Guava Bootloading Documentation

Written By: Team Guava

Last Updated: 1/30/2020

Supplies:

- Arduino Uno
- 1x Micro USB cable
- 1x USB Type A/B cable
- Bare arduino
    - 2x Capacitors (*22 pF*)
    - Crystal Clock (*8 MHz*)
    - 1x Resistor (*100* Ω)
    - 1x Push button
    - Microcontroller of Choice (*ATMEGA1284*)
    - Misc wires

Software:

- Arduino IDE
- Microcontroller library (Mighty)

Getting Started:

Microcontrollers require a bootloader to install programs onto them. Fortunately most microcontrollers come bootloaded after purchasing them, but an Arduino Uno can be used to bootload them yourself. To begin bootloading, you will want to construct your bare arduino with your microcontroller of choice (Team Guava uses the *ATMEGA1284*) or find a working PCB. The Arduino Uno will be used as an ISP (In-System Programmer) to burn the bootloader, and power the circuit. You can follow the instructions on the Arduino website to construct your bare arduino through this link or through the documentation on the wiki.

Configuring the Arduino Uno:

To configure the Arduino Uno for in-system-programming, it will need to be connected to the bare Arduino or PCB that your microcontroller is located on. Bootloading requires the SPI (Serial Peripheral Interface) protocol as compared to I2C protocol, and their differences can be found through this link. For the purposes of this tutorial, we will be using the *ATMEGA1284*, but should still work for any microcontroller with SPI pins. A schematic for the bootloading configuration can be found in Figure 1 below:

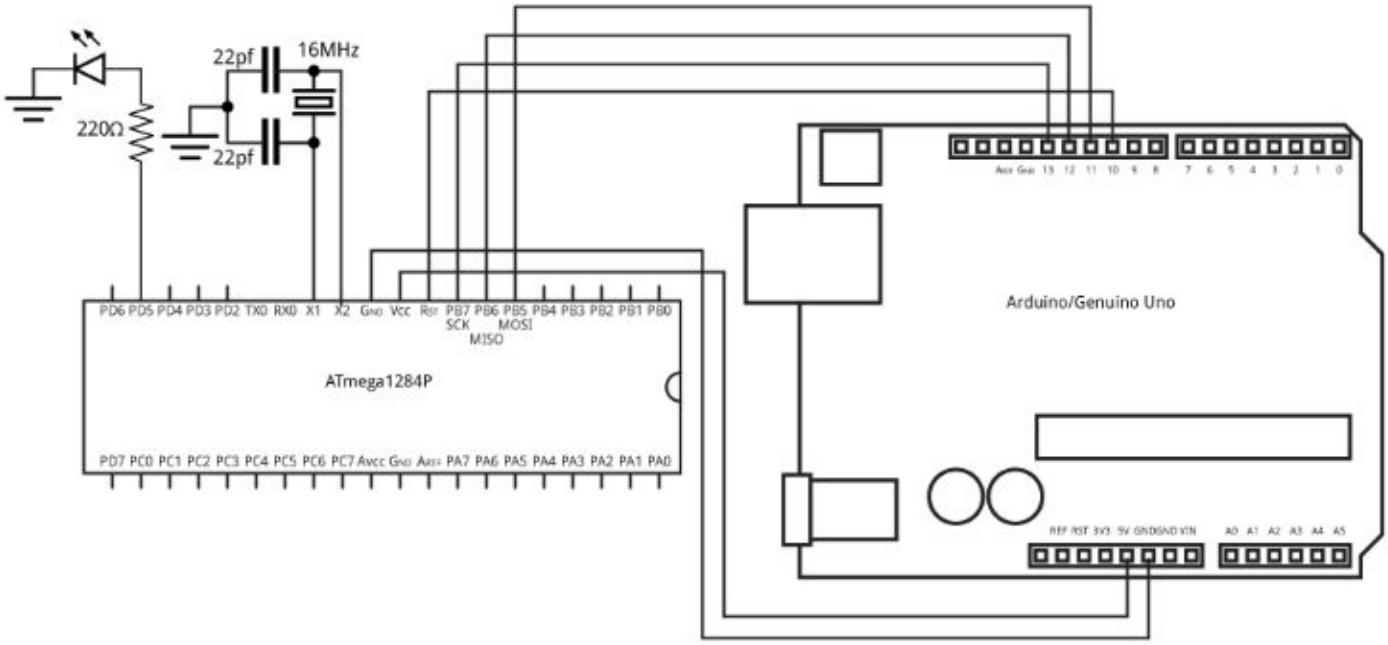

Figure 1: Connecting an Arduino Uno to an ATMega1284 for ISP programming

You may be using a different microcontroller than the ATMega1284 (such as the ATMEGA328), in which the pins will be located in different places. In that case, you will want to follow the following table for connecting the Uno to your bare arduino:

Microcontroller Pin	Arduino Uno Pin
RESET	10
MOSI	11
MISO	12
SCK	13
VIN	5V or 3.3V
GND	GND

Figure 2: Pin configurations for connecting a microcontroller to the Arduino Uno for ISP Programming

Burning the Bootloader:

Once the Arduino Uno is configured for ISP programming, open up the Arduino IDE software and install the core for your microcontroller. You can find the cores from this github repository or this one, depending on which microcontroller you are using, and install them through this tutorial. To verify that it installed properly, check under Tools → Boards and find the submenu labelled, “MightyCore”. Beneath this menu should be the list of supported microcontrollers, where you can check to see if your microcontroller is listed. You will also want to verify that the Arduino IDE is recognizing the Arduino Uno, which can be viewed under Tools → Ports. If the Arduino Uno is plugged into the computer it should appear here.

There are two main phases in bootloading using the Arduino Uno. In the first phase, you will need to upload the ArduinoISP sketch to the Uno. This can be found under File → Examples → 11.ArduinoISP. Make sure the Arduino Uno is plugged into the computer and select, “Arduino / Genuino Uno”, under Tools → Boards. Change the port to match the one that the Uno is connected to under Tools → Ports. We will be using the “AVR mkII” programmer for this first phase, which can be found under Tools → Programmer. Once these steps are verified, upload the ArduinoISP code to the Uno by holding Ctrl + U, or by clicking on the sideways arrow towards the top of the window.

If it does not upload correctly, check the ports and board selection. A screenshot of the correct configuration can be found below in Figure 3:

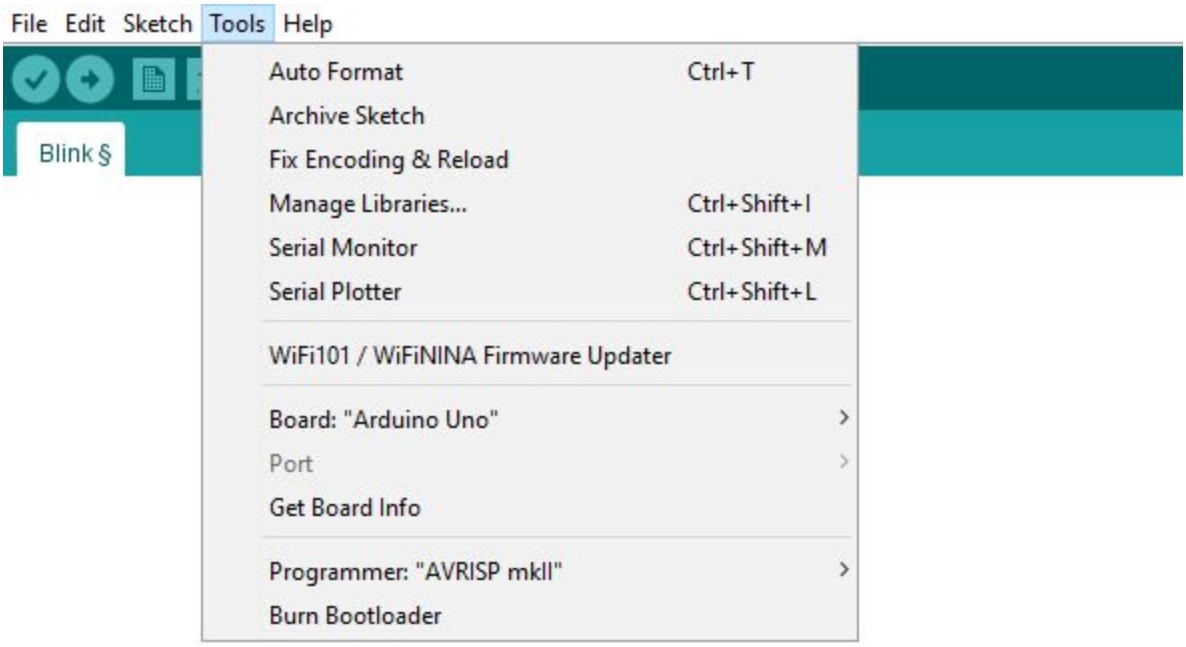

Figure 3: Settings for uploading Arduino.ISP to the Arduino Uno

In the next phase, we will be burning the bootloader onto the microcontroller. We will be using the same port from the first phase, but you will need to change the board to the microcontroller that you are using under Tools → Boards. For the purposes of this tutorial we will be selecting the "ATmega1284" option under the MightyCore submenu. Change the programmer to "Arduino as ISP" under Tools → Programmer so that the Arduino Uno will be used as an ISP. Check to make sure that the other settings such as Clock or Variant match the configuration you are using. The other settings are best left at the default values, but can be changed if you know what you are doing. To burn the bootloader onto the microcontroller, click on, "Burn Bootloader" under the Tools menu.

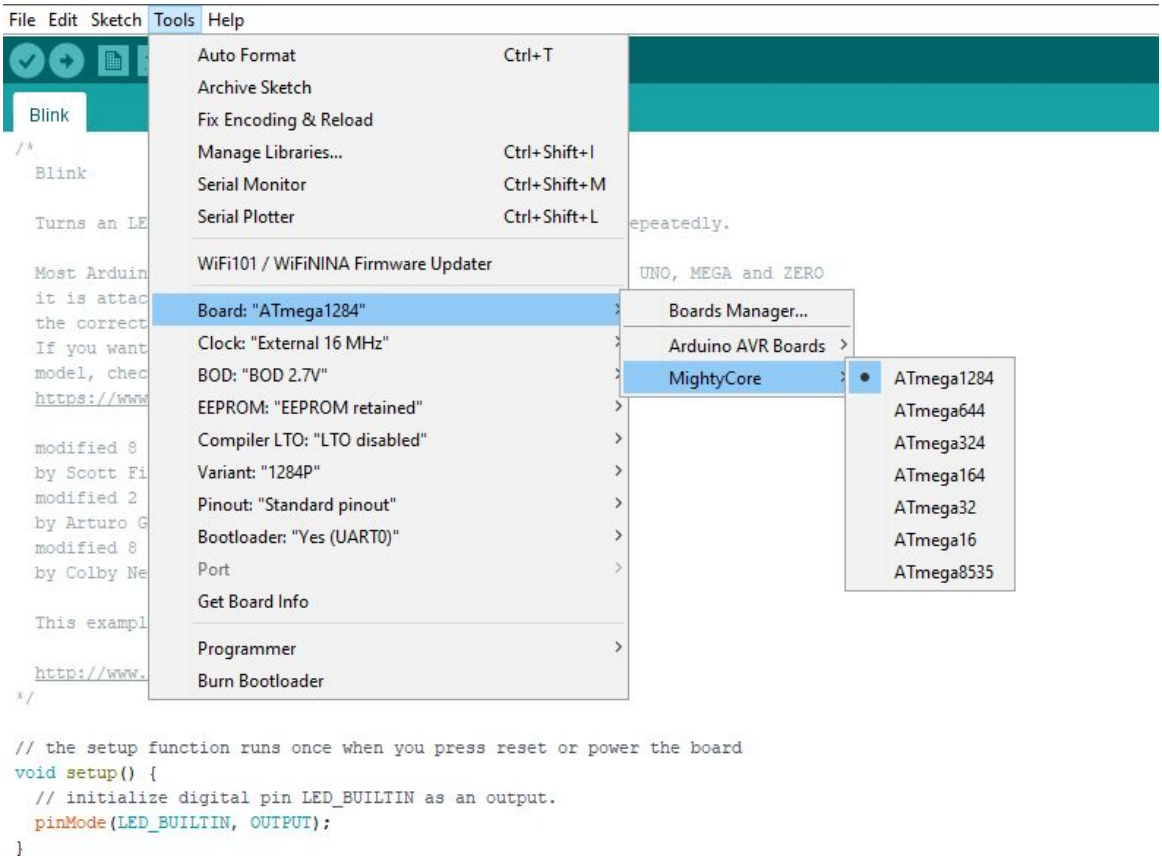

Figure 4: Settings for bootloading the ATmega1284

After this step, your microcontroller should be bootloaded properly. You can test it by uploading a sketch through the FTDI or by pressing the reset button and checking if the LED flashes.

TLDR:

1. Build bare arduino
2. Connect Arduino Uno to bare arduino using table in Figure 2
3. Install appropriate microcontroller core
4. Open ArduinoISP file under File → Examples → 11.ArduinoISP
5. Under Tools make sure the following settings are selected:
    a. Boards → “Arduino / Genuino Uno”
    b. Port → “COM# (Arduino / Genuino Uno)”
    c. Programmer → “AVR mkII”
6. Upload code
7. Change the settings under Tools again to the following:
    a. Boards → “(ATmega####)”
    b. Port → “COM# (Arduino / Genuino Uno)”
    c. Programmer → “Arduino as ISP”
8. Burn the bootloader under Tools → Burn Bootloader

References:

Bootloading tutorial:

- http://www.technoblogy.com/show?19OV#bootloader

I2C and SPI Differences:

- https://aticleworld.com/difference-between-i2c-and-spi/

MightyCore Github:

- https://github.com/MCUdude/MightyCore

MiniCore Github:

- https://github.com/MCUdude/MiniCore

Core Installation Tutorial:

- https://elementztechblog.wordpress.com/2016/10/28/mightycore-an-arduino-core-for-the-atmega16-atmega32-atmega324-and-more/

Standard, Bobuino, and Sanguino Pinouts:

- https://github.com/MCUdude/MightyCore#pinout

Further Readings:

ATMEGA328P Bootloader:

- https://www.circuito.io/blog/atmega328p-bootloader/
- https://www.arduino.cc/en/Tutorial/BuiltInExamples/ArduinoToBreadboard

ISP's:

- https://www.quora.com/What-is-in-system-programming
- https://en.wikipedia.org/wiki/In-system_programming

Communication Protocols (I2C, SPI, UART):

- https://www.seeedstudio.com/blog/2019/09/25/uart-vs-i2c-vs-spi-communication-protocols-and-uses/
- https://www.rfwireless-world.com/Terminology/UART-vs-SPI-vs-I2C.html