PARALLAX INC

Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

MoBoStamp-pe BS2pe Motherboard (#28300)

General Description

The MoBoStamp-pe provides a compact, professional-grade platform for BASIC Stamp applications. With the MoBoStamp-pe and the assortment of available daughterboards, you will be able to integrate and package one-off or multiple application systems with ease. The onboard AVR coprocessors permit the offloading of compute-intensive and background tasks from the BASIC Stamp, yielding a high performance level, while retaining the BASIC Stamp's ease of programming. They come preprogrammed for digital I/O, analog input, pulse-width-modulated output, and frequency counting. Each coprocessor interfaces to both the BASIC Stamp and one daughterboard each and may be reprogrammed by the user. Potential uses include background serial I/O, floating point processing, and background servo pulsing.

Before using your MoBoStamp-pe, please read and understand this entire document.

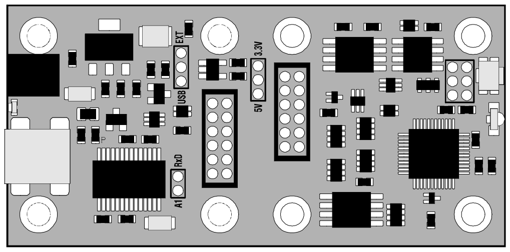
Features

- Compact size: designed to fit available packaging.
- BS2pe BASIC Stamp chip for high performance with low current consumption.
- 32K x 8 EEPROM for program and data storage.
- Built-in USB interface, capable of powering the entire board.
- Two sockets for plug-compatible daughterboards, enabling the easy integration of sensors and interface options.
- Two user-programmable Atmel AVR coprocessors, preprogrammed for digital I/O, PWM output, analog input, and frequency measurement.
- Multi-mode power sourcing: board can run from USB or external power.
- Multiple Vdd levels: 3.3V and 5V, jumper selectable.
- User-programmable multi-color LED for status indication.
- Programming header for future interface to an SX-KEY. This will allow programming the SX chip directly.

Application Ideas

- Robotics
- Remote Sensing
- Data Acquisition
- Industrial Control
- Desktop Appliances

What's Included



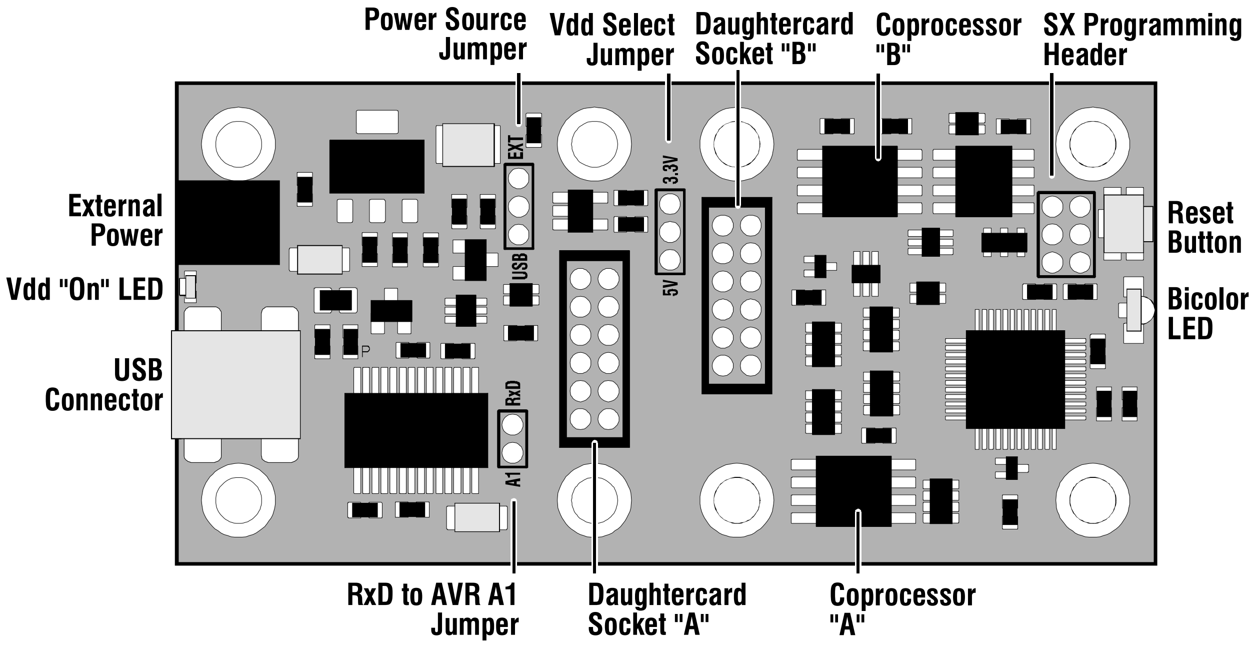
1 ea. MoBoStamp-pe Motherboard, preconfigured with:

4 ea. 3/16" dia. x 5/16" Threaded Standoffs

8 ea. 4-40 x 3/16" Panhead Machine Screws

3 ea. 2mm Jumpers

Interface Connections and Jumpers



External Power Connector

The external power connector enables the connection of a filtered, unregulated 6-9VDC power supply to power the motherboard. The plug and cable for this connector may be obtained separately from DigiKey, using the part numbers given under "Specifications" near the end of this document. Power applied here supplies both the onboard 5V regulator and the Vin connection to Daughterboard Socket "A".

Vdd "On" LED

This LED comes on when the board's Vdd is active. When the board is powered externally, this will occur whenever external power is applied. When the board is powered from the USB port, the LED will come on once the board has been connected to a USB port and the onboard USB chip enumerated by the PC.

For applications (such as light-sensing) where such a light source is undesirable, or for situations requiring minimum current draw, there is a "bow-tie" trace on the bottom of the board (under the external power connector) which can be cut to disable the LED.

USB Connector

The USB connector is the mini-B type. Compatible "A-to-mini-B" cables may be obtained at most local computer stores or from Parallax as part number 805-0006.

Power Source Jumper

The power source jumper can be placed to select either external or USB power. When external power is selected (upper position, marked **EXT**), the onboard regulator provides 5V to the rest of the board from Vin. Vin can come from either the external power connector or a daughterboard plugged into Daughterboard Socket "A". The USB chip, however, will continue to be powered by the host PC through the USB cable.

When USB power is selected (lower position, marked **USB**), the host PC provides 5V to the rest of the board from the USB cable. **Note:** *The USB "5V" supply can range anywhere from 4V to 5V.* When an accurate supply voltage is critical (e.g. for analog applications), it's best either to use an external supply or, if all circuitry can run from 3.3V, to select a Vdd of 3.3V.

Vdd Select Jumper

The Vdd select jumper allows the BASIC Stamp, the EEPROM, and both AVR coprocessors to run from either 3.3V or 5V. By selecting 5V (lower position, marked **5V**), Vdd is provided by the power source jumper, without further regulation.

By selecting 3.3V (upper position, marked **3.3V**), the onboard 3.3V regulator supplies these chips, as well as the "Vdd" pin on each daughterboard connector. In this position, the "5V" receptacle on each daughterboard connector continues to receive 5 volts.

RxD to AVR A1 Jumper

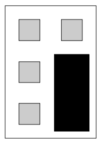
This jumper allows the incoming serial data from the USB interface to be connected to one of the AVR coprocessor pins. This will permit a properly programmed AVR to receive serial data in the background, relieving the BASIC Stamp of the task.

SX Programming Header

This six-pin header includes all the signals (plus RST) required to reprogram the SX with something other than the BS2pe interpreter. This function will be available via an adapter that the Parallax SX Key plugs into.

Important: Overwriting the BS2pe firmware is permanent. It is not possible to revert to BASIC Stamp operation once this has been done.

For normal operation, this header should always be configured with a jumper in the lower right-hand corner, thus:



Reset Button

The tiny reset button, when pressed, will reset the BASIC Stamp chip and both AVR coprocessors. It's designed more to be used in conjunction with a paperclip and a pinhole enclosure opening than with a finger; hence, its diminutive size. During debugging, resetting the board via the debug screen's DTR button will likely be easier.

Bicolor LED

This red/green LED is illuminated by pulling BASIC Stamp port P13 low for red or P14 low for green. Pulling both low simultaneously will show, at a distance, as amber.

Daughterboard Sockets

The daughterboard sockets receive the specially-designed daughterboards which, in conjunction with the onboard AVR coprocessors, perform sensing, interfacing, and other functions. They are labeled "A" and "B".

The daughterboards plug in parallel to the motherboard and will typically have connectors facing the ends of the motherboard. The daughterboard sockets are identical, with one exception: the "A" socket carries the Vin signal, while the "B" socket does not. This permits a daughterboard plugged into the "A" socket to power the whole system. That way, daughterboards can be designed, for example, with 24V-input DC-DC converters for industrial systems, or simple wall-transformer inputs for hobby systems. The Parallax PWR-I/O-DB card (part #28301) is an example of the latter.

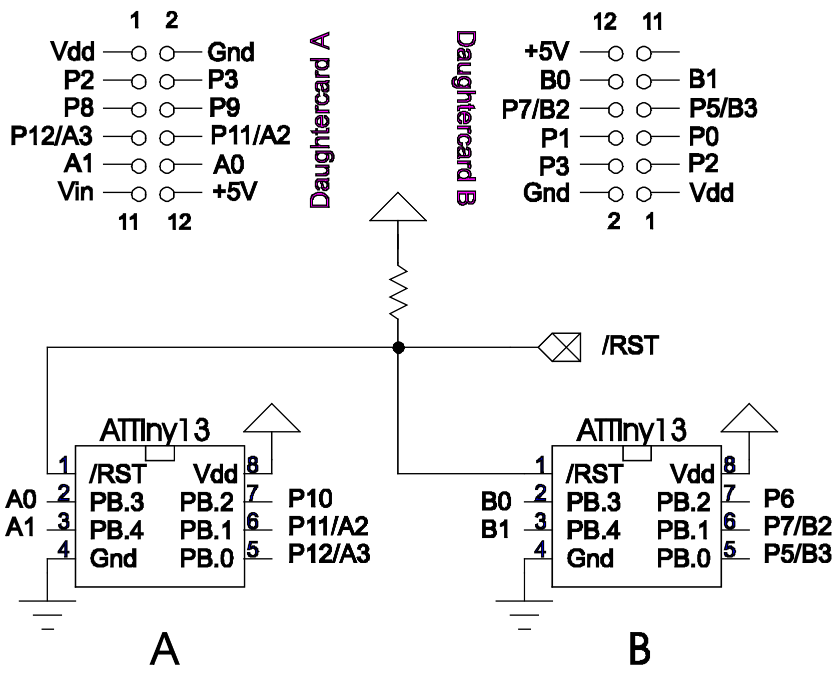
Important: Never insert jumper wires into the daughterboard sockets. Unlike 0.1" header sockets, these are not big enough to receive most jumper wires without damage.

The "A" daughterboard is sometimes referred to as the "Interface" daughterboard, since it is frequently used to interface to the outside world, with its connectors on the same end of the motherboard as the USB connector. Likewise, the "B" daughterboard is often called the "Sensor" daughterboard, since that's the socket where sensors are more likely to be plugged in. In reality, these names are arbitrary, since the capabilities of the two sockets are virtually identical.

In addition to the Vin line for socket "A", each daughterboard socket includes +5V, Vdd, Gnd, and eight signal lines. It is convenient to think of the signal lines in pairs. Their functions can be described as follows:

- **Common pulled-up pair:** These lines connect to both sockets and also to BASIC Stamp ports P2 and P3. They are pulled up to Vdd via 4.7K pull-ups. They can be used for daughterboard-to-daughterboard communication, without BASIC Stamp intervention. They can also be used with open collector drivers as poll inputs to the BASIC Stamp or as actual interrupts to a raw SX system.
- **Individual pulled-up pair:** These lines connect ports P8 and P9 to socket "A" and P0 and P1 to socket "B". They are pulled up to Vdd via 4.7K pull-ups. These ports can be used with the BS2pe firmware as an I2C interface. Daughterboards with I2C peripherals will use these lines as SDA and SCL.
- **AVR/BASIC Stamp shared pair:** Two lines to each daughterboard connect to both the BASIC Stamp and to the daughterboard's associated AVR coprocessor. These lines are not pulled up. Socket "A" receives P11/A2 and P12/A3. (A2 and A3 are pin designators for coprocessor "A", as shown below.) Socket "B" receives P7/B2 and P5/B3. A2, A3, B2, and B3, can be programmed as PWM outputs from the AVR chips.
- **AVR exclusive pair:** Two lines to each daughterboard connect exclusively to its AVR coprocessor. These are A0 and A1 for socket "A" and B0 and B1 for socket "B". These pins can be programmed as analog inputs.

The foregoing description is summarized in the following schematic fragment:



Coprocessors

The two AVR coprocessors interface to the daughterboards and to the BASIC Stamp as shown above. P6 and P10 are reserved for BASIC Stamp/AVR communication. These lines are pulled up to Vdd, so that open-collector comms (e.g. OWIN and OWOUT) may be utilized. The actual interface details will depend on the AVR firmware. See the GPIO-3 document for the firmware that comes preinstalled on the motherboard.

Getting Started

Before connecting your MoBoStamp to a PC, you should download and install the FTDI drivers necessary to operate the USB interface. These can be obtained, along with installation instructions, here:

http://www.parallax.com/html_pages/downloads/software/ftdi_drivers.asp

For programming the BASIC Stamp, you will need the Parallax BASIC Stamp Editor software, available for download here:

http://www.parallax.com/html_pages/downloads/software/software_basic_stamp.asp

Finally, to use the general-purpose I/O firmware pre-installed in the AVR coprocessors, download the GPIO user's guide here:

http://www.parallax.com/detail.asp?product_id=28300

Once these tasks have been completed, make sure your MoBoStamp is jumpered for USB power, and plug it into your PC. You should hear a "boo-beep" signal from the PC, indicating that the device has been recognized, and the green Vdd "on" LED should light. You are now ready to start programming.

Open the BASIC Stamp Editor, and key in the following program:

```
' {$STAMP BS2pe}
' {$PBASIC 2.5}

DO
  HIGH 14
  LOW 13
  PAUSE 500
  HIGH 13
  LOW 14
  PAUSE 500
LOOP
```

Hit Ctrl-R to load and run the program. You should see the bicolor LED alternately flash red and green. You've just run your first MoBoStamp program!

AVR Coprocessors

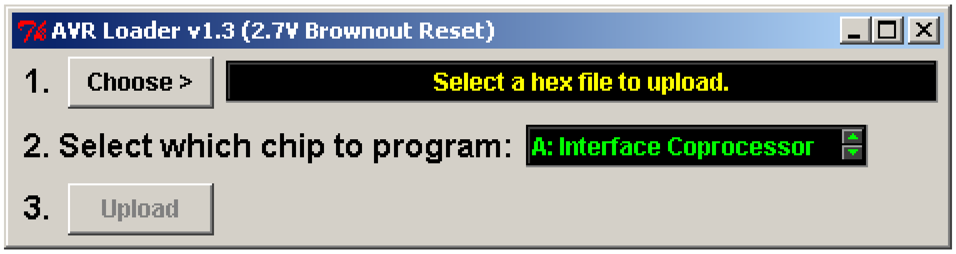
The coprocessors on the MoBoStamp board are Atmel ATTiny13s. They run on an internal 9.6 MHz RC clock and approach 9 MIPS (million instructions per second), depending on the actual firmware used. These controllers include 10-bit ADC (analog-to-digital conversion) capability, as well as built-in timers for PWM (pulse-width modulation) output, among other timing functions.

Each coprocessor connects to both the BASIC Stamp and one each of the daughterboards. This permits the BASIC Stamp's interaction with a daughterboard to be intermediated by the coprocessor, possibly in the background, thus relieving the BASIC Stamp from much of the work it would otherwise handle by itself. In addition, the built-in ADC and PWM functions permit the coprocessors to serve as analog peripherals for motherboards that require analog interfacing.

Different daughterboards will require, and be provided with, different AVR firmware. This firmware will come in the form of pre-assembled hex files, which can be uploaded to the coprocessors' onboard flash memory. This is done with the AVR Loader program, LoadAVR.exe, available here:

http://www.parallax.com/detail.asp?product_id=28300

To use the loader program, copy it to any folder on your hard drive, and double-click on it to start it up. You will first be asked to locate the BASIC Stamp Editor program, Stampw.exe. Once you've found and selected it, the following window will appear:


AVR Loader v1.3 (2.7V Brownout Reset)

1. Choose > Select a hex file to upload.

2. Select which chip to program: A: Interface Coprocessor

3. Upload

In step 1, click the "Choose" button to bring up a file dialog. Locate the desired AVR .hex file, and click "Open". In step 2, select which coprocessor to load: "A", "B", or both. In step 3, just click the button, and Stampw.exe will be invoked. Once the AVR loader program is uploaded to the BASIC Stamp, a debug window will pop up showing the AVR programming progress. If everything goes okay, the AVR chip(s) will be programmed with the new code.

Important: Once the AVR loading process has begun, do not interrupt it: let it run to completion.

In the event that you install an updated BASIC Stamp Editor after using LoadAVR at least once, you will want to make sure that the loader uses the new editor. This can be accomplished in two ways:

1. Make sure to uninstall the older BASIC Stamp Editor, or
2. Delete the file LoadAVR.ini that appears in the same directory as LoadAVR.exe.

Either action will force LoadAVR.exe to request the location of the new editor software the next time it is run.

It is possible, and even encouraged, for users to write their own firmware for the AVR controller. You can download a complete development system for the AVR processors free of charge from the Atmel website:

http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725

Coprocessor programs written for the ATTiny13, should use the internal 9.6Mhz RC clock. The other fuse settings configured automatically by the loader program are:

- Self-programming: Disabled
- Debug wire: Disabled
- Brown-out reset: Enabled, 2.7V
- Serial programming: Enabled
- Preserve EEPROM during chip erase: Enabled
- Watchdog timer always on: Disabled
- Divide clock by 8: Disabled
- Startup time: 4mSec after reset

These settings cannot be changed.

Startup Time

Due to the 4mSec startup time for the AVR coprocessors, a BASIC Stamp program may well begin before the AVRs come out of reset. For this reason, you should always put a `PAUSE 5` at the beginning of each program that uses one or more coprocessors. This will give the coprocessors time to start before being accessed.

Serial Data Echo

When the BASIC Stamp starts up, the Editor requires that it echo any characters sent to it by the PC. Hence, BASIC Stamps are configured in hardware to do just that. But this is not always a desirable thing during the execution of an application program. Therefore, a provision has been made to disable the echo. Port P4 is reserved for this purpose and, when pulled low (e.g. `LOW 4`), will disable the echo from RxD to TxD.

Port Summary

Port	Description and Typical Use	Daughterboard	4.7K pullup?
P0	General-purpose pin or SDA.	B	Yes
P1	General-purpose pin or SCL.	B	Yes
P2	General-purpose pin or poll input.	A & B	Yes
P3	General-purpose pin or poll input.	A & B	Yes
P4	Receive data echo enable. Pull low to disable echo.	(none)	Yes
P5	General-purpose I/O. Shared with AVR "B", port B3.	B	No
P6	Communication port for coprocessor "B".	(none)	Yes
P7	General-purpose I/O. Shared with AVR "B", port B2.	B	No
P8	General-purpose pin or SDA.	A	Yes
P9	General-purpose pin or SCL.	A	Yes
P10	Communication port for coprocessor "A".	(none)	Yes
P11	General-purpose I/O. Shared with AVR "A", port A2.	A	No
P12	General-purpose I/O. Shared with AVR "A", port A3.	A	No
P13	Red LED. Pull low to turn on.	(none)	No
P14	Green LED. Pull low to turn on.	(none)	No
P15	Reserved for AVR programming. **DO NOT USE!**	(none)	No

Specifications

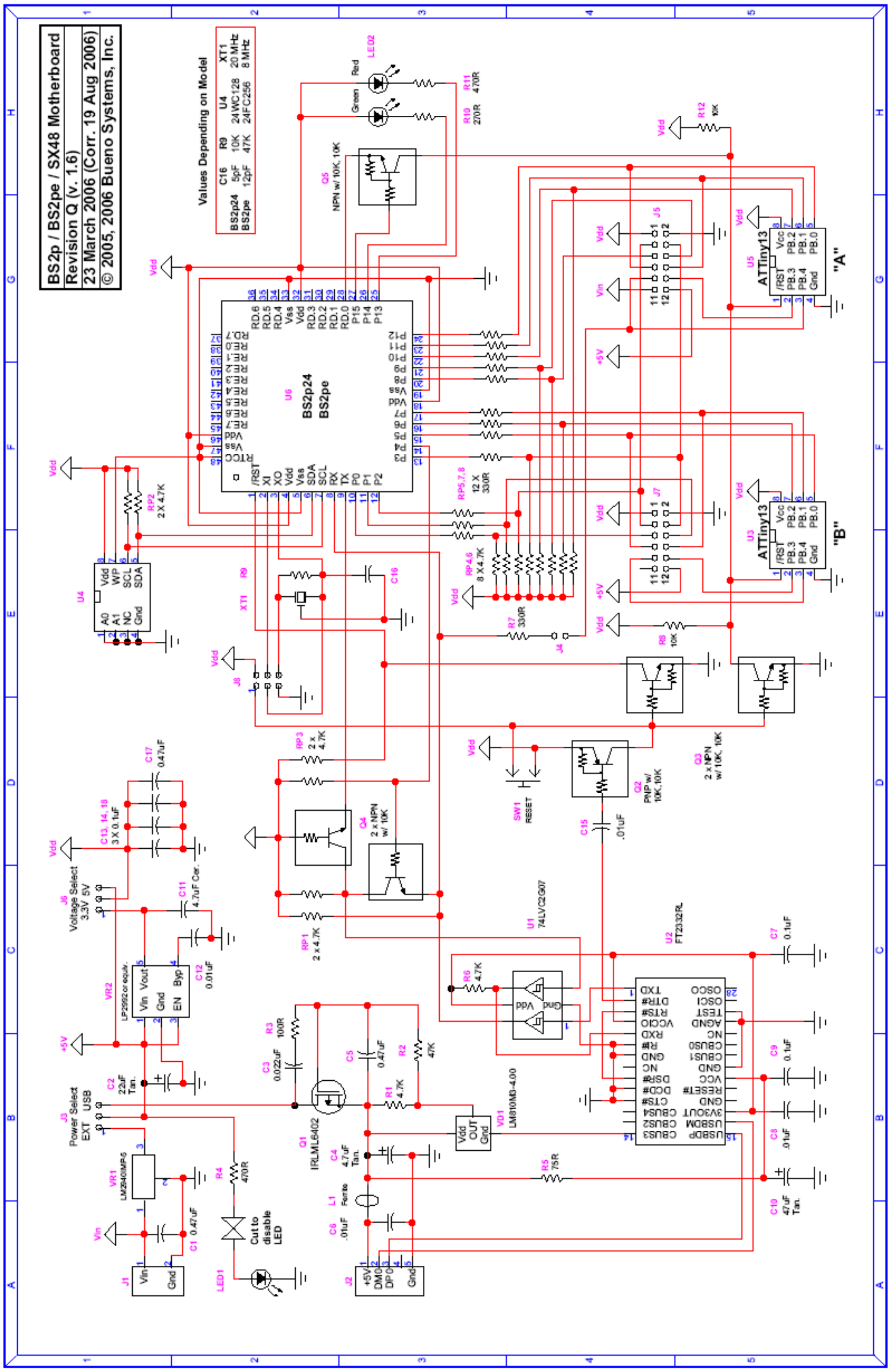
Circuit Board Size	2.75" x 1.35"
Nominal Daughterboard Size	1.35" x 1.35"
External Supply (optional)	6 – 9 VDC, 18mA minimum (3.3V, power LED disabled, no daughterboards)
USB Supply (from PC)	4 – 5 VDC
BASIC Stamp Clock	8MHz
Coprocessor Clocks	9.6MHz
USB Connector	Mini-B
Daughterboard Connectors (2mm)	Mating Header: Hirose DF11-12DP-2DSA(24)
External Power Connector (2mm)	Mating Receptacle: Shell: Hirose DF3-2EP-2C (DigiKey H4035-ND) Black Wire: DigiKey H2BXT-10112-B4-ND Red Wire: DigiKey H2BXT-10112-R4-ND

Sample Programs

Sample programs, written for various daughterboards, are available from the Parallax website:

http://www.parallax.com/detail.asp?product_id=28300

Schematic


BS2p / BS2pe / SX48 Motherboard
Revision Q (v. 1.6)
23 March 2006 (Corr. 19 Aug 2006)
© 2005, 2006 Bueno Systems, Inc.
Values Depending on Model
C16 R9 U4 XT1
BS2p24 5pF 10K 24WC128 20 MHz
BS2pe 12pF 47K 24FC256 8 MHz
U6 BS2p24 BS2pe
U5 ATTiny13 "A"
U3 ATTiny13 "B"
U4
U2 FT232RL
U1 74LVC2G07
VR1 LM2940IMP-5
VR2 LP2992 or equiv.
Q1 IRLML6402
Q2 PNP w/ 10K,10K
Q3 2 x NPN w/ 10K, 10K
Q4 2 x NPN w/ 10K
Q5 NPN w/ 10K, 10K
VD1 LM810M3-4.00
SW1 RESET
J1
J2
J3 Power Select EXT USB
J4
J5
J6 Voltage Select 3.3V 5V
J7
J8
LED1
LED2
Cut to disable LED
L1 Ferrite
RP1 2 x 4.7K
RP2 2 X 4.7K
RP3 2 x 4.7K
RP4,6 8 X 4.7K
RP5,7,8 12 X 330R
R1 4.7K
R2 47K
R3 100R
R4 470R
R5 75R
R6 4.7K
R7 330R
R8 10K
R10 270R
R11 470R
R12 10K
C1 0.47uF
C2 22uF Tan.
C3 0.022uF
C4 4.7uF Tan.
C5 0.47uF
C6 .01uF
C7 0.1uF
C8 .01uF
C9 0.1uF
C10 47uF Tan.
C11 4.7uF Cer.
C12 0.01uF
C13, 14, 18 3 X 0.1uF
C15 .01uF
C17 0.47uF
Vdd
Vin
+5V
Green
Red

AVR Firmware: GPIO, Version 3

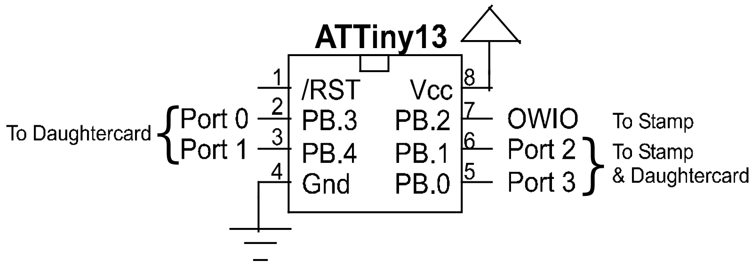
Introduction

This document describes the use of AVR firmware that is used in conjunction with the BS2p/BS2pe BASIC Stamp motherboard. This firmware can be uploaded to either or both of the motherboard's two AVR coprocessors as file **GPIO3.hex**. Once loaded, the coprocessor is capable of communicating with the Stamp using PBASIC's **OWOUT** and **OWIN** commands. This communication takes place on the AVR's OWIO pin (see illustration below) to read data from, and write data to, the other four I/O pins. These four pins, two of which are shared with the Stamp, also connect to an attached daughterboard. By utilizing the capabilities of the AVR coprocessor, interaction is afforded with the daughterboard in ways that augment the Stamp's capabilities and offload from it the more mundane and processor-intensive tasks.

The functions available with this firmware include:

- Digital input on all four ports.
- Digital output on all four ports.
- 1 MHz frequency counter on all four ports.
- Up to 37.5 KHz PWM output on two ports.
- 10-bit analog-to-digital input on two ports.
- Analog comparator input on three ports, comparing with either the fourth port or a 1.1- volt reference.
- 32 bytes of RAM, which can be written and read.

In addition, the type and frequency of the PWM outputs can be configured on the fly.
The AVR (Atmel ATTiny13) pinout is shown below:



OWIO connects to the Stamp and has a pull-up resistor to Vdd. Communication is bi-directional via a protocol using open-collector signaling. Ports 2 and 3 also connect to the Stamp without external pull-ups, as well as to an attached daughtercard. Ports 0 and 1 connect to an attached daughtercard only. Ports 0 and 1 are capable of analog input. Ports 2 and 3 are capable of unattended PWM output. Ports 0, 1, and 2 can be used as positive comparator inputs, comparing with either Port 3 or an internal 1.1-volt bandgap reference. All ports can be used for digital I/O and frequency counting.

Command Protocol

Communication with the AVR is via one-byte commands sent using the Stamp's **OWOUT** statement, possibly followed by additional data bytes or by reads using **OWIN.** An example command might be:

```
    OWOUT Owio, 0, [$25]
```

This statement will write a digital "1" to port 2, causing it to go high. Some commands are used to read from the AVR. An example would be:

```
    B VAR Bit
    OWOUT Owio, 0, [$10]
    OWIN Owio, 4, [B]
```

This reads a one-bit value (high or low) from Port 1 into variable B.

Any reset (low pulse lasting longer than 160 microseconds) sent via the OWIO pin will reset the AVR's protocol state machine, interrupting any transaction in progress (with one exception, described under "Counter Input" below). It will not affect the states of any of the pins, however. A reset can be incorporated into an **OWOUT** statement by choosing the second argument appropriately. For example:

```
    OWOUT Owio, 1, [$14]
```

resets the protocol engine before writing a zero to Port 1. A reset is usually a good idea in the first transaction of a Stamp program. It should also be considered when communication with the AVR is infrequent or done in electrically noisy environments. A too-liberal use of resets, however, can not only slow a program down, but it can also mask program errors associated with AVR communications.

Finally, the AVR comes out of a hardware reset more slowly than the Stamp does. So don't start talking to it right away in your Stamp program. To make sure the AVR is ready for communicating, put a 5ms PAUSE at the beginning of your Stamp program:

```
    PAUSE 5
```

This will prevent out-of-the gate misfires.

Firmware Identification

To identify the firmware currently extant in the AVR (assuming it uses the same protocol), send the following command ($DD):

1	**1**	**0**	**1**	**1**	**1**	**0**	**1**

The AVR will respond with three bytes of data. The first two are characters representing the name of the firmware. The third is a version number. This information can be used by a Stamp program to make sure the correct AVR firmware is loaded. The following program snippet, where **I**, **J** and **K** are byte variables, prints out this information:

```
    OWOUT Owio, 0, [$DD]
    OWIN Owio, 0, [I, J, K]
    DEBUG " Device: ", I, J, ", Version: ", DEC K
```

For this firmware, the following line is printed:

```
Device: GP, Version: 3
```

Outputs

Outputting signals from the AVR is a one-step process. It involves sending the AVR a command telling it what's to be done. Some commands require an additional data byte after the command byte. If either or both of Ports 2 and 3 are used as outputs, their corresponding shared Stamp pins must be set to inputs to avoid bus conflicts.

Digital Output

The command for causing a port to drive high or low is as follows:

0	**0**	**Addr**	**0**	**1**	**0**	**N**

Addr is the two-bit address (00 = 0, 01 = 1, 10 = 2, 11 = 3) of the destination port. **N** is the bit value (0 or 1) to write. A zero drives the pin low; a one drives it high.
The following program segment, sets all four pins low:

```
    FOR I = 0 TO 3
      OWOUT Owio, 0, [I << 4 + $04]
    NEXT
```

PWM Output

Ports 2 and 3 can be configured as pulse-width modulated outputs having 256 possible duty cycles from 0% to 99.6% (or 100%: see below). A PWM output can be low-pass filtered and used to form a rudimentary digital-to-analog converter (DAC). Or, it can be used with an appropriate driver to modulate inductive loads directly, without filtering. The maximum PWM frequency attainable with the AVR is 37.5 KHz, which can be filtered with fairly small-valued components. The PWM output command has the following format:

0	**0**	**Addr**	**0**	**1**	**1**	**Inv**

Data byte: $00 - $FF (0 – 255)

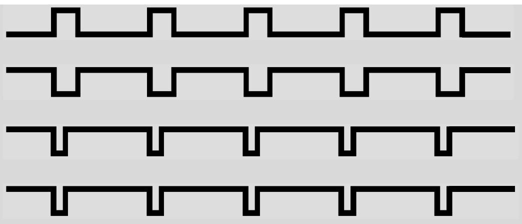
Addr is the two-bit address (10 = 2, 11 = 3) of the destination port. Writes to Ports 0 and 1 are ignored, since they do not support PWM output. The **Inv** bit selects the sense of the PWM: A zero here selects positive-going pulses; a one selects negative-going pulses. The data byte selects the desired relative pulse width. Example commands and the pulse trains they produce are shown below:

```
    OWOUT Owio, 0, [$26, 64]

    OWOUT Owio, 0, [$27, 64]

    OWOUT Owio, 0, [$37, 32]

    OWOUT Owio, 0, [$36, 224]
```



The first two lines write to Port 2; the second two, to Port 3. Lines 1 and 4 create positive pulses; lines 2 and 3, negative pulses. Notice that a positive pulse with a duty cycle of 224/256 = 87.5% looks a lot like a negative pulse with a duty cycle of 32/256 = 12.5%.

It is also possible to select from several fixed overall frequencies, using the frequency command:

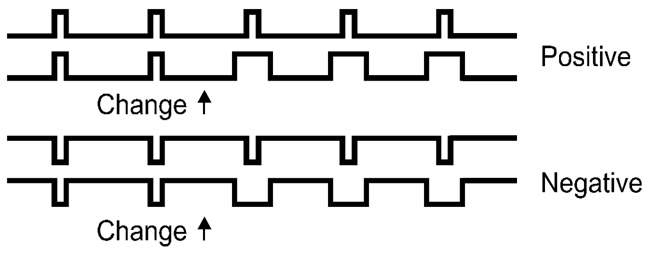
1	**1**	**1**	**1**	**Freq**

Freq is a four-bit number ranging from 0 through 9 that specifies the actual PWM frequency. The selected frequency applies to both Ports 2 and 3 equally. It is not possible to specify a different frequency for each port. The **Freq** values and their associated nominal frequencies are:

Freq	Frequency	PBASIC Code
0	37,500.00 Hz	`OWOUT Owio, 0, [$F0]`
1	18,823.53 Hz	`OWOUT Owio, 0, [$F1]`
2	4,687.50 Hz	`OWOUT Owio, 0, [$F2]`
3	2352.94 Hz	`OWOUT Owio, 0, [$F3]`
4	585.94 Hz	`OWOUT Owio, 0, [$F4]`
5	294.12 Hz	`OWOUT Owio, 0, [$F5]`
6	146.48 Hz	`OWOUT Owio, 0, [$F6]`
7	73.52 Hz	`OWOUT Owio, 0, [$F7]`
8	36.62 Hz	`OWOUT Owio, 0, [$F8]`
9	18.38 Hz	`OWOUT Owio, 0, [$F9]`

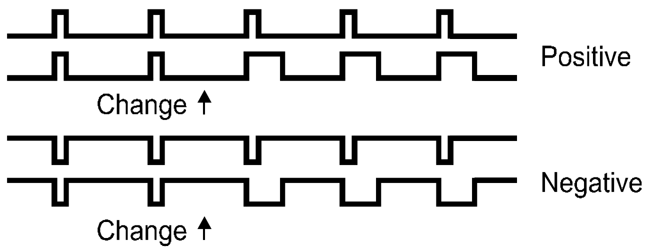
Remember, these are nominal frequency values. They are derived from the AVR's internal RC clock, so they can vary as much as ±10%. Frequency values larger than 9 are ignored, and no changes are made if a larger value is attempted.

Finally, the odd-numbered frequency values cause the PWM to behave somewhat differently than the even-numbered ones do. The odd-numbered values designate "phase correct" pulses. This is to say that when the duty cycle changes, the change is made symmetrically about the center of the pulse, like this:



This type of PWM is often used for DC motor speed controllers or other inductive load drivers, where phase shifts may be undesirable.

In the even-numbered frequencies, the leading edges of the pulses line up, regardless of any changes made to the duty cycle, thus:



This type of PWM is completely adequate for low-pass filtering to obtain a voltage output.

The other difference between the two PWM modes is the base length of each pulse. In the even-numbered modes, the base length is 256 units. So duty cycles can range from 0/256 to 255/256, positive or negative. The odd numbered modes have a base length of 255 units. So duty cycles can range from 0/255 to 255/255, positive or negative.

Inputs

Reading signals from the AVR's pins is a two-step process. First the appropriate command is sent via **OWOUT**. Then **OWIN** is used to get the actual data. If you are reading signals from a daughterboard on Ports 2 and/or 3, be the corresponding shared Stamp pin(s) are configured as input(s) to avoid bus conflicts.

Digital Input

The following command format is used for reading digital pin values:

0	**0**	**Addr**	**0**	**0**	**0**	**0**

Addr is the two-bit address (00 = 0, 01 = 1, 10 = 2, 11 = 3) of the source port. Once issued, **OWIN** must be used in bit mode to read the state of the chosen port pin.
The following program segment reads the state of pin 3:

```
    State VAR Bit
    OWOUT Owio, 0, [$30]
    OWIN Owio, 4, [State]
```

In some cases it may be desired just to tri-state the port pin without reading it. This can be accomplished by sending a reset after the command byte, as follows:

```
    OWOUT Owio, 2, [$20]
```

All this does is to make Port 2 float as an input pin.

Counter Input

By invoking the counter input command, pulses arriving on any selected AVR port pin may be counted over a programmed time interval. The AVR can count pulses arriving at up to a 1MHz rate. Minimum high and low times for these pulses are 500ns each. The counter input command has the following format:

0	**0**	**Addr**	**0**	**0**	**H**	**L**

Addr is the two-bit address (00 = 0, 01 = 1, 10 = 2, 11 = 3) of the source port. **H** and **L** determine which bytes (high-order and/or low-order) of the final count to return. At least one of these bits must be a one. If only the low-order byte is selected and if the actual count is greater than 255, 255 is returned. If only the high byte is selected, and the actual count is greater than 65535, 255 is returned. If both bytes are selected, and the actual count is greater than 65535, 65535 is returned.

This command starts the counting process immediately. Counting continues until the next falling edge from the Stamp on OWIO. This will typically be a reset pulse. But unlike all other resets, this one does not reset the protocol engine: it merely stops counting and sets up to send the count byte(s) back to the Stamp via its **OWIN** command. It's most convenient just to use an **OWIN** prefaced by a reset pulse (*i.e*. **OWIN Owio, 1, ...**).

The following example counts pulses on Port 2 for a duration of 50ms and computes the actual frequency, which then gets printed out:

```
    Result VAR Word
    OWOUT Owio, 0, [$23]
    PAUSE 50
    OWIN Owio, 1, [Result.HIGHBYTE, Result.LOWBYTE]
    DEBUG "Frequency: ", DEC Result / 50
    DEBUG ".", DEC2 Result // 50 * 2, " KHz", CR
```

Remember that the actual number of bytes read by **OWIN** *must* agree with the number requested by the **H** and **L** parameters.

Comparator Input

An analog voltage on any of Ports 0, 1 or 2, may be compared with either an internal 1.1-volt reference, or with an analog voltage on Port 3. The following command performs the comparison:

0	**0**	**Addr**	**1**	**Ref**	**0**	**0**

Addr is the two-bit address (00 = 0, 01 = 1, 10 = 2) of the input port. (If Port 3 is chosen, the result of the comparison will always be zero.) **Ref** determines what to compare the pin to. A zero selects Port 3; a one, the internal 1.1-volt reference.

After this command is issued, a 1-bit **OWIN** should be used to read the result of the comparison. A "one" means the **Addr**essed port pin was higher than the reference; a "zero", equal or lower. In the following example, Port 1 is compared with Port 3. The result is read into the Bit variable **C**:

```
    C VAR Bit
    OWOUT Owio, 0, [$18]
    OWIN Owio, 4, [C]
```

Analog Input

An analog voltage on either Port 0 or Port 1 may be converted to a digital value by the AVR's 10-bit A-to-D converter and read by the Stamp. The command for doing so is as follows:

0	**0**	**Addr**	**1**	**Ref**	**H**	**L**

Addr is the two-bit address (00 = 0, 01 = 1) of the analog input port. **Ref** determines the voltage reference source (highest voltage) for the A-to-D conversion. A zero selects Vdd; a one, the internal 1.1-

volt reference. **H** and **L** determine how to return the digitized value. At least one of these bits must be a one. The following table defines how **H** and **L** are interpreted:

H	**L**	**Actual 10-bit value (ADC)**	**Returned Bytes(s)**
0	1	0 – 255	ADC[7..0]
		256 - 1023	255
1	0	0 - 1023	ADC[9..2] (*i.e.* ADC / 4)
1	1	0 - 1023	ADC[9..8], ADC[7..0]

As soon as the command is issued, the desired A-to-D conversion begins. Because this conversion may not be finished before the next read, it is necessary to poll the AVR by reading a single bit before the conversion result can be read. If this bit is a "one", the AVR is still busy. If it's a "zero", the conversion is complete, and the result byte(s) may be read.

The following example reads the voltage on Port 0, using Vdd as a reference, and returns the entire 10-bit value:

```
    Voltage VAR Word
    Busy    VAR Bit
    OWOUT Owio, 0, [%00001011]
    DO : OWIN Owio, 4, [Busy] : LOOP WHILE Busy
    OWIN Owio, 0, [Voltage.HIGHGYTE, Voltage.LOWBYTE]
```

If you use Vdd as the voltage reference, be sure you have a reliable voltage source. If the motherboard is jumpered to use the USB's 5-volt source as Vdd, this voltage could be as low as 4.2 volts. Under these circumstances, it would be better either to use 3.3V for Vdd or to lower the analog input voltage with a resistor divider and use the internal 1.1-volt reference. An alternative would be to add a 2.5V bandgap reference to the other analog input and read both the unknown voltage and the 2.5-volt reference voltage, as compared to Vdd. Then you can compute the unknown voltage in such a way that the Vdd terms cancel.

Important Note: No voltage input on an AVR analog input pin may exceed Vdd by more than 0.5 volts, regardless of which reference you use. Use a resistor divider, if necessary, to keep such voltages in range.

RAM Writes and Reads

The AVR has 32 spare bytes of RAM available. The values stored here are zeroed on a hardware reset, but persist across multiple protocol resets. This section describes how to access the RAM from PBASIC.

RAM Writes

To write a single byte to RAM, use the **E**nter command ($En):

1	**1**	**1**	**0**	**0**	**Addr**

The number **Addr** can range from **0** to **7** and represents a shorthand notation for writing a *single byte* to the address represented by **Addr**. The command should be followed immediately by the single byte value to be written.

The following example writes the value **77** at address **2**:

```
    OWOUT Owio, 0, [$E2, 77]
```

To write multiple bytes or to write to addresses beyond **7**, use the **E**nter **A**ddress command ($EA), followed by the beginning address to write to:

1	**1**	**1**	**0**	**1**	**0**	**1**	**0**

0	**0**	**0**	**Addr**

The address value, **Addr**, can range from **0** through **31** (**$1F**). This address byte should be followed by one or more bytes of data, which are stored sequentially, beginning with the chosen address. This continues until a reset is received. Data received for addresses beyond **$1F** are ignored.

The following example writes the values **123** and **64**, beginning at address **$13**. Notice the use of the "reset after data", which is used to terminate data reception:

```
    OWOUT Owio, 2, [$EA, $13, 123, 64]
```

RAM Reads

To read a single byte from RAM, use the **D**ump command ($Dn):

1	**1**	**0**	**1**	**0**	**Addr**

The number **Addr** can range from **0** to **7** and represents a shorthand notation for reading a *single byte* from the address represented by **Addr**. The command should be followed immediately by a read of a single byte.

The following example reads the value stored in location **5** and saves it in the variable **Dat**:

```
    OWOUT Owio, 0, [$D5]
    OWIN Owio, 0, [Dat]
```

To read multiple bytes or to read from addresses beyond **7**, use the **D**ump **A**ddress command ($DA):

1	**1**	**0**	**1**	**1**	**0**	**1**	**0**

0	**0**	**0**	**Addr**

The address value, **Addr**, can range from **0** through **31** (**$1F**). This address byte should be followed by a read of one or more bytes of data, which are retrieved sequentially, beginning with the chosen address. This continues until a reset is received. Data read from addresses beyond **$1F** are assigned the value zero.

The following example reads three values beginning at location **15**, and assigns them to the variables **I**, **J**, and **K**. Notice the use of the "reset after data" in the **OWIN** statement to terminate the read operation.

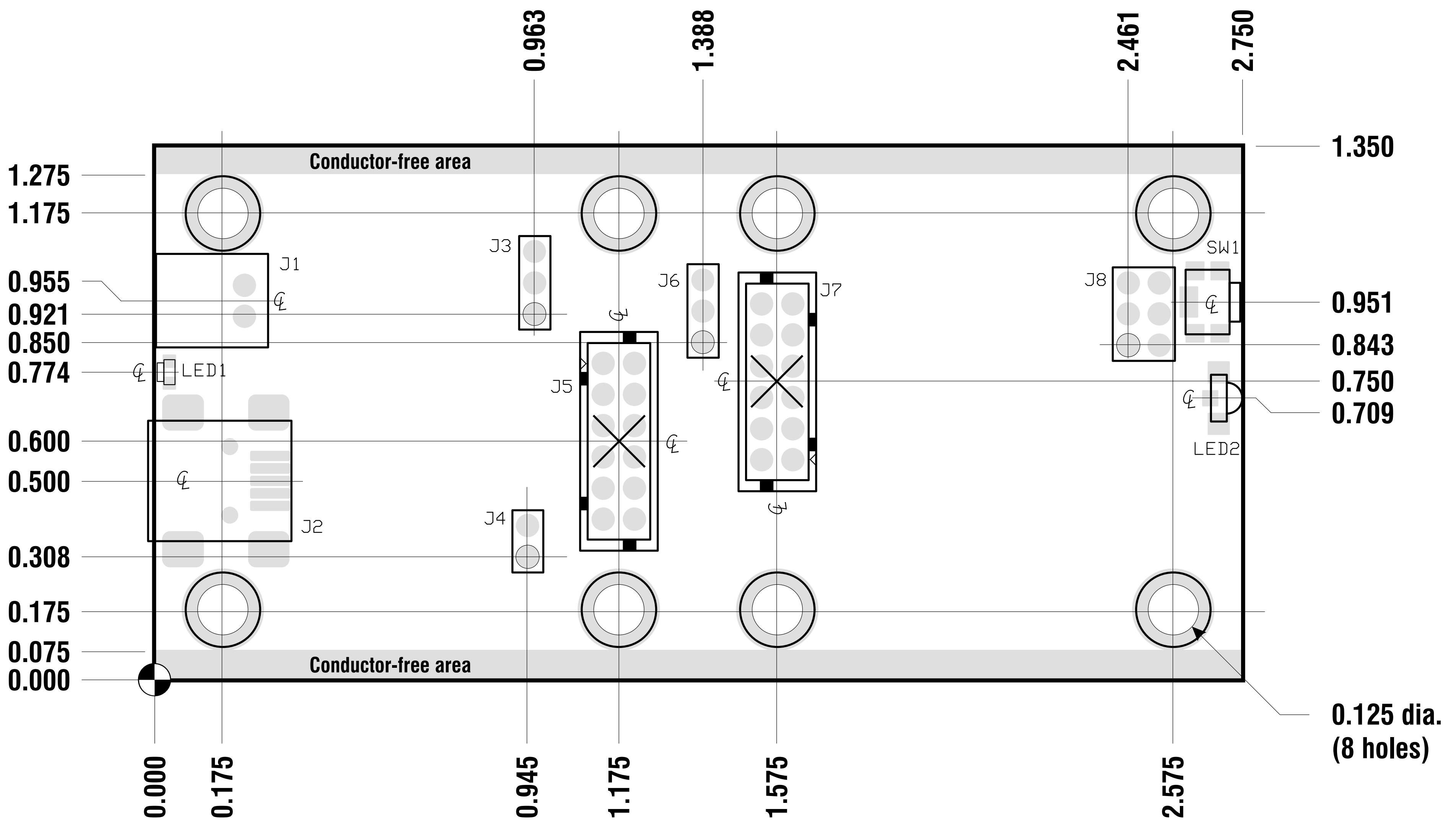
```
    OWOUT Owio, 0, [$DA, 15]
    OWIN Owio, 2, [I, J, K]
```

Summary

The following table summarizes the GPIO commands:

Allowed Port No.				Command (p is port no.)	Description	Followup
3	2	1	0			
✓	✓	✓	✓	00pp0000 = $p0	Digital Input	Read bit
✓	✓	✓	✓	00pp0001 = $p1	Counter input, LSB	Read byte (reset before data)
✓	✓	✓	✓	00pp0010 = $p2	Counter input, MSB	Read byte (reset before data)
✓	✓	✓	✓	00pp0011 = $p3	Counter input, Word	Read 2 bytes (reset before data)
✓	✓	✓	✓	00pp0100 = $p4	Digital output 0	
✓	✓	✓	✓	00pp0101 = $p5	Digital output 1	
✓	✓			00pp0110 = $p6	PWM output positive	Output PWM value (one byte)
✓	✓			00pp0111 = $p7	PWM output negative	Output PWM value (one byte)
	✓	✓	✓	00pp1000 = $p8	Compare to Port 3	Read bit
		✓	✓	00pp1001 = $p9	ADC[7..0], Vdd ref	Read bit until 0, then read byte
		✓	✓	00pp1010 = $pA	ADC[9..2], Vdd ref	Read bit until 0, then read byte
		✓	✓	00pp1011 = $pB	ADC[9..0], Vdd ref	Read bit until 0, then read 2 bytes
	✓	✓	✓	00pp1100 = $pC	Compare to 1.1V ref	Read bit
		✓	✓	00pp1101 = $pD	ADC[7..0], 1.1V ref	Read bit until 0, then read byte
		✓	✓	00pp1110 = $pE	ADC[9..2], 1.1V ref	Read bit until 0, then read byte
		✓	✓	00pp1111 = $pF	ADC[9..0], 1.1V ref	Read bit until 0, then read 2 bytes
				$F0 - $F9	Set PWM frequency	
				$En, n = 0-7.	Save 1 byte to RAM.	Output byte value to save.
				$EA	Save data to RAM	Output address, data byte(s), reset
				$Dn, n = 0-7	Read 1 byte from RAM	Read byte
				$DA	Read data from RAM	Output address, read byte(s), reset
				$DD	Read firmware ID	Read three bytes

MoBoStamp-pe Dimensions



Coordinates are in inches, relative to lower-lefthand corner of board.
Reference point of daughterboard connectors (J5, J7) is center of hole pattern.
Nominal daughterboard size is 1.350" x 1.350", with upper-lefthand corner coincident to motherboard (0,0) or (2.750,1.350), above.
Daughterboards should be free of conductors (traces and groundplanes) within 0.075" of side edges (top and bottom, above).
Daughterboards should be free of conductors (traces and groundplanes) within 0.138" of mounting hole centers.
All vertical headers have a 2mm pitch.
Mating plugs for J5 and J7 are Hirose DF11-12DP-2DSA(24).
Mating cordset for J1 is Parallax #800-28300 (Hirose 2-pin DF3 series).

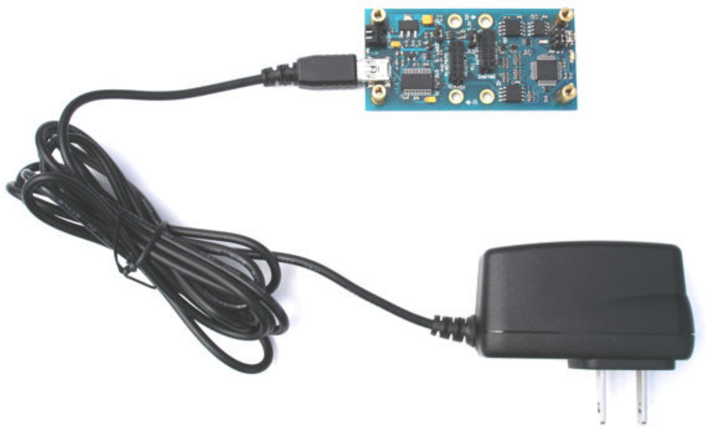
TITLE: MoBoStamp-pe Dimensions	
File Name: mobostamp-pe_dim.cdr	
Date: 4 Feb 2008 (Upd. 7 Mar 2008)	
Drwn: PCP	Rev: Q (v1.6)
Materials:	

propeller@phipi.com

App Note: *Standalone Power for Parallax Motherboards via the USB Connector*

Introduction

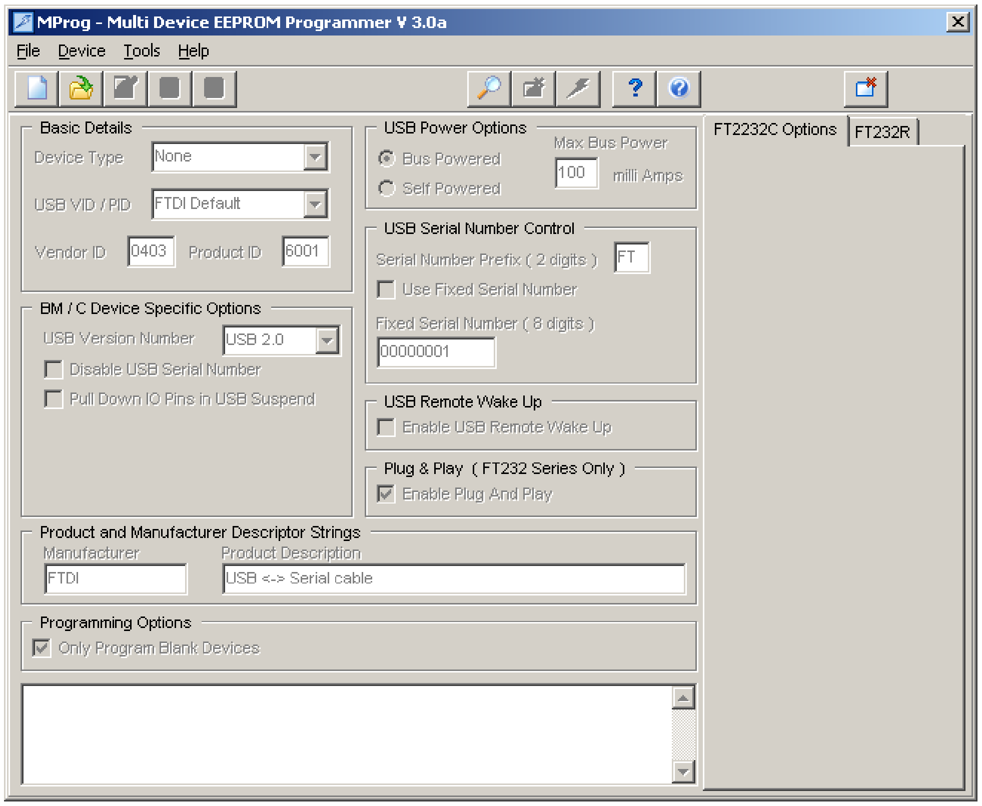
Parallax motherboards, like the MoBoStamp-pe (p/n 28300), along with some of their daughterboards, accommodate a variety of power sources. For example, they can be powered from a PC's USB port, from the onboard **Vin** connector via the MOBO Power Cable (p/n 800-28300), from an external RS232 device via the (upcoming) RS232-DB daughterboard (p/n 28315), or from a wall transformer via the PWR-I/O-DB daughterboard (p/n 28301). This application note describes yet another way: using a 5V regulated DC wall transformer that includes a USB mini-B connector. Such a power supply is made by Phihong (among others). The Phihong unit (p/n PSAA05A-050 for the U.S. model) may be obtained from Mouser Electronics (www.mouser.com). It is shown plugged into a MoBoStamp-pe in the photo below.



Setup

For power from the USB port to be switched onto the motherboard's 5V supply rail, the onboard FTDI USB interface chip must undergo a handshaking transaction with an attached PC. Once this takes place, it will turn on the USB-supplied power to the rest of the board. Therefore, simply plugging one of the Phihong power supplies into the motherboard's USB port will not power up the board, since there's no PC with which to do the handshaking. But the FTDI chip can be configured (*i.e.* tricked) to bypass this important function and power up the downstream electronics whenever it receives power from the USB connector.

To configure the FTDI chip, you will need to obtain the **MPROG** utility from FTDI's website: http://www.ftdichip.com/Resources/Utilities.htm. Download and install the **MPROG** utility, then start it up. You will se a screen like the following:



MProg - Multi Device EEPROM Programmer V 3.0a

File Device Tools Help

Basic Details
Device Type: None
USB VID / PID: FTDI Default
Vendor ID: 0403 Product ID: 6001

BM / C Device Specific Options
USB Version Number: USB 2.0
Disable USB Serial Number ☐
Pull Down IO Pins in USB Suspend ☐

USB Power Options
[✓ Bus Powered] Self Powered
Max Bus Power: 100 milli Amps

USB Serial Number Control
Serial Number Prefix (2 digits): FT
Use Fixed Serial Number ☐
Fixed Serial Number (8 digits): 00000001

USB Remote Wake Up
Enable USB Remote Wake Up ☐

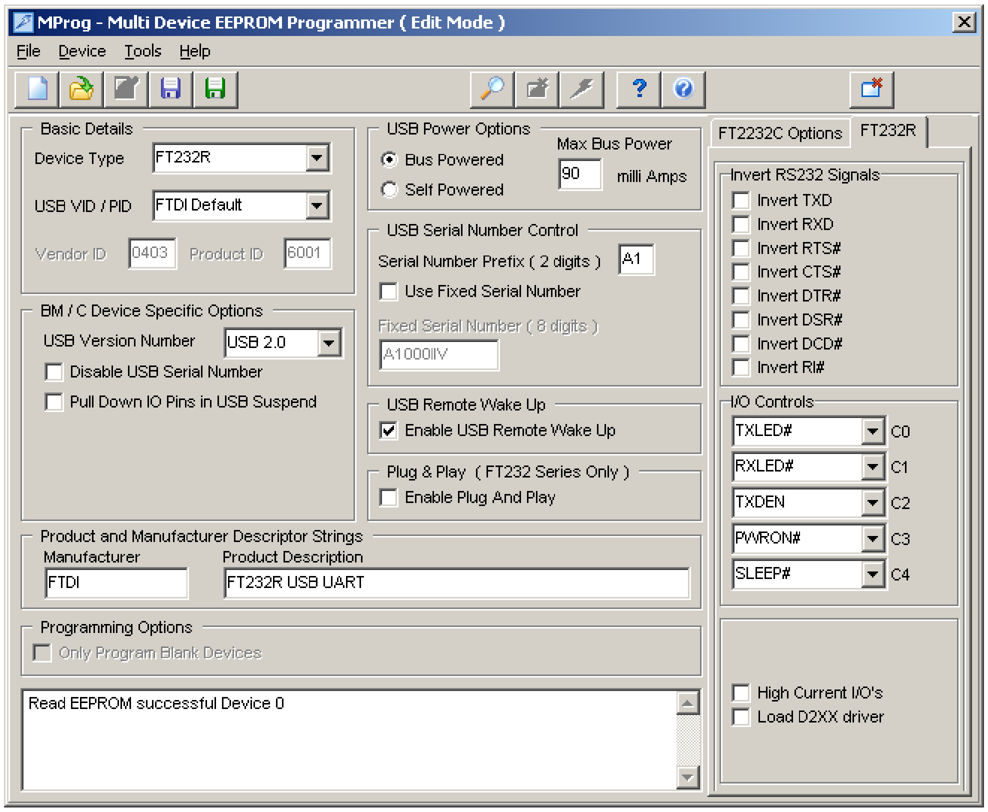
Plug & Play (FT232 Series Only)
Enable Plug And Play ☑

Product and Manufacturer Descriptor Strings
Manufacturer: FTDI
Product Description: USB <-> Serial cable

Programming Options
Only Program Blank Devices ☑

FT2232C Options | FT232R

Now, connect your motherboard via the USB port. Then click **Tools -> Read and Parse.** You will see a screen that looks something like this:



MProg - Multi Device EEPROM Programmer (Edit Mode)

File Device Tools Help

Basic Details
Device Type: FT232R
USB VID / PID: FTDI Default
Vendor ID: 0403 Product ID: 6001

BM / C Device Specific Options
USB Version Number: USB 2.0
Disable USB Serial Number ☐
Pull Down IO Pins in USB Suspend ☐

USB Power Options
[✓ Bus Powered] Self Powered
Max Bus Power: 90 milli Amps

USB Serial Number Control
Serial Number Prefix (2 digits): A1
Use Fixed Serial Number ☐
Fixed Serial Number (8 digits): A1000IIV

USB Remote Wake Up
Enable USB Remote Wake Up ☑

Plug & Play (FT232 Series Only)
Enable Plug And Play ☐

Product and Manufacturer Descriptor Strings
Manufacturer: FTDI
Product Description: FT232R USB UART

Programming Options
Only Program Blank Devices ☐

```
Read EEPROM successful Device 0
```

FT2232C Options | FT232R

Invert RS232 Signals
Invert TXD ☐
Invert RXD ☐
Invert RTS# ☐
Invert CTS# ☐
Invert DTR# ☐
Invert DSR# ☐
Invert DCD# ☐
Invert RI# ☐

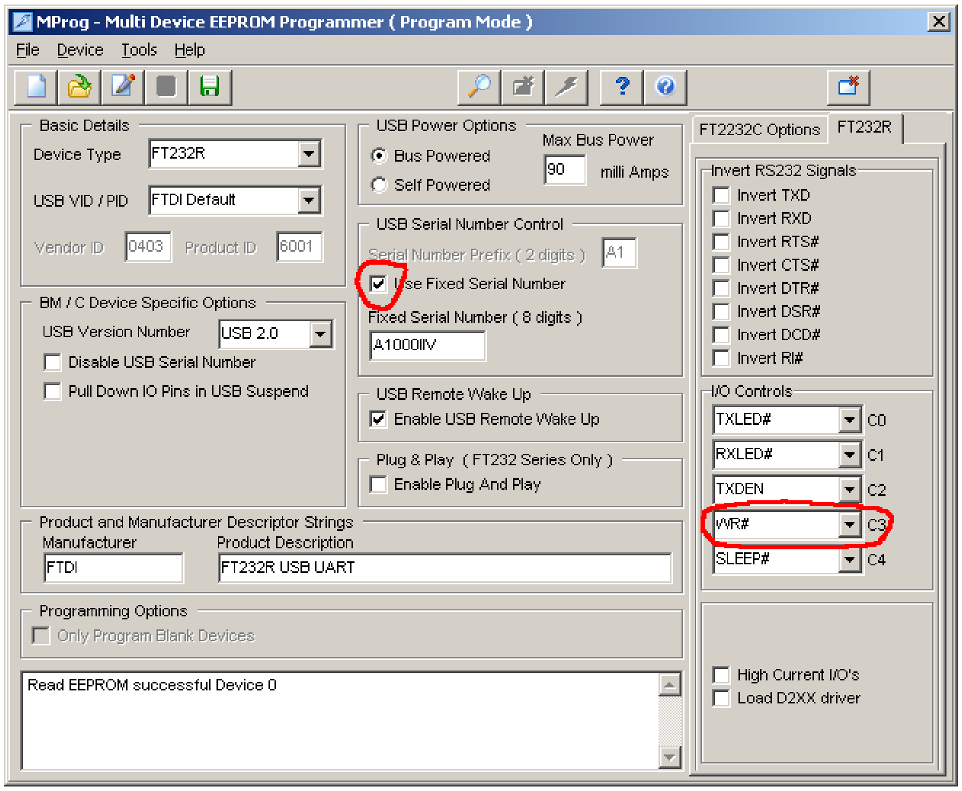
I/O Controls
TXLED#	C0
RXLED#	C1
TXDEN	C2
PWRON#	C3
SLEEP#	C4

High Current I/O's ☐
Load D2XX driver ☐

There are two things you will need to set:

1. Check the box that says, “Use Fixed Serial Number”. This will keep the serial number of your FTDI chip from changing. *Do not change the serial number in the “Fixed Serial Number (6 digits)” box.*
2. Under **C3** in the righthand column, select **WR#** instead of the default **PWRON#** setting.

Your screen will look like this:



Basic Details
Device Type: FT232R
USB VID / PID: FTDI Default
Vendor ID: 0403 Product ID: 6001

BM / C Device Specific Options
USB Version Number: USB 2.0
☐ Disable USB Serial Number
☐ Pull Down IO Pins in USB Suspend

USB Power Options
● Bus Powered
○ Self Powered
Max Bus Power: 90 milli Amps

USB Serial Number Control
Serial Number Prefix (2 digits): A1
☑ Use Fixed Serial Number
Fixed Serial Number (8 digits): A1000IIV

USB Remote Wake Up
☑ Enable USB Remote Wake Up

Plug & Play (FT232 Series Only)
☐ Enable Plug And Play

Product and Manufacturer Descriptor Strings
Manufacturer: FTDI
Product Description: FT232R USB UART

Programming Options
☐ Only Program Blank Devices

Read EEPROM successful Device 0

FT2232C Options | FT232R

Invert RS232 Signals
☐ Invert TXD
☐ Invert RXD
☐ Invert RTS#
☐ Invert CTS#
☐ Invert DTR#
☐ Invert DSR#
☐ Invert DCD#
☐ Invert RI#

I/O Controls
C0: TXLED#
C1: RXLED#
C2: TXDEN
C3: WR#
C4: SLEEP#

☐ High Current I/O's
☐ Load D2XX driver

Now do a **File -> Save As ...** and save your new settings under whatever filename you want. (This is necessary to enable **MPROG** for programming.) Finally, to program the new settings, do a **Device -> Program**. At this point, your motherboard should be capable of receiving power from the Phihong wall transformer. (Be sure the EXT/USB jumper is set to USB.)

You will also be able to program the BASIC Stamp, as usual, via the USB port. However, before connecting the USB cable, it will be best to unplug any daughterboards that consume a lot of power, since the usual power-up sequence will not be followed.

It is also possible to allow the Phihong unit to power the system and to *prevent any further programming of the BASIC Stamp*. This can be useful when others are using your motherboard system and you don’t want them to change the program. This is done simply by selecting **TXDEN** in the above screen for **C3**, instead of **WR#**. Of course, this will prevent *you* from reprogramming the BASIC Stamp, too. So, to reverse this lockout, you will need to run **MPROG** again and restore **C3** to either **WR#** or **PWRON#**.