Programming with AVRISP Mkii

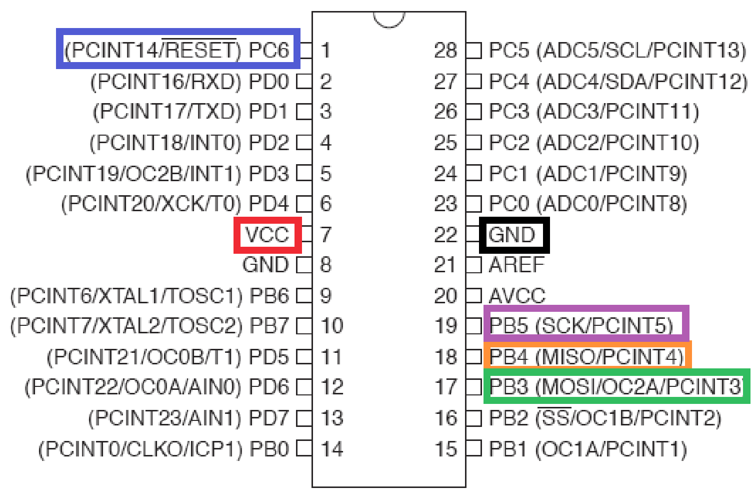
Table of Contents

1. Installing the USB Drivers

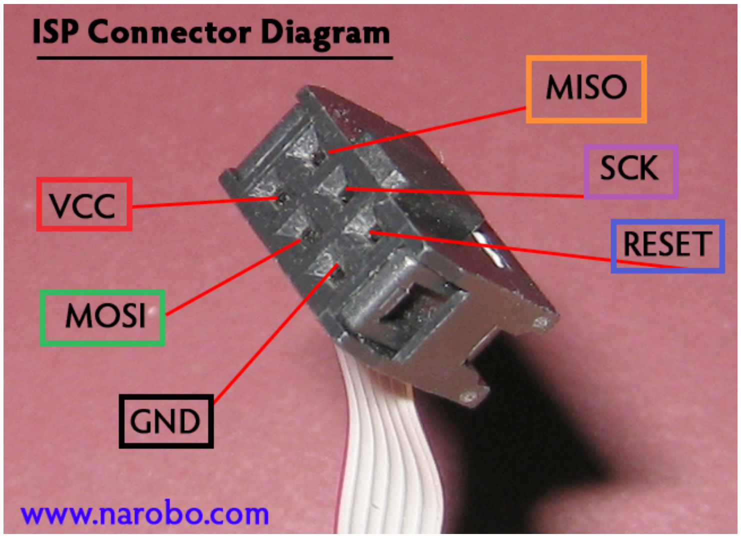
The first step is to install the USB drivers for the AVRISP mkii. This is done during initial install or can be done by modifying the installed program (see guide: USB Driver Installation Guide). Atmel recommends installing the USB drivers BEFORE plugging the USB into the ISP (I didn't experience any problems with plugging it in beforehand other than it not being usable). If the USB drivers are installed correctly, when plugged in, the AVRISP internal led will be solid green, and external led will be solid red. If the drivers are not installed, the internal light should blink once and then stay off.

2. Connecting to a Device

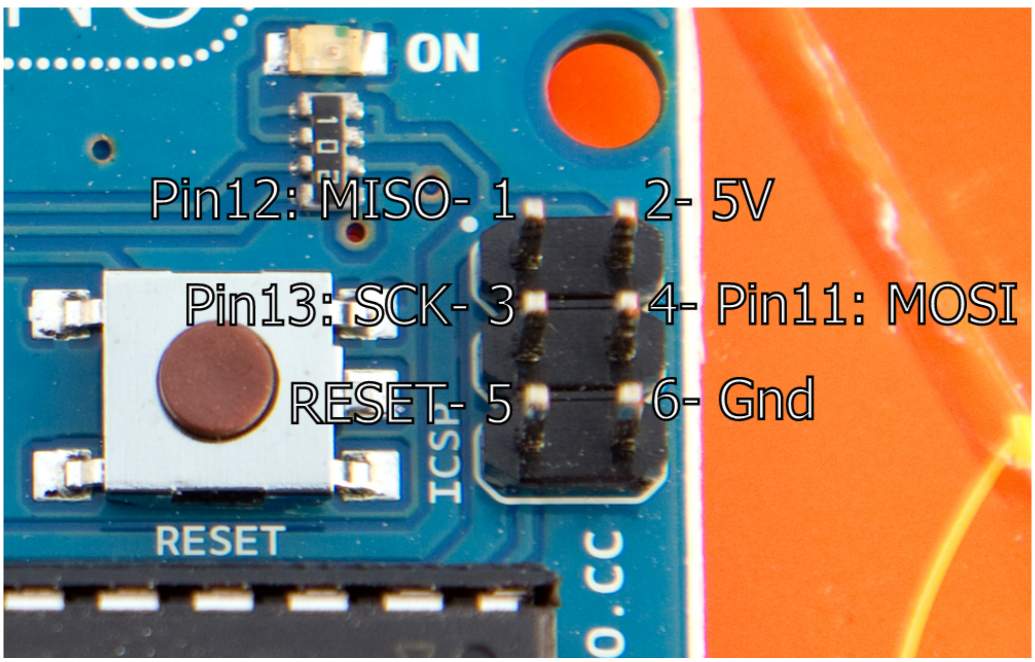
An ISP will communicate with a target device through the SPI using the pins for MOSI, MISO, VCC, GND, SCK, and RESET.



This is the pin layout for a typical DIP AVR microprocessor (ATMEGA 168 or 328) with the connections highlighted to correspond with the following picture. A DIP can easily be programmed on a breadboard as long as you provide an external clock source (16Mhz crystal) and supply voltage to the microprocessor. The AVRISP mkii will **NOT** provide power to the microprocessor.

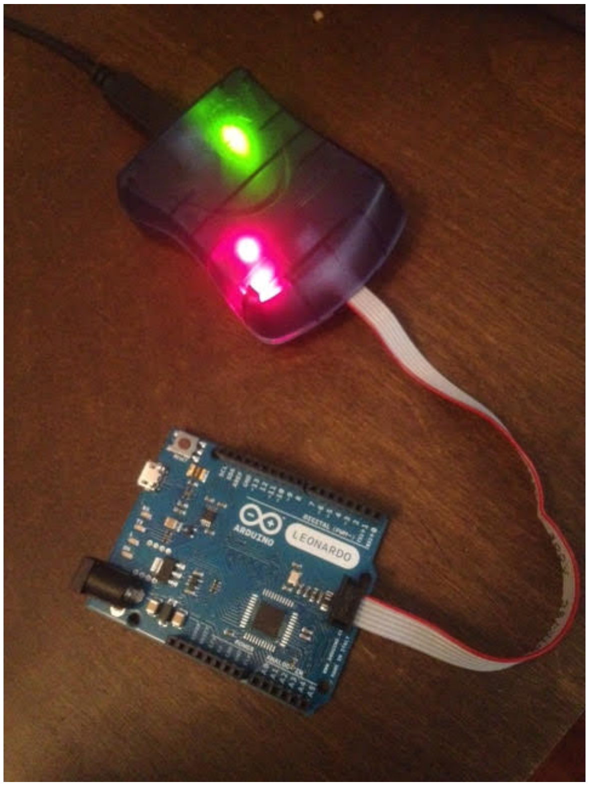
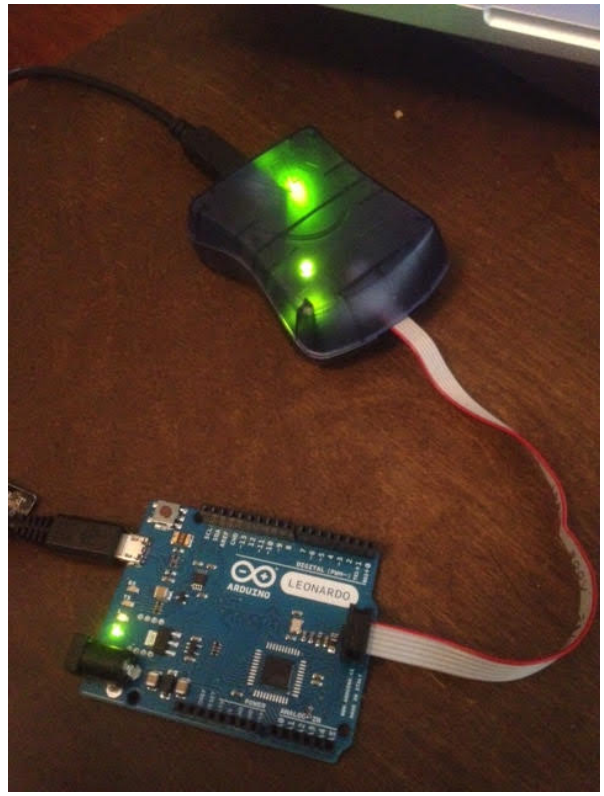


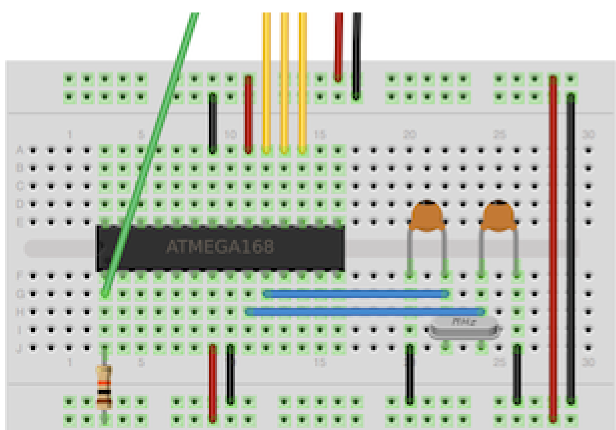
This is the plug of the ISP with the pins marked and color coded. Note one end of the ribbon cable is colored **red** to help with orientation (on the side with VCC and MISO).



Many PCBs that incorporate microprocessors will also include an ICSP header (Arduino Duemilanove shown) which is where ISPs will be connected. This allows a user to reprogram a microprocessor without removing it from the board, and in the case of a surface mounted chip, makes programming much easier. Note the white dot in the corner (next to MISO) which signifies the orientation of the header. The proper connection is made when the red wire on the AVRISP ribbon cable is in the same orientation as the dot. For this picture, the red wire would be on the top and the ribbon cable would extend to the right.

However you have done it, if the target device is connected correctly and powered, the external led of the AVRISP should now be solid green. At this point you can start communicating with the target device provided that it has an external clock source.

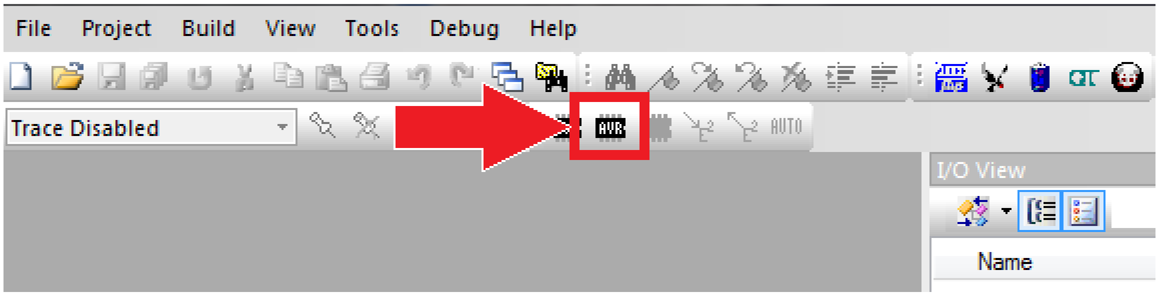
No power :(	Power Provided :)
	



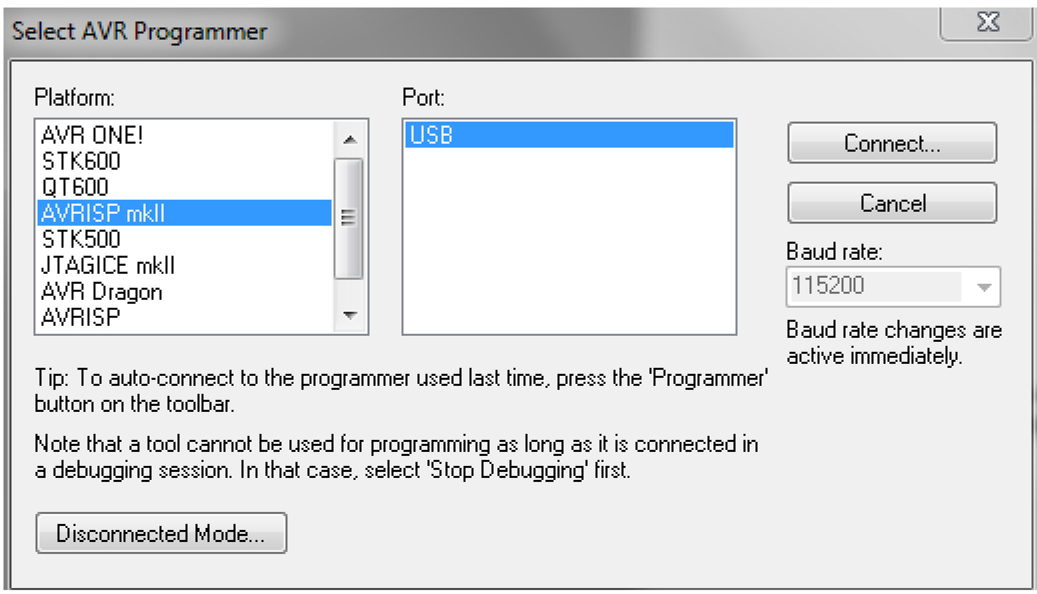
If you are programming a chip on a breadboard, this setup is recommended to provide an external clock to the microprocessor. The wires leading up will connect to the proper ISP pins. The crystal used in this case is 16 MHz and the capacitors are 22pF.

3. AVRStudio4 Environment

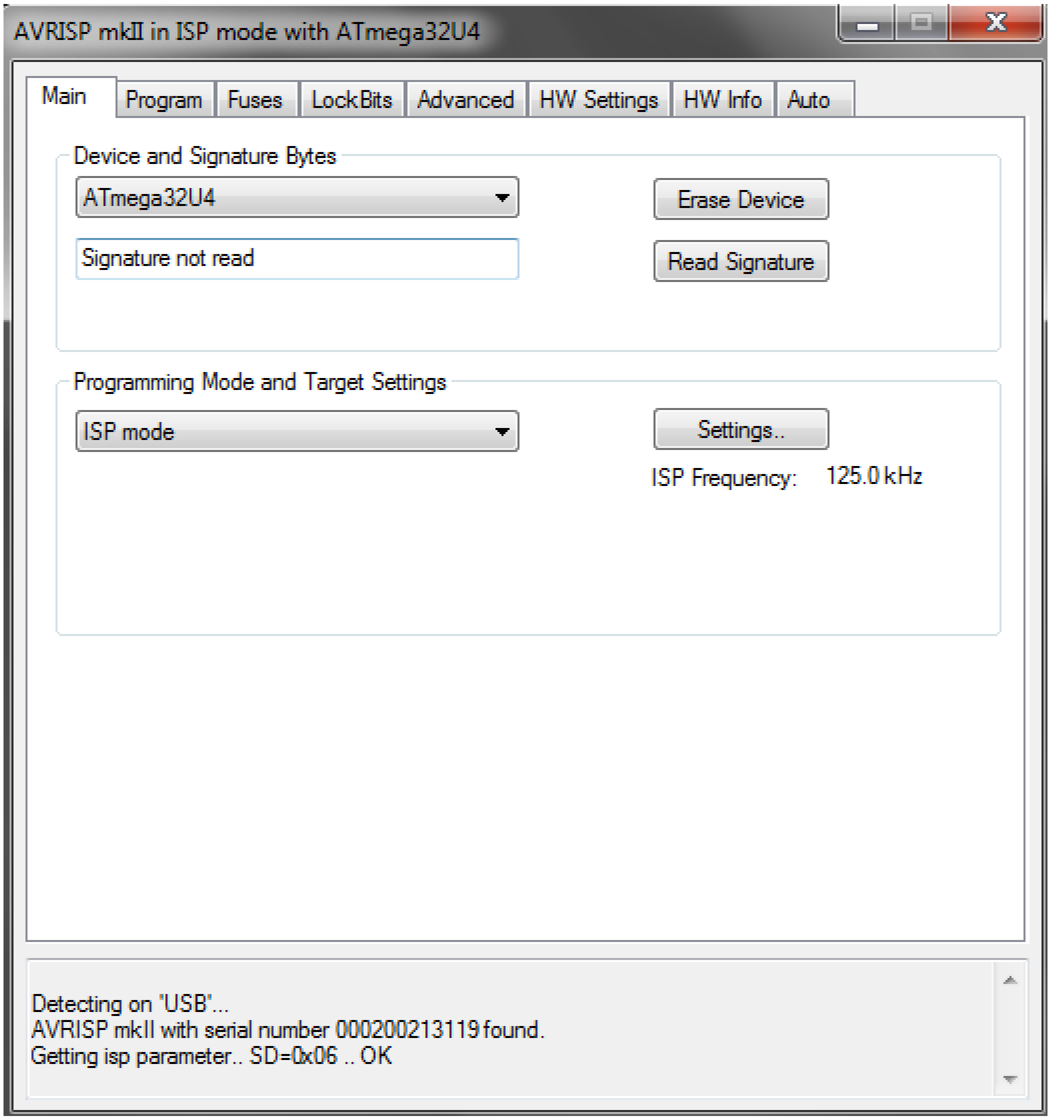
I will provide a small walkthrough for the AVR Studio v4.18.692 environment, but this was the guide that I used initially and provides more detailed information (link below).
http://www.societyofrobots.com/member_tutorials/book/export/html/290



To open the programming window, select **Tools** >> **Program AVR** >> **Connect…** or press the 'AVR' button on the toolbar.



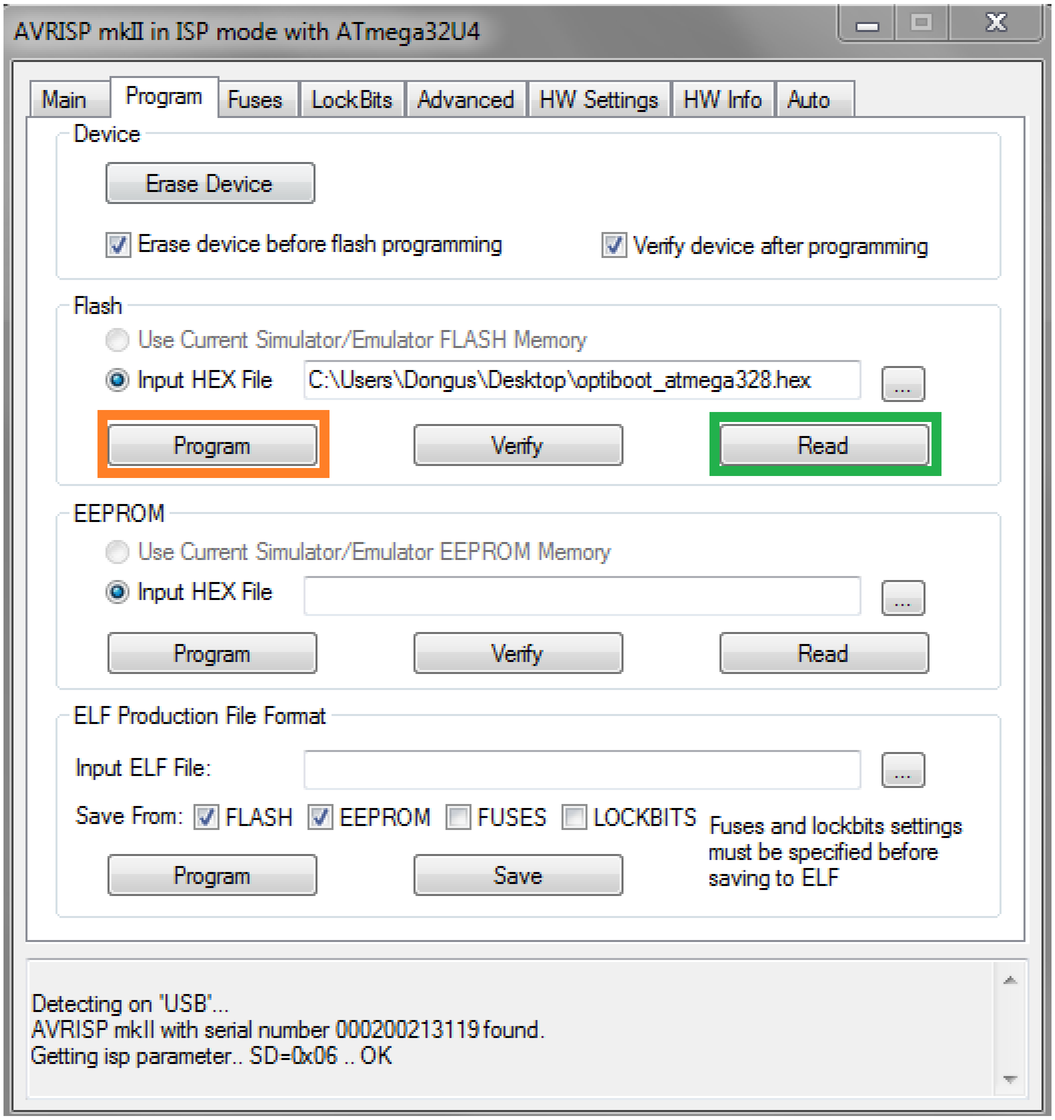
The window that opens will give you a selection of connections. For this example we are using AVRISP mkii through a USB port. Press **Connect…** to continue.



From the 'Main' tab, you can select the connected microprocessor (ATmega32U4 for Arduino Leonardo). From here you can erase the contents of the chip, leaving FPM empty.

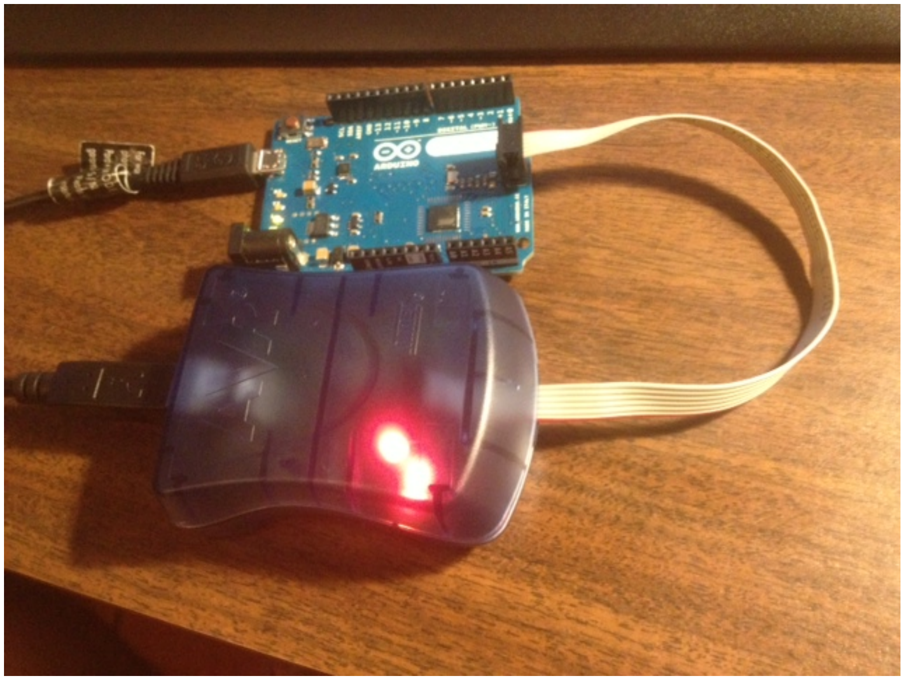
Note: If you are planning to upload through AVRDUDE or use Arduino IDE, you should not erase the chip as this will remove the Arduino firmware, making the chip unresponsive in the Arduino IDE. If you accidentally remove the firmware, I provide a quick guide for restoring it in the next section.

In the **Settings…** you can adjust the ISP frequency. It is required that you use a frequency which is less than ¼ that of the target device. I am using ATMega32U4 which has an internal clock of 16 MHz so the default of 125 kHz is fine. A higher ISP frequency will allow you to read/write to targets faster, but going above ¼ of the target frequency may cause problems with communication.



Under the “Program” tab of the programming window, you may select a hex file to be **written to the FPM of the device**.

Another useful tool is to **read the FPM from the device to a .hex file** which can be opened in a basic text editor. I will provide examples and explanations of reading devices in the last section.



It is kind of hard to tell from the picture, but while the device is being read from or written to, the green internal LED will strobe periodically and the external LED will be solid **orange**. When the process is done, the ISP's leds will both be solid **green**.

4. Restoring the Arduino Firmware

If you accidentally wiped the Arduino Firmware from a chip or would like to teach a chip to be an Arduino, you can use the AVRISP mkii and AVRStudio4 to replace the code. If you happen to have an extra Arduino laying around, you can repurpose it as an ISP and use it to restore the bootloader to another Arduino.

Optimized Arduino Firmware

http://www.societyofrobots.com/member_tutorials/book/export/html/290
This link contains downloads for optimized boot codes (.hex files) for Arduino microcontrollers. They use less room (allowing for larger sketches) and operate at higher baud rates, allowing the uC to boot up and start its program quicker. These are NOT the original Arduino firmwares.

Section 4 of this document (page 7) shows the setup for uploading the firmware to an Arduino device. Make sure that the code you are programming to the board is the correct code for the microprocessor used. Simply select the correct file and then press the **Program** button.

Restoring in Arduino IDE

The original Arduino firmware can be restored in the Arduino IDE using our AVRISP mkii or ArduinoISP if you have a second Arduino board available (link below).
http://arduino.cc/en/Tutorial/ArduinoISP

Note: Using Arduino's setup means that the Arduino used as an ISP WILL provide power to the target device so you should NOT use external/USB power. Doing so may run the risk of permanently damaging one or both of the Arduinos.

5. Reading a .hex File

Most of this information is taken from the Wikipedia article on Intel HEX encoding (link below). The code example is from Arduino Leonardo loaded with the “Blink” sketch and original Arduino firmware.

http://en.wikipedia.org/wiki/Intel_HEX

A record (line of text) consists of six fields (parts) that appear in order from left to right:

1. **Start code**, one character, an ASCII colon ':'.
2. **Byte count**, two hex digits, indicating the number of bytes (hex digit pairs) in the data field. The maximum byte count is 255 (0xFF). 16 (0x10) and 32 (0x20) are commonly used byte counts.
3. **Address**, four hex digits, representing the 16-bit beginning memory address offset of the data. The physical address of the data is computed by adding this offset to a previously established base address, thus allowing memory addressing beyond the 64 kilobyte limit of 16-bit addresses. The base address, which defaults to zero, can be changed by various types of records. Base addresses and address offsets are always expressed asbig endian values.
4. **Record type,** two hex digits, *00* to *05*, defining the meaning of the data field.
5. **Data**, a sequence of *n* bytes of data, represented by 2*n* hex digits. Some records omit this field (*n* equals zero). The meaning and interpretation of data bytes depends on the application.
6. **Checksum**, two hex digits, a computed value that can be used to verify the record has no errors.

Example 1: A single line from a .hex file.

Start code Byte count Address Record type Data Checksum

:10000000 0C9476010C949E010C949E010C949E011C

Byte no.   0 1 2 3 4 5 6 7 8 9 A B C D E F (16 total)

Example 2: Arduino Leonardo with “Blink” sketch programmed.

Start code | Byte count | Address | Record type | Data | Checksum

Code from .hex file	Notes
:100000000C9476010C949E010C949E010C949E011C :100010000C949E010C949E010C949E010C949E01E4 :100020000C949E010C949E010C94B20C0C94390D0E :100030000C949E010C949E010C949E010C949E01C4 :100040000C949E010C949E010C949E010C949E01B4 :100050000C949E010C949E010C949E010C948D04B2 :100060000C949E010C94C6070C9414080C949E01E9 ...	Beginning of File. The Address is the byte address of the first byte and data contains 0x10 (or 16) bytes per line. Data is the code translated into machine code. Record Type 00 is used for data which is loaded to the flash memory.
... :1020700044004552524F523A2054454E53494F4E18 :102080004520332E3356004572726F726520667597 :10209000736500FFD8CBEF0100E1000000000000F5 :1020A00001010000000001107E909B006D706BE06CD :1020B00039073D07000000004009E909D608070973 :1020C000E70830090000000009E0EE909400FFD0EF0 :1020D000F00EFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF10 :1020E000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00 :1020F000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00 :10210000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00 :10211000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00 :10212000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0 ...	This is the end of the code uploaded through AVRDUDE. The program uploaded occupies addresses 0x0000 through 0x20D4 which is 8404 bytes. At the end of the uploaded code, the data is filled with 0xFF which indicates unused space.
... :106FE000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFB1 :106FF000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFA1 :107000000055C000006EC000006CC000006AC00000E7 :1070100068C0000066C0000064C0000062C00000DC :1070200060C000005EC00000F2C400005AC0000052 :1070300058C0000056C0000054C0000052C00000FC :1070400050C0000078C000004CC000004AC00000E2 :1070500048C0000046C0000044C0000042C000001C ...	This is the beginning of the Arduino Boot section. Note that it is located at the end of FPM starting at byte address 0x7000

... :107FD000004C004C004300000000FFFFFFFFFFFFCC :107FE000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFA1 :107FF000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF91 :00000001FF	The final byte address is 0x7FFF which means there are 32,768 bytes or 32KB in the FPM (ATMEGA32U4 used in example). At the end of the file, the Record Type = 0x01, indicating that it is at the end of the file.