BS2 Mouse

It is no secret that our PAK-VI lets you connect a PS/2 keyboard to a Basic Stamp. The example code you can find in our document library uses the PAK-VI to generate Morse code using a standard keyboard. I've seen other people use the PAK-VI and a wireless keyboard to command a robot. I've also heard of people interfacing "keyboard wedge" scanners using the PAK-VI.

When I first designed the PAK-VI, I tested using the chip with a PS/2 mouse. It worked, but I didn't really develop the idea and although the literature says it can be done, I've never really provided an example. Well, this month marks the end of that! I'm going to show you how to interface a mouse to a BS2 or any micro using the PAK-VI.

Why would you want a mouse on a Basic Stamp? A track pad or a track ball can allow you adjust analog values in a very intuitive way. In addition, you can gut a mouse and get a very nice X/Y position system. I've always thought this would be a cheap way to "close the loop" on a homemade CNC mill, for example. With some mechanical linkage, you could even use a mouse to track the movement of a robot.

Inside the Mouse

The normal PS/2 mouse sends three byte packets. This is unlike the keyboard which sends scan codes. When you reset the mouse, it sets its output off. If you want to listen to the mouse, you have to enable it. Since none of this relates to ASCII characters, you'll want to set the PAK-VI to raw mode before you do anything.

The mouse has two modes you can set. Stream mode just spits position packets out as the mouse moves. The PC uses this mode, but for the Stamp it isn't very handy. Luckily, the mouse also supports a remote mode. In this mode, the mouse accumulates position information. When you ask for the information, the mouse sends a three byte packet and resets the position information. That way you don't have to constantly monitor the mouse for input. However, the mouse can only track differences of +/- 255 counts, so you have to read the mouse often enough to prevent overflow (there is an error indicator if you wait too long).

The first byte of each packet contains several status bits, defined in my program like this:

```
yover var p1.bit7 ' y overflow

xover var p1.bit6 ' x overflow

ysign var p1.bit5 ' y sign
```

```
xsign var p1.bit4 ' x sign

midbtn var p1.bit2 ' button flags

rightbtn var p1.bit1

leftbtn var p1.bit0
```

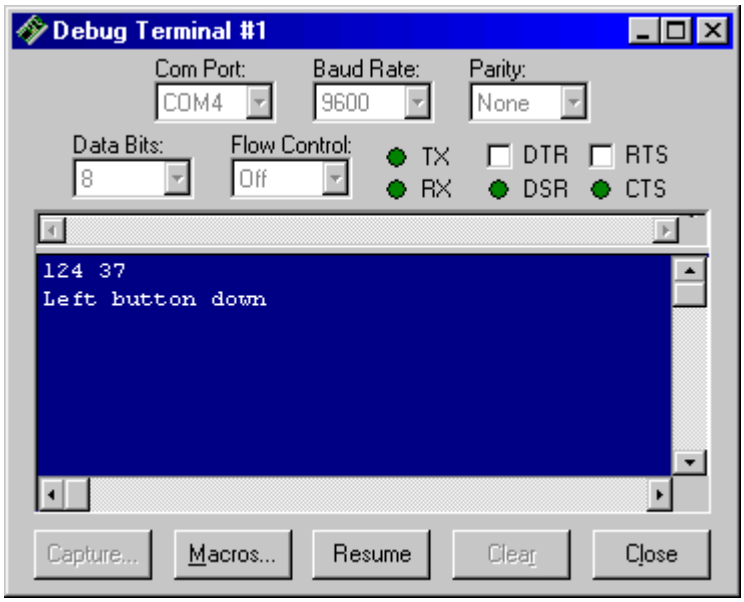
As you might expect, the *xover* and *yover* variables tell you if you've had an overflow. The flags that end with *btn* indicate when a particular button is pressed.

The *xsign* and *ysign* flags are 1 when the corresponding value is negative (meaning the mouse direction is "backwards" from the positive direction. However, when a quantity is negative, the mouse sends the data (the 2nd byte is the x data and the 3rd byte is the y data) in 2's compliment format. So you either have to sign extend the data or convert it to a magnitude before you use it. My code converts it to a magnitude.

The connections required are Stamp pin 15 to the PAK's output, pin 14 to the PAK's input, and pin 13 to Enable2.

Testing

I hooked an Alps Glidepoint trackpad to a PAK-VI and ran the program below. Here's the screen shot:



The program shows the accumulated X and Y positions, plus the button status. As you read through the program here are a few fast facts to keep in mind:

1. Sending $02 to the PAK-VI puts it in raw mode

1. Sending $0B to the PAK-VI causes it to send the next byte directly to the mouse
1. $F0 sets remote mode
1. $EB requests a motion packet
1. $FA indicates a successful reply from the mouse
1. The expression `((xreg-1)^$FF)` computes the magnitude of a negative number in xreg

Other commands that I did not use include:

1. 0xF3 - Set the sample rate. The mouse responds with "acknowledge" (0xFA) then reads one more byte. After receiving the sample rate, the mouse responds with 0xFA and resets the movement counters. Valid sample rates are 10, 20, 40, 60, 80, 100, and 200 samples/sec.
1. 0xF2 - Get device ID. The mouse responds with "acknowledge" (0xFA) followed by 0 for a standard PS/2 mouse. The mouse also resets its movement counters.
1. 0xE8 - Set resolution. The mouse sends back an 0xFA and then waits for another byte. A 0 byte sets 1 count/mm, a 1 sets 2 counts/mm, and 2 sets 4 counts/mm, and 3 sets 8 counts/mm.

The Program

```
'{$STAMP BS2}
'  Mouse code for BS2 and PAK-VI

pakout con 15
pakin con 14
baud con 84
fpin con 13

keyin var byte
' This assumes a 3 byte packet
' Some mice can do 4 byte packets
' (mainly those with wheels) but
' you have to turn on that mode
p1 var byte   ' mouse status
xreg var byte ' raw x motion
xmov var word ' x accumulator
yreg var byte ' raw y motion
ymov var word ' y accumulator

' flags in p1
yover var p1.bit7   ' y overflow
```

```
xover var p1.bit6   ' x overflow
ysign var p1.bit5   ' y sign
xsign var p1.bit4   ' x sign
midbtn var p1.bit2  ' button flags
rightbtn var p1.bit1
leftbtn var p1.bit0

high fpin

' Reset PAK and wait for device reset
serout pakout,baud,[$FF]
pause 500
xmov=0
ymov=0
' Enter raw mode
serout pakout,baud,[2]
' Set remote mode
serout pakout,baud,[$0B,$F0]
serin pakin\fpin,baud,[keyin]
' should get $FA
if keyin=$FA then top
Debug "Didn't get $FA from remote mode set",cr

top:
' read movement packet
serout pakout,baud,[$0B,$EB]
serin pakin\fpin,baud,[keyin,p1,xreg,yreg]
if keyin=$FA then parse
debug "Didn't get $FA on mouse poll",cr
goto top

parse:
if yover or xover then overflow
if xsign=0 then xplus
' convert negative number
xmov = xmov - ((xreg-1)^$FF)
goto doy
xplus:
xmov=xmov+xreg
doy:
if ysign=0 then yplus
' convert negative number
ymov=ymov- ((yreg-1)^$FF)
goto ydone
yplus:
ymov=ymov+yreg
ydone:

' Display results
debug cls, sdec xmov, " ", sdec ymov ,cr
if leftbtn=0 then checkmid
Debug "Left button down",cr
checkmid:
if midbtn=0 then checkr
Debug "Mid button down",cr
checkr:
if rightbtn=0 then top
Debug "Right button down",cr
goto top
```

```
overflow:
Debug "Overflow occured",cr
goto top
```