599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office: (916) 624-8333
Fax: (916) 624-8003

General: info@parallax.com
Technical: support@parallax.com
Web Site: www.parallax.com
Educational: www.stampsinclass.com

96 dpi Serial Inkjet Printer Development Kit (#27949)

1.0 Introduction

1.1 Description

Parallax Inc., the Hewlett-Packard Company's Specialty Printing Systems/Inkjet Supplies Operation, and Matt Gilliland cooperated in the development of the Parallax Serial Inkjet Printer Kit. The hardware and documentation are designed around the HP Thermal Inkjet 1.0 design (96 dpi). Using the Serial Inkjet Printer board with a microcontroller (TTL) or PC (USB mini B) connection, printing 1/8" tall 96 dpi characters is as easy as waving the inkjet print cartridge 1/16" to 1/8" over a piece of paper at a constant rate.

The Serial Inkjet Printer Kit:

- makes the Parallax Boe-Bot® robot a mobile printer (mounting hardware is included in this kit);
- provides an alternative method of displaying data (versus a serial LCD or PC monitor); and
- is an open-sourced reference design which OEMs and hobbyists may freely use in their printing applications, either as-is or in their own printed circuit board designs.

The book Inkjet Applications – Circuits with the BASIC Stamp 2 and SX Microcontrollers by Matt Gilliland is an understandable, in-depth reference on this specific print technology. This book includes multiple circuit examples, code examples for the BASIC Stamp® and SX (using SX/B compiler) microcontrollers, and tips for obtaining the best print quality.

1.2 Documentation Updates

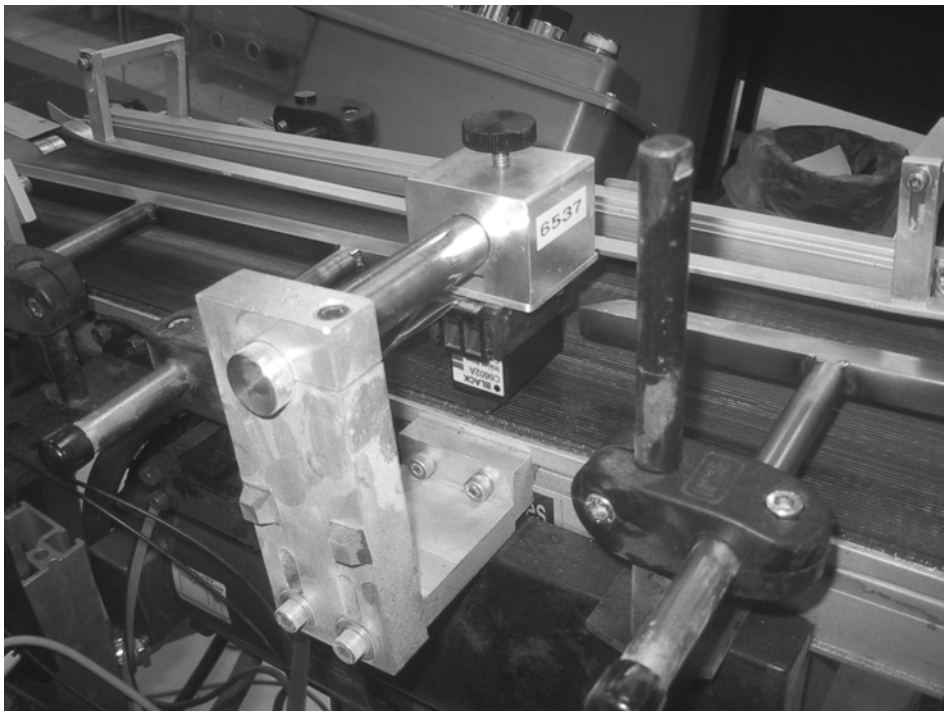
Improvements to code and documentation are common in the early stages of a hardware release. Visit http://www.parallax.com/detail.asp?product_id=27949 to download new documentation and current BASIC Stamp and SX code files for this product. All code examples provide in the document are also available for download from this same link. Send any comments or errors found in this documentation to editor@parallax.com.

1.3 Application Ideas

Common uses for the 96 dpi print technology currently include receipt printing, data printing and production-line labeling. Some new application ideas for the Serial Inkjet Printer hardware may include:

- using a material other than ink;
- printing on media other than paper, such as plastic, skin or wood;
- robotic graffiti and painting 'bots that point the inkjet nozzle horizontally; and
- bar-code printing.

Parallax and Hewlett-Packard take no responsibility for your applications. In fact, many of these could be harmful to human health or require approval by the Food and Drug Administration. Review the appropriate Material Safety Data Sheets (MSDSs) for more information on chemical and safety considerations.



Customer Application: Hewlett Packard's Manufacturing Operations in Boise, Idaho uses a BASIC Stamp 2 and their own design (similar to our Serial Inkjet Printer) to print labels on the 51604/51605 inkjet cartridge boxes. Process support engineers and technicians put their heads together and configured this solution with the goals of high up time, simple integration, small footprint, and better reliability than their prior date code labeling process. Now, a BASIC Stamp and SX chip manage the inkjet to print the date code on every 51604A/51605B/51605R inkjet box as they are conveyed on a belt under the inkjet print cartridge. More details of this application, including a video, are available on the Parallax web site (www.parallax.com under Resources / Customer Applications / Industrial Commercial / Assembly Date Line Coder.

1.4 Serial Inkjet Printer Kit Contents

If parts are missing from your kit, contact Parallax immediately for replacements. The 96 dpi Serial Inkjet Printer Development Kit (#27949) includes the following contents, which may also be purchased individually.

96 dpi Serial Inkjet Printer Development Kit (#27949) Parts and Quantities Subject to Change Without Notice		
Stock code	**Description**	**Qty**
27948	Serial Inkjet Printer Circuit Board	1
70017	Inkjet Applications – Circuits wih the BASIC Stamp 2 and SX Microcontrollers by Matt Gilliland (ISBN 0-9720159-3-0)	1
30014	HP 51604A Black Inkjet Cartridge	1
556-27948	HP Q7453A Inkjet Cartridge Holder	1
720-27948	Aluminum Boe-Bot Inkjet Mounting Bracket	1
805-00002	3-pin F/F 14" 26 AWG red/white/black cable	1
713-00001	Standoff, aluminum, 5/8" long, ¼" diameter	2
700-00003	4-40 nut, zinc plated	2
700-00028	4-40 panhead screw, Phillips, ¼" long	6

A complete Bill of Materials for the Serial Inkjet Printer printed circuit board (#27948) is available in Section 6.0 for developers who wish to implement the same circuit in their own OEM designs.

2.0 Serial Inkjet Printer Hardware

2.1 Features

Parallax's Serial Inkjet Printer hardware is designed with the following features:

- Font storage in a 64 K Byte EEPROM, which may be loaded from a BASIC Stamp or USB connection
- Simple protocol for printing a 64-byte string of characters
- Timing and sequence management for printing a 12x8 or 12x12 character
- LEDs for visual feedback during experimentation
- Step-up 6 to 24 VDC power supply for inkjet nozzle firing via Darlington array
- 2x7 0.1" header (unpopulated) for direct control of inkjet nozzle (only used if customer has a desire to bypass the pre-programmed SX28AC/SS firmware)
- Connector for the HPQ7453A Printer Cartridge Holder
- Open-sourced designed for OEMs who are making their own 96 dpi printing products

Brief Overview of Inkjet Firing Timing, Sequence and Font Storage A short explanation of the signals required to activate the inkjet cartridge nozzles will lend an appreciation of the Parallax Serial Inkjet Printer.

Firing inkjet nozzles requires a supply voltage of 22.5 - 24 VDC and series of pulses ranging from 4.5 - 6.0 µs, with inactive periods of 500 µs before firing the same nozzle again. Additionally, only two nozzles may be fired at one time. The nozzles must be paired to assure their lowest proximity. Firing adjacent nozzles (such as 1 and 2, or 5 and 6) causes droplet degradation from air pressure. So, nozzles in low proximity are fired together (such as 1 and 7, 2 and 8, etc.). Additionally, a refractory period of 500 µs must be provided before firing the same nozzle pair again. To create a column of 12 dots (such as the first line in the letter "B"), the column can be fired as quickly as:

= (the number of nozzles fired) x (the length in microseconds of each pulse)
+ (the number of dead-times) x (the length of each dead-time)

= (12 nozzles) x (6.0 µs pulse width) + (11 dead-times) x (0.5 µs dead-time)

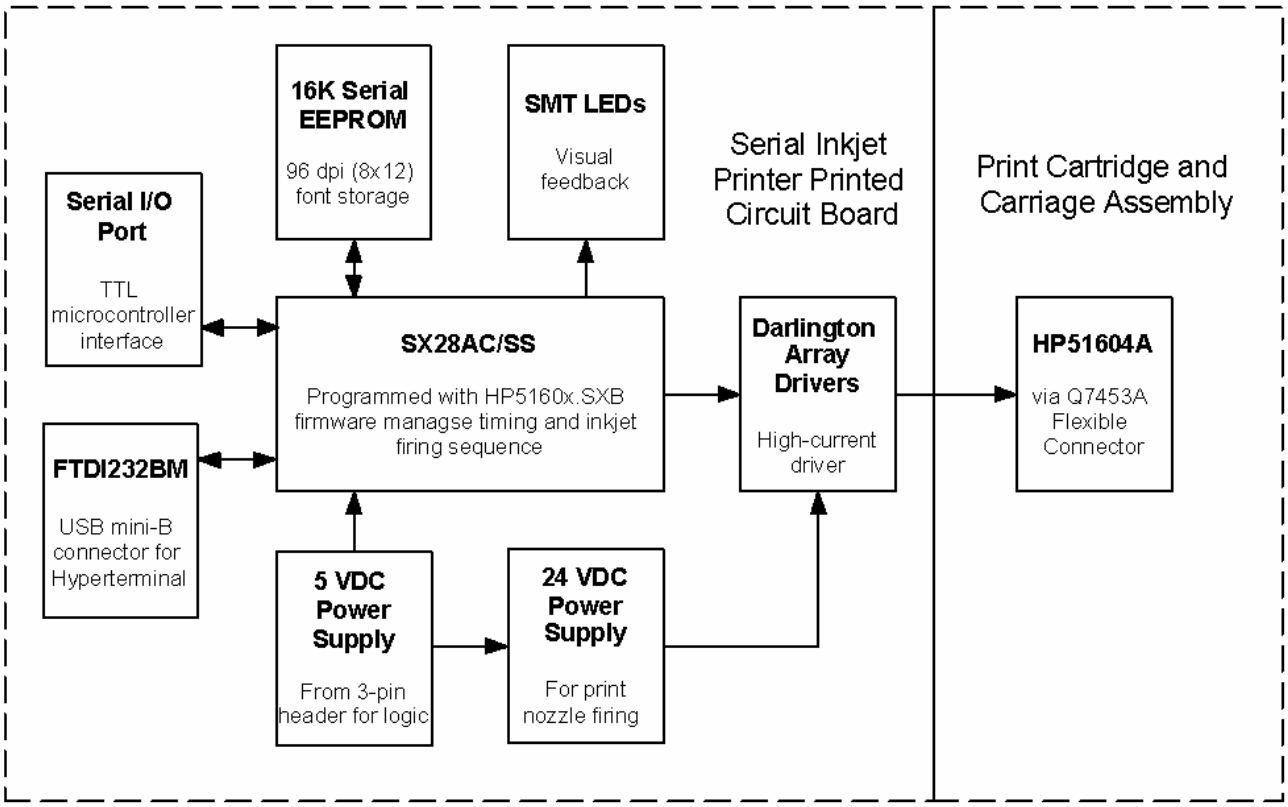
≈ 78 microseconds

Then you can move to the next column.

A complete set of fonts occupies 20-80 K bytes depending on the number of characters defined. Many 8-bit microcontrollers (such as the BASIC Stamp and SX) have limited memory, making an external EEPROM an ideal place for storing fonts.

The Parallax Serial Inkjet Printer manages all of the nozzle firing parameters and font storage from a simple serial input.

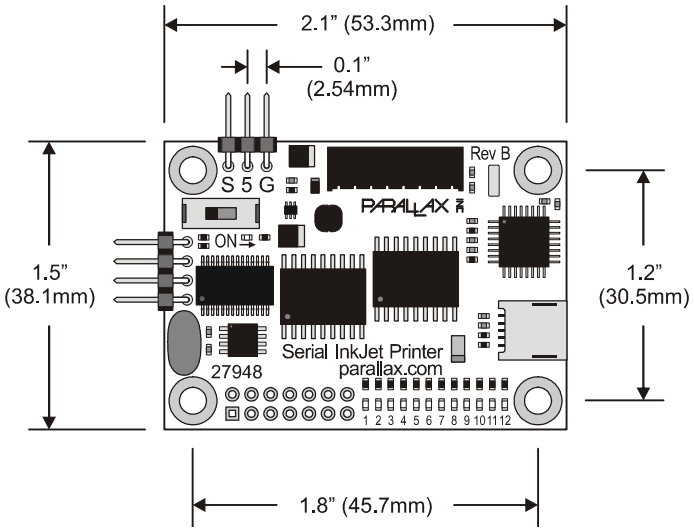
2.2 Functional Block Diagram



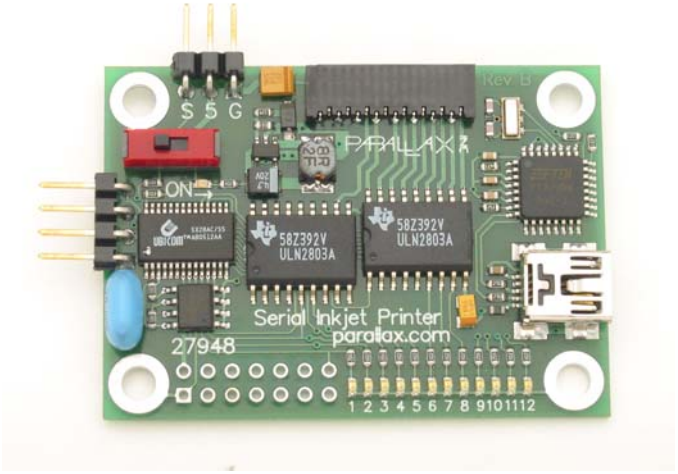
2.3 Technical Specifications

Size	1.5” x 1.8” (38.1 mm x 45.7 mm)
Operating Temperature	0-70° C
Supply Power	6 VDC
Current Draw	50 mA idle and 65 mA maximum
Interfaces	5V logic level UART or USB (virtual COM port) with 9600 bps, 8-bit, no parity true open-mode baud rate
Firmware	SX28AC/SS is programmed with HP5160x.SXB firmware (available for download)
Connectors	▪ 3-pin, 0.1” space for UART control (S) , 5V (5), Vss (G) ▪ 4-pin, 0.1” space for Parallax SX-Key ▪ USB Mini-B for PC USB virtual COM port interface (requires FTDI VCP drivers) ▪ 2 x 7 0.1” spaced breakout holes for direct connection to print head connector to bypass SX28AC/SS’s HP51604x.SXB firmware ▪ 16-position 1 mm right-angle connector for Q7453A Inkjet Cartridge Holder

2.4 Mechanical Drawing and Photograph



Note: the 3-in header is comprised of square pins that are .025” square.



3.0 Quick Start Guide

3.1 Command Set

The Serial Inkjet Printer serial interface is a 9600 bps, 8-bit, no parity, true, open baud rate. See the BASIC Stamp PBASIC Syntax Guide (in the BASIC Stamp Windows Editor on-line Help) to calculate baud modes for different BASIC Stamp models. The Serial Inkjet Printer has the following built-in command set (further described in Section 6.0) for OEMs.

All commands to the Serial Inkjet Printer are prefaced with an ESC (ASCII 0.27) character.

Command	Description
ESC	Escape (ASCII 027) preface puts Serial Inkjet Printer in waiting mode for receiving commands
>	Ready prompt (ASCII 062) from Serial Inkjet Printer to receive next command
STX	Start-of-text (ASCII 002) indicates the start of a string to print (up to 64 pre-loaded characters)
ETX	End-of-text marker (ASCII 003)
C	Inter-column delay (ASCII 067) or "feed rate" of the inkjet nozzle, between 0.1 and 25.5 ms (i.e., a value of 15 is 1.5 ms)
N	New character (ASCII 078) is used only with the HP_Load_Font.bs2 program to map characters in the EEPROM.
T	Timing value for LED controls, specified in five millisecond units. The purpose of this command is educational, allowing the user to "view" the nozzle output sequence on the surface-mount LEDs.
D	Display (0 or 1); 0 indicates standard printing mode (only two nozzles fired at one time), 1 indicates LED display mode (up to 12 nozzles at one time).

Using the T and D commands will destroy the inkjet printer cartridge if it is connected to the Serial Inkjet Printer board. These commands are present for visualizing the nozzle firing routines, strictly for educational use only. To experiment with the T and D commands, first disconnect the inkjet cartridge holder from the Serial Inkjet Printer. While researching circuits and testing code for the Inkjet Applications book, Matt Gilliland destroyed over 80 nozzles on more than 30 inkjet printer cartridges with his experimentation.

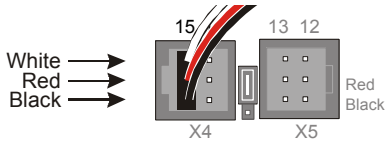
The Serial Inkjet Printer will store 64 bytes of characters, which all must be pre-defined using the HP_Load_Font.bs2 program description in Section 4.1.

3.2 Simple_Print.BS2 Example

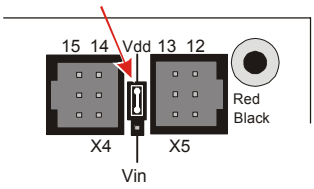
The Simple_Print.BS2 program is a short example of printing strings from a BASIC Stamp 2 module.

1. Connect the Serial Inkjet Printer to the BASIC Stamp 2's I/O pin P15 using the 3-pin cable.

   - White: P15
   - Red: Vdd
   - Black: Vss



2. If you are using a Board of Education, set the jumper to Vdd (or, you may use Vin only if you are using 6 V from 4 AA batteries in a battery pack for the power supply).



3. Insert the printer cartridge into the print cartridge holder.
4. Connect the ribbon cable to the Serial Inkjet Printer.
5. Load the Simple_Print.BS2 example program.
6. Cycle power to the Serial Inkjet Printer, or to both the SIP and the BASIC Stamp simultaneously.
7. Wave the print cartridge over a piece of paper to print the characters.

The operation of this BASIC Stamp code is predicated on receiving a prompt character from the Serial Inkjet Printer Module. If the BASIC Stamp doesn't receive this prompt, it will do nothing. Shortly after applying power to the Serial Inkjet Printer Module, it transmits the prompt to the host, in this case the BASIC Stamp. Considering this, it is necessary that Serial Inkjet Printer Module is powered up either simultaneously with, or sometime after, the host powers up. If the BASIC Stamp module is powered up after the Serial Inkjet Printer module, executing your code will have no effect because the BASIC Stamp will be waiting in vain for the Serial Inkjet Printer's prompt.

```
' =============================================================================
'
'   File....... Simple_Print.BS2
'   Purpose....
'   E-mail..... support@parallax.com
'   Started....
'   Updated.... 25 OCT 2005
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================


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


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


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

Inkjet          CON     15                      ' to HP driver
```

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

T9600           CON     84
Open            CON     $8000
Baud            CON     Open + T9600

Prompt          CON     ">"                     ' ready prompt from driver
STX             CON     2                       ' start of text
ETX             CON     3                       ' end of text
Esc             CON     27                      ' OEM configuration command


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


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


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

Reset:
  SEROUT Inkjet, Baud, [Esc, "C", 15]           ' intercolumn delay 1.5 ms
  SERIN  Inkjet, Baud, 500, No_Inkjet, [WAIT(Prompt)]


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

Start:
  SEROUT Inkjet, Baud, [STX]
  SEROUT Inkjet, Baud, ["PRINT UP TO SIXTY FOUR BYTES OF CHARACTERS"]
  SEROUT Inkjet, Baud, [ETX]
  END


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

No_Inkjet:
                                                ' not-ready code here
  END
```

3.2 Simple_Print_DATA.BS2 Example

For an example program that stores strings in the EEPROM and uses a pointer to retrieve and print, download Simple_Print_DATA.BS2 from http://www.parallax.com/detail.asp?product_id=27949.

3.3 HyperTerminal with USB

The Serial Inkjet Printer's USB-mini B connection may be used for control and font downloading via HyperTerminal. This requires installation of the FTDI virtual COM port drivers, available for download at http://www.parallax.com/detail.asp?product_id=28802). The same command set used by the BASIC Stamp may be issued via USB. However, these examples are not yet available and will be provided when the Edit-Ink program is completed (see Using Edit-Ink in Section 4.2). Check back to the Serial Inkjet Printer web page on Parallax's web site for HyperTerminal and Edit-Ink examples.

4.0 Downloading New Characters

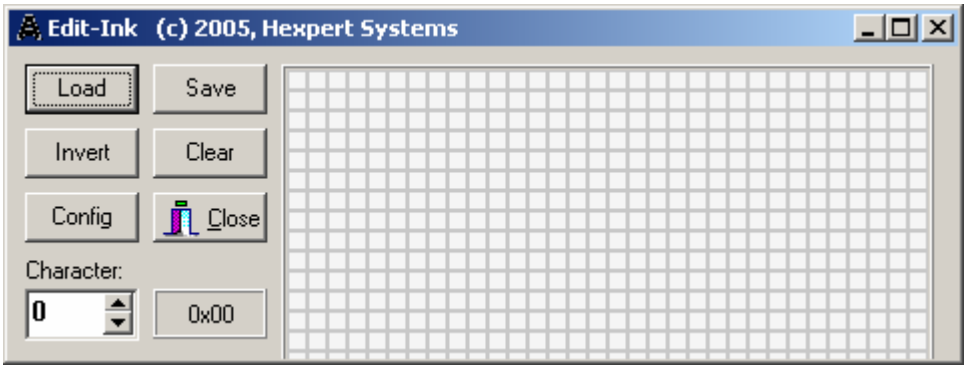
4.1 Using the HP_Load_Font.BS2 Program

The example BASIC Stamp program HP_Load_Font.BS2 transfers font maps into the HP5160x driver board EEPROM. New characters may be downloaded to the 64KB EEPROM by modifying this source code. Once a character is mapped and appears in this program, it may be printed. The default EEPROM character set is all capitals. HP_Load_Font.BS2 supports all Parallax BASIC Stamp 2 modules (BS2, BS2e, BS2sx, BS2p BS2pe, BS2px) with the use of #SELECT…#CASE directives. This program can be downloaded from http://www.parallax.com/detail.asp?product_id=27949.

4.2 Edit-Ink Font Development Program

Edit-Ink is a simple to use font development program created especially for the 51604x inkjet cartridge. You can download it for free from http://www.parallax.com/detail.asp?product_id=27949. Edit-Ink is a handy tool used to make a special character, or an entirely new font. It will create any sized character up to 12 drops high by as many as 32 drops long.

Chapter 7 of the Inkjet Applications – Circuits with the BASIC Stamp 2 and SX Microcontrollers book shows how to use Edit-Ink to create characters. Once a character is created, it may be saved with a file name. It may then be copied into the HP_Load_Font.BS2 program to be stored in the Serial Inkjet Printer's EEPROM.



Edit-Ink Font Creator

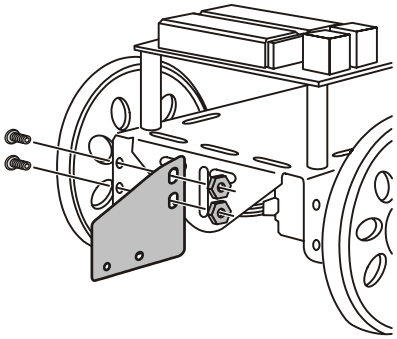
5.0 Boe-Bot Mobile Printing with the Inkjet Mounting Bracket

5.1 Parts Required

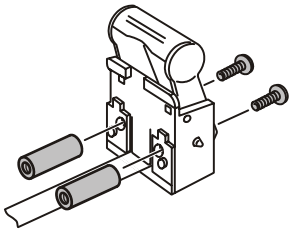
These parts are included in the 96 dpi Serial Inkjet Printer Development Kit, exclusively for mounting to the Boe-Bot Robot:

- (1) Boe-Bot Inkjet Mounting Bracket
- (6) 4/40 panhead machine screw, 1/4" long
- (2) 5/8" long 4/40 threaded standoffs
- (2) 4/40 nuts

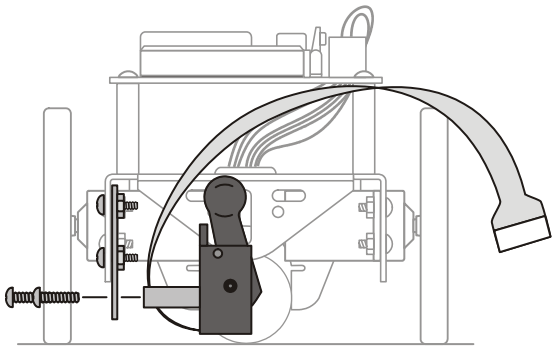
5.2 Installing the Inkjet Mounting Bracket on the Boe-Bot Robot



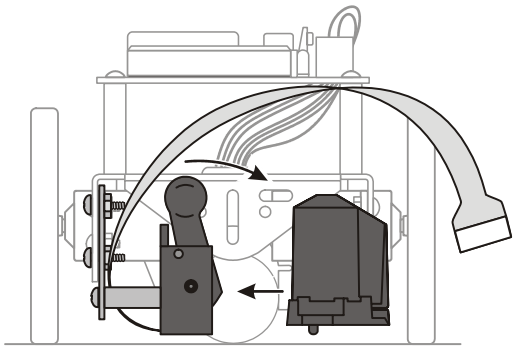
Step 1: Using (2) 4/40 1/8" long panhead machine screws and (2) 4/40 nuts, attach the mounting bracket to the inside front of the Boe-Bot chassis.



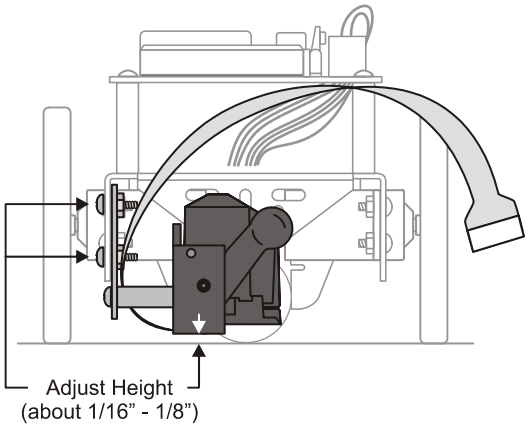
Step 2: Attach two 5/8" long 4/40 threaded standoffs to the Q7453A Inkjet Cartridge Holder with (2) 4/40 ¼" long panhead screws.



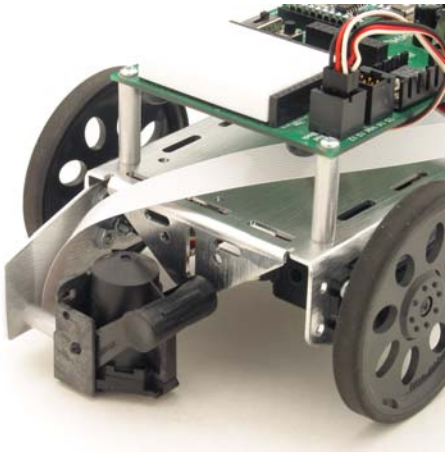
Step 3: Attach the inkjet cartridge holder and standoff assembly to the bracket using the remaining (2) 4/40 ¼" long machine screws.



Step 4: Remove the protective film from the 50604A Inkjet Cartridge. Insert the 51604A Inkjet Cartridge into the cartridge holder. Close the latch and the inkjet cartridge will be secured.



Step 5: Adjust the height of the inkjet nozzle to be between 1/16"-1/8" above the surface.



Step 6: Picture of Boe-Bot with the inkjet cartridge installed on the front of the chassis.

5.3 Mounting the Serial Inkjet Printer circuit board on the Boe-Bot Robot

There are a variety of ways to mount the Serial Inkjet Printer board to a Boe-Bot, so Parallax chose to leave this decision (and the hardware) up to the customer. There are at least three solutions for mounting the Serial Inkjet Printer to the Boe-Bot:

- Using double-sided foam sticky tape, mount it underneath the Board of Education.
- Using two of the Serial Inkjet Printer circuit board's mounting holes and 4/40 screws/standoffs, attach it to the front two slots in the Boe-Bot chassis.
- Press the Serial Inkjet Printer into the breadboard and wire the I/O (to S), 5 V (to 5 V) and ground (to Vss).

5.4 Programming the Boe-Bot Robot with Printer_Bot.BS2

The Printer_Bot.BS2 source code supports all Parallax BASIC Stamp 2 modules (BS2, BS2e, BS2sx, BS2p BS2pe, BS2px) with the use of #SELECT...#CASE directives for baud rates and timing. The program maintains Boe-Bot speed while sending messages to the Serial Inkjet Printer.

```
' =============================================================================
'
'   File....... Printer_Bot.BS2
'   Purpose....
'   E-mail..... support@parallax.com
'   Started....
'   Updated.... 25 OCT 2005
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================


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


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


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

```
LServo          PIN     12
RServo          PIN     13

Inkjet          PIN     15                      ' to HP driver


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

#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T9600       CON     84
  #CASE BS2SX, BS2P
    T9600       CON     240
  #CASE BS2PX
    T9600       CON     396
#ENDSELECT

SevenBit        CON     $2000
Inverted        CON     $4000
Open            CON     $8000
Baud            CON     Open + T9600

LStop           CON     750                     ' left servo speed control
LFwd            CON     650
LRev            CON     850

RStop           CON     750                     ' right servo speed control
RFwd            CON     850
RRev            CON     650

Prompt          CON     ">"                     ' prompt from HP driver
STX             CON     2                       ' start of text
ETX             CON     3                       ' end of text
Esc             CON     27                      ' escape


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

servoTimer      VAR     Byte                    ' for servo updates
moveTimer       VAR     Word                    ' movement timer
lSpeed          VAR     Word                    ' left servo speed
rSpeed          VAR     Word                    ' right servo speed
msgPntr         VAR     Word                    ' points to msg in EE
char            VAR     Byte                    ' character to print


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

Msg1            DATA    STX, "PARALLAX PRINTER-BOT", ETX
Msg2            DATA    STX, "SX/B RULES....", ETX


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

Reset:
 ' SERIN  Inkjet, Baud, 500, No_Inkjet, [WAIT(Prompt)]

  LOW LServo
  LOW RServo


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

Main:
```

```
  lSpeed = LFwd
  rSpeed = RFwd
  moveTimer = 100
  GOSUB Move_Robot

  msgPntr = Msg1
  GOSUB Print_String

  moveTimer = 100
  GOSUB Move_Robot

  SEROUT Inkjet, Baud, [Esc, "C", 15]

  moveTimer = 10
  GOSUB Move_Robot

  msgPntr = Msg2
  GOSUB Print_String

  moveTimer = 100
  GOSUB Move_Robot

Hold_Robot:                                 ' stop here
  lSpeed = LStop
  rSpeed = RStop
  DO
    PULSOUT LServo, lSpeed                  ' refresh servos
    PULSOUT RServo, rSpeed
    PAUSE 20
  LOOP

  END


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

No_Inkjet:

  ' handle no response from Inkjet here

  GOTO Reset

' -------------------------------------------------------------------------

' This subroutine should be called every millisecond for automatic
' update, or "servoTimer" preloaded with value when that is not possible

Update_Servos:
  servoTimer = servoTimer + 1               ' update servo timer
  IF (servoTimer >= 10) THEN                ' check timer
    PULSOUT LServo, lSpeed                  ' refresh servos
    PULSOUT RServo, rSpeed
    servoTimer = 0                          ' reset the timer
  ENDIF
  RETURN

' -------------------------------------------------------------------------

' This subroutine controls movement timing of the Boe-Bot.
' -- set "moveTimer" in 2 ms units (destroyed by subroutine)
' -- set "lSpeed" and "rSpeed" to control servos

Move_Robot:
  DO WHILE (moveTimer > 0)
    PAUSE 1                                 ' timer pad
```

```
    moveTimer = moveTimer - 1                   ' update movement timer
    GOSUB Update_Servos
  LOOP
  RETURN

' -----------------------------------------------------------------------------

' This subroutine sends a string to the print module while maintaining
'  the current Boe-Bot movement/speed.
' -- set "msgPntr" to address of message to print

Print_String:
  DO
    READ msgPntr, char                          ' get character from msg
    SEROUT Inkjet, Baud, [char]                 ' send to print driver
    msgPntr = msgPntr + 1                       ' point to next char
    GOSUB Update_Servos
  LOOP UNTIL (char = ETX)                       ' done at ETX
  RETURN

' -----------------------------------------------------------------------------
```

6.0 OEM Developer Guide

6.1 Product Development with this Design

OEMs may freely use the Parallax Serial Inkjet Printer design in their own applications. Parallax does not offer technical support this purpose, though we provide the complete Bill of Materials (BOM), schematic, and the SX28AC/SS firmware (HP51604x.SXB). OEMs are responsible for making their own PCB design unless other arrangements are made with Parallax to obtain our Serial Inkjet Printer PCB files.

6.2 Bill of Materials

The Parallax design uses parts readily available from common electronic suppliers including Digi-Key and Mouser. The Parallax internal BOM is shown below with our own stock codes. Parallax does not sample OEMs any of these components though they are all available for purchase.

Parallax Stock Code	Description	Qty
150-11020	Resistor, surface mount, 0603, 5%, 1/8 W, 1 kΩ	13
150-11021	Resistor, surface mount, 0603, 5%, 1/16 W, 10 kΩ	1
150-11045	Resistor, surface mount, 0603, 5%, 1/16 W, 220 Ω	1
150-14712	Resistor, surface mount, 0603, 5%, 1/16 W, 470 Ω	1
150-14720	Resistor, surface mount, 0603, 5%, 1/16 W, 4.7 kΩ	2
172-01330	Resistor, surface mount, 0603, 5%, 1/10 W, 13 kΩ	1
172-01520	Resistor, surface mount, 0603, 5%, 1/10 W, 1.5 kΩ	1
172-02240	Resistor, surface mount, 0603, 5%, 1/10 W, 220 kΩ	1
172-02700	Resistor, surface mount, 0603, 5%, 1/10 W, 27 Ω	2
200-11041	Capacitor, surface mount, ceramic, 0603, .1 µF, 16 V	3
213-01030	Capacitor, surface mount, ceramic, 0603, 20%, 25 V, 10 nF	1
213-02700	Capacitor, surface mount, ceramic, 0603, 5%, 50V, 27 pF	2
216-01060	Capacitor, tantalum, type A, 20%, 16 V, 10 µF	1
217-02250	Capacitor, surface mount, tantalum, type B, 20%, 16 V, 2.2 µF	1
217-04750	Capacitor, surface mount, tantalum, type B, 20%, 16 V, 4.7µF	1
250-02060	Resonator, 20 MHz, DIP	1
250-16050	Resonator, 6 MHz	1
275-10002	Ferrite 1 A 60 Ω 0603 surface mount	1
275-10822	Inductor, surface mount, 20%, 8.2 µH, 0.84 A	1
300-27948	Printed Circuit Board, Serial Inkjet Printer	1
350-10001	LED, surface mount, 0603, super red clear	1
350-10003	LED, SMT, 0603, green clear	12
400-00016	Switch, SPDT, slide, 3 position	1
451-00302	3 pin right-angle header	1
451-04002	40 pin single row right-angle header	1
452-00020	Connector, FPC/FFC, 16 position, 1MM, right angle	1
452-10006	Connector, USB mini-B right angle	1
500-10010	Darlington transistor array HV 18-SOIC	2
501-13001	Diode, Schottky, 30 V, 500 mA, SOD123	1
602-00017	24LC64 Serial EEPROM 64 K 2.5 V 8-SOIC	1
604-00031	FTDI 232 BM USB to serial converter	1
604-00044	DC/DC converter step-up SOT23-5	1
SX28AC/SS	SX28AC/SS	1

6.3 SX28AC/SS Driver Code Description (HP5160x.SXB)

6.3.A Overview

Parallax ships the Serial Inkjet Printer hardware with pre-programmed firmware (HP5160x.SXB) in the SX28AC/SS microcontroller. Developers are free to modify this firmware as desired using the Parallax SX-Key or SX Blitz. You may reprogram the SX28AC/SS using the 4-pin header, but to use the SX-Key's built-in debug facilities you will need to desolder the through-hole resonator.

The primary purpose of the driver program is to allow the user to print a string of characters using the embedded font set. Secondarily, the program accepts commands to change the timing between character columns (to accommodate different paper feed rates), and it allows the user to download a new character definition to the onboard EEPROM.

For educational purposes, the HP51604.SB driver also accepts commands (T and D) to configure print cartridge nozzle timing for "visualizing" the signals on the Serial Inkjet Printer's surface-mounted LEDs. If you send commands to demonstrate control using the LEDs, be certain that your inkjet cartridge is not connected to the Serial Inkjet Printer board. If it is, you will destroy the resistors and throw the inkjet printer cartridge. **If it is, you will destroy the Inkjet's nozzles.**

For simplicity and ease-of-maintenance, the driver program is written almost entirely in high-level SX/B (free BASIC compiler for the SX microcontroller). The single exception is a fixed multiplication routine which is very simple in SX assembly language. To minimize code space used, subroutines are used as shells for complex SX/B commands (e.g., **PAUSE, SEROUT**, etc.). Since SX/B is an inline compiler, placing these commands in a single location minimizes the amount of assembly code generated, as well as giving the programmer additional flexibility with these commands.

6.3.B Program Operation

The program takes advantage of the SX/B compiler's normal initialization sequence which clears all RAM variables to zero, and sets all I/O pins to an input state. After the internal initialization is complete the driver program begins at **`Start`**. This section enables the SX's internal pull-ups for all unused pins to reduce current consumption, and sets the nozzle control pins to outputs.

After essential setup is complete, the program waits for 250 milliseconds to allow the host (e.g., BASIC Stamp microcontroller) to perform its own initialization. After this delay, key driver control variables are initialized:

- **`nozDelay`**: Sets the nozzle on-time (default value is 5), in units specified by **`nozTiming`**.
- **`nozTiming`**: Sets the units used for the nozzle on-time; 0 (default) = microseconds, 1 = units of five milliseconds (for training and non-printing demo modes).
- **`colDelay`**: Sets the inter-column delay. This value is expressed in units of 0.1 milliseconds, so the default value of 20 provides a 2.0 millisecond delay between printed columns. The purpose of this variable is to allow for a constant character width based on the paper feed rate.
- **`printMode`**: Sets the method in which the column data is sent to the outputs: 0 (default) activates two outputs at one time, 1 (demo mode, or for POV displays) activates all 12 column outputs at the same time (**this mode must not be used with a cartridge installed**).

At this point the serial input pin (**Sin, RC.7**) from the USB connection is scanned. If this port is high then the USB port is connected and active, therefore the USB port will be used by the program for serial I/O. If the **Sin** pin is low, the USB port is not connected and the bi-directional serial pin (**SerIO, RA.0**) will be used instead. Note that the serial port selection is based only on the activity of the USB port at driver start-up, and cannot be changed during the program.

Once the active serial port has been determined the program drops into the core at **Main**. A five-millisecond delay allows the host to get ready for the prompt (">" character) that is sent on the active serial port. By placing this delay at the top of the loop redundant code is eliminated from the locations that direct the program back to **Main**. After the prompt is transmitted the program waits for serial input and places the received by into **char**.

6.3.C Receiving Text

The receipt of <**STX**> (ASCII 002) indicates the start of a string to print and causes the program to jump to **Get_Text** where subsequent bytes are received and buffered. The program will loop at **Get_Text**, placing each character received into the text buffer until 64 characters have been buffered or **<ETX>** (ASCII 003), the end-of-text marker, has been received.

6.3.D Printing Text

When the text buffer has been filled or the input terminated with **<ETX>** the program drops through to **Print_Text**. This section of the program does a great deal of work, and can be broken down into the following steps:

While characters are in the text buffer:

- Retrieve a character
- Create pointer to character map in EEPROM
- Copy character map (24 bytes) from EEPROM to map buffer (RAM)
- Loop through 12 columns for the character
    - ⇨ Loop through nozzle sequence to print column
    - ⇨ Delay (**colDelay**) between columns

Retrieving a character from the text buffer is trivialized with the **GET_BUF** subroutine (details below). A pointer to the starting address (in EEPROM) of the character map is created with **MAKE_ADDR** and then 24 bytes of character map data are moved from the 24LC64 EEPROM to RAM for faster access during the character print loop. Each character map resides on a 32-byte boundary in the EEPROM to allow the use of page mode read.

With the character map in RAM the program drops to **Print_Char** that handles the actual firing of the print-head nozzles. Each column requires two bytes, so two consecutive reads from the map buffer are required retrieve the column data. When **printMode** is set for using a cartridge a loop is used to read the nozzle firing sequence from a **DATA** table at location **MaskMaps**. These values are used to mask the raw column data in order to form the specified firing sequence for the nozzles. Though the print-head has 12 nozzles for a column, only two may be fired simultaneously. The mask table causes the nozzles to be fired in the order recommended by Hewlett Packard for the HP5160x print-head. After the current active nozzles have been selected with the mask, a call to **FIRE_NOZLS** takes care of firing, timing, and disabling the nozzles.

When all of the characters in the buffer have been printed the program returns to **Main** where it prints a new prompt for the host controller.

6.3.E Receiving and Processing Commands

Commands to the printer driver are prefaced with the **<Esc>** (ASCII 027) character. When **<Esc>** is received after the prompt the program is redirected to **Get_Cmd** where the program will wait for the command character. Valid commands are processed; any other character will be ignored and the program will be redirected back to **Main**.

On receipt of **<c>** or **<C>** (column delay) after **<Esc>** the program jumps to **Col_Delay** and waits for one additional byte that is used as the inter-column delay when printing. This value is expressed in units of 0.1 milliseconds allowing for delays of 0.1 to 25.5 milliseconds. If the user attempts to set the inter-column delay to zero the program will change this value to one.

On receipt of **<n>** or **<N>** (new character) after **<Esc>** the program jumps to **New_Char_Map**. The program expects the **<N>** command to be followed by the ASCII code of the character to re-map and then 24 bytes of map data. The diagram below shows the format of the character map data.

		C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
RC.5	N12	1.5	3.5	5.5	7.5	9.5	11.5	13.5	15.5	17.5	19.5	21.5	23.5
RC.4	N11	1.4	3.4	5.4	7.4	9.4	11.4	13.4	15.4	17.4	19.4	21.4	23.4
RC.3	N10	1.3	3.3	5.3	7.3	9.3	11.3	13.3	15.3	17.3	19.3	21.3	23.3
RC.2	N9	1.2	3.2	5.2	7.2	9.2	11.2	13.2	15.2	17.2	19.2	21.2	23.2
RC.1	N8	1.1	3.1	5.1	7.1	9.1	11.1	13.1	15.1	17.1	19.1	21.1	23.1
RC.0	N7	1.0	3.0	5.0	7.0	9.0	11.0	13.0	15.0	17.0	19.0	21.0	23.0
RB.5	N6	0.5	2.5	4.5	6.5	8.5	10.5	12.5	14.5	16.5	18.5	20.5	22.5
RB.4	N5	0.4	2.4	4.4	6.4	8.4	10.4	12.4	14.4	16.4	18.4	20.4	22.4
RB.3	N4	0.3	2.3	4.3	6.3	8.3	10.3	12.3	14.3	16.3	18.3	20.3	22.3
RB.2	N3	0.2	2.2	4.2	6.2	8.2	10.2	12.2	14.2	16.2	18.2	20.2	22.2
RB.1	N2	0.1	2.1	4.1	6.1	8.1	10.1	12.1	14.1	16.1	18.1	20.1	22.1
RB.0	N1	0.0	2.0	4.0	6.0	8.0	10.0	12.0	14.0	16.0	18.0	20.0	22.0

Example:
- "P" (ASCII 80)
- EE Start: 2560

A simple loop is used to receive the character map and hold it in a RAM buffer. When all bytes have been received the program transfers this data to the 24LC64 EEPROM with the code at **Transfer_2EE**.

Again, the character code is used to create the starting address in EE and then page mode is used to transfer the 24-bytes from the RAM buffer to the EEPROM.

On receipt of **<t>** or **<T>** (timing) after **<Esc>** the program jumps to **Set_Nozzle_Timing**. The program expects the **<T>** command to be followed by the timing value, specified in five millisecond units. The purpose of the **<T>** command is educational, allowing a student to view the nozzle output sequence (via the SIP LEDs). The **<T>** command must not be used with a cartridge connected, otherwise cartridge damage will occur. Standard nozzle timing (5 us) can be restored with **<Esc><T><0>**.

On receipt of **<d>** or **<D>** (display) after **<Esc>** the program jumps to **Set_Print_Mode**. The program expects the **<D>** command to be followed by 0 or 1; 0 indicates standard printing mode (only two nozzles at one time), 1 indicates LED display mode (up to 12 nozzles at one time). The **<D>** mode is for educational purposes and may not be used with a cartridge installed.

6.3.F Subroutine Code

What follows is a detailed description of each of the subroutines used in this program. The purpose of this discussion is to shed additional light on these code segments so that the end-user can modify the program, or use these subroutines in other applications.

WAIT_US
This subroutine allows for program delays in microseconds and, in fact, is a shell for the SX/B **PAUSEUS** instruction. The user is required to pass at least one byte to the subroutine, and may pass two with the second byte serving as a delay multiplier (for extended delays). If the second parameter is not provided the code sets this value to one, and then validates both values (for greater than zero) before attempting to use **PAUSEUS**.

WAIT_MS
This subroutine is identical to **WAIT_US** except that the timing units are expressed in milliseconds.

FIRE_NOZLS
This subroutine controls nozzle firing and expects two bytes: the lower nozzle mask and the upper nozzle mask. Each byte should have just one bit set, otherwise too many nozzles will be fired simultaneously. Since the upper bank of nozzles is connected to port RC of the SX28, and this port is connected to the USB serial connection, Bit6 of the upper nozzles value is set to prevent a false start bit from being generated when the values are move to the nozzle ports (RB and RC).

Actually, nozzle timing is controlled by the current print mode. In standard (cartridge) mode the selected nozzles are filed for five microseconds; in LED mode the nozzle timing is controlled by the value sent to the module with the **<Esc><D>** sequence (in units of five milliseconds). After nozzle firing, all are turned off and a 0.5 microsecond pause is inserted per HP specifications for the standard nozzle firing sequence.

RX_BYTE
This subroutine will receive one byte from the active serial port and return it to the caller.

TX_BYTE
This subroutine will transmit one byte on the active serial port.

UPPER_CASE
This subroutine is used to convert a lowercase letter (“a” to “z”) to its uppercase counterpart. Any non-alpha character is unaffected.

PUT_BUF

This subroutine is used to put a byte into the 64-byte print buffer (circular FIFO). Since the SX28 uses banked memory, this routine calculates the correct bank and offset for the current buffer pointer (**head**). This routine also modifies the variable **bufCnt** (buffer count) so that the program will prevent writing to the buffer when it is full.

GET_BUF

This subroutine is used to retrieve a byte from the 64-byte print buffer (circular FIFO). Since the SX28 uses banked memory, this routine calculates the correct bank and offset for the current buffer pointer (***tail***). This routine also modifies the variable ***bufCnt*** (buffer count) so that the program will prevent reading from the buffer when it is empty.

6.4 SX28AC/SS Driver Code Listing (HP51604x.SXB)

```
' ================================================================================
'
'   File...... HP5160x.SXB
'   Purpose... Simple Driver for HP 5160x Inkjet Cartridge
'   Author.... Parallax, Inc. -- Copyright (c) 2005, All Rights Reserved
'   E-mail.... support@parallax.com
'   Started...
'   Updated... 11 OCT 2005
'
' ================================================================================


' --------------------------------------------------------------------------------
' Program Description
' --------------------------------------------------------------------------------
'
' Simple driver for HP5160x Inkjet cartridges.  Communication is at 9600
' baud and is half-duplex.  The module will output a ">" when it is ready
' to accept data.
'
' To print text: <STX>Some text<ETX>
' -- Note that the text buffer is 32 bytes; printing will start as soon
'    as 32 characters are received and others will be ignored.
'
' Commands are prefaced with <Esc>
'
' <ESC><C><byte_value>
'   -- sets the inter-column delay
'   -- units are in 0.1 milliseconds
'   -- range is 1 to 255 (0.1 ms to 25.5 ms)
'
' <Esc><N><ASCII_Code><24 bytes>
'   -- use to download new 12x12 character map for "ASCII_Code"
'   -- refer to HP-CharMap.PDF for map definition
'
' ********************************************************
'  NOTE: You must remove the Inkjet cartridge before using
'        any of the following commands -- damage will occur
'        if the cartridge is left in place while using these
'        commands
' ********************************************************
'
' <Esc><T><byte_value>
'   -- sets the nozzle timing delay
'   -- units are in 5 milliseconds
'   -- range is 1 to 255 (5 ms to 1.25 s)
'   -- 0 resets to print speed delay (5 us)
'   -- DO NOT USE when cartridge is connected
'
' <Esc><D><mode>
'   -- use to set display mode (1) for POV displays
'   -- DO NOT USE when cartridge is connected
'
' <Esc><L><low_byte><high_byte>
'   -- displays 12-bit value on LEDs
'   -- DO NOT USE when cartridge is connected


' --------------------------------------------------------------------------------
' Device Settings
' --------------------------------------------------------------------------------
```

```
DEVICE          SX28, OSCXT2, TURBO, STACKX, OPTIONX
FREQ            20_000_000
ID              "HPINK0.3"


' -----------------------------------------------------------------------
' IO Pins
' -----------------------------------------------------------------------

SerIO           VAR     RA.0                    ' bi-directional serial
SIn             VAR     RC.7                    ' from USB
SOut            VAR     RC.6                    ' to USB (pull-up)

SDA             VAR     RA.2                    ' I2C connections
SCL             VAR     RA.3

NozLo           VAR     RB                      ' RB.0 - RB.5
NozHi           VAR     RC                      ' RC.0 - RC.5


' -----------------------------------------------------------------------
' Constants
' -----------------------------------------------------------------------

Stamp           CON     0                       ' serial connection
USB             CON     1
Baud            CON     "OT9600"                ' for bi-direction Stamp

BufMax          CON     64

SlaveWr         CON     $A0                     ' for 24LC64
SlaveRd         CON     $A1
Ack             CON     0
Nak             CON     1

Prompt          CON     ">"                     ' ready for input prompt

STX             CON     2                       ' start of text
ETX             CON     3                       ' end of text
Esc             CON     27

FireTm          CON     5                       ' in microseconds
DemoFire        CON     5                      ' in milliseconds (for LEDs)
FastFire        CON     0                       ' fire in us (printing)
SlowFire        CON     1                       ' fire in 5 ms (demo)
ColTiming       CON     10                      ' default column delay

PrintCart       CON     0                       ' cartridge mode
PrintLeds       CON     1


' -----------------------------------------------------------------------
' Variables
' -----------------------------------------------------------------------

flags           VAR     Byte
sPort           VAR     flags.0                 ' serial port, 1 = USB
ackNak          VAR     flags.1                 ' for I2C routines
nozTiming       VAR     flags.2                 ' standard or demo (x 5 ms)
printMode       VAR     flags.3                 ' standard or full-column

idx             VAR     Byte                    ' loop control
serByte         VAR     Byte                    ' serial in/out
bufIO           VAR     Byte                    ' buffer I/O byte
```

```
head            VAR     Byte                    ' for serial buffer
tail            VAR     Byte
bufCnt          VAR     Byte
bufA            VAR     Byte(16)                ' 64 byte serial buffer
bufB            VAR     Byte(16)
bufC            VAR     Byte(16)
bufD            VAR     Byte(16)

eeLo            VAR     Byte                    ' EE location
eeHi            VAR     Byte

mapPntr         VAR     Byte
charMapLo       VAR     Byte(12)                ' character nozzle map
charMapHi       VAR     Byte(12)

nozMask         VAR     Byte                    ' nozzle mask for sequence
tmpNozLo        VAR     Byte                    ' lower nozzle bits
tmpNozHi        VAR     Byte                    ' upper nozzle bits
nozDelay        VAR     Byte                    ' nozzle firing delay
colDelay        VAR     Byte                    ' column delay (x0.1 mS)

tmpIdx          VAR     Byte                    ' subroutine work vars
temp1           VAR     Byte
temp2           VAR     Byte
temp3           VAR     Byte


' =============================================================================
  PROGRAM Start
' =============================================================================

' "Standard" firing sequence
' -- mask is used for NozLo and NozHi to fire two nozzles at a time

MaskMaps:
  DATA  %010000                                 ' nozzles 5 & 11
  DATA  %000001                                 ' nozzles 1 & 7
  DATA  %000100                                 ' nozzles 3 & 9
  DATA  %100000                                 ' nozzles 6 & 12
  DATA  %000010                                 ' nozzles 2 & 8
  DATA  %001000                                 ' nozzles 4 & 10


' -----------------------------------------------------------------------------
' Subroutine Declarations
' -----------------------------------------------------------------------------

WAIT_US         SUB     1, 2                    ' delay in microseconds
WAIT_MS         SUB     1, 2                    ' delay in milliseconds
FIRE_NOZLS      SUB     2                       ' fire two nozzles
RX_BYTE         SUB                             ' serial rx
TX_BYTE         SUB     1                       ' serial tx
UPPER_CASE      SUB     1                       ' convert char to uppercase
PUT_BUF         SUB     1                       ' put byte in buffer
GET_BUF         SUB                             ' get byte from buffer
MAKE_ADDR       SUB     1                       ' make ee address from char
PUT_EE          SUB     3                       ' put byte in 24LC64
GET_EE          SUB     2                       ' get byte from 24LC64
I2C_START       SUB                             ' generate I2C Start
I2C_STOP        SUB                             ' generate I2C Stop
I2C_OUT         SUB     1                       ' write byte to SDA
I2C_IN          SUB     1                       ' read byte from SDA
PUT_MAP         SUB     1, 2                    ' put byte into char buf
GET_MAP         SUB     0, 1                    ' get byte from char buf
```

```
VAL_2LEDS       SUB     1, 2                    ' put value on LEDs


' -----------------------------------------------------------------------------
' Program Code
' -----------------------------------------------------------------------------

Start:
  PLP_A = %1100                                 ' pull-up unused pins
  PLP_B = %00111111
  PLP_C = %10111111                             ' pull-up SOut
  TRIS_B = %11000000                            ' nozzles are outputs
  TRIS_C = %11000000

  WAIT_MS 250                                   ' let host initialize
  nozDelay = FireTm                             ' set default timing
  nozTiming = FastFire                          ' use microseconds
  colDelay = ColTiming                          ' default inter-col delay
  printMode = PrintCart                         ' default to cartridge mode
  sPort = Sin                                   ' 0 = BASIC Stamp, 1 = USB

Main:
  WAIT_MS 5                                     ' let host be ready
  TX_BYTE Prompt                                ' send prompt
  serByte = RX_BYTE                             ' get byte from host
  IF serByte = STX THEN Get_Text                ' text input
  IF serByte = Esc THEN Get_Cmd                 ' command input
  GOTO Main                                     ' invalid entry

Get_Text:                                       ' accept text string
  bufCnt = 0                                    ' clear buffer
  DO WHILE bufCnt < BufMax
    bufIO = RX_BYTE                             ' get character
    IF bufIO = ETX THEN EXIT                    ' test for end
    PUT_BUF bufIO                               ' save in buffer
  LOOP

Print_Text:
  DO WHILE bufCnt > 0                           ' loop through buffer
    bufIO = GET_BUF                             ' get a character

EE_2MapBuf:
    MAKE_ADDR bufIO                             ' make starting address
    I2C_START                                   ' reset EE address pointer
    I2C_OUT SlaveWr                             ' send slave ID
    I2C_OUT eeHi                                ' send start address
    I2C_OUT eeLo
    I2C_START
    I2C_OUT SlaveRd                             ' start block read
    FOR idx = 0 TO 22                           ' read block
      bufIO = I2C_IN Ack
      PUT_MAP bufIO, idx                        ' move to map buffer
    NEXT
    bufIO = I2C_IN Nak                          ' final byte requires Nak
    PUT_MAP bufIO, 23
    I2C_STOP

Print_Char:
    mapPntr = 0                                 ' reset map pointer
    FOR idx = 0 TO 11                           ' loop through columns
      tmpNozLo = GET_MAP                        ' read nozzle data
      tmpNozHi = GET_MAP
      IF printMode = PrintCart THEN
        FOR tmpIdx = 0 TO 5                     ' loop through nozzle set
          READ MaskMaps + tmpIdx, nozMask       ' read mask
```

```
          temp1 = tmpNozLo & nozMask            ' mask active nozzle(s)
          temp2 = tmpNozHi & nozMask
          FIRE_NOZLS temp1, temp2               ' fire the nozzles
        NEXT
      ELSE
        FIRE_NOZLS tmpNozLo, tmpNozHi           ' fire LEDs in display mode
      ENDIF
      WAIT_US colDelay, 100                     ' delay between columns
    NEXT
  LOOP
  GOTO Main

Get_Cmd:
  serByte = RX_BYTE                             ' get command letter
  serByte = UPPER_CASE serByte                  ' convert to uppercase
  IF serByte = "C" THEN Set_Column_Delay
  IF serByte = "N" THEN DnLoad_New_Char_Map
  IF serByte = "T" THEN Set_Nozzle_Timing
  IF serByte = "D" THEN Set_Print_Mode
  IF serByte = "L" THEN Put_Word_On_Leds
  GOTO Main                             ' invalid command


Set_Column_Delay:
  colDelay = RX_BYTE                            ' get column delay
  IF colDelay = 0 THEN
    colDelay = ColTiming                        ' restore default
  ENDIF
  GOTO Main


DnLoad_New_Char_Map:
  serByte = RX_BYTE                             ' get character code
  FOR idx = 0 TO 23                             ' get 12 x 2 map
    bufIO = RX_BYTE                             ' get a byte
    PUT_MAP bufIO, idx                          ' save to map buffer
  NEXT

Transfer_2EE:                                   ' move char buffer to EE
  MAKE_ADDR serByte                             ' make starting address
  I2C_START                                     ' reset EE address pointer
  I2C_OUT SlaveWr                               ' send slave ID
  I2C_OUT eeHi                                  ' send start address
  I2C_OUT eeLo
  FOR idx = 0 TO 23
    bufIO = GET_MAP idx                         ' get byte from map
    I2C_OUT bufIo                               ' write to EE
  NEXT
  I2C_STOP
  DO                                            ' let write cycle finish
    I2C_START
    I2C_OUT SlaveWr
  LOOP UNTIL ackNak = Ack
  GOTO Main


Set_Nozzle_Timing:
  nozDelay = RX_BYTE                            ' get nozzle timing
  IF nozDelay = 0 THEN
    nozDelay = FireTm                           ' set default timing mode
    nozTiming = FastFire                        ' use microseconds
  ELSE
    nozTiming = SlowFire                        ' use x5 ms
  ENDIF
  GOTO Main
```

```
Set_Print_Mode:
  serByte = RX_BYTE                             ' get mode bit
  printMode = serByte.0                         ' set mode flag
  GOTO Main


Put_Word_On_Leds:
  temp1 = RX_BYTE
  temp2 = RX_BYTE
  VAL_2LEDS temp1, temp2
  GOTO Main


' -----------------------------------------------------------------------------
' Subroutine Code
' -----------------------------------------------------------------------------

' Use: WAIT_US microseconds {, multiplier}
' -- multiplier is optional

WAIT_US:
  temp1 = __PARAM1                              ' get microseconds
  IF __PARAMCNT = 1 THEN                        ' if no multiplier
    temp2 = 1                                   '   set to 1
  ELSE                                          ' else
    temp2 = __PARAM2                            '   get multiplier
  ENDIF
  IF temp1 > 0 THEN                             ' no delay if either 0
    IF temp2 > 0 THEN
      PAUSEUS temp1 * temp2                     ' do the delay
    ENDIF
  ENDIF
  RETURN

' -----------------------------------------------------------------------------

' Use: WAIT_MS milliseconds {, multiplier}
' -- multiplier is optional

WAIT_MS:
  temp1 = __PARAM1                              ' get milliseconds
  IF __PARAMCNT = 1 THEN                        ' if no multiplier
    temp2 = 1                                   '   set to 1
  ELSE                                          ' else
    temp2 = __PARAM2                            '   get multiplier
  ENDIF
  IF temp1 > 0 THEN                             ' no delay if either 0
    IF temp2 > 0 THEN
      PAUSE temp1 * temp2                       ' do the delay
    ENDIF
  ENDIF
  RETURN

' -----------------------------------------------------------------------------

' Use: FIRE_NOZLS lowNozzle, hiNozzle
' -- "lowNozzle" and "hiNozzle" should have only one bit set each

FIRE_NOZLS:
  temp1 = __PARAM1                              ' get active nozzles
  temp2 = __PARAM2
  temp2.6 = 1                                   ' prevent false start bit
  NozLo = temp1                                 ' move to outputs
```

```
  NozHi = temp2
  IF nozTiming = PrintCart THEN
    WAIT_US FireTm                              ' standard fire timing
  ELSE
    WAIT_MS nozDelay, DemoFire                  ' demo (user set) timing
  ENDIF
  NozLo = %00000000                             ' nozzles off
  NozHi = %01000000
  PAUSEUS 0.5                                   ' nozzle dead time
  RETURN

' -----------------------------------------------------------------------------

' Use: aByte = RX_BYTE

RX_BYTE:
  IF sPort = Stamp THEN
    SERIN SerIO, Baud, temp1                    ' rx from Stamp connection
  ELSE
    SERIN SIn, Baud, temp1                      ' rx from USB port
  ENDIF
  RETURN temp1

' -----------------------------------------------------------------------------

' Use: TX_BYTE aByte

TX_BYTE:
  temp1 = __PARAM1                              ' copy outgoing byte
  IF sPort = Stamp THEN
    SEROUT SerIO, Baud, temp1                   ' tx on Stamp connection
  ELSE
    SEROUT SOut, Baud, temp1                    ' tx on USB connection
  ENDIF
  RETURN

' -----------------------------------------------------------------------------

' Use: aChar = UPPER_CASE aChar
' -- converts lower-case letter to upper-case

UPPER_CASE:
  temp1 = __PARAM1                              ' get character
  IF temp1 >= "a" THEN                          ' "a".."z"?
    IF temp1 <= "z" THEN
      temp1 = temp1 - $20                       ' yes, make upper-case
    ENDIF
  ENDIF
  RETURN temp1

' -----------------------------------------------------------------------------

' Use: PUT_BUF aByte
' -- puts byte into 64-byte circular buffer
' -- ignored if no room

PUT_BUF:
  temp1 = __PARAM1                              ' make copy of byte to save
  IF bufCnt < BufMax THEN                       ' room in buffer?
    temp2 = head >> 4                           ' get bank pointer
    temp3 = head & %00001111                    ' get position in bank
    IF temp2 = %00 THEN
      bufA(temp3) = temp1
    ENDIF
    IF temp2 = %01 THEN
```

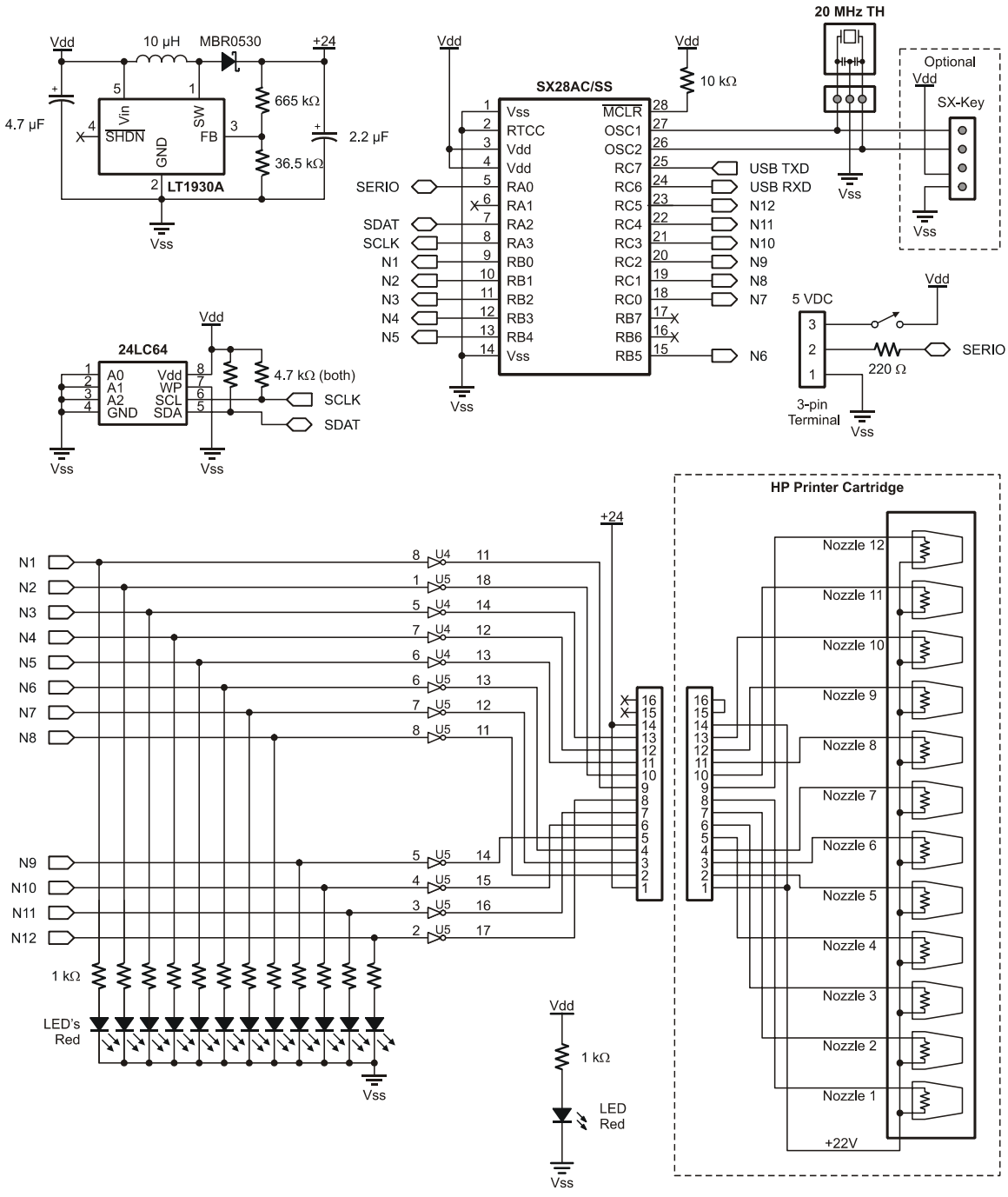
```
      bufB(temp3) = temp1
    ENDIF
    IF temp2 = %10 THEN
      bufC(temp3) = temp1
    ENDIF
    IF temp2 = %11 THEN
      bufD(temp3) = temp1
    ENDIF
    INC head                                    ' update head pointer
    head.6 = 0                                  ' wrap pointer
    INC bufCnt                                  ' update buffer count
  ENDIF
  RETURN

' -----------------------------------------------------------------------

' Use: aByte = GET_BUF
' -- gets byte from 64-byte circular buffer
' -- returns 0 if buffer empty

GET_BUF:
  IF bufCnt = 0 THEN                            ' anything in buffer
    temp1 = 0                                   ' no, clear return value
  ELSE
    temp2 = tail >> 4                           ' get bank pointer
    temp3 = tail & %00001111                    ' get position in bank
    IF temp2 = %00 THEN
      temp1 = bufA(temp3)
    ENDIF
    IF temp2 = %01 THEN
      temp1 = bufB(temp3)
    ENDIF
    IF temp2 = %10 THEN
      temp1 = bufC(temp3)
    ENDIF
    IF temp2 = %11 THEN
      temp1 = bufD(temp3)
    ENDIF
    INC tail                                    ' update tail pointer
    tail.6 = 0                                  ' wrap if needed
    DEC bufCnt                                  ' update buffer count
  ENDIF
  RETURN temp1

' -----------------------------------------------------------------------

' Use: MAKE_ADDR theChar
' -- multiples 'theChar' by 32

MAKE_ADDR:
  eeLo = 0                                      ' clear address
  eeHi = 0
  IF __PARAM1 > 0 THEN
    ASM
      MOV  eeLo, __PARAM1                       ' copy character value
      MOV  __PARAM1, #5                         ' prep to shift 5 times
      CLC
      RL   eeLo                                 ' shift the low byte
      RL   eeHi                                 ' shift high (use C)
      DJNZ __PARAM1, $-2
    ENDASM
  ENDIF
  RETURN

' -----------------------------------------------------------------------
```

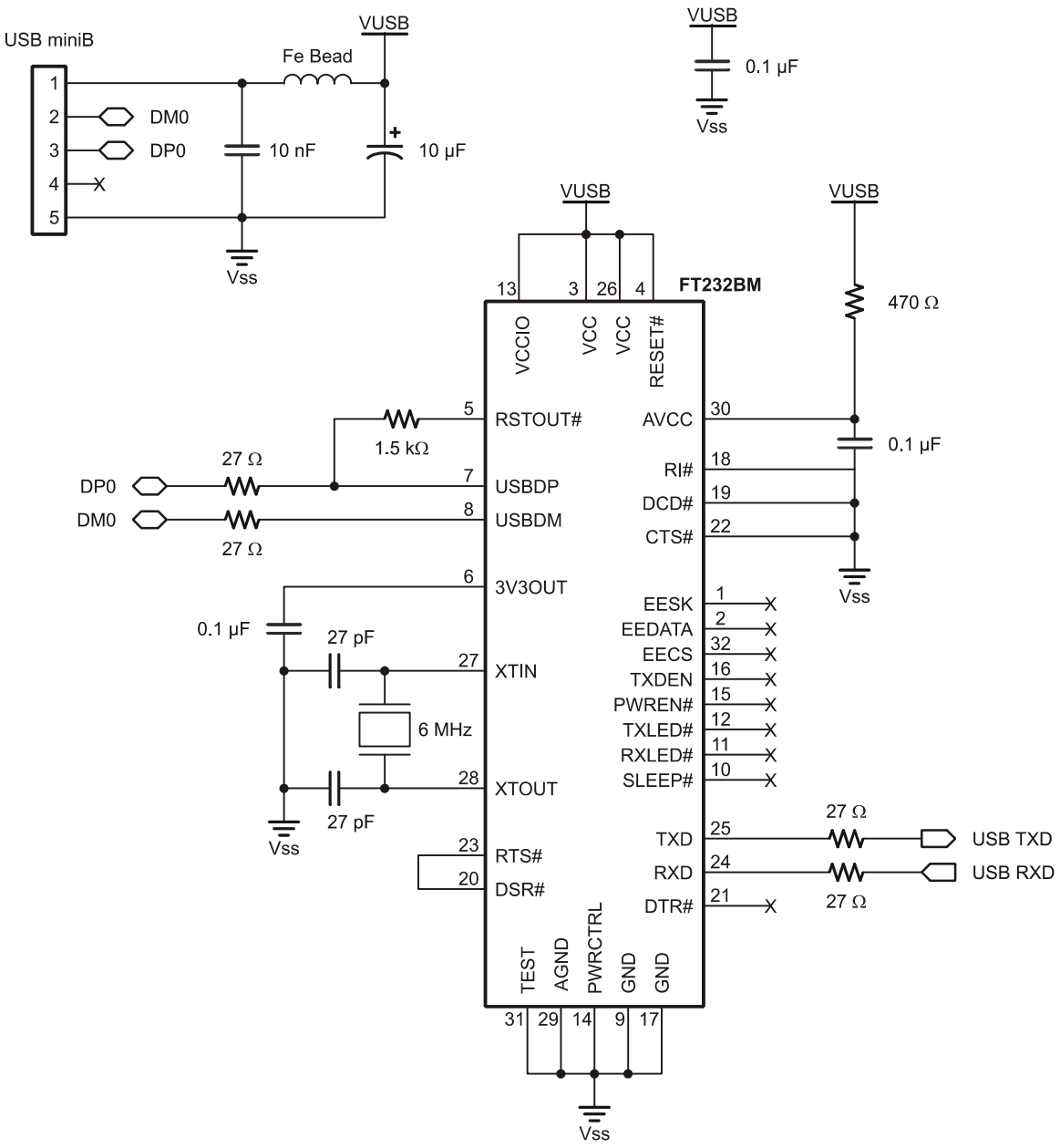
```
' Use: PUT_EE addrLo, addrHi, aByte
' -- writes 'aByte' to 24LC64 at location addrHi/addrLo

PUT_EE:
  temp1 = __PARAM1                              ' copy parameters
  temp2 = __PARAM2
  temp3 = __PARAM3

  I2C_START
  I2C_OUT SlaveWr                               ' send slave ID
  I2C_OUT temp2                                 ' send addrHi
  I2C_OUT temp1                                 ' send addrLo
  I2C_OUT temp3                                 ' send data byte
  I2C_STOP                                      ' finish
  DO                                            ' let write cycle finish
    I2C_START
    I2C_OUT SlaveWr
  LOOP UNTIL ackNak = Ack
  RETURN

' -----------------------------------------------------------------------------

' Use: aByte = GET_EE addrLo, addrHi
' -- reads 'aByte' from 24LC64 at location addrHi/addrLo

GET_EE:
  temp1 = __PARAM1                              ' copy parameters
  temp2 = __PARAM2

  I2C_START
  I2C_OUT SlaveWr                               ' send slave ID
  I2C_OUT temp2                                 ' send addrHi
  I2C_OUT temp1                                 ' send addrLo
  I2C_START
  I2C_OUT SlaveRd
  temp1 = I2C_IN Nak
  I2C_STOP
  RETURN temp1

' -----------------------------------------------------------------------------

' Use: I2C_START
' -- generates I2C start condition on SDA/SCL pins

I2C_START:
  I2CSTART SDA
  RETURN

' -----------------------------------------------------------------------------

' Use: I2C_STOP
' -- generates I2C stop condition on SDA/SCL pins

I2C_STOP:
  I2CSTOP SDA
  RETURN

' -----------------------------------------------------------------------------

' Use: I2C_OUT aByte
' -- writes 'aByte' to SDA pin
' -- affects global var "ackNak"

I2C_OUT:
```

```
  I2CSEND SDA, __PARAM1, ackNak
  RETURN

' -----------------------------------------------------------------------

' Use: aByte = I2C_IN AckBit
' -- reads 'aByte' from SDA pin
' -- address pointer must be preset before call

I2C_IN:
  ackNak = __PARAM1.0
  I2CRECV SDA, temp3, ackNak
  RETURN temp3

' -----------------------------------------------------------------------

' Use: PUT_MAP theByte {, pointer}
' -- puts byte into character map buffer
' -- uses and increments "mapPntr"

PUT_MAP:
  temp1 = __PARAM1                              ' get byte to save
  IF __PARAMCNT = 2 THEN                        ' new pointer
    mapPntr = __PARAM2                          ' yes, update
  ENDIF
  temp2 = mapPntr >> 1                          ' divide to create index
  IF mapPntr.0 = 0 THEN                         ' even byte?
    charMapLo(temp2) = temp1                    '  yes, get from low
  ELSE
    charMapHi(temp2) = temp1                    '  no, get from hi
  ENDIF
  INC mapPntr
  RETURN

' -----------------------------------------------------------------------

' Use: theByte = GET_MAP {pointer}
' -- gets byte from character map buffer
' -- uses and modifies "mapPntr"

GET_MAP:
  IF __PARAMCNT = 1 THEN                        ' new pointer?
    mapPntr = __PARAM1                          '  yes, update
  ENDIF
  temp1 = mapPntr >> 1                          ' divide for index
  IF mapPntr.0 = 0 THEN                         ' even byte?
    temp2 = charMapLo(temp1)                    '  yes, get from lo
  ELSE
    temp2 = charMapHi(temp1)                    '  no, get from hi
  ENDIF
  INC mapPntr
  RETURN temp2

' -----------------------------------------------------------------------

' For program development and trouble-shooting only
' -- do not use when cartridge is attached to driver board
' -- SOut bit is always set to 1 to prevent false str

' Use: VAL_2LEDS loByte {, hiByte}

VAL_2LEDS:
  temp1 = __PARAM1                              ' get low byte of value
  IF __PARAMCNT = 2 THEN                        ' hi byte passed?
    temp2 = __PARAM2                            '  yes
```

```
  temp2 = temp2 & $4F                           '   strip high bits
ELSE
  temp2 = $40                                   '  no, clear high LEDs
ENDIF
NozLo = temp1 & %00111111                       ' output low byte
NozHi = temp2 << 2                              ' output high nib
NozHi.0 = temp1.6                               ' put upper bits on RC
NozHi.1 = temp1.7
RETURN
```

6.5 Serial Inkjet Printer Board Schematic




USB miniB
VUSB
Fe Bead
1
2
3
4
5
DM0
DP0
10 nF
10 µF
Vss
VUSB
0.1 µF
Vss
VUSB
VUSB
FT232BM
13
3
26
4
VCCIO
VCC
VCC
RESET#
470 Ω
5
RSTOUT#
1.5 kΩ
27 Ω
DP0
7
USBDP
DM0
8
USBDM
27 Ω
6
3V3OUT
0.1 µF
27 pF
27
XTIN
6 MHz
28
XTOUT
27 pF
Vss
23
RTS#
20
DSR#
30
AVCC
0.1 µF
18
RI#
19
DCD#
22
CTS#
Vss
1
EESK
2
EEDATA
32
EECS
16
TXDEN
15
PWREN#
12
TXLED#
11
RXLED#
10
SLEEP#
27 Ω
25
TXD
USB TXD
24
RXD
USB RXD
27 Ω
21
DTR#
TEST
AGND
PWRCTRL
GND
GND
31
29
14
9
17
Vss

7.0 Inkjet Print Cartridge Material Safety Datasheets

To obtain the Inkjet Print Cartridge Material Safety Datasheets from Hewlett Packard, browse to:

http://www.hp.com/go/msds

Once you are at that page, look for the “SPS, or Specialty Printing Systems” link. Then browse for “51604AREVB” (black ink), “51605BREVB” (blue ink), or “51605RREVB” (red ink).