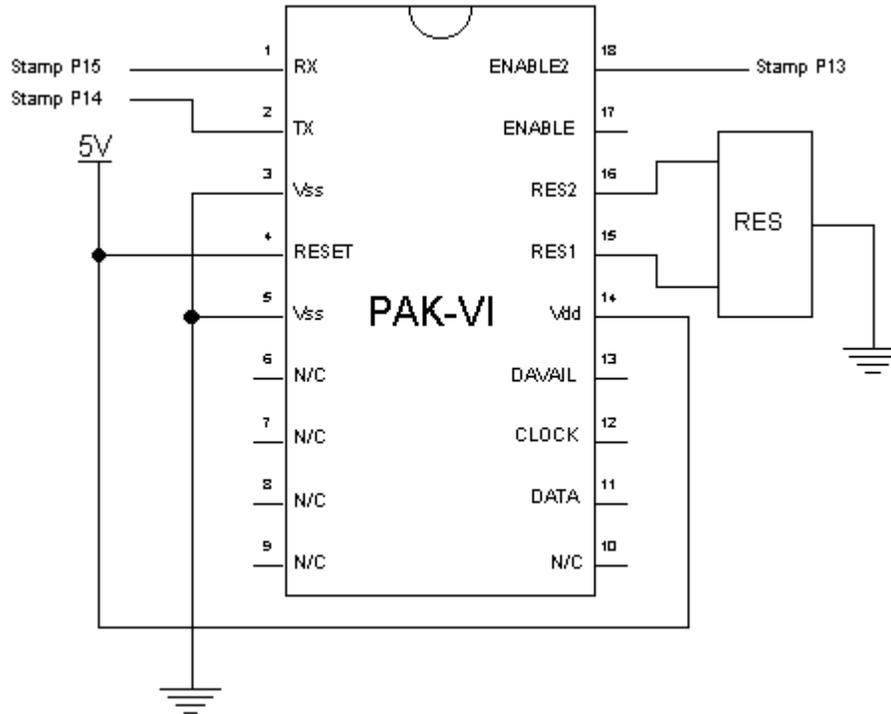
The PAK-VI works with Basic Stamp flow control so you don't have to worry about missing characters. The PAK-VI buffers 16 characters (or scan codes in raw mode) and the keyboard itself can usually buffer 16 more scan codes.

If you are not in raw mode, operation is very simple. The PAK even controls the status LEDs (shift lock, caps lock, and scroll lock) for you. Extended keys (like the function keys) emit one byte which makes it very easy to process them using the Stamp.

For this application, the Stamp doesn't send many commands to the PAK (just a software reset command). However, there are commands to switch modes, set scroll, num, and caps lock, and send commands directly to the keyboard.

You can use the PAK-VI to communicate with a keyboard or a mouse. Wireless keyboards might be of interest to those building robots. You can read more about the PAK-VI [here](#) or by downloading its datasheet.

The Circuit



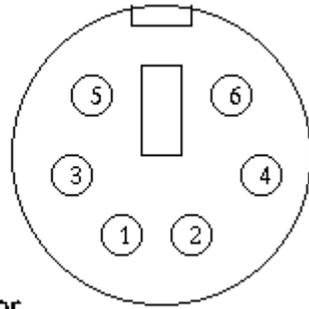
Note 1: The RES is a 50Mhz ceramic resonator, included with the PAK-VI.

Note 2: Stamp pin 0 drives the transmitter's PTT (optional), Stamp pin 1 is the key input to the transmitter, and Stamp pin 6 is the sidetone (connect to a piezo speaker). Just to demonstrate the circuit you only need a speaker on Stamp pin 6 (you could also drive LEDs with pins 0 and 1).

Under the Hood

The circuit is pretty simple. The PAK sends ASCII codes to the Stamp. It would be possible to not connect pin 15 of the Stamp since only a reset command is sent to the PAK. You can easily connect the PAK to a keyboard using a PS/2 motherboard socket. You can find these in computer stores for about \$5. They allow you to connect a PS/2 keyboard to an ordinary motherboard that has pins for the keyboard. Just cut the header off and make your connections.

PS/2



Looking into
male connector

- 1 data
- 2 reserved
- 3 GND
- 4 +5V
- 5 clock
- 6 reserved

The code operates in a loop. It reads each character and examines it to see if it is a special command:

Command	Key	Meaning
\$CA	Shift+F11	Set callsign
\$CB	Shift+F12	Set custom macro
\$91	Up arrow	Increase speed
\$97	Down arrow	Decrease speed
\$9A	Del	Switch to receive mode

If the code finds one of these codes, it jumps to the correct routine. Otherwise, it processes the character using the charproc routine.

To handle a normal character, the code looks up two bytes from EEPROM. One is the number of elements (dits and dahs) in the character. The other is the actual dit/dah pattern. If the number of elements is \$FF, the item is a space.

Macro handling is interesting. There is a table that contains a pointer to each macro. The callsign macro is exactly long enough to hold a long U.S. callsign (e.g., WD5GNR/2). The custom macro appears at the end so it can grow as needed. The program limits the length of the custom macro to 50 characters. Press enter when you are done setting either the callsign or custom macro.

In the preprogrammed macros, you can use a \$FF character to signal the program to send the callsign.

The Code

```
' CW Keyboard using BS2 and PAK6

pakout con 15 ' pak communications
pakin con 14
baud con 84 ' baud rate for pak
```

```
fpin con 13      ' pak handshake
ptt con 0       ' turns transmitter on/off (no QSK)
key con 1       ' Keys transmitter
spkr con 6      ' Makes beep for sidetone
```

```
freq con 1000   ' 1000 Hz sidetone
```

```
dot var word
dash var word
notx var bit
keyin var byte
taddr var word
maddr var word
len var byte
didah var byte
```

```
notx=0
dot=100
dash=300
dotmin con 10
dotmax con 1000
```

```
high fpin
low ptt
low key
```

```
serout pakout,baud,[$FF]  ' reset
pause 500
top:
' read character
serin pakin\fpin,baud,[keyin]
```

```
debug hex2 keyin
' check for commands (Del=Rx,up=speed+,dn=speed-)
if keyin=$CA then setcall
if keyin=$CB then setmac
if keyin=$91 then speedup
if keyin=$97 then speeddn
if keyin=$9A then rx
```

```
high ptt ' ptt on (may have already been on)
gosub charproc ' process character
goto top
```

```
' switch to receive mode
rx:
low ptt
goto top
```

```
' use arrow key up to increase speed
speedup:
dot=dot-1
goto speed
speeddn:      ' arrow down decreases speed
dot=dot+1
speed:
```

```

if dot>=dotmin then minok
dot=dotmin
minok:
if dot<=dotmax then maxok
dot=dotmax
maxok:
dash=dot*3
debug ?dot
goto top

' do an ordinary character, space, or macro
charproc:
if keyin<"a" or keyin>"z" then nouc
keyin = keyin & $5F ' force upper case
nouc:
taddr=keyin+ltbl
Read taddr,len
if len=0 then ret
if len=$FF then cspace
if len=$FE then wspace
if len>=$80 then macro
' ordinary char
taddr=keyin+ctbl
Read taddr,didah ' read pattern
cloop:
gosub element
didah=didah<<1
len=len-1
if len<>0 then cloop
ospace:
pause dash' element space
return
' word space
ospace:
pause dash*2
return
' do a particular element
element:
if notx then nokey0
high key
nokey0:
if (didah & $80)=$80 then dah
freqout spkr,dot,freq
debug "."
goto elspace
dah:
freqout spkr,dash,freq
debug "--"
ospace:
if notx then nokey1
low key
nokey1:
pause dot
ret:
return
' send a macro
macro:

```

```

maddr=2*(keyin-$80)+macrotbl
read maddr,taddr
read maddr+1,maddr
maddr=maddr*256+taddr
mloop:
read maddr,keyin
debug "Macro: ",keyin,cr
if keyin=0 then ret
if keyin<>$FF then norm
gosub sendcs
goto nextm
norm:
gosub charproc
nextm:
maddr=maddr+1
goto mloop

' callsign macro
csmacro:
maddr=csmac
goto mloop

csaddr var word
' Send the callsign macro
sendcs:
csaddr = csmac
scsloop:
read csaddr,keyin
if keyin=0 then ret
gosub charproc
csaddr=csaddr+1
goto scsloop

' Set callsign
setcall:
notx=1
csaddr = csmac
scl:
if csaddr=csmac+8 then scfin
serin pakin\fpin,baud,[keyin]
if keyin<>8 then nobx
if csaddr=csmac then scl
csaddr=csaddr-1
goto scl
nobx:
if keyin<>13 then nocecho
scfin:
write csaddr,0
gosub sendcs
notx=0
goto top
nocecho:
write csaddr,keyin
csaddr=csaddr+1
gosub charproc
goto scl

```



```
data $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
data $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
data $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
data $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
data $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
```

```
' This table has a pointer to each macro
macrotbl data macro0&$FF,macro0>>8,macro1&$FF,macro1>>8
data macro2&$FF,macro2>>8,macro3&$FF,macro3>>8
data macro4&$FF,macro4>>8,macro5&$FF,macro5>>8
data macro6&$FF,macro6>>8,macro7&$FF,macro7>>8
data macro8&$FF,macro8>>8,macro9&$FF,macro9>>8
data csmac&$FF,csmac>>8,umacro&$FF,umacro>>8
```

```
' These are the macros, last one is user defined
csmac data "NOCALL/0",0 ' callsign is replaced at runtime
macro0 data "CQ CQ CQ DE ", $FF, " ", $FF, " ", $FF,0
macro1 data "DE ", $FF,0
macro2 data "QRZ? DE ", $FF,0
macro3 data "TNX UR 599",0
macro4 data "73 AND TNX DE ", $FF,0
macro5 data 0
macro6 data 0
macro7 data 0
macro8 data 0
macro9 data 0
umacro data 0
```