Column #101 September 2003 by Jon Williams:

PlayStation Control Redux

I often say to friends, and in fact have stated right here in my column, that I feel like I'm one of the luckiest guys in the whole world. Now, don't worry, I won't bore you with the myriad reasons I could use to back up that statement, but I will say that on of them is the job I get to do and all the neat people I get to meet in the course of doing that job.

A small part of my job with Parallax involves training. Back in June I had the very good fortune to be invited to teach BASIC Stamps at IBM's EXITE (Exploring Interests in Technology and Engineering) camp for girls here in the Dallas/Fort Worth area. It was fun, if not a very tough assignment. Not the course material, mind you, I'm pretty good with BASIC Stamps. The audience ... a whole different story: twenty-odd thirteen-year-old girls who didn't have a background in electronics or computer programming.

Perhaps it's a sign that at 41 I am finally getting old, but teenagers today seem significantly more sophisticated and cynical than when I was a kid. I knew that if I was going to last all four days that I would have to start strong. So, on day one (Monday), I retrieved a Sony PlayStation® game controller from my backpack and held it in the air for the girls to see. "Does anyone know what this is?" I asked. Of course, every single one of them knew what it was and responded accordingly. Then, I planted the seed that I hoped would get and maintain their attention. "No, it isn't. This, my new friends, is a robot controller and I will prove it to you on Thursday." It seemed to work. With the minor exception of a couple of "bad attitude"

girls, the rest seemed genuinely interested in how they could control a robot with a device that they all knew very well.

Thank You, Aaron

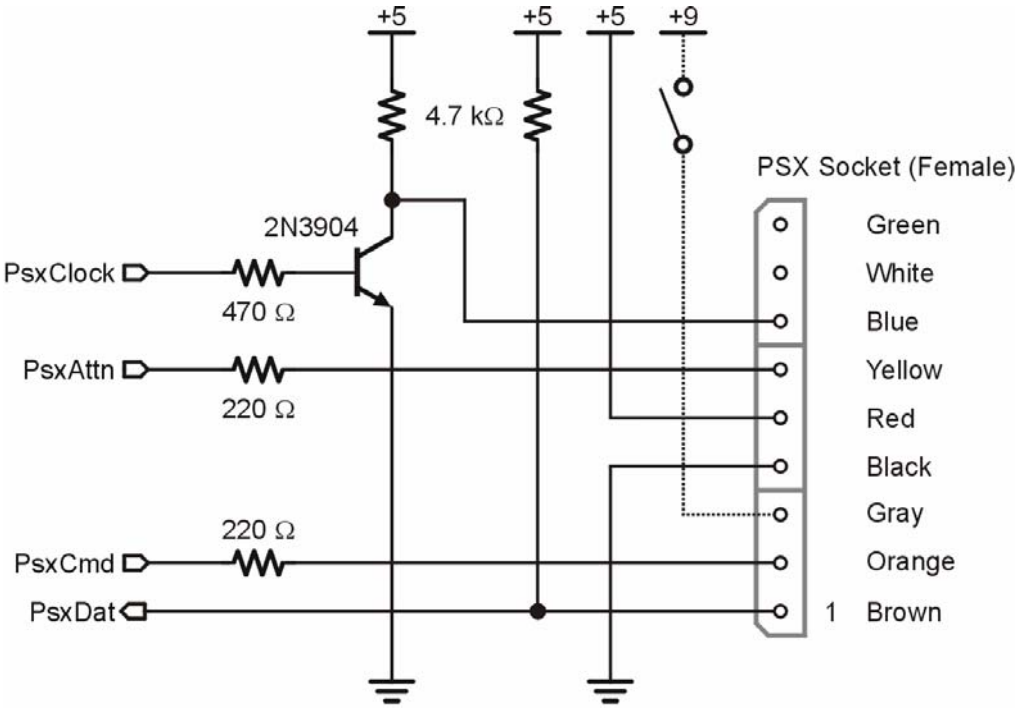
Before I go any further let me admit that I got started with the PlayStation controller because of Aaron Dahlen's neat article published in Nuts & Volts back and June. About a year ago, I had some interest in the controller, but never got around to doing anything with it. So my thanks go to Aaron for his work and getting me off my duff. My hope is that I can expand on Aaron's information so you can get more use from the controller.

Building An Interface

The toughest part about working with the PlayStation controller is building the mechanic/electrical interface, and most of that is very simple; it's the connector that is the tough part. The socket that accepts the odd 9-pin controller plug is a Sony product, and not something you can purchase readily. I did find a couple of places on the Internet that sell repair parts for games, but these were not new parts – they had been removed from damaged PlayStations, and none were cheap.

I solved this problem by purchasing a $10 PlayStation extension cable and carefully hacking the socket from one end of it. The socket is attached to a PCB with professional strength contact cement and a bit of hot glue (be careful!) is used to secure the wires. Figure 101.1 shows the schematic for my version of Aaron's interface. This circuit varies from Aaron's only in [trivial] component values and the use of 220-ohm resistors on any BASIC Stamp pin that is used as an output – I do this for safety in case the controller fails. Also note that the 9V connection to pin 3 of the controller is optional (dual-shock motor power) and not required for our experiments.

Figure 101.1: PlayStation BASIC Stamp Interface



Shifting Bits – Automatic and Manual Modes

Part of the reason I decided to write about the PlayStation controller after Aaron's excellent article has to do with an oddity in the values returned when using `SHIFTIN` (more on that in a minute) and I wanted to create code that would tell the BASIC Stamp what kind of controller is attached. Controller type detection is not possible using the `SHIFTOUT/SHIFTIN` method since the PlayStation controller sends its device type to the game console (or our Stamp) when the data request byte (`$42`) is being sent to the controller. Figure 101.2 shows how data is exchanged between the master and the controller, and how the overlap occurs on the second byte of the transfer.

Figure 101.2: Data Exchange Between Master and Controller Showing Overlap on Second Byte of Transfer

Knowing the type (or current mode) and if the device is ready can be very useful, so how can we retrieve this data? We do it by creating a manual function to shift data to and from the controller simultaneously. While this may sound a bit complex, it really isn't. Back in the BS1 days we had to create our own shift functions for synchronous devices; we'll just build on those strategies.

Here's a bit of code we can use to send a byte to and receive a byte from the controller.

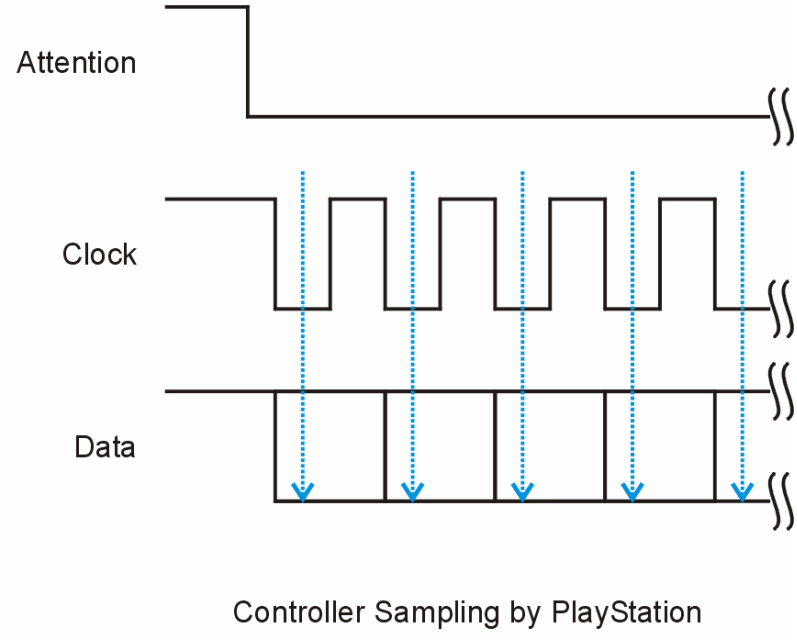
```
PSX_TxRx:
  FOR idx = 0 TO 7
    PsxCmd = psxOut.LOWBIT(idx)
    PsxClk = ClockMode
    psxIn.LOWBIT(idx) = PsxDat
    PsxClk = ~ClockMode
  NEXT
  RETURN
```

As you can see, the code is quite straightforward. A `FOR-NEXT` loop accommodates eight bits. A bit from the command byte is put onto the `PsxCmd` pin before the controller clock line is pulled low. While the clock line is low, the `PsxDat` line can be scanned and the bit stored in psxIn. The clock line is returned high and the process is repeated for all bits. Notice that this routine works LSB to MSB.

Before we continue with the code, let me address an issue that occurs when using `SHIFTIN` to retrieve all six data bytes from the PlayStation controller. If you've run Aaron's demo program you have probably noticed that the left joystick Y-axis (up-down) does not span the entire range; its range is 128 to 255. This seemed odd to me and after a bit of investigation, I believe I know why this is happening.

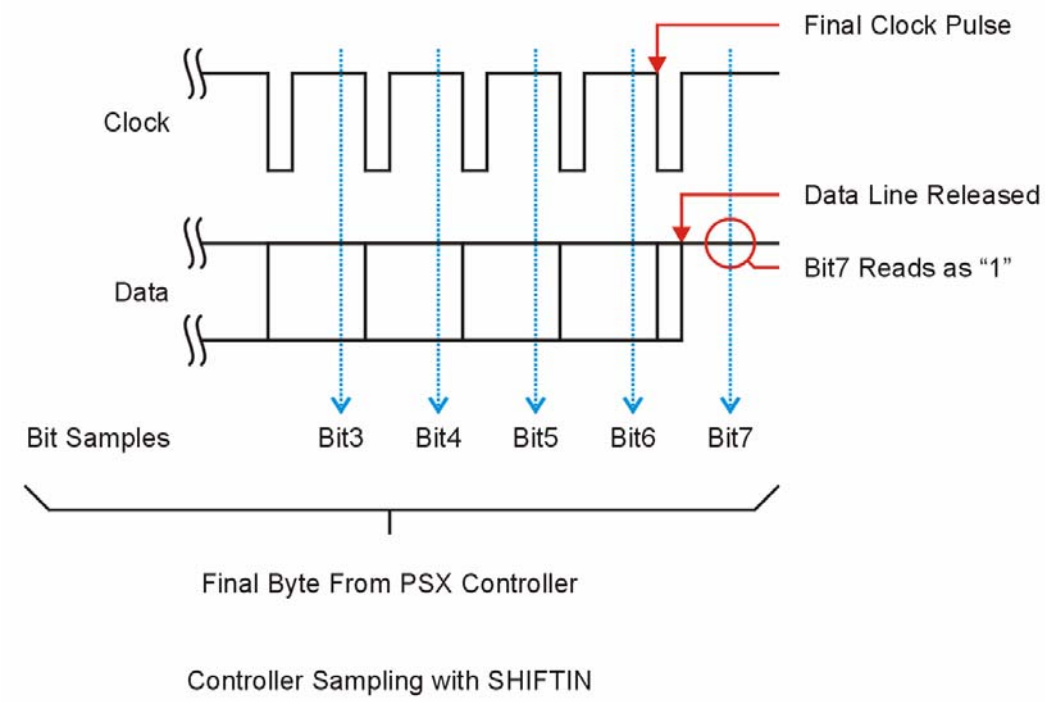
Take a look at Figure 101.3. This graphic shows the how the PlayStation controller sampling is handled. Notice that the bit sampling – as with our code sample above – takes place while the clock line is being held low.

Figure 101.3: PlayStation Controller Last Few Bits of the Last Byte



Now take a look at Figure 101.4. This diagram shows how the controller is sampled when using `SHIFTIN`; specifically the last few bits of the last byte of the sequence.

Figure 101.4: PlayStation Controller Sampling Occurs While Clock Line is Low



Notice that the BASIC Stamp doesn't actually sample the data line until after the clock line is released. This normally isn't an issue, but becomes one for the PlayStation controller. What happens is that the controller can count the clock pulses and when it detects the release of the final clock pulse it releases the data line – which is pulled up through a 4.7K resistor so the last bit (Bit7) of the final byte in the sequence is always 1. This explains why we never see a value lower than 128 on the left joystick Y axis. The manual code does not suffer this problem. That said, I confirmed my suspicion about `SHIFTIN` by structuring the manual code so that it dropped the clock line before sampling – it behaved exactly like `SHIFTIN` does.

The manual version is slower however, by a significant factor so we have to make a choice between speed and accuracy. If you're using a digital controller or don't need the left joystick inputs, they you can get away with using `SHIFTIN`. If you need to verify the device type, or need both joysticks, then you'll need to use manual code. Let's go ahead and take a look at the routine that transfers the full controller packet.

```
Get_PSX_Packet:
  LOW PsxAtt
  psxOut = $01 : GOSUB PSX_TxRx
  psxOut = $42 : GOSUB PSX_TxRx
  psxId = psxIn
  psxOut = $00 : GOSUB PSX_TxRx
  psxStatus = psxIn
  GOSUB PSX_TxRx : psxThumbL = psxIn
  GOSUB PSX_TxRx : psxThumbR = psxIn
  GOSUB PSX_TxRx : psxJoyRX = psxIn
  GOSUB PSX_TxRx : psxJoyRY = psxIn
  GOSUB PSX_TxRx : psxJoyLX = psxIn
  GOSUB PSX_TxRx : psxJoyLY = psxIn
  HIGH PsxAtt
  RETURN
```

The routine starts as we expect by pulling the `PsxAtt` line low. This activates the controller – it works just like a chip select line (just as the controller behaves as a multi-byte shift register). The first byte out is `$01` (start), followed by `$42` (get data). At this point the controller type is available in psxIn and gets transferred to `psxId` for later use. The next byte out is `$00`; the return value at this point is the controller status which should be `$5A` ("ready"). We can use this byte to detect the presence of the controller. If it was unplugged, for example, the `psxId` and `psxStatus` bytes would both be `$FF`. The next six bytes in are the button states and joystick values. At the end of the sequence we disconnect the controller from the buss by taking the `PsxAtt` line high.

Speeding It Up

I was very happy to get the manual code working and be able to detect the controller type and read all of the joystick data – until I timed it to find that it takes nearly 145 milliseconds on a stock BS2. That's just way to long. Look at Figure 101.2 again. The only time we really need to use the manual (slow) shifting is when something is coming back at the same time and a command byte is going out, and on the final byte so that we can read all eight bits properly. Combining techniques, we get this:

```
Get_PSX_Packet_Fast:
  IF (ClockMode = Direct) THEN Get_PSX_Packet
  LOW PsxAtt
  SHIFTOUT PsxCmd, PsxClk, LSBFIRST, [$01]
  psxOut = $42 : GOSUB PSX_TxRx
  psxId = psxIn
  SHIFTIN PsxDat, PsxClk, LSBPOST, [psxStatus]
```

```
  SHIFTIN PsxDat, PsxClk, LSBPOST, [psxThumbL]
  SHIFTIN PsxDat, PsxClk, LSBPOST, [psxThumbR]
  SHIFTIN PsxDat, PsxClk, LSBPOST, [psxJoyRX]
  SHIFTIN PsxDat, PsxClk, LSBPOST, [psxJoyRY]
  SHIFTIN PsxDat, PsxClk, LSBPOST, [psxJoyLX]
  GOSUB PSX_TxRx : psxJoyLY = psxIn
  HIGH PsxAtt
  RETURN
```

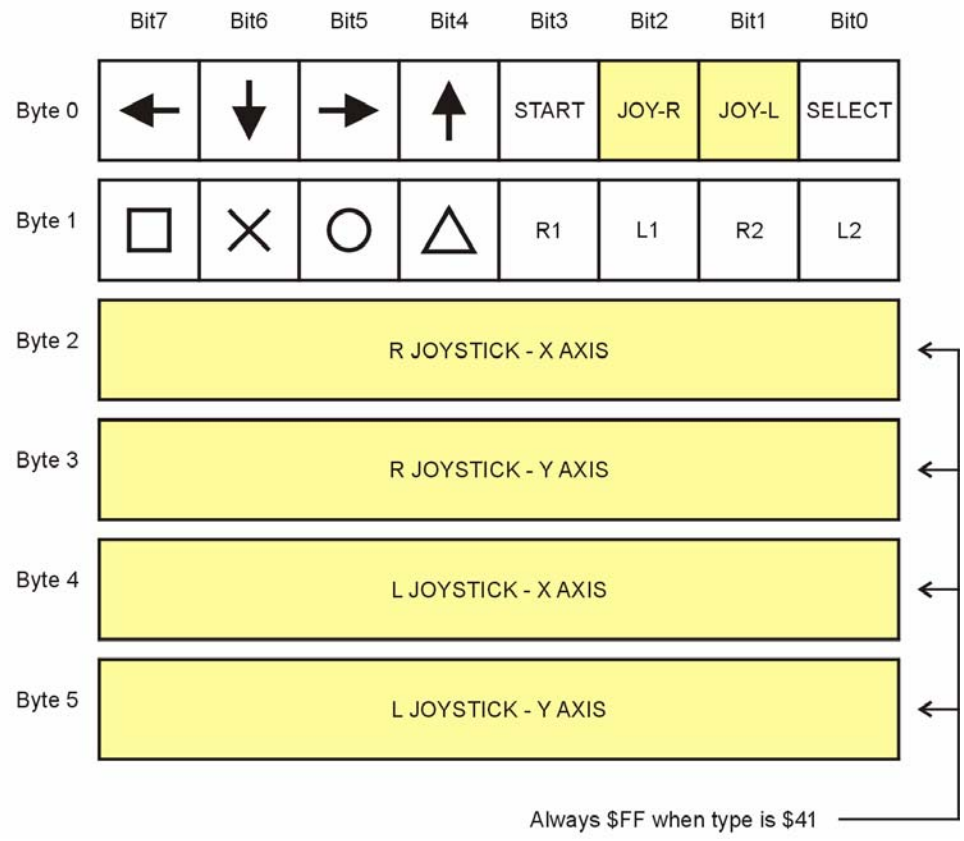
Another thing this routine does is check the interface type we're using based on a constant definition. The transistor inverter can be eliminated, but what this means is that we must use manual shifting only. If we set the `ClockMode` value for direct (just a current limiter, no inverter) then it will force the program to the *slow* version of the code.

By combining techniques, over 100 milliseconds in execution can be shaved off the routine – this is valuable time when controlling a robot. The rest of the demo program (please download from the Nuts & Volts site) displays the controller type and values from it. Figure 101.5 shows the breakdown of the data packet. Note that the shaded areas are only applicable when the controller type is `$73` (analog). My experience is that the Sony analog controller, when running in digital mode, will disable the analog joysticks. However, a clone controller made by Pelican Accessories simulates the button presses with the analog joysticks when running in digital mode. I did confirm Aaron's observation that the clone controller does not have the same resolution on the analog joysticks as the Sony, so be wary of that.

Let the Robot Roll

Finally, what about the robot I promised the girls at the IBM EXITE camp? Well, as you can see by the photo in Figure 101.6 I kept my promise and brought a robot that uses the PlayStation controller. The young lady in the photo is one the stars of the BASIC Stamp class; her name is Diana and she did a great job with the BASIC Stamp despite never working with electronics or computer programming. She also had no trouble driving the Boe-Bot with the PlayStation controller – just as we would expect.

Figure 101.5: Data Packet Breakdown



Let this be a lesson to all of us: It's often a good idea to design toward customer expectations; in fact, it is usually the best idea. I've worked with a lot of young engineers who – in an effort to make their mark on the world – design things differently just for the sake of being different. I'm going to suggest you be careful to avoid this [ego-driven] trap, as it often leads to customer frustration and disappointment for you, the hard-working engineer. Remember that being original doesn't mean you have to be "different."

Okay, now you have a "standardized" interface device that is well known to a wide range of customers and several options for using it. How might you use this standard interface to create something original?

Another Contest

Would you like to have a spare Stamp for your collection? Perhaps try one that you don't currently own? Okay, let me tell you how you can get a brand-new Stamp at no charge: You simply need to send me working BS2 code for the Sony Dual-Shock controller that selectively activates the motors (the 9V connection to pin 3 of the controller is for the motors). I am actively pursuing this myself, but publishing deadlines for the article prevented me from getting it working. Once I do – or someone shows me how – I will make that code available to everyone, and the person who sends me the code will get his or her choice of new BASIC or Javelin Stamp. Any takers?

Until next month, Happy Stamping.

Figure 101.6: Diana Drives a Boe-Bot with PlayStation Controller

```
' =============================================================================
'
'   File...... PSX_Demo.BS2
'   Purpose... PlayStation Controller Interface
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallax.com
'   Started...
'   Updated... 17 JUL 2003
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------

' This program demonstrates the essential interface between the BASIC
' Stamp and a Sony PlayStation (or compatible) game controller.  This
' code assumes that the clock signal is inverted between the Stamp and
' the controller to allow simpler [less sophisticated] interface with
' SHIFTOUT and SHIFTIN.
'
' Note: The interface and portions of code are based on previous work by
'       Aaron Dahlen.


' -----[ Revision History ]------------------------------------------------


' -----[ I/O Definitions ]-------------------------------------------------

PsxAtt          PIN     8                       ' PSX joystick interface
PsxClk          PIN     9
PsxCmd          PIN     10
PsxDat          PIN     11


' -----[ Constants ]-------------------------------------------------------

Inverted        CON     1                       ' inverted clock signal
Direct          CON     0                       ' no inverter in clock line
ClockMode       CON     Inverted


' -----[ Variables ]-------------------------------------------------------

idx             VAR     Nib                     ' loop counter
psxOut          VAR     Byte                    ' byte to controller
psxIn           VAR     Byte                    ' byte from controller
```

```
' joystick packet

psxID           VAR     Byte                    ' controller ID
psxThumbL       VAR     Byte                    ' left thumb buttons
psxThumbR       VAR     Byte                    ' right thumb buttons
psxStatus       VAR     Byte                    ' status ($5A)
psxJoyRX        VAR     Byte                    ' r joystick - X axis
psxJoyRY        VAR     Byte                    ' r joystick - Y axis
psxJoyLX        VAR     Byte                    ' l joystick - X axis
psxJoyLY        VAR     Byte                    ' l joystick - Y axis


' -----[ EEPROM Data ]------------------------------------------------


' -----[ Initialization ]---------------------------------------------

Setup:
  HIGH PsxAtt                                   ' deselect PSX controller
  OUTPUT PsxCmd
  PsxClk = ~ClockMode                           ' release clock
  OUTPUT PsxClk                                 ' make clock an output


' -----[ Program Code ]-----------------------------------------------

Main:
  DO
    GOSUB Get_PSX_Packet_Fast                   ' type and packet
    DEBUG HOME, "Type = "

    IF (psxStatus = $5A) THEN
      DEBUG IHEX2 psxId, " (", IHEX2 psxStatus, ")",
            CLREOL, CR, CR,
            BIN8 psxThumbL, " ", BIN8 psxThumbR, " "

      IF (psxId <> $41) THEN
        DEBUG DEC3 psxJoyLX, " ", DEC3 psxJoyLY, " ",
              DEC3 psxJoyRX, " ", DEC3 psxJoyRY
      ELSE
        DEBUG CLREOL
      ENDIF
    ELSE
      DEBUG "Unknown.  No response.", CR, CLRDN
      PAUSE 1000
    ENDIF
    PAUSE 100
  LOOP

  END
```

```
' -----[ Subroutines ]-----------------------------------------------------

' This routine REQUIRES inverted clock signal from
' Stamp to PSX controller

Get_PSX_Buttons:
  IF (ClockMode = Direct) THEN Get_PSX_Packet   ' redirect if not inverted
  LOW PsxAtt
  SHIFTOUT PsxCmd, PsxClk, LSBFIRST, [$01, $42]
  SHIFTIN  PsxDat, PsxClk, LSBPOST, [psxThumbL, psxThumbL, psxThumbR]
  psxId = $41
  HIGH PsxAtt
  RETURN


' This routine manually creates the clock signal,
' so it can be used with a direct (via 220 ohm resistor)
' connection to the clock input
'
' Execution time on BS2 is ~145 ms.

Get_PSX_Packet:
  LOW PsxAtt                                    ' select controller
  psxOut = $01 : GOSUB PSX_TxRx                 ' send "start"
  psxOut = $42 : GOSUB PSX_TxRx                 ' send "get data"
  psxId = psxIn                                 ' save controller type
  psxOut = $00 : GOSUB PSX_TxRx
  psxStatus = psxIn                             ' should be $5A ("ready")
  GOSUB PSX_TxRx : psxThumbL = psxIn            ' get PSX data
  GOSUB PSX_TxRx : psxThumbR = psxIn
  GOSUB PSX_TxRx : psxJoyRX = psxIn
  GOSUB PSX_TxRx : psxJoyRY = psxIn
  GOSUB PSX_TxRx : psxJoyLX = psxIn
  GOSUB PSX_TxRx : psxJoyLY = psxIn
  HIGH PsxAtt                                   ' deselect controller
  RETURN


' Transmit psxOut to, and receive psxIn from the
' PSX controller

PSX_TxRx:
  FOR idx = 0 TO 7
    PsxCmd = psxOut.LOWBIT(idx)                 ' setup command bit
    PsxClk = ClockMode                          ' clock the bit
    psxIn.LOWBIT(idx) = PsxDat                  ' get data bit
    PsxClk = ~ClockMode                         ' release clock
  NEXT
  RETURN
```

```
' This routine combines manual and built-in shifting
' routines to get the best speed and all valid data.
'
' Execution time on BS2 is ~40 ms.

Get_PSX_Packet_Fast:
  IF (ClockMode = Direct) THEN Get_PSX_Packet   ' redirect if not inverted
  LOW PsxAtt                                    ' select controller
  SHIFTOUT PsxCmd, PsxClk, LSBFIRST, [$01]      ' send "start"
  psxOut = $42 : GOSUB PSX_TxRx                 ' send "get data"
  psxId = psxIn                                 ' save controller type
  SHIFTIN PsxDat, PsxClk, LSBPOST, [psxStatus]  ' should be $5A ("ready")
  SHIFTIN PsxDat, PsxClk, LSBPOST, [psxThumbL]
  SHIFTIN PsxDat, PsxClk, LSBPOST, [psxThumbR]
  SHIFTIN PsxDat, PsxClk, LSBPOST, [psxJoyRX]
  SHIFTIN PsxDat, PsxClk, LSBPOST, [psxJoyRY]
  SHIFTIN PsxDat, PsxClk, LSBPOST, [psxJoyLX]
  GOSUB PSX_TxRx : psxJoyLY = psxIn
  HIGH PsxAtt                                   ' deselect controller
  RETURN
```