

Bot Brain I

Tak Auyeung

December 5, 2004

1 Disclaimer

The author/designer of BotBrain I is not responsible for any damage, injury, financial loss, frustration or emotional distress as a result of utilizing the design. In other words, I am not liable for *anything* as a result of using this board. If you are looking for a mature product that is approved or certified by any agency, look elsewhere.

Although BB1 has many features that compare to commercial/industrial embedded controllers, the inventor has never intended to manufacture, test, distribute or support it as such.

2 Licencing

BB1 is free 'jelloware' under the GPL. Refer to <http://www.gnu.org/copyleft/gpl.html> for details. In a nutshell, you are allowed to take the design and modify it. However, you must make the modified version available for others to change, much like I allow you to modify mine.

Note that the GPL does *not* prohibit you from making and selling PCBs based on this design. You only have to keep my copyright notice in the documentation of the PCB, *and* make the design available to anyone who wishes to use or modify it.

2.1 Why the GPL?

Some people ask me why make this design available under the GPL. Afterall, some companies sell such products for good profit (3 to 4 times mark up). Well, I am not in the business of manufacturing controller boards. In other words, don't even try to contact me to purchase assembled, tested and shrink-wrapped controller boards. I am mostly in the business of education, with just a little bit of consulting.

GPL is perfect for education purposes because others can now take my design, study and refine it without worrying about violating the law (as long as the GPL is observed). Then, *I* can take the improved version for my own use! See, by giving first, I'll take later.

GPL is also good for consulting purposes. Afterall, I am selling software and hardware *design* solutions. In other words, I charge for my time and other resources needed to develop a solution, but I have no interest in selling any hardware to my clients. Besides, many clients prefer the option to produce hardware components at cost. The GPL makes clients feel comfortable about the future since anyone can take the design and modify it (as long as the modified version is also released under the GPL). This means my clients need not worry (too much) that I may not be available to work on future projects.

2.2 What to do if you want to use it?

If you have an industrial or commercial application, and you think either BB1 or a simple modified version can do it, you have several options.

The first option is to make use of the GPL. You can request the schematic and board layout source file (in EAGLE) and download all GPLed software related to BB1. In other words, you don't have to

spend a cent to acquire the design for your application. Of course, you are still prohibited by the GPL to distribute BB1 and all other GPLed designs as your own. Even if you modify/improve the board, you still need to retain my copyright notice *and* make your modified version available via the GPL. For as long as you respect the GPL, you can still sell manufactured PCBs to others. It is just that someone else can request the schematic and board layout from you and start selling the same PCBs.

The second option is to collaborate with the inventor and get some extra help for consulting fees. As far as licencing goes, everything is still GPLed. However, the inventor may be able to help you improve BB1 and/or its software for your application. This can save you time (especially time-to-market) if you do not have an embedded engineering team at your company. Afterall the inventor *should* know the product better than others.

The last option is to contact the inventor or another embedded engineer consultant for a custom design that is licensed to you and you only (or any other special licensing agreed by the inventor). This way, *your* design remains as *yours* and no one else can take it and start to manufacture and sell your PCB. Of course, this option tends to take up more consulting hours because the embedded engineer (including the inventor) needs to redesign from scratch.

As far as the logistics of PCB manufacturing goes, you can consult PCB houses that do the whole nine yard. Please note that the board layout of the current BB1 may not be friendly to automated pick-and-place.

2.3 How to get one?

As the designer of this board, I do not intend to make a business from the sales of these boards. If you want one, assembled or not, contact me at tauyeung@ieee.org. I may or not may not have the parts available given any moment.

If you want one in kit form, I can send the PCB and the necessary parts in a ziplock bag. If you want one assembled, you can either hire someone else to assemble the board or pay me to do so. It won't be cheap since my usual consulting fees is not cheap.

2.4 Potential Distributors

If you are a distributor and want to sell the BB1, you can do so. I can provide the part list and the gerber files so you can order the PCBs from almost any PCB fabrication company and the parts from any electronic part distributors. However, remember that the BB1 is licensed under the GPL. This means you must make it clear that the copyright belongs to me.

Print the following in at least 8 point fonts and put it at an obvious place in the package.

This BotBrain I is manufactured by ----
under the General Public License (<http://www.gnu.org/licenses/gpl.htm>)
Copyright 2002 (C) Tak Auyeung (tauyeung@ieee.org)

Under the GPL, you do not need to give me any portion of the profit from selling BB1s. On the other hand, BB1 does not come with any warrantee, so I am not responsible for any damage/injury from the sold BB1s. Furthermore, the GPL allows *anyone* to approach me and get the same part lists and gerber files. Furthermore, the GPL does not prohibit me from selling kits and assembled BB1s.

The bottom line is that if you wish to make money from selling BB1s, you have to price it right. Otherwise, someone else will sell BB1s for less and take away your business. I *suggest* the following pricing scheme:

- include cost of raw materials (PCB, components, packaging, etc), use your actual cost (take into considering the discount you get from ordering large quantities)
- include cost of assembly and testing

- include cost of overhead (advertisement, sales, marketing,...)
- compute a “total cost” per board
- compute cost of raw materials for individual hobbyists (low quantity pricing)
- assign a dollar amount per hour for a hobbyist, \$20 to \$50 is probably about right, most experienced hobbyists will need at least two hours to assemble a BB1, four or more hours is more realistic for hobbyists who have not performed SMT soldering before
- combine the hobbyist cost of materials and cost of time as a “total worth” of a BB1
- sell at the “total worth”, the difference from the “total cost” is your net profit from each BB1
- have “group discounts” available for robot clubs.

3 Overall Design Objective

The BotBrain I (hence BB1) is a PCB designed for general purpose miniature robot control, but with sufficient resource for specific robots such as Micromouse and Sumo type robots. Potential users of BB1 include college students with electronics and computer programming background, as well as amateur robot enthusiasts who may have little background in programming or electronics.

4 Feature Overview

BB1 is, more or less, AICB (another inexpensive controller board). With modern inexpensive MCUs (microcontrollers), anyone with a degree in electrical engineering can design a controller board. On the other hand, BB1 does have certain ‘unique’ features less commonly found in other controller boards.

4.1 Direct GP2D12 Connection

If you use the GP2D12 distance sensor (triangulation type), you can connect the sensor to BB1 directly. In fact, you can connect up to 8 of these sensors to BB1.

4.2 Direct R/C Servo Connection

If you use R/C servos (modified for full rotation or not), you can connect their connectors to BB1 directly. There are four such connectors on BB1.

4.3 Current Controlled Pulsed Low-side Switches

This is one of the most unique features of BB1. BB1 provides 8 multiplexed current controlled low-side switches. This means software can turn on any one of the eight low-side switches and expect no more than a determined current sunk. The amount of current is controlled by SIP (single inline package) resistor packs that are easily replaced.

This feature is very useful to utilize LED-type devices. Many sensors rely on either an LED or IrED (infrared emitting diode) to emit some energy, then use either a photodiode or phototransistor to sense the transmitted or reflected energy. While most LEDs and IrEDs have a 20mA maximum *continuous* current rating, they can also be pulsed at up to 1A when duty cycle and pulse length are controlled.

Duty cycle and pulse length can be controlled by software, the current controlled circuit controls the amount of current. Note that this feature almost completely eliminate the need of current limiting resistors. Such resistors are only needed only if the the voltage drop across the low-side switches exceeds power dissipating limits.

4.4 Synchronous Input Shift Registers

Of the two digital input shift registers, one can be synchronized with the current controlled pulse. This provides hardware synchronization of the end of pulse and data capturing. The result is great simplification of software with precise latch timing.

Note that the current controlled circuit is designed to work closely with 8 digital inputs (implemented by a shift register). Depending on the voltage drop per diode, up to 64 reflective or transmissive sensors can be used with only 8 current-controlled switches and 8 digital inputs!

4.5 1A (per channel) Inductive-capable Low Side Switches

Okay, 1A isn't that much current, still it is quite a bit more than TTL/CMOS levels. BB1 has 8 independent such switches. The MOSFET chosen for this feature has a very low on-resistance (less than 0.1Ω), which make these switches very efficient.

4.6 Port Headers

The chosen MCU (ATMega128) has plenty of communication ports. BB1 brings all such ports out to headers. This includes the following:

1. serial ISP (in-system-programming) port
2. first UART port
3. second UART port
4. SPI port
5. two-wire (I2C) port

Each header provides VCC and GND to power external circuits. The UART ports also provide a transmitter-enable signal so that common carrier protocols (such as RS-485 and most RF protocols) can be implemented by daughterboards.

4.7 Universal Motor Ports

The previously mentioned R/C servo compatible headers are the first three pins of universal motor ports. Each universal motor port also provide VCC (regulated 5V) and RESET. This allows engineers to design external motor control boards. There are many ways to encode motor control in a pulsed signal (even for stepper motors and general DC motors).

5 Implementation

Due to routing resources and board area limitations, the actual implementation differs from the original design, just a little.

Refer to <http://www.drta.org/develops/bb1combo.pdf> for the board layout (1:1). If so needed, please contact me for the Eagle schematics and board layout files. The top components are shown in <http://www.drta.org/develops/bb1-top.pdf> and bottom components are shown in <http://www.drta.org/develops/bb1-bot.pdf>.

Most of what you need to know are in the schematic capture and the board layout. Refer to <http://www.drta.org/develops/bb1sch.pdf> for the schematics of the board.

This section describes the implementation.

5.1 UMPs

The universal motor ports are on the west edge of the board. Each UMP is configured as follows:

pin #	function
1	control
2	VDD
3	GND
4	VCC
5	reset

Table 1: UMP pinout

The ‘control’ line is directly connected to the MCU. As a result, it is a TTL/CMOS level signal. VDD is the unregulated voltage. GND is the system ground. VCC is regulated 5V. ‘reset’ is the reset signal to the entire system.

Each UMP is designed such that most R/C servos can be directly connected to just the first three pins (the northmost three pins). For custom DC motor and stepper motor control, all five pins can be connected to a daughterboard, which interprets the ‘control’ signal and determines what to do.

The following table indicates how UMPs are connected to the pins of the ATmega128.

UMP connector	ATmega128 pin
JP1	pin 15, OC1A (PB5)
JP2	pin 16, OC1B (PB6)
JP3	pin 17, OC1C (PB7)
JP4	pin 5, OC3A (PE3)

Table 2: UMP ‘control’ to ATmega128 pins

5.2 Analog Port

JP15 and JP16 combine to form the analog port. This port is configured so that 3-pin connectors can access ground, VCC and analog input pins directly. The northmost row of JP15 is ground. All pins except the westmost and eastmost pins of the second row of JP15 connect to ADC lines of the ATmega128. The westmost and eastmost pins of the second row of JP15 connect to VCC. From west to east, the signals are ADC0 to ADC7

All pins of JP16 connect to VCC.

This configuration allows one directly connect a Sharp GP2D12 distance sensor to the connector.

5.3 General Purpose Input/Output

JP21 is the GPIO port. The following table lists the pinout.

This connector can be used for any application that only requires TTL/CMOS level signals. Note that PE4 to PE7 can also serve as special function pins.

5.4 Slow Input Port 0

JP8 is slow input port 0. Signals connected to this port first pass through schmitt triggers (negating) before captured by a shift register. This port is special because it can be set up (via a jumper) to synchronize with the pulsing circuit. The following table summarizes the pinout.

Note that the input pins are not pulled up. If these signals are connected to phototransistor-type sensors, external resistors must be provided to provide pull-up or pull-down functions.

JP21 pin #	function
1	VCC
2	pin 51, PA0
3	pin 50, PA1
4	pin 49, PA2
5	pin 48, PA3
6	pin 47, PA4
7	pin 46, PA5
8	pin 45, PA6
9	pin 44, PA7
10	pin 6, PE4, INT4, OC3B
11	pin 7, PE5, INT5, OC3C
12	pin 8, PE6, INT6, T3
13	pin 9, PE7, INT7, IC3

Table 3: GPIO (JP21) Pinout

pin #	function
1	VCC
2	bit 0
3	bit 1
4	bit 2
5	bit 3
6	bit 4
7	bit 5
8	bit 6
9	bit 7
10	ground

Table 4: Slow Input Port 0 (JP8) Pinout

5.5 Slow Input Port 1 (JP9)

Slow input port 1 (JP9) is daisy chained to JP8 to read as bit 8 to bit 15. The load (/LD) of this shift register is independent of the pulsing circuit, making this more general purposed than JP8. The pinout of this port is identical to that of JP8.

5.6 Slow Logic Output Port 0 (JP10)

Slow logic output port 0 maps to the output of a shift register. The following table summarizes the pinout of this port.

These output pins are only suitable for CMOS/TTL level applications.

5.7 High Voltage High Current Port 1

This port (JP11) provides 8 channels of low-side drivers. Each channel is controlled by a shift register daisy chained to the shift register used in the Slow Logic Output Port (JP10). The entire southmost row connects to VDD (unregulated input voltage). The northmost row connects to each individual low side driver. The westmost one is bit 0, the eastmost one is bit 7.

These low-side drivers are protected by diodes against electro-magnetic kickback, and therefore they are suitable for driving inductive loads.

pin #	function
1	VCC
2	bit 0
3	bit 1
4	bit 2
5	bit 3
6	bit 4
7	bit 5
8	bit 6
9	bit 7
10	ground

Table 5: Slow Logic Output Port 0 (JP8) Pinout

5.8 Constant Current Pulsed Port

This port (JP7) provides 8 channels of pulsed constant-current low-side drivers. Only one of the 8 channels may be active at any time because driver control is implemented by a demultiplexer. The following table summarizes the pinout.

pin #	function
1	VDD
2	channel 0
3	channel 1
4	channel 2
5	channel 3
6	channel 4
7	channel 5
8	channel 6
9	channel 7
10	VDD

Table 6: Constant Current Pulsed Driver (JP7) Pinout

5.9 Logic Level Pulsed Port (JP6)

In some applications, it is useful to use LEDs to indicate the state of each input corresponding to the constant-current pulsed drivers. The logic level pulsed port (JP6) provides logic level pulses for each of the 8 constant-current pulsed driver so an external circuit (involving an 8-bit latch) can be used to take snapshots for display purposes.

The following table summarizes the pinout of JP6.

5.10 Communication/Programming Port 0

JP12 is communication/programming port 0. It can be used for communication (asynchronous) or in-system programming (synchronous). The following table summarizes the pinout.

The first 5 pins can be used for a stripped programming cable. Using the first 6 pins, one can implement a buffered programming cable. Pin 7 of the connector is used for communication boards that have control over the transmitter (radio modem, RS-485 and IR).

pin #	function
1	VCC
2	channel 0
2	channel 0
3	channel 1
4	channel 2
5	channel 3
6	channel 4
7	channel 5
8	channel 6
9	channel 7
10	ground

Table 7: Constant Current Pulsed Driver (JP7) Pinout

pin #	function
1	reset
2	RXD0, PDI, PE0
3	TXD0, PDO, PE1
4	SCK
5	ground
6	VCC
7	TXEN0 (PE2)

Table 8: Communication/Programming Port 0 Pinout

5.11 Communication Port 1

JP14 is the other communication port. JP14 is suitable for asynchronous communication. The following table summarizes the pinout.

pin #	function
1	
2	RXD1, PD2, INT2
3	TXD1, PD3, INT3
4	
5	ground
6	VCC
7	TX1EN (PD4)

Table 9: Communication Port 1 (JP14) Pinout

5.12 I2C (two-wired clocked communication)

JP13 is a port for I2C or two-wired clocked communication. The pinout is summarized in the following table.

Note that SCL and SDA should be pulled up for this bus to work. The pull up resistors are not included because BB1 can serve as a slave to another system.

pin #	function
1	VCC
2	SCL
3	SDA
4	ground

Table 10: I2C Port (JP13) Pinout

5.13 SPI Port

JP22 is a port for SPI (another clocked synchronous communication protocol). The pinout is summarized in the following table.

pin #	function
1	VCC
2	SS/SLV0
3	SCK
4	MOSI
5	MISO
6	SLV1 (PG2)
7	SLV2 (PC4)
8	SLV3 (PC3)
9	SLV4 (PC2)
10	ground

Table 11: SPI Port (JP22) Pinout

This port is designed to allow the BB1 serve as a master with up to 5 slaves (selected by SLV0 to SLV4).

6 Programming BB1

BB1 is powered by Atmel's ATMega128. As such, any software development tool designed for the ATMega128 should work for BB1. This section discusses known tools (both software and hardware) used to program BB1.

6.1 Software

I use AVR-GCC to develop software for BB1. Not only is it free, it is also derived from one of the most tested C compilers. AVR-GCC, like BB1, is distributed under the GPL.

AVR-GCC is available for most Unix-like systems as well as Win32 type systems. In other words, it is available for almost all computers now that Macintosh OS X is unix-like.

You can check out <http://www.avrfreaks.net> for more information about using AVR-GCC, especially on a Win32 platform. Linux users can get AVR-GCC as a package (at least with Debian systems).

For serious debugging, you can use Atmel's AVRStudio. AVRStudio can interact with AVR-GCC for an almost seamless development tool chain. In other words, you can write C programs and compile them using AVR-GCC, then test and debug the code with the simulator built into AVRStudio. Consult user's guides written by AVRFreaks for more information.

The GNU debugger (AVR-GDB) can also be used. However, for the time being, BB1 has limited support for debugging a program running on a board. Future version of BB1 will most likely include

a JTAG port so that inexpensive in-circuit emulators (compared to multi-thousand dollar ones) can be used to debug programs running on BB1.

6.1.1 Getting Free Stuff (Linux)

Although Debian includes most of the tools as packages, it is better to get the latest. At least, it is better to have an option to get the latest builds.

First, you need a few tools on your Linux box such as autoconf, automake and other utilities for building applications. You can often get these tools as packages of your distribution.

Next, you need to get the source code and make all the necessary tools. Generally speaking, there are a few steps to install a package from source.

1. download the package, I assume you will download to `/tmp`
2. uncompress the package:

```
cd /usr/local
tar xzvf /tmp/some_package.tar.gz
```

if the package ends with `.bz2`, use `xjvf` instead of `xzvf`.

3. go to the package direction:

```
cd /usr/local/some_package_dir
```

4. configure the package (general options for avr tools shown, some packages require more customized options):

```
./configure --target=avr --prefix=/usr/local/avr
```

5. make the files:

```
make
```

6. install the files:

```
make install
```

7. optionally, if you do not include `/usr/local/avr/bin` in your `PATH`, you can create symbolic links to the files at `/usr/local/avr/bin`:

```
ln -f -s --target-directory=/usr/bin /usr/local/avr/bin/*
```

Start with `binutils`. Go to <http://mirrors.rcn.net/pub/sourceware/binutils/releases> and download the latest package.

After you build `binutils`, you can compile the C compiler, `gcc`. Go to <http://www.planetmirror.com/pub/gnu/gcc/>, click on the latest version, then download the latest `gcc-core` package. There is no point in getting the other languages because there is no library support for them (as of July, 2002).

You can following the general procedure to build a package, except to Use the following command to configure the package:

```
./configure --target=avr \  
--prefix=/usr/local/avr --disable-nls \  
--enable-languages=c
```

After you get the C compiler compiled, you can build the C libraries. Go to <http://www.amelek.gda.pl/avr/libc> and get the latest version.

Follow the general procedure to build the package except to Use the following command to configure the package:

```
./configure -v --target=avr --host=avr \  
            --build='./config.guess' \  
            --prefix=/usr/local/avr
```

I also suggest that you periodically update your `uisp`. This package (source) is found at <http://savannah.gnu.org/download/>

6.1.2 My Patched Versions

I have prepared a set patched source distributions that work better for the ATmega128 (compared to the 'stock' version). These files can be found at <http://www.mobots.com/avr-gnu/>.

You can unzip, untar, configure, make and make install just like the stock distributions.

If anyone is interested in integrating these changes back to the main branch, please be my guest!

6.2 Hardware

Because BB1 uses the same physical port for programming and UART 0, it does not use the same programming connector as Atmel's STK200. To overcome this problem, I designed a STK200 compatible programming board for BB1 that allows the use of any programmer software that works with the STK200.

The programming interface board connects to the parallel port of a PC, the other end connects to the programming port of BB1. Because only 6 wires are used, you can use 6 conductor modular-plug cables or CAT5 cables for this purpose. You probably want to use a Molex-type connector on the BB1 side for easy connection and disconnection.

The programming interface board is also distributed under the GPL. The cost of this board should be less than US\$10, with most of the cost going to the DB25 connector. You can see the board layout, top components and the schematic online.

If you already own an STK200 board, you can also custom make a cable to connect from the 2x5 header of the STK200 to the 1x7 header of BB1.

For those who need the best debugging tool, yes, AVRStudio supports JTAG ICE, and AVR-GDB supports JTAG ICE with AVaRICE (refer to <http://avarice.sourceforge.net> for more details). If you cannot wait for the next revision, you can make a special cable to adapt the JTAG ICE 2x5 interface to four of the ADC pins that double as JTAG pins. Read Atmel's document on the JTAG ICE. Don't forget to connect ground and RESET (probably to the programming/UART 0 port or the SPI port)!

7 Assembling and Soldering

All sections of this page has been modified so all references to "solder wick" is replaced by "desoldering braid" as per the request of Leydig, Voit and Mayer Ltd., representing Illinois Tool Works Inc (ITW). The objection was based on the phrase "solder wick" resembles the registered trademark "SODER-WICK" and it might harm the protected trademark.

As a result, let me just clarify once and for all that any desoldering braid of the right size will work in this tutorial. In addition, I was not referring to or endorsing any product from ITW in the previous version of this section in the tutorial.

Leydig, Voit and Mayer Ltd. and ITW can now rest assured that the registered trademark "SODER-WICK" is now protected (from drtak.org).

That said, I wonder how many entries one will find from a search engine looking for “solder wick” as a phrase. I don’t think ITW will like the answer. Lawyers, on the other hand, may like the answer because this means more work. Make that *much* more work.

7.1 Stuff that you need

1. temperature controlled soldering iron/station
 - (a) do not use fixed power soldering irons since they have no temperature control
 - (b) Metcal soldering stations work well, but they are expensive
 - (c) Hakko and Weller soldering stations are good compromise solutions
2. circuit board clamp/holder
 - (a) makes sure the board does not slide when it is being soldered
 - (b) flexible holders also allow all kinds of movement to adjust the working angle
3. tweezers
 - (a) useful for holding small parts
 - (b) do *not* use plastic tweezer!
 - (c) spring-loaded (to close) ones are useful, but they are not required
4. solder wire
 - (a) ‘44’ type works, but it contains lead and requires cleaning
 - (b) no-clean type uses noncorrosive flux so you don’t have to wash the board after soldering
 - (c) leadless type is safer if you are soldering at home, especially in the kitchen or where children visit often
5. desoldering braid
 - (a) the size doesn’t really matter
 - (b) most come with some flux already included
 - (c) if you use solder wire that has built-in no-clean flux, make sure your desoldering braid is also of no-clean type
6. flux
 - (a) make sure it is for electronics work and not for plumbing!
 - (b) if you are using no-clean solder wire and desoldering braid, make sure you also use no-clean flux
7. cleaning alcohol/flux remover
 - (a) most are organic solvents, don’t inhale!
 - (b) most are flammable
8. snipper for cutting desoldering braid, wire cutters should work as well
9. masking tape
10. low power magnifying aide

- (a) must be hands free so you can use both hands for soldering!
- (b) 2x to 3x leave sufficient working distance
- (c) best to use a magnifying visor with prism so you don't have to cross eye
- (d) cheap clip-on reading attachments from drug stores also work, but most of these do not come with prism

11. high power magnifying aide

- (a) needs not be hands free for inspection purposes
- (b) a 8x or 10x jewelry loupe works well
- (c) a lit one works even better, but it is not necessary
- (d) higher power (beyond 8x) is not necessary better because it shortens the working distance

12. lighting

- (a) may need to reposition the light source, flexibility is important
- (b) cheap swing arm desk lamps work great
- (c) no need to have magnifying glass on the lamp

7.2 Soldering Order

In theory, you can solder components in any order you like. However, a carefully determined order can make the process easier and less prone to problems.

The rule of thumb is easy: solder components from thin to thick. This means solder the TQFP and TSSOP components first, then SOT23, then SO,... Headers, jumpers and sockets are typically soldered last.

(This is changed as of 2004/11/29.)

For an inexperienced person, soldering components on the BB1 can be a long process. To reduce the amount of frustration, it is best to solder some, test some, and then solder more. The order to solder components is outlined as follows:

- Break off the two parts. Separate the BB1 from the programmer board.
- Solder the programmer board. Use a 6 or 7 pin header for the connector that connects to the BB1. If you use a 6 pin header, be sure to just leave pin 7 unconnected. The first pin (pin 1) is the one with a square pad.
- Prepare a programmer cable. The cable length should not exceed 7 or 8 inches. Use good connectors on both sides. The crimp type connectors often work well. You *can* use a 14-conductor IDE cable with IDE connectors. However, you need to remember which half you use.
- Voltage Regulation:
 - Solder the jumper or switch, power connections, input capacitor (1210 sized 1uF), regulator (8SOIC 73500), and output capacitor (“tin can” 10uF). The SOT23 diode is not required and it is usually not included. Also, solder the pull-up resistor (10k Ohm) for the reset line. The 0805 pull-up resistor should read 103 on the top (**10** times 10 to the power of **3**).
 - Pin 1 of the regulator should be closer to the edge of the board.
 - Set a power supply to a relatively low voltage. If possible, also set current limit to 10mA or less. If current limiting is not possible, use a 10mA inline fuse.

- Power up the board, and measure the output voltage of the regulator. If possible, use an oscilloscope to make sure that the regulated voltage is DC.
- Power Good LED:
 - Solder the LED and current limiting resistor. The dark green bar of the LED should be closer to the edge of the board. The current limiting resistor should read 301 (for 30 times 10 to the power of 1).
 - Change current limit to 20mA or so. If you use an inline fuse instead, use a similar value.
 - Power up the board again, this time you should see the “power good” LED light up.
- MCU, oscillator (resonator) and programming header
 - Solder the MCU. Read the next section to learn how to solder this TQFP component. The caption on the chip should align with the label of the PCB.
 - Solder the the oscillator (3-pin device)
 - Solder the programming header. Note that although there are 7 through holes, only the first 6 are needed. The pin with a square pad is pin 1.
 - Change current limiting to 80mA, or change the inline fuse to 100mA or so.
 - Power up the board, and observe current consumption. It should be around 20mA or so.
- Software controlled LEDs
 - Solder the two software controlled LEDs. It helps to use different colors for these two LEDs. The dark green bar should be close to the edge of the PCB.
 - Solder the two current limiting resistors (300 Ohm). The pads for these resistors are on the bottom side of the PCB.
 - Program the MCU’s fuse bits. Read section 8 for more information. It is normal to see an increase of current draw (by 20mA or so) once the fuse bits are programmed.
 - Now you can run the blinky test program. If the LEDs blink, everything is good up to this point!

7.3 TQFP, TSSOP and high pin-count SO Component

This procedure is quite simple and it only requires good vision in the inspection phase. The approach is quite simple. First, the component must be place correctly on the PCB. Then, solder is generously applied to the pins/pads. Yes, this does easily result in solder bridges on TQFP components. After solder is roughly and generously applied, use desoldering braid to mop up exceed solder.

Isn’t it possible that the desoldering braid may remove *too much* solder and end up with pins not contacting their pads? Yes, but *only if* the component was not flat on the PCB to start with, or that the temperature of the soldering tip/desoldering braid is incorrect. At the correct temperature, molten solder has adhesion and surface tension to help it attach to the pin and the pad. If the component is flat on the PCB, the vertical gap between a pin and a pad should not be large enough, relative their surface areas, to let the desoldering braid remove enough solder to disconnect them.

The best feature of this approach is that you really don’t need a spool of fine solder wire nor a fine soldering tip. *However*, you do need a temperature controlled soldering iron/station with a peak power of 40W or more. The more accurately and responsively your soldering tip stays at a set temperature (about 325°C in most cases), the more efficiently and safer you can use this approach.

1. Cut or tear a strip of masking tape, make sure it extends about 2cm on each side of the component. Do *not* use Scotch (TM) tape or other types of tape because they are more likely to leave some residue.

2. Place the strip of masking tape in the middle of the component horizontally, exposing the corner pins.
3. Optionally use magnifying aide, align pins to pads and place the component on the PCB, you can use the exposed tape as handles, *do not tack the strip of tape down on the PCB*
4. Gently push the exposed tape onto the PCB, just enough so the component cannot move freely, make sure you allow some degree of movement.
5. Use a higher power magnifying device to see if all pins are properly aligned, concentrate on the corner pins.
6. Nudge the component until the pins are aligned; remove the tape and component and restart from scratch if there is too much to nudge. *An improperly placed component causes a lot of grieve down the line, spend a few more minutes here!*
7. Once the pins are fully aligned, solder the corner pins first.
8. Remove the masking tape, then solder all the other pins. You can optionally use a lower power magnifying aide for this step.
9. It doesn't matter whether you get too much or too little solder on individual pins at this stage. Don't worry if one pin seems to have too much solder, two pins share one blob of solder or one pin seems to have too little solder
10. After you roughly put solder on the pins, prepare desoldering braid by cutting the used part of it and apply a little flux on the desoldering braid. It doesn't matter which size of desoldering braid you use, just make sure the end is cleanly cut so it can contact the pins/pads correctly.
11. Preheat the desoldering braid by putting the soldering tip on it for some time, until the flux on it starts to 'smoke'. This step may take a second for a properly chosen soldering iron/station, or it may take *much* longer for a lesser soldering iron/station.
12. Put the heated/smoking desoldering braid on one end of a row, and the soldering tip on the desoldering braid.
13. Make sure the desoldering braid stays hot! Otherwise, you may end up soldering the desoldering braid on the PCB, and that can be quite a mess to clean up with a low power soldering iron.
14. Slowly and steadily, move the desoldering braid *and* soldering tip across the row.
15. You should observe excess solder 'mopped' from the pins and 'dry' pins getting soldered. At the same time, the desoldering braid should get desoldered up to a distance of roughly 3mm to 10mm, depending on the thickness of braid strand and flux.
16. When you are done with one row, rotate the board and perform the same on the next row.
17. When you are done mopping all the rows, clean the PCB and inspect the solder joints with a higher power loupe (8× to 10×).
 - (a) Inspect at an angle, I recommend tilting the board 45 degrees.
 - (b) Watch for solder bridges (solder contacting two pins), wick some more until there is not enough solder to bridge the pins.
 - (c) Watch for solder that looks dull (instead of shiny), apply some flux and 'reflow' this particular pin/pad.
 - (d) Rotate the board 45 degrees and keep it tilted 45 degrees.
 - (e) Watch for a pad not contacting a pin, reapply solder to connect the pad to the pin.

7.4 0805, 1206, SOT23 and other small pin count chips

1. Apply solder to one of the pads. For a part with more than three pins, apply solder to a corner pin. If you are right-handed, apply solder to the right hand pad(s). Rotate the PCB so the chip has a right-left orientation instead of top-down orientation.
2. Use a pair of tweezers to handle the component.
3. Place the soldering iron on the pad to keep the solder molten, then place the component on the pad.
4. Remove the soldering iron when solder flows up the component.
5. Let the solder solidify.
6. Release the component from the tweezers.
7. Double check the orientation and alignment. You can easily fix any problem at this stage just by reflowing (melting) the only soldered pin/pad and use the tweezers to adjust the alignment.
8. Solder the other pin(s).

7.5 What is the big deal about a cold solder joint?

Well, it is a *big* deal. For digital circuits, a cold solder joint is hardly noticeable. However, for analog circuits, such as the constant current driver on BB1, a cold solder joint make a big difference.

The difference is due to the extra resistance of a cold solder joint compared to a good solder joint. If your LEDs seem to be on all the time, you need to check the solder joints of components of the constant current driver.

8 Initial Testing

After you assemble the boards, it is time to give the system a test spin. I cannot guarantee your system will work. In fact, I cannot even guarantee that the initial setup will not damage your current computer. In other words, the risks of killing your PC is yours to take.

That said, let's go ahead and describe how you can test the system.

Note that the ATMega128 ships with configuration bits selecting the internal oscillator at 1MHz. This means your external 16MHz ceramic resonator doesn't work at this stage. This is normal. You can communicate with the MCU via in circuit programming, then switch to the external resonator.

8.1 Powering Up BB1

At any point in this section, if current draw exceeds 100mA, disconnect power immediately!

Do not connect BB1 to anything at this stage. Use a current meter to monitor the power consumption of BB1 as you power up the board. Make sure you know which pin is ground and which pin is input voltage!

The board should draw about 80mA initially, then drop to about 40mA to 50mA. The LED at the lower right corner should also light up.

8.2 Connecting the Programmer Board

At any point in this section, if current draw exceeds 100mA, disconnect power immediately!

This is the board that can cause damage to the parallel port of your PC. The board itself is designed right, but if you have soldering problems, you can still damage the PC.

Continue to monitor the current draw of BB1. Disconnect power first, then plug in the connector from the programmer board. Do not connect the programmer board to the PC just yet. Now power up the board and monitor the current draw. The programmer board should not cause much noticeable increase of current draw.

If everything is okay at this stage, power off the board again. Now, connect the programmer board to the PC's parallel port, then power up BB1 (connected to the programmer board) and monitor current draw. You should see no change of current draw compared to the previous case (with the programmer board not connected to the PC).

8.3 Testing

You can use uisp or PonyProg2000 to test this initial setup. I use `uisp -dprog=stk200 dlpt=/dev/parport0 --erase` to test the setup. You should not receive any error messages.

You can also use PonyProg2000 for testing. Go to the 'Setup' menu and select 'Interface Setup...'. In the dialog box, select 'Parallel', then use the drop down box to select 'Avr ISP API' for NT-based Windows and Linux, or select 'Avr ISP I/O' for non-NT-based Windows.

Press 'Probe' to see if the system checks everything out okay. For some reason, PonyProg2000 under Linux always complains 'Test Failed' as of version 2.05a. I'd ignore that message for now and just click 'OK'.

Make sure you go to the 'Device' menu and select 'AVR Micro' and then 'AVR Auto'. Then the real test begins. Go to the 'Command' menu and click on 'Read All'. If everything is working, PonyProg2000 should read in the contents of the chip properly.

Note that PonyProg automatically recognizes the chip as 'ATMega103'. This is normal because the ATMega128 ships with Mega103 compatibility set. To utilize Mega128 only features (such as to disable Mega103 compatibility mode), you must manually select ATMega128 as the type. We'll talk about this more in the next section.

If you can follow up to here and successfully read back the contents of the chip, congratulations!

8.4 Selecting the External Resonator

8.4.1 uisp

uisp accesses the lock/fuse bits as the 'fuse' segment. The byte sequence is as follows:

1. fuse low bits: one byte
2. fuse high bits: one byte
3. lock bits: one byte
4. fuse extension bits: one byte

I strongly recommend that you read back the current configuration before attempting to change anything. You can use the following command: `uisp -dprog=stk200 -dlpt=/dev/parport0 --segment=fuse --download`. You may want to redirect the output to a file so you save it. Off-the-shelf ATMega128s should give you something like the following as the output:

```
Atmel AVR ATmega128 is found.  
Downloading: fuse
```

S00700006675736545
S1080000E19999FFFDEC
S5030001FB
S9030000FC

Pay attention to the middle line that reads S108. . . . Here's an analysis of this line.

- S1 means it is a data packet
- 08 means it has eight bytes following (hexadecimal)
- 0000 is the address/offset, this means the beginning
- E1 is the first byte: fuse low bits
- 99 is the second byte: fuse high bits
- 99 is the oscillator calibration byte
- FF is the lock bits
- FD is the fuse extension bits
- EC is the checksum of this packet

If you save the output to a Motorola s-record file, then you can change that file and upload it to modify the settings.

8.4.2 PonyProg2000

If you use PonyProg2000, you can access the lock bits and fuse bits graphically. First second Mega128 as the device. If you use the 'auto' setting, PonyProg2000 only recognizes the Mega128 as a Mega103. Next, select 'Configuratio and Security Bits...' under the 'Command' menu.

Now, be very careful or you may need to get a new MCU!

Click on 'Read' to read back the current settings. Make sure you do not accidentally click on 'Write', or you will lose the ability to program the chip again without using a parallel programmer! This is because the dialog box starts with all boxes unchecked, which includes JTAGEN and SPIEN. If you write this back, it prohibits the device from being programmed via JTAG and SPI. It'll be a hassle to remove the chip and get access to a parallel programmer.

Whatever you do, make sure SPIEN remain checked when you click the write button!!!

With PonyProg2000, you need to know what is checked and what is cleared. For lock and fuse bits, 'checked' means 'programmed' or a value of 0. A unchecked checkbox means 'unprogrammed' or a value of 1. Yes, checked means 0 and unchecked means 1.

The following is a screen capture of what PonyProg2000 should look like when you configure the fuse bits.

8.4.3 Changing the Fuse Bits

I will use 0s and 1s in this discussion. PonyProg2000 users should be careful when converting 0s and 1s to programmed and unprogrammed.

CKSEL3:0 should be changed to 1111₂, 1101₂, or 1011₂. CKOPT should be changed to 0. SUT1:0 should be changed to 10. These options should be very safe to use. For uisp, the previous output should be changed as follows:

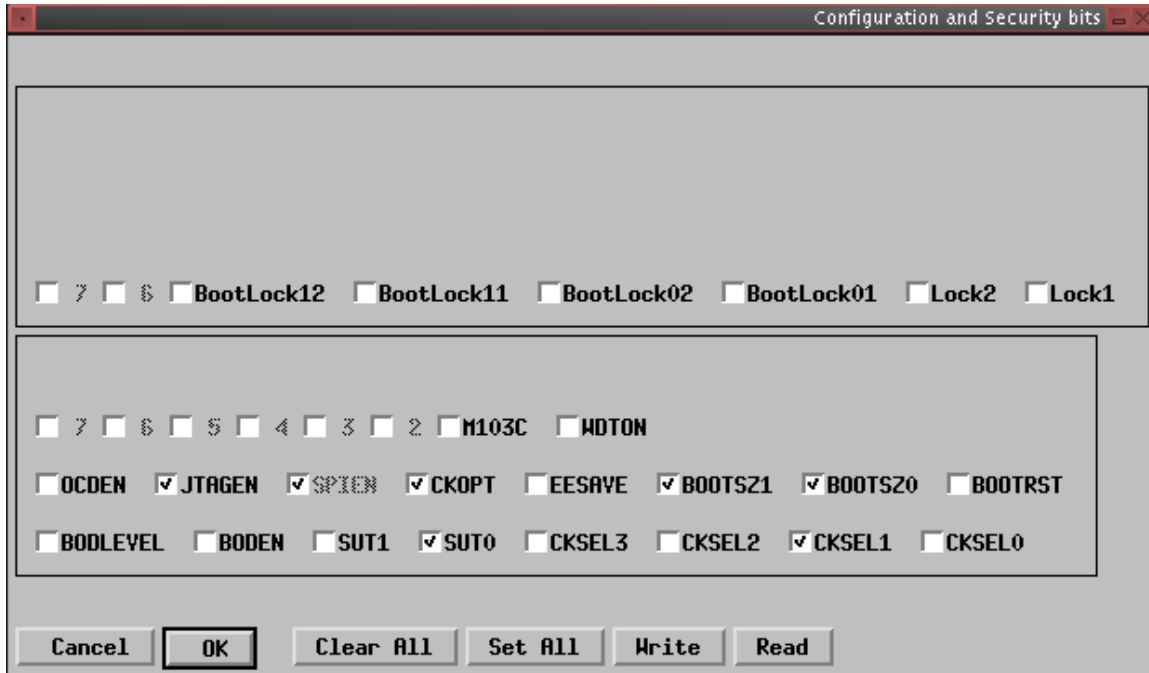


Figure 1: Screen shot of PonyProg2000 fuse setup dialog box

```
S00700006675736545
S1080000ED8999FFFDEC
S5030001FB
S9030000FC
```

For uisp, issue a `--upload` command and use `if=` to specify the modified filename.

Note that you should observe an increase of power consumption once the external resonator is used. Current draw should now be about 70mA to 80mA.

You may want to disable the Mega103 compatibility mode to utilize all features available (such as to turn on the software controlled LEDs).

8.5 Things that can go wrong

- Symptom: the board was accessible when it powers up, but subsequent operations fail. Cause: this is most likely due to the reset pin tied to ground, or it is not pulled up. Certain operations require that the reset pin be toggled to become effective.

9 Coming Attractions...

Three daughter boards will be coming soon. The first one is an LCD/keypad board, the second one is a pulse-width modulated high current driver board, and the third one is a stepper motor control board.