SWTPC PR-40 Printer + Sphere

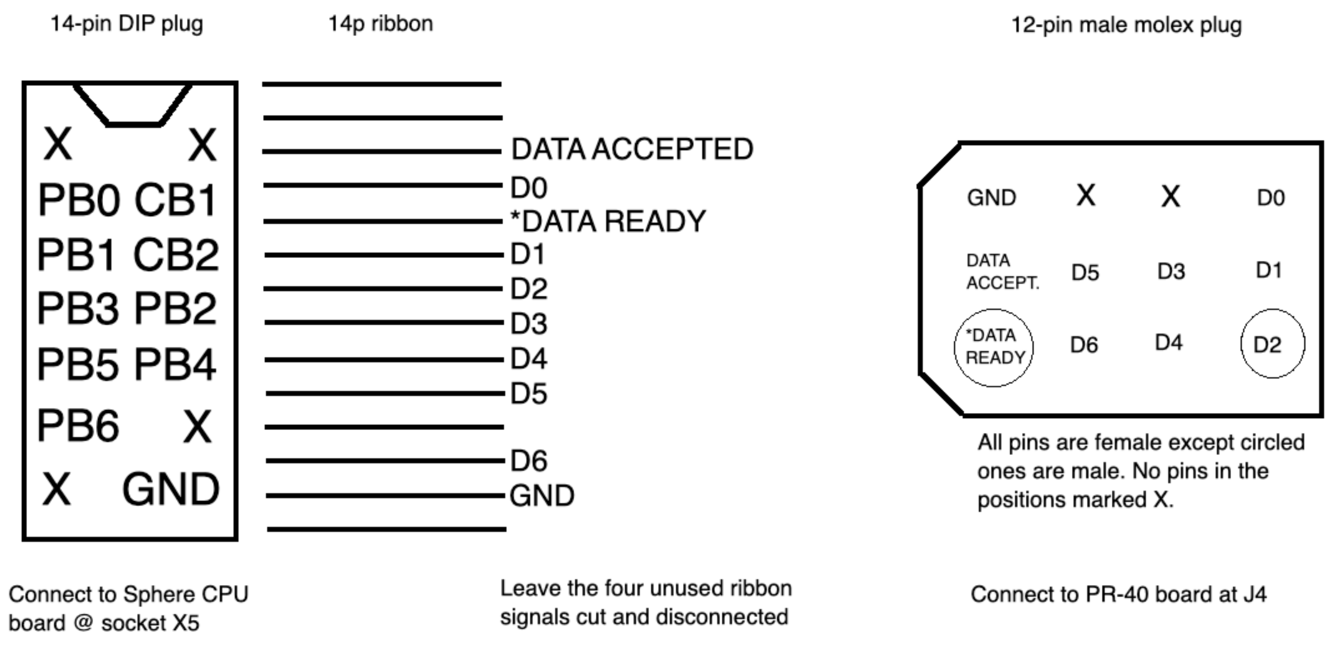
Ben Zotto, 2024 (revision 1)
https://sphere.computer

Southwest Technical's famed little impact printer kit makes for a useful companion to a Sphere computer—many original Sphere customers thought so too: users published driver software and other hints. The printer is designed to be driven by one port of a 6820 PIA chip (or similar) and fortunately even the most basic Sphere setup has exactly what is required: a spare 6820 PIA port on its CPU board. All you need to do is make the right cable and write some software to drive it.

Interconnection

If you are using the KBD/2 (or equivalent via adapter) like almost all Sphere systems, then "port A" of the 6820 PIA chip on the CPU board is used to interface with that keyboard via socket X4. [See note below] But the Sphere leaves open its PIA's "port B" for user connection via socket X5. All of the required signals are already available at that socket, we just need to create the correct cable.

The PR-40 printer expects 7 data lines in, one strobe line in (called "data ready") and one interrupt or done line out ("data accepted") which is how the printer talks back to the computer. We will set this up from our Sphere PIA as PB0-PB6 (data output to printer), plus control lines CB1 (data accepted) and CB2 (data ready strobe). We use one of the standard 14-pin DIP plugs on the Sphere side, and then on the printer side is a 12-position male Molex plug. We will connect the nine signal lines as above, plus the signal ground, using ten of the 12 positions in the Molex and leaving the other wires disconnected.


14-pin DIP plug
14p ribbon
12-pin male molex plug
X X
PB0 CB1
PB1 CB2
PB3 PB2
PB5 PB4
PB6 X
X GND
DATA ACCEPTED
D0
*DATA READY
D1
D2
D3
D4
D5
D6
GND
GND X X D0
DATA ACCEPT. D5 D3 D1
*DATA READY D6 D4 D2
All pins are female except circled ones are male. No pins in the positions marked X.
Connect to Sphere CPU board @ socket X5
Leave the four unused ribbon signals cut and disconnected
Connect to PR-40 board at J4

NOTE for the diagrams above that both plugs are shown in the conventional "through the plug looking at the socket" orientation with "pin 1" on the respective connectors in the upper left corner of each.

Crimp a DIP plug onto one side of a 14-connector ribbon cable that's a few feet long to run to the printer (try to keep this as short as is reasonable: the 6820 PIA will be driving unbuffered TTL signals across the length of the cable. If you run into signal integrity issues, try swapping the chip for a 6821 PIA which has improved electrical characteristics but the same interface).

On the other end of the ribbon cable, carefully split apart the signal wires for the last couple inches. See the diagram above for the interconnection pattern. **Be SURE** the DIP plug uses the typical configuration of routing DIP pin 14 and pin 7 signals to the outermost lines of the ribbon cable. SOME versions of these IDC DIP plugs are shifted, such that pin 1 and pin 8 end up on the outer wires. If you have the latter kind, you'll need to recompute which signal goes where on the ribbon cable; the diagram above will be wrong.

The Molex plug is the 0.093" size, and those crimp pins are not made for wires as small as the 28AWG ribbon cable wires. So be careful when crimping the pins on to get a decent grip, and then I recommend additionally using a daub of solder to really secure the wire to the pin material and make a reliable connection.

Notice that the Molex plug itself is male, but the pins inside it vary to mate with the printer's board correctly. My printer socket needed 8 female crimp pins and two male ones for this connection. I assume all these printers are the same for their J4 connection, but double check yours before fitting the pins.

There will be four unused signals in the ribbon cable. Simply cut these off and make sure they will not make electrical contact anywhere.

It's good practice once the cable is constructed to do a continuity check between the DIP pins on one side and the corresponding Molex pins on the other side to make sure everything is connected well. Be sure to plug the DIP plug into the Sphere (X5 on the CPU board) in the correct orientation—unlike the Molex, you can definitely do this the wrong way around by accident.

NOTE: If you are instead using the older KBD/1 keyboard, it connects differently, leaving both ports of the PIA available, and you can still use port B. BUT if this is your setup, you may not already have a 6820 PIA chip installed at location E3. Just install one there (or a 6821) to use with the printer. Nothing changes.

Connection Check

The simplest way to check that the connection is up and more or less working is to use the Sphere PDS debugger to set four values in memory (actually the PIA port registers) to get the printer to respond. After connecting the printer and Sphere, turn on the printer and the computer.

Open location $F042. This will initially reflect the data direction register for PIA port B (should be $00 after reset). Set this to $FF (all pins as outputs). Then the control register at $F043 should be set to $3F (opens the data register and sets the data strobe high). Then go back to location $F042 and set it to $0D, the carriage return character—this will be latched in the PIA for the printer to see. Finally, change address $F043 (control register) to $37. This toggles the strobe low which tells the printer that the carriage return data just configured is valid. Immediately upon setting this value, the printer should run the motor for one carriage return cycle (nothing will print).

If this happens, then your basic connection is in place: the PIA is working as expected, and the data and control lines are connected to the right places on the printer and you can move on to diagnostics.

If not, something could be wrong with either the cable or the printer. You can use a basic multimeter to check the TTL voltages at the pins of the Molex connector: you should see the data pins have the stable latched bit pattern of the carriage return character, and the strobe should be low (after following the steps above). If that all looks as expected, the problem probably lies with the printer.

Diagnostic program

There's an assembly listing in the SWTPC manual that basically runs the printer constantly and prints out the whole character set that it supports. The listing is designed to target a SWTPC 6800 computer running MIKBUG firmware and using its standard parallel board. I've adapted this program to run on the Sphere-- it's almost exactly the same, but the PIA address and entry point are different.

You can load and run this by manually entering bytes or by converting to cassette data, etc.

```
$0000                ; SWTPC PR-40 Diagnostic
$0000                ;
$0000                ; Adapted for Sphere from SWTPC PR-40 manual by Ben Zotto, 2024.
$0000                ;
$0000                ; Prints constant alternating lines showing the full character set.
$0000                ;
$0000                ; Uses the port B interface of the CPU's PIA.
$0000                ; CB1 for "data accepted" input, CB2 output for "data ready" strobe.
$0000                ;
$0000                ; Don't let this run indefinitely, the print head solenoids can overheat.
$0000
$0000                PRPIA   EQU     $F042       ; port B of the onboard PIA
$0000
$0000                        ORG     $200
$0200
$0200  CE F0 42       START   LDX     #PRPIA      ; Setup PIA (assumes from reset!)
$0203  C6 FF                  LDA B   #$FF
$0205  E7 00                  STA B   0, X        ; all data bits to output
$0207  86 3F                  LDA A   #$3F
$0209  A7 01                  STA A   1, X        ; data reg; CB1 interrupt on +ve transition; CB2 high
$020B
$020B  86 0D          FSTLIN  LDA A   #$0D
$020D  8D 1A                  BSR     OUTCHR      ; emit carriage return
$020F  86 20                  LDA A   #$20        ; first line starts with '!' char
$0211  4C             LOOP1   INC A
$0212  81 40                  CMP A   #$40
$0214  27 04                  BEQ     NXTLIN
$0216  8D 11                  BSR     OUTCHR
$0218  20 F7                  BRA     LOOP1
$021A  86 0D          NXTLIN  LDA A   #$0D
$021C  8D 0B                  BSR     OUTCHR
$021E  86 3F                  LDA A   #$3F        ; second line starts with '@' char
```

```
$0220  4C           LOOP2   INC A
$0221  81 60                CMP A   #$60
$0223  27 E6                BEQ     FSTLIN      ; restart with a first line
$0225  8D 02                BSR     OUTCHR
$0227  20 F7                BRA     LOOP2
$0229
$0229              ; OUTCHR - routine to emit one character (in A)
$0229              ;   assumes PIA base is in X
$0229              ;
$0229  A7 00        OUTCHR  STA A   0, X        ; send char data
$022B  C6 37                LDA B   #$37
$022D  E7 01                STA B   1, X        ; bring CB2 (data ready strobe) low
$022F  C6 3F                LDA B   #$3F
$0231  E7 01                STA B   1, X        ; return CB2 (data ready strobe) to high
$0233  6D 01        LOOP3   TST     1, X        ; done printing this char?
$0235  2A FC                BPL     LOOP3
$0237  E6 00                LDA B   0, X        ; clear the interrupt
$0239  39                   RTS
```

Software notes

If you want to write programs that do anything with the printer besides test it, you'll want to have a bit more context. The functioning of a PIA is outside the scope of this document, but in short, you want to program the control register to trigger an interrupt (or flag) when CB1 goes high so that you know when you're able to print the next character. You also want to control the output level of CB2 (the data strobe). The "OUTCHR" routine in the diagnostic program is perfectly functional for this purpose, and you can reuse it as a module. The $3F and $37 are full bit patterns that control port B of the PIA; it's a bit confusing because you must rewrite the whole bit pattern even if you're changing only a few bits. $3F is the version that sets CB2 high; $37 brings CB2 low. The other stuff is set up as you want it to be.

Once the PIA is configured correctly snd you're up and running, you just need to set a character value in the PIA data register ($F042) and then write a $37 followed by a $3F to the control register ($F043). That sequence latches the output character and toggles the data ready strobe, and the printer should accept the character. You then loop on the high bit of the control register, which will itself go high when the printer is ready for the next character.

Varsity programmer note: You can put the PIA in a different mode where it automatically strobes CB2 low for one cycle of its E clock and then sets it high again. This should work with the printer, because the E clock cycle time (= phi 2 clock on Sphere) is longer than 1 microsecond, which is the printer's minimum strobe duration. This means you don't have to toggle CB2 manually via the control register for every character output, so saves you a handful of bytes in the print subroutine, but it won't save you any wall clock time when printing.

NOTE ON INTERRUPTS: the sample code and control values here also instruct the PIA to interrupt the actual CPU (in the sense of causing an IRQ). That's how the SWTPC sample does it so I've left that here, but this will cause havoc if interrupts are actually enabled on your Sphere CPU at the time you run it and you don't have interrupt handling code ready. It works normally because the M6800 interrupts are inhibited (off) by default when the CPU comes out of reset and the Sphere firmware never enables them. If you are writing code for another environment that might enable interrupts, either issue an "SEI" instruction before doing printer stuff or alter the control bits to set the flag but not cause an IRQ.