

Atmel ISP Programmer using Pro Micro (or Leonardo)

© 2021 Rick Groome

Overview

This project implements an Atmel In System Programmer (ISP) programmer using only a low cost (\approx \$5-\$17) Pro Micro (Sparkfun DEV-12640 or equivalent clone) (or Leonardo) module with an ATMega32U4 processor chip and software implemented using the Arduino IDE compiler and environment. It is based on a project by Randall Bohn from the 2011 timeframe. It was attempted to use the original code/sketch with the Pro Micro module, but it was quickly determined that it would not work without some modification. There were also a number of features that would be nice to have that were not part of the original design. As mentioned, this project is for the Pro Micro module, but can also be used with the Arduino Leonardo module. Any text that refers to the Pro Micro module implies the Leonardo module also.

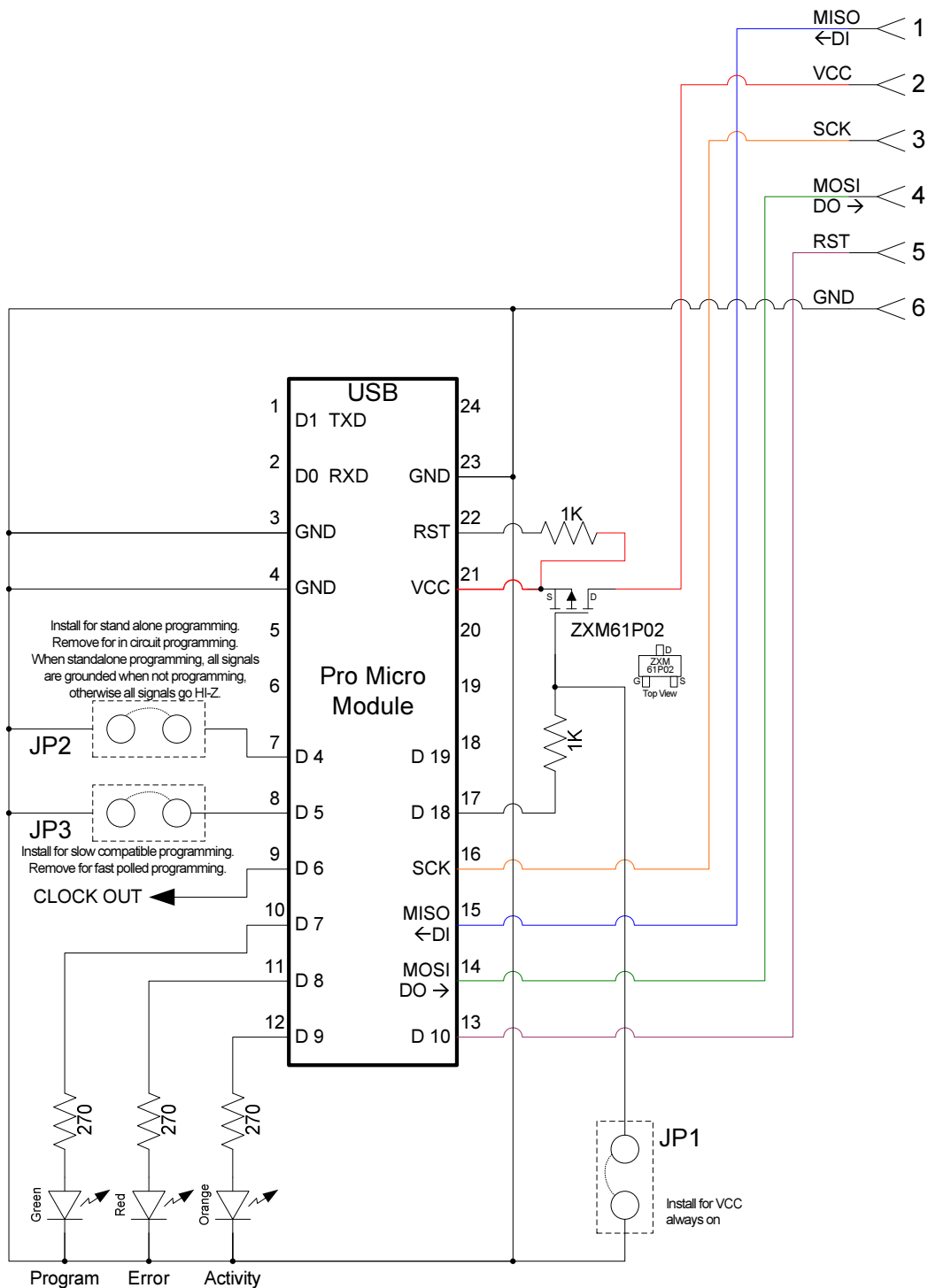
This documentation is divided into a number of sections. The first section shows and details the hardware design. Subsequent sections detail the Arduino code for the host Pro Micro module, a system to integrate the programmer into the Arduino IDE environment, and finally a batch file that can be used as a convenient and simpler interface to the AVRdude software.

Hardware

The hardware used for this project is an Arduino Pro Micro module and wiring to a number of sockets where the chip to be programmed would be installed. Any construction technique that connects the Pro Micro module to the target device is acceptable; however a proposed implementation is shown here.

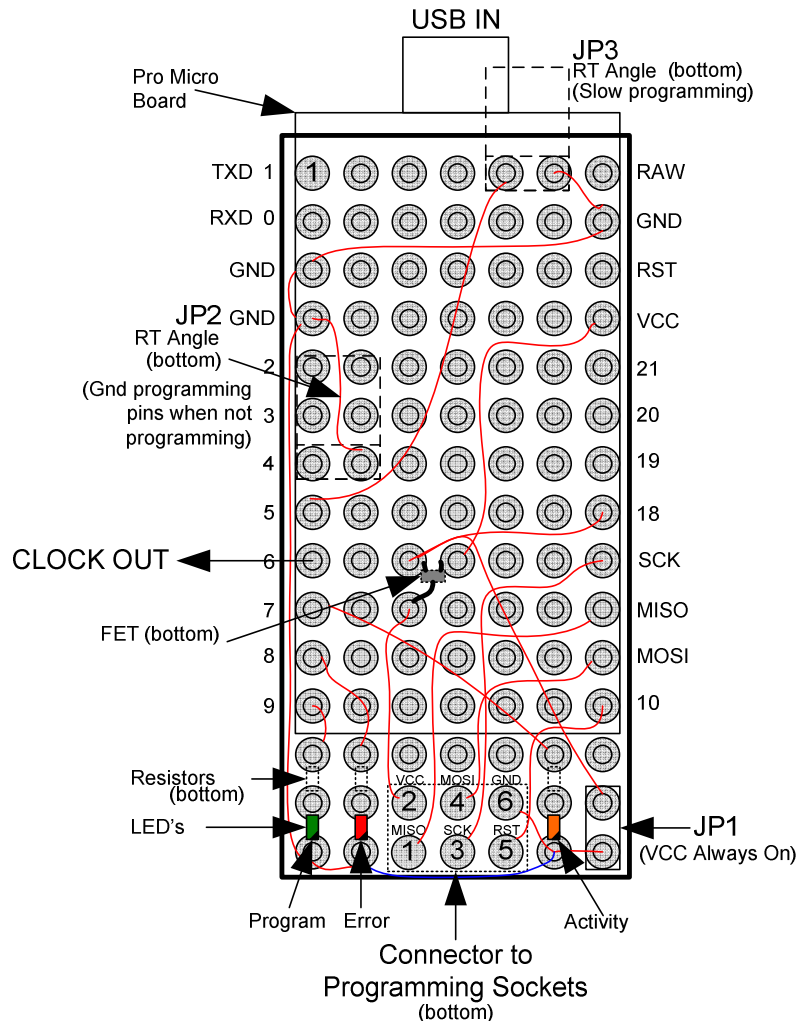
For the system detailed here a couple of small pad per hole perf boards were used with point to point wiring using wire wrap wire – One for the socket for the Pro Micro module and another one to hold either a couple or a number of sockets to put the part to be programmed in. Circuit boards could be created for these perf boards but it is really not necessary as usually only one or two programmers are needed. In the design detailed here the perf board for the Pro Micro module is only slightly larger than the Pro Micro module itself and contains a 6 pin connector that connects to the programming socket board and has the three jumpers and three LEDs on it. It also has a FET power switch for the programming sockets. The perf board for the programming sockets gets its signals from the input 6 pin connector and then is wired to each programming socket implemented on the board. Using this scheme, the programmer can be used to program parts that are not in a system or by just using the programmer module itself, without the programming sockets part, it can be connected directly to an in-circuit system provided the system has the rather standard 6 pin interface used by this design.

The Pro Micro module contains the Pro Micro board and a number of jumpers, LED's and a connector that is to be connected to a target socket board or to an embedded processor. The schematic of this circuit is detailed next. Obviously, the user can choose to include the LEDs and jumpers and the FET switch or not. If not included then just leave them out, however the pin for the JP3 jumper may need to be grounded if not using the polled mode of programming (see next section) for slower target processors or target processors that do not support polling. If the FET switch is not used, then connect the 6pin connector pin 2 to the Pro Micro module pin 21 directly.



The next diagram details the physical layout of the Pro Micro module. It shows a proposed layout and a wiring diagram for the point to point wiring using wire wrap wire. The LED's, resistors and the FET are surface mount parts (typically 0805 parts) while other parts are thru hole parts. Parts shown with dotted lines are located on the bottom of the board, while other parts are on the top. The Pro Micro board itself is set up to plug into this board so that it can be removed if needed. Directly mounting the Pro Micro board to the perf board is discouraged in case it has to be reprogrammed using its serial programming pins.

Pro Micro ISP



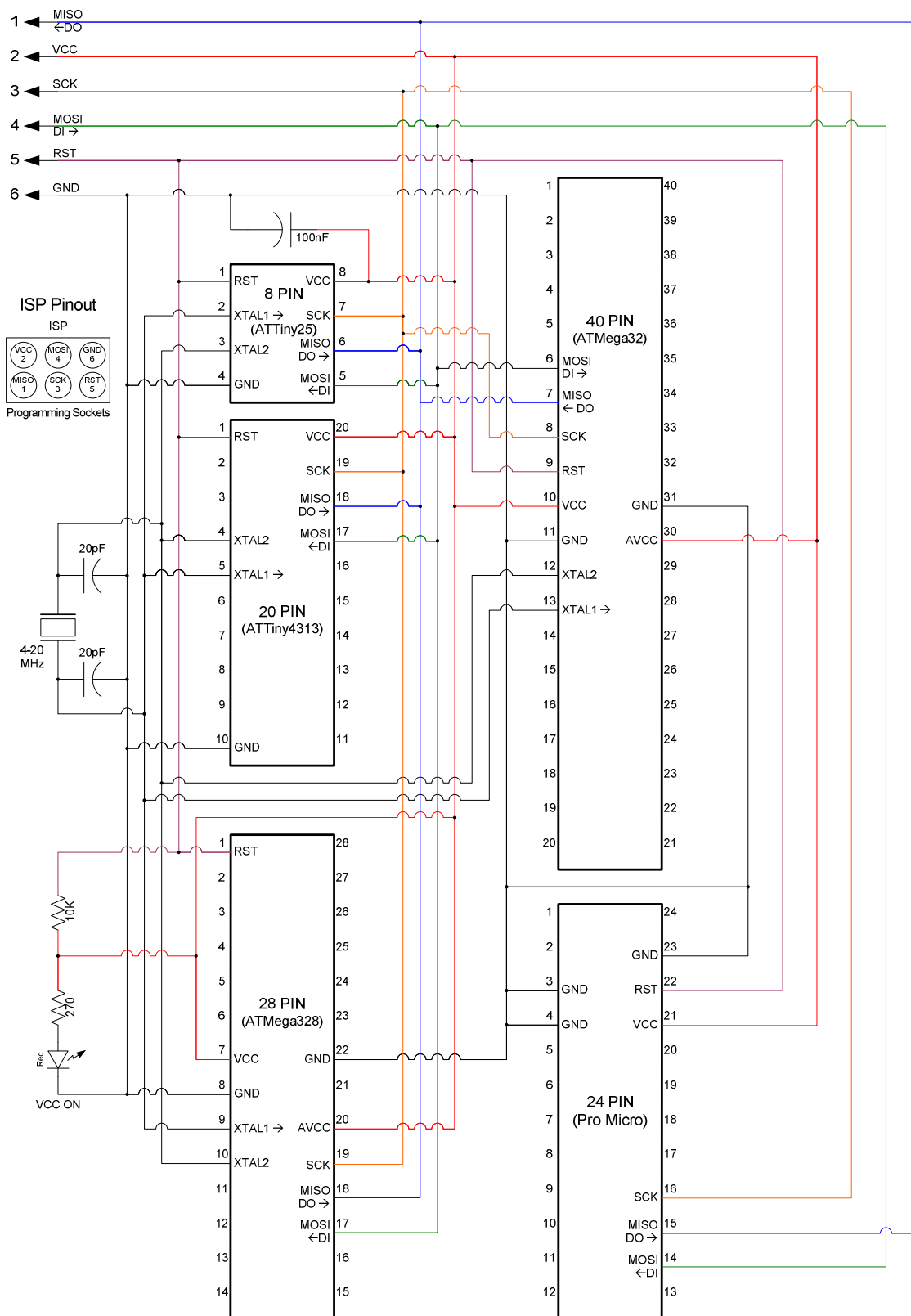
Once the Pro Micro ISP module has been fabricated, the board for the programming sockets can be constructed if out of circuit programming is needed. This can be a “free form” board and need only include the sockets that are needed for the particular application. A schematic showing the interface for 5 different Atmel AVR parts is shown next. This can be adjusted by either removing unneeded sockets or adding sockets for AVR parts not shown here. The construction technique shown here is similar to the Pro Micro ISP board – Perf board and direct point to point wiring using wire wrap wire.

The programming socket board shown here contains a crystal and a couple of capacitors for it. This circuitry is not needed for most applications, but if the chip to be programmed has its fuses (probably inadvertently) set for external oscillator, the part will not program without the crystal oscillator. With the crystal, the fuses can be set for any oscillator type and it will still program. The crystal can be any crystal from about 4MHz to about 20MHz, but slower crystals are preferred so that slower target processors will still work. If the target part is always set to internal oscillator, the crystal is not needed.

There is also a 100KHz clock output from the Pro Micro ISP Module if needed for the target processor, but connection to target processor (if needed) is left to the user. This signal could be used instead of the

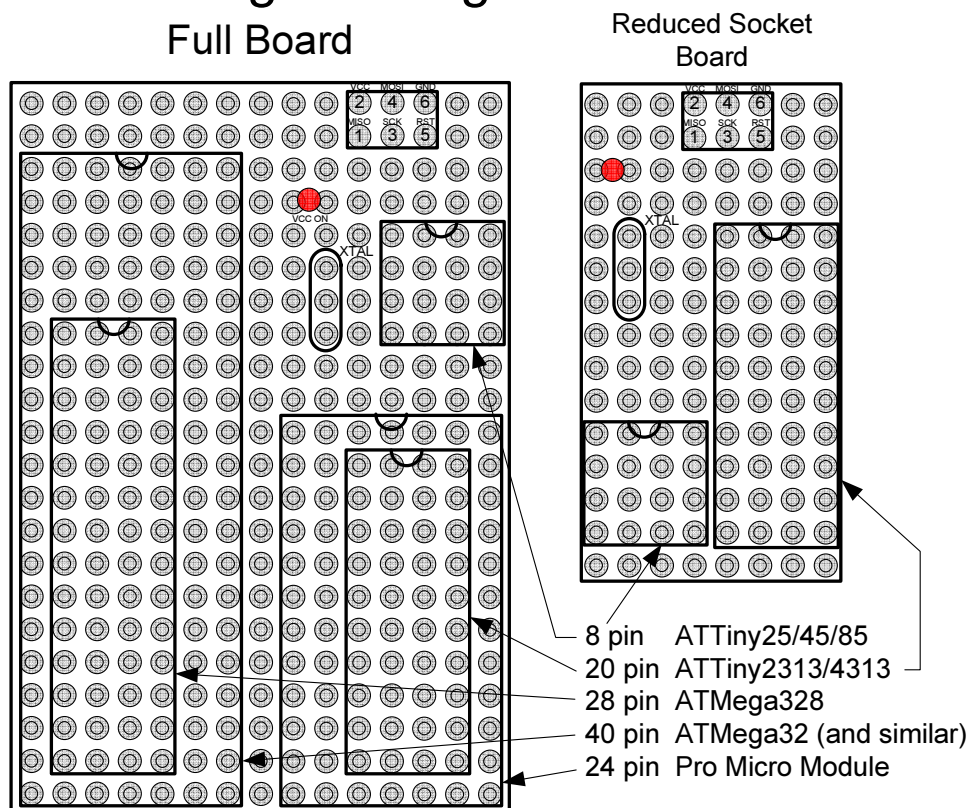
crystal/capacitors and should be connected to the target processor's "XTAL1" or "CLKI" input, if needed to provide a clock signal to the target processor. Like the other programming signals, this signal is active only when the programmer is in operation (from AVRdude or an IDE upload operation).

Programming Sockets



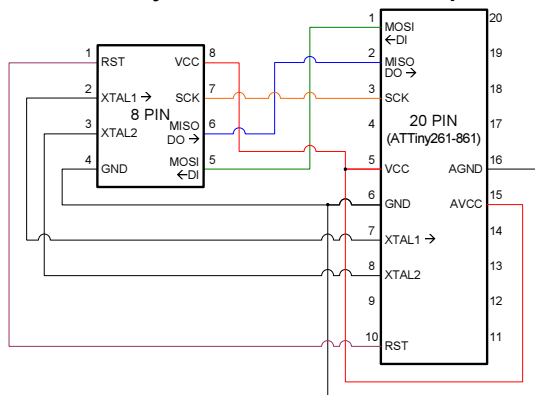
The physical layout of the programming sockets board is shown next. Two boards are shown – One that contains 5 sockets and another that only contains 2 sockets, demonstrating that the implementation of this board is really adaptable to what the user needs. The LED used on this board is an LED with an internal resistor, simplifying the wiring somewhat. In the sketch shown below the 28 pin, the 8pin and the 20pin sockets are IC sockets, while the other sockets and the crystal socket are strip IC connectors like Digikey part 1212-1114-ND so that sockets for some parts can be inside of sockets for other parts making the board size a bit smaller.

Programming Sockets

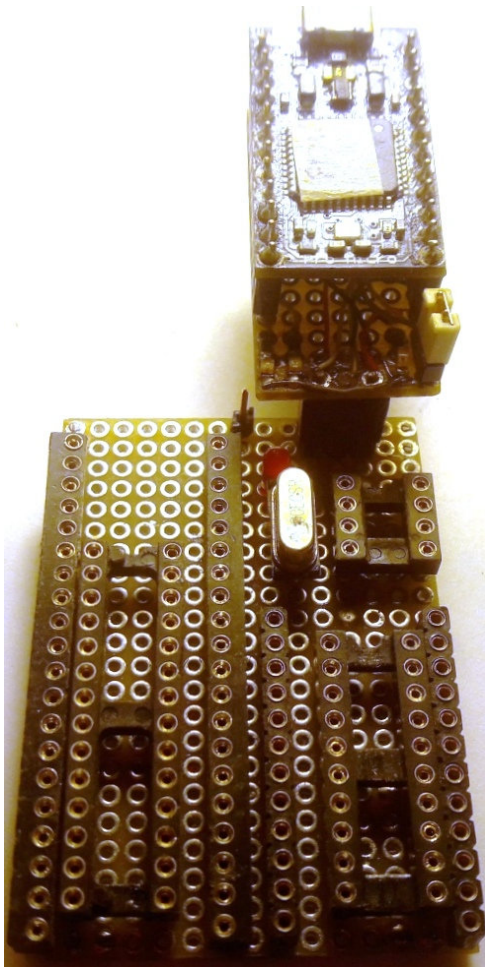


Once a programming sockets board is built, it may later be discovered that some new AVR processor needs to have a socket added. This can be done by creating a different programming socket board or by creating an adapter that converts one of the existing sockets to the new pin-out needed for the new target device. A sample of this is adding programming ability for an Atmel ATTiny261 device. None of the existing sockets matched the needed pin-out, so an adapter was created to connect the ATTiny261 to the existing 8 pin connector. A schematic of such an adapter is shown next.

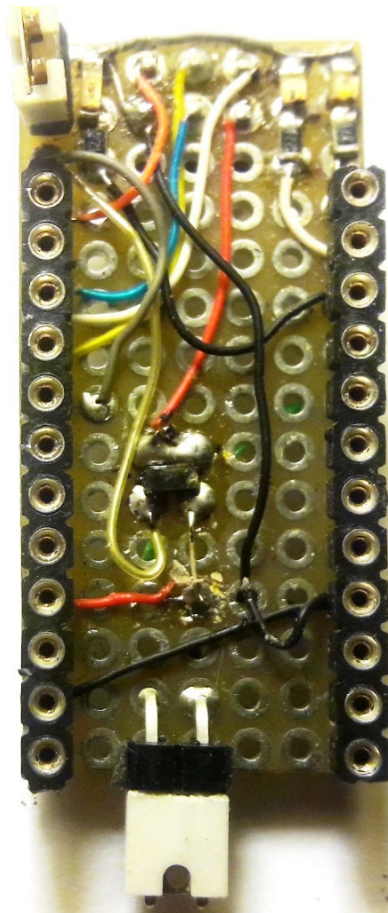
ATTiny 261/461/861 Adapter



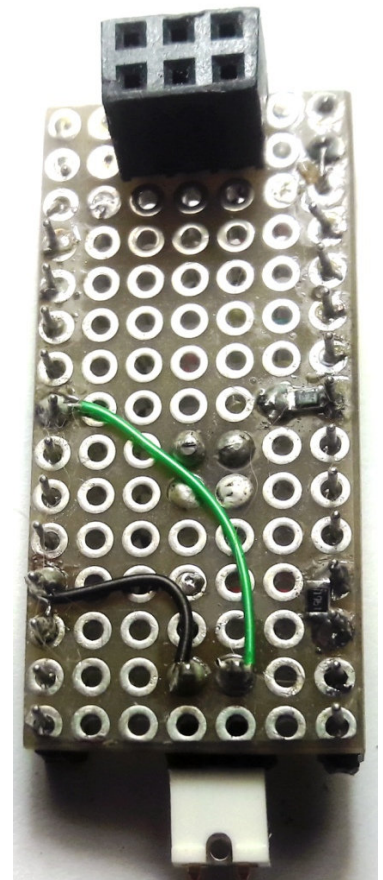
The photos shown next detail the author's construction of the three perf boards constructed by the author. With the diagrams above and the pictures shown it may assist in showing how the boards were laid out.



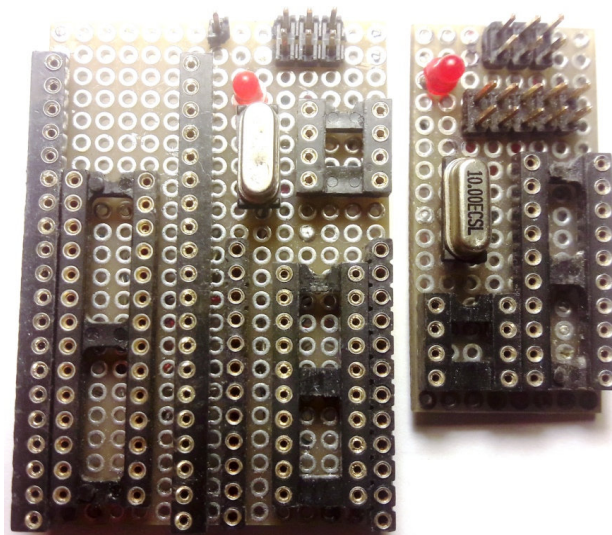
Complete Programmer



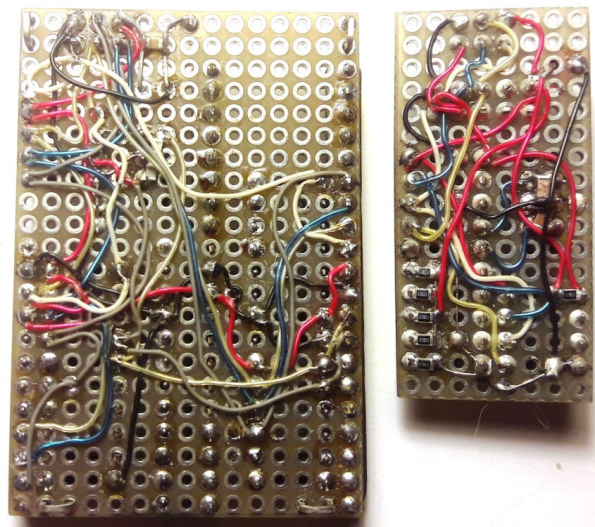
Pro Micro ISP (top)



Pro Micro ISP (bottom)



Programming sockets (top)



Programming sockets (bottom)

Once the hardware is created, attentions can then be turned to the software and using the product.

Arduino Programmer Code


The Arduino sketch/code for this programmer is based upon a work by Randall Bohn back in 2008-2011. It implements a STK-500 like protocol and is used to allow AVRdude (used by the IDE or as a stand alone programmer) to program an AVR microprocessor.

While the original sketch by Mr. Bohn was generic and usable on a number of microprocessor boards, it is not completely compatible with the Pro Micro board. Therefore a number of changes have been made to it and the other support files to integrate this Pro Micro programmer into AVRdude and the Arduino IDE. There have also been a number of enhancements to create a better programmer.

The following changes/enhancements have been implemented in the code supplied in this repository:

1. It communicates with the host computer and the Arduino IDE with the USB virtual serial port. A single USB connection supplies power for the programmer and target device and also is used to communicate with the Pro Micro programmer. This can be changed to use a "real" serial port by changing a #define statement in the sketch, but this is left to the user. The programmer using the USB virtual com port can be used at any baudrate except 1200 baud.
2. The LED outputs are now on different pins. Some of the original pins are inaccessible on the Pro Micro Module. See Hardware section for the pin assignments for this code set.
3. Removed the simulated "heartbeat" code and replaced it with simple on off interrupt code at 1 sec interval on the pin that the LED is wired to (Activity). (The pin that the LED is wired to on the Pro Micro doesn't do AnalogWrite as was implemented in the original code.)
4. Added target socket power on/off. Output of this is hooked to the gate of a P-CH FET, source is connected to power, and the drain is connected to the VCC of target device. The hardware is using a ZXM62P02 P-channel FET for this switch. This allows for the target socket to be "cold" when not programming a device.
5. Added code to hold unused pins of the Pro Micro programmer at GND. Note: This is done with DIO primitive instructions (e.g. DDRB,etc) and is set up for the Pro Micro module only.. Change (in "setup") if using a different device. This makes the programmer a bit more static resistant.
6. Added code for JP2. If JP2 is installed, then all DIO bits go to GND when not programming (e.g. cold socket) and the heartbeat (activity) light blinks faster. If JP2 is removed, then all target socket bits go to hi impedance state when not programming and heartbeat light blinks slower. With JP2 removed, the programmer can be used for in-circuit applications.
7. Reworked the program LED so it doesn't consume more programming time when PROG_FLICKER is true. Now this is done with heartbeat and uses pmode as the state... 0=off, 1=on, 2=blink. The LED will be off when not running, on when reading, and blinking when programming.
8. Added JP3 to allow for faster programming time for devices that will support it. (Replaces SPI_CLOCK with SPI_SLOWCLK and SPI_FASTCLK.) If JP3 is installed, use slower programming time without polling which should be compatible with all devices, even with a slow clock. If JP3 is removed then use faster programming time and poll device to wait for programming page done (if device supports it).

9. Added a 100KHz oscillator using Timer4 that can be used by user as clock input to target micro, in case it's inadvertently programmed to Ext Clk or crystal oscillator. This signal can be applied to XTAL1 or CLKI to provide clock to chip. This signal (like others) can go to GND when not in use (JP2 installed)(NOT in circuit format) or Hi-Z (JP2 NOT installed) (in circuit). Oscillator is completely implemented in hardware, with no software required, other than to set up timer4 to an auto reloading PWM mode.

To program the ArduinoISPPM.ino file into the Pro Micro device double click on the ArduinoISPPM.ino file. This will bring up the Arduino IDE. Then in the tools menu select Board = Leonardo, Programmer = ArduinoISP, and select the “Port” to the comport that the Pro Micro device is located on (These settings work for Pro Micro module also). Once this is done, simply press the Upload button “” and the programming software will be uploaded to the Pro Micro device. Once this is done, the Pro Micro programmer is set up. Install it in the socket in whatever programming hardware has been created.

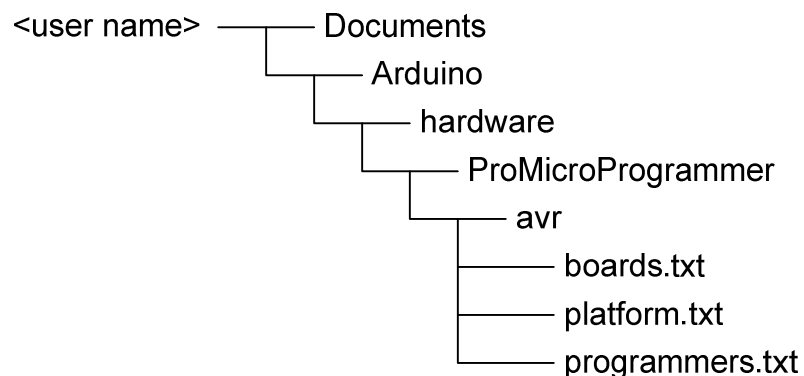
Once the hardware is setup and the programmer controller has been programmed, the programmer can be used to program parts either in the Arduino IDE or with AVRdude and the PGMATMEL batch file program.

Integration into the Arduino IDE

The Pro Micro programmer can be easily integrated into the Arduino IDE environment by simply copying the files and folders in this repository into the Hardware directory of the Arduino folder (e.g. %USERPROFILE%\Documents\Arduino) .

The three files “boards.txt”, “platform.txt”, and “programmers.txt” make up the files needed to integrate the Pro Micro programmer into the Arduino IDE environment.

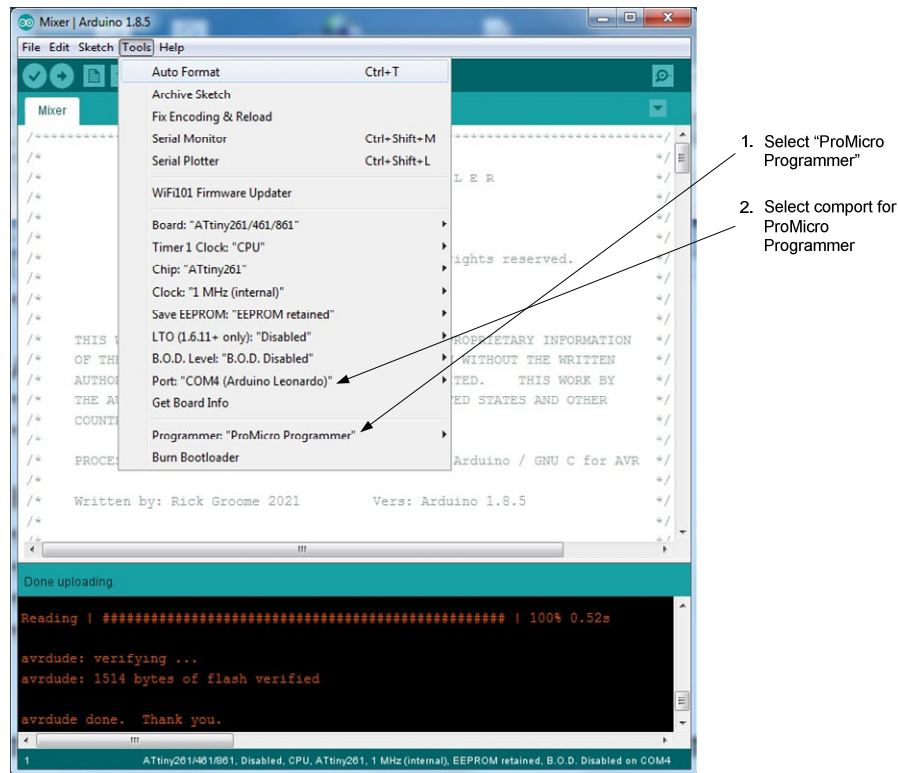
This folder is part of the Arduino IDE and is usually located in the “My Documents” folder within the “Arduino” folder. If a “hardware” folder does not exist in the Arduino folder, then it should be created. The diagram shown next is the hierarchy that the folders and files must be in to allow the Arduino IDE to use this programmer.



The files in the ProMicroProgrammer\avr folder control how the programmer is integrated into the Arduino IDE environment. The “boards.txt” file is basically blank, but is required so that the IDE environment “sees” the programmer. The “platform.txt” file contains information to integrate

AVRdude into the IDE environment. It may need modification only if you have installed the Arduino IDE program in a non default location on the PC. The “programmers.txt” file contains the specific protocol parameters for the Pro Micro programmer.

Once the files/folders are copied onto the PC, then the Arduino IDE can be set up to use this programmer. This is detailed in the next screen shot. (be sure to restart the IDE after copying the files).



Once this is setup, the project can be uploaded to the part by simply pressing the “upload” button within the IDE. This will program flash on the part, but will not program EEPROM memory or the fuses. To program these memory areas (if needed) you can use the supplied PGMATMEL batch file, detailed next.

Batch file for Use with AVRdude

While one can directly use the very excellent program AVRdude to program and read parts, this is not a trivial task. The parameters that it requires to do a specific programming function are long and complex and require that the user know where the two needed files are and the specific format for other parameters. A typical command line for AVRdude can span several lines, depending on what needs to be done.

To ease interface to the AVRdude program a batch file has been created that allows one to program a part with only a couple of simple parameters – The name of the part being programmed and the file name to program the part with. The same batch file can also be used to read a part into an Intel hex file by merely adding the /R switch to the command line. This batch file can be used to completely program a part including all memory sections (flash, EEPROM, fuses, etc) or can be used to just program a portion of it -- Most notably only the fuses, which the version that is integrated into the Arduino environment cannot do.

This batch file is called PGMATMEL.bat. By simply calling it with the name of the chip to be programmed and the file name to program the part with, the batch file massages the supplied parameters to create an AVRDude command line and then calls the AVRDude program to program or read the part.

Using the batch file, the following programmers are supported: The Pro Micro programmer detailed here, the Arduino ISP programmer (a small USB board from Arduino) and the Pro Micro board itself (like uploading a sketch to it). For the Pro Micro board itself the EEPROM and fuses can also be programmed (with caution about setting invalid values that will “brick” the part) or read. See the ‘/S’ option for further details.

In addition to programming the flash portion of the target microprocessor, the PGMATMEL batch file will also write and read the EEPROM memory and write the fuses and lock byte and read the fuses. It looks for a file with the same name as the flash hex file but with the name suffix “_ep” for the EEPROM and a file with the base name of the flash hex file but with the suffix “_f” and extension “.txt” for the fuses/lock byte. As an example, if the Intel hex flash file for programming the flash is called Myfile.hex, then the file Myfile_ep.hex is the Intel hex file for programming the EEPROM and Myfile_f.txt is the text file used to program the fuses and/or the lock byte. When reading a part all of these files are created. When programming a part, if only some of the three files exist, then only those memory areas will be programmed. All fuses except the lock byte are stored in the fuse file (Myfile_f.txt in the sample). Reading a lock byte doesn’t make sense, so it can’t be read, only programmed. A command line call to this batch file can be created that only programs the fuses and lock byte without using a file and specifying the fuse values on the command line. See the samples below for syntax of this operation.

The program is called with the following parameters (Items in [] are optional.)

PGMATMEL [switches] uPName HexFile [switches] [AVRDudeOpts]

where

'PGMATMEL' is this file (PGMATMEL.bat)

'switches' are any of (The switches and AVRDude options can be in any location on the command line) (SOME SWITCHES ARE CASE SENSITIVE)

/? or ? or /H[elp]	Show a detailed help screen. Display pauses after each screen. To output continuously also add the /Q switch as in ‘/? /Q’
/Q	Run quietly. No console output unless error. No prompt before programming. No pause at end of running (unless error). No output from AVRDude unless error.
/q	Run somewhat quietly. No prompt before programming. No pause at end of running (unless error). All AVRDude output included.
/R	Read contents of part into the filename mentioned.
/O	When reading, overwrite existing files without asking.
/T	Enter AvrDude Terminal mode. If specified then other parameters are ignored. (File name not needed but a uPName is required)
/P[COMxx]	Use Pro Micro ISP programmer on Com port xx. (Replace xx with com number, e.g. COM4) If only /P then use the com port # specified as default in batch file. If no /P or /S option then use the Arduino ISP programmer.

/S[COMxx] Use Pro Micro ISP programmer on Com port xx to program itself. (Replace xx with com number, e.g. COM4) If only /S then use the com port # specified as default in batch file. If no /S or /P option then use the Arduino ISP programmer. Do NOT try to reprogram the bootloader section when using this device. (No addresses beyond 0x7000 in flash file.)

/F[Fuses] Fuse values to program the part with. Format is /Flow, high, extended, lock (e.g. "/F0x44,0x55,0x66") where the first value is the fuses for the low fuses, the second value is for the high fuses, and third value is for the extended fuses and the final byte is the lock byte.
If fuses are specified by the /F option they override fuse settings in the fuse file (if it exists).
Fuse values must be in hex and start with '0x'. Use '*' for fuses you don't want to specify (do not use blank). (Example: /F0x01,*,0x03 will program Low and Extended fuses to 01 and 03 but will program the High fuse from the fuse file if it exists or won't program the High fuse if the fuse file does NOT exist.)

'upName' The name of the Atmel chip being programmed / read. This is typically something like ATTINY25 or ATMEGA328 or similar. See AVRdude documentation for exact names. (If an invalid name is entered, the AVRdude program will list all the names when run.) (This is a required parameter)

'HexFile' The file name of the file to program the part with or read the file into. It is assumed to be an Intel Hex file. See filename formats below. If the filename is "NUL" then flash/EEPROM will not be programmed. (This can be used to program fuses via the command line /F switch.)

HexFile is a file path without an extension (.hex assumed). It may be just a file name in the current directory or have a full path and refer to a file in some other directory (as in "C:\SYS\MYSTUFF\MyFile"). This program will use the following file names for each memory area when programming or create these files when reading a part:

(Assumes "Filepath" was the filename entered)

Filepath.hex An Intel hex file for flash memory

Filepath_ep.hex An Intel hex file for EEPROM memory

Filepath_f.txt A text file to program the fuses/lock byte with

The format of the file for programming the flash and EEPROM memory is Intel hex.

The file for programming the fuses is a text file with the fuses listed on a single line, in order and separated by commas. Any fuse that shouldn't be programmed should be '*' – A blank (as in '0x22,,0x44') is NOT acceptable (enter as '0x22,*,0x44'). Extra spaces can be used to separate values, if desired.

When reading a part, each of these three files will be created. If any of the files exist when reading, a prompt will be presented to ask if it's ok to delete/replace the files.

When programming a part, if only some of the three files exist, then only those memory areas will be programmed.

(This is a required parameter)

'AVRdudeOpts' Any other options that may be required for AVRdude program. If the AVRdude option is preceded by '-' then it is placed in the AVRdude command line AFTER all parameters specified by this program. If the AVRdude option is preceded by '--' then it is put in the AVRdude command line BEFORE parameters specified by this program.
Ex: '-v --n' will put '-n' before any parameters specified by this program and '-v' after

any parameters specified by this program. Because some of these options require a space separated parameter, if this is needed, use `-*` or `--*` for the parameter as in `'-B -*20'` for `'-B 20'`. These AVRdude options are rarely needed. (The AVRdudeOpts can be in any location on the command line)

It should be noted that when using the `'/S'` option (Pro Micro programming itself) any code that is downloaded should NOT contain the bootloader code (any code beyond 0x7000) as this will try to overwrite the running bootloader and crash the system. If this is done the only way to fix it is to reprogram the bricked part with another programmer to restore the bootloader section.

Some examples of parameters to this program are as follows:

1. `PGMATMEL attiny261 myfile`
Programs an ATTiny261 with myfile.hex using ArduinoISP programmer. Program will pause before programming the part and upon completion of running.
2. `PGMATMEL /PCOM4 attiny261 myfile /q`
Programs part myfile.hex using Pro Micro programmer on COM4. Program will not output any messages and will not pause at the end of execution (unless error) but will display all the AVRdude messages.
3. `PGMATMEL /P attiny261 myfile /R /Q /O`
Reads part using Pro Micro programmer on default com port into file(s) myfile.hex, myfile_ep.hex and myfile_f.txt. No output unless error. Overwrite files if they exist without asking.
4. `PGMATMEL /P attiny261 NUL /F0x62,*,0xFF /Q`
Program the low and extended fuse with the values 0x62 and 0xFF on the selected part with the Pro Micro programmer on the default com port. This command does not program flash memory or EEPROM and does not require a file to operate (it uses NUL [no] file). No output unless error. Using a command line similar to this can be handy to program the fuses after using the programmer integrated into the Arduino IDE environment.
5. `PGMATMEL /P attiny261 /t`
Run AVRdude terminal program for an ATTiny261 part using the Pro Micro programmer on the default com port. Produces a prompt that will then accept AVRdude commands. (Type '?' for help). End terminal mode by entering 'quit'
6. `PGMATMEL /Q /P attiny261 NUL /F*,*,0x3F`
Program the lock byte with the value 0x3F on the selected part with the Pro Micro programmer on the default com port. This command does not program flash memory, EEPROM or fuses and does not require a file to operate (it uses NUL [no] file). No output unless error. Using a command line similar to this can be handy to lock a part after using the programmer integrated into the Arduino IDE environment.
7. `PGMATMEL /SCOM4 atmega32u4 myfile /q`
Programs part myfile.hex using Pro Micro programmer on COM4 in self programming mode. This is like just uploading code in the Arduino IDE. Program will not output any messages and will not pause at the end of execution (unless error) but will display all the AVRdude messages. Make sure that myfile.hex does not contain code for the bootloader (any data at/above 0x7000 is bootloader code), but only code for the application to be programmed. This is a good way to make duplicate parts without the IDE and also programs the EEPROM, fuses and lock bytes per the file(s) specified.

For use in Windows (no cmd box) a shortcut can also be created to this program and then enter the parameters in the 'Target' box after the name of this batch file.

Example:

TARGET: "<path to batch file>\PGMATMEL" /Q /PCOM3 attiny2313 myfile /F0x33, *, 0x55)

Then by double clicking on the shortcut, the program will run and program the part.

For the filename, either a full path can be used in the "Target" entry above or the 'Start in' box can contain the path to the file to program.

A shortcut could also be created that will ask for parameters by entering the following in the target box:

```
cmd.exe /v:on /c set /P param=Parameters? &&
"%USERPROFILE%\Documents\Arduino\PGMATMEL" !param!
```

(This is all on one line)

One question that comes up repeatedly is "Where can I find the hex files that the Arduino IDE produced when compiling the sketch?" The answer to this depends on the version of the Arduino IDE used, but is typically in the folder "%USERPROFILE%\AppData\Local\Temp\arduino_build_XXXXX" where "XXXXX" is some random number that the Arduino IDE created. If you look in the "temp" folder, the sub directory should be relatively easily locatable. The Arduino IDE MUST be running and the compile button pressed before the file will be available in this temp directory. When the Arduino IDE closes, it deletes this directory. The Intel hex file for the flash memory is typically called something like <projectname>.ino.hex. As far as is known, the hex file for the EEPROM memory is not created by the Arduino IDE environment, but it might be a part of the .elf file located in the same temp directory. The fuses are also not saved by the Arduino IDE. To create these files you can program a part with the Arduino IDE or this batch file and then set the fuses with the PGMATMEL batch file, and then perform a read operation (/R) to read the device into the three files mentioned. After that, all that is needed is the three files and the PGMATMEL batch file to create more target devices.

The batch file has a customization section that may need to be altered to be usable on other PCs and with the COM port that the PC may have assigned the Pro Micro programmer. This section is near the top of the batch file and consists of a few entries:

1. The location of the AVRDude program files
With the supplied batch file it defaults to
DUDESW=%ProgramFiles(x86)%\Arduino\hardware\tools\avr
2. The location and name of the AVRDude program.
With the supplied batch file it defaults to **DUDEPGM**=%DUDESW%\bin\avrdude.exe
3. The location and name of the AVRDude configuration file.
With the supplied batch file it defaults to **DUDECFG**=%DUDESW%\etc\avrdude.conf
4. The default Com port to use with the Pro Micro programmer module.
With the supplied batch file it defaults to **PORT**=COM4
This can be overridden on the command line. If this is set properly it just saves some typing.
5. The names of the memory sections that the PGMATMEL batch file will use for creating the command line to the AVRDude program.
With the supplied batch file it defaults to **FUSESECTS**=lfuse, hfuse, efuse, lock
While these names are suitable for most processors, they may have to be changed if a particular microprocessor has different memory section names in the AVRDude program. It

should be noted that the order that the fuse names are listed in this variable is the order that they will be listed in the fuse text file and should be entered on the command line.

Using this program (batch file) greatly simplifies using AVRDUDE and is relatively intuitive to use, being far less cryptic than using AVRDUDE directly.

Final Notes:

1. When working with AVRDUDE directly, the programmer type used in AVRDUDE should use the programmer type "arduino" (not "stk500v1" or "arduinoisp") because even though this programmer “looks” like a stk500v1 it will not work without the USB DTR signal going hi. (stk500v1 does not raise DTR), which the “arduino” type implements. For the Arduino IDE using the programming hardware files detailed next, this is already setup correctly and is not a concern.
2. Be careful when selecting the programmer type when working on other projects/sketches. If “ArduinoISP” is mistakenly selected as the programmer type, the programming hardware’s Pro Micro module will be programmed and not the target device, as was probably intended. To prevent this, if multiple Pro Micro devices exist, the device used for the programmer can be programmed (as a target device) to have the lock bits set so that it can’t inadvertently be reprogrammed. Otherwise the ArduinoISPPPM sketch can be reloaded in the programmers Pro Micro device to restore operation. Another alternative is to reprogram the programming Pro Micro device without the bootloader so that it can’t inadvertently be reprogrammed via the USB interface.
3. If the fuses are inadvertently programmed to a mode that requires a crystal or external clock input and the programmer socket board does not have a crystal on it or the crystal doesn’t seem to be working, the Pro Micro part being used as the programmer outputs a 100KHz signal on pin Digital 5. This signal can be fed into the external clock input (or XTAL1) to provide a clock source so that the part can be reprogrammed.
4. In addition to having sockets for the parts that are intended to be programmed, it is a good idea to have a socket for a second Pro Micro device. While this is not needed, it can help get one out of a jam if the Pro Micro part being used as the programmer is inadvertently programmed to a state that would “brick” it. In this case one can program the second part with the proper code and then use that device to reprogram the “bricked” part. This should never happen if you never select the ArduinoISP option in the IDE and never use the /S switch in PGMATMEL.bat. (but accidents can happen)

The PGMATMEL batch file and other documentation for this project is on Github at <https://github.com/Rick-G1/Atmel-ISP-Programmer-using-ProMicro>