Programming Cyduino

In order to use Cyduino (or any similar hardware we have developed), it is necessary to load programs into the Atmega chip on the PCB. The simplest approach is to use the Arduino board as an intermediate programming platform.

1. Plug an Arduino UNO[1] into the computer using the USB interface.
2. Write the code and upload it to the Arduino. (We assume there are no errors.)
3. Unplug the Arduino from the USB interface to remove power.
4. Carefully[2] remove the Atmega chip from the socket of the Arduino board.
5. Carefully plug the chip into the socket on the Cyduino.
6. Apply power to the Cyduino. The chip should boot up and start running.

This works because the program is stored in flash memory on the Atmega. The memory is non-volatile, so the code will stay in place until is overwritten. We can move the chip around as much as we want.

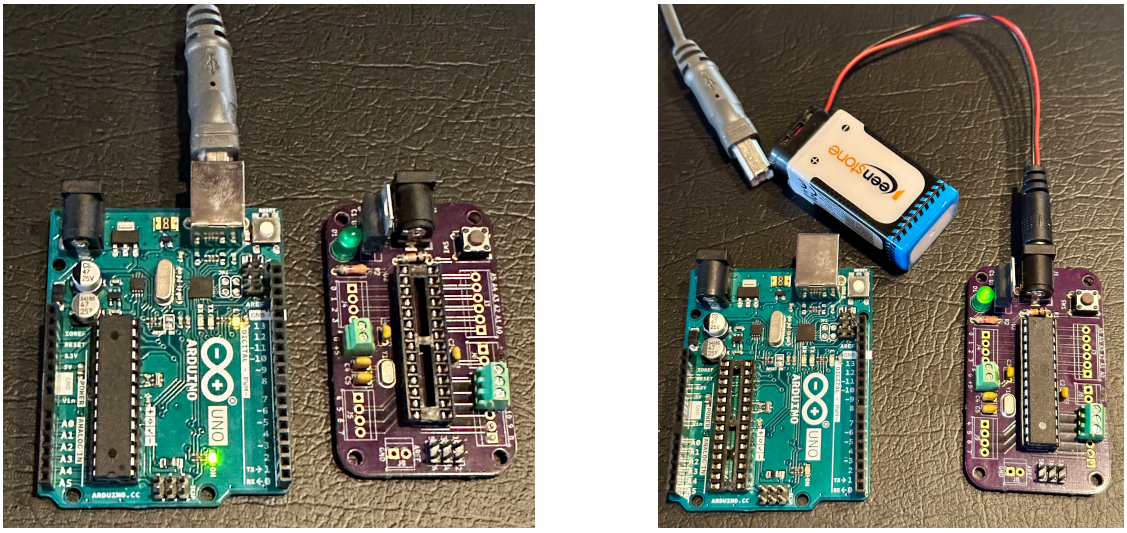


Figure 1. Arduino loaded with code on the left. Atmega chip, with code, transferred to Cyduino.

---

[1] For the descriptions of all the procedures in this document, we assume only Arduino Uno R3 hardware is used. Other types of Arduinos can be used, but the different hardware may require different connections. If using any Arduino other than Uno, RTFM to determine the correct connections.

[2] When removing a large chip, it is easy to bend or break the pins. Or we might fumble around and jam the pins into our fingertips. Use a small flathead screwdriver to pry up one end of the chip until the pins on that end are about halfway out of the socket. Then move the screwdriver other end of the chip and pry those pins part-way out. Go back and forth a couple of times until all of the pins are completely free. When plugging the chip into the socket, spend a minute to bend the the pins as needed so that are well aligned with the socket holes . Then insert the chip slowly. It is very easy to bend pins during insertion.

Boot loaders

The whole point of Cyduino is to allow us to make lots of hardware units without the space and cost overhead of using Arduinos. To make many Cyduinos, we will need many Atmega328P chips. In general, that shouldn't be a problem[3] — Atmega328P chips cost only two or three dollars, so if we can find them, we can stock up.

The problem is that a new, "bare" Atmega328P will not work if plugged into Ardunio board. The bare Atmega chip is missing a key snippet of code, known as a "boot loader". The boot loader code allows the Arduino to communicate through USB interface. The boot loader must be in place *before* the Atmega chip is plugged into the Arduino board. It is a bit of a chicken-and-egg situation — the boot loader (or any other code) cannot be installed until the boot loader is in place.

One approach is to buy Atmega328P chips that already have the boot loader installed. Pre-loaded chips are available from some vendors. Generally, the pre-loaded chips cost and extra two or three dollars more than the bare chips — not a lot, but it adds up.

However, we are striving to be professionals, so we should really know how to handle this boot loader issue ourselves. It is not hard — the Arduino app has all the software needed to install boot loaders, but we need some extra hardware.

ICSP

The extra hardware comes in the form of a "programmer", which is used to upload the boot loader code to the bare Atmega chip. The programmer makes use of the "In-Circuit Serial Programming" (ICSP) interface, also known as the "In-System Programming" (ISP) interface.

An ICSP is a synchronous serial interface, meaning that a clock signal is sent to keep the transmission synchronized between the sending and receiving devices. The transmission channel requires four connections, MOSI, MISO, SCK, and ground. The standard six-pin ICSP interface also includes a power connection (typically 5 V) and a "chip select" connection. A six-pin ICSP pin header is located near the Atmega chip socket on both the Arduino Uno and the Cyduino.

The ICSP is used to connect the "programmer" or "controller", to a bare Atmega chip, which we will call the "target". Then the Arduino IDE can upload the boot loader code through the programmer to the target.

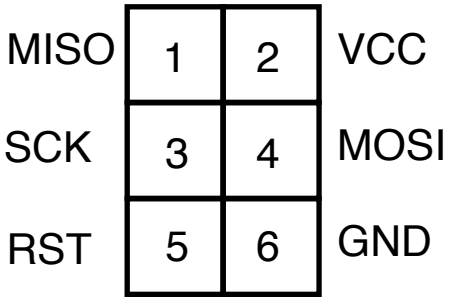


Figure 2. ICSP header pins viewed from above.

[3] Except when we are still in the throes of a Covid-induced semiconductor shortage.

Table 1. ICSP pin names[4], functions, Arduino board connections, and Atemga328P chip pins.

ICSP pin	ICSP names	ICSP function	Arduino connection	Atmega pin
1	MISO	Programer In, Target Out	12	18
2	VCC	Power	5 V	7
3	SCK	Clock for ICSP	13	19
4	MOSI	Programmer Out, Target In	11	17
5	RESET	Chip reset	Reset	1
6	GND	Ground	Gnd	8 or 21

Using the Arduino as a programmer

Most ICSP programmers are made using microcontrollers. Fortunately, we have a ready-made, general-purpose microcontroller platform at our disposal — we can use an Arduino as a programmer! It has all the right hardware and the Arduino developers have helpfully provided code that will make the Arduino function as an ICSP programmer.

If we are lucky enough to have two Arduinos available, we can set up one as the programmer and use the other to hold the target chip. If we have only one Arduino available, we can use it as programmer and then put together a bread-board Cyduino for the target chip. We could also use a Cyduino PCB to hold the target chip to receive the boot loader, but as we will see shortly, the idea of putting boot loaders onto Cyduinos is actually pointless.

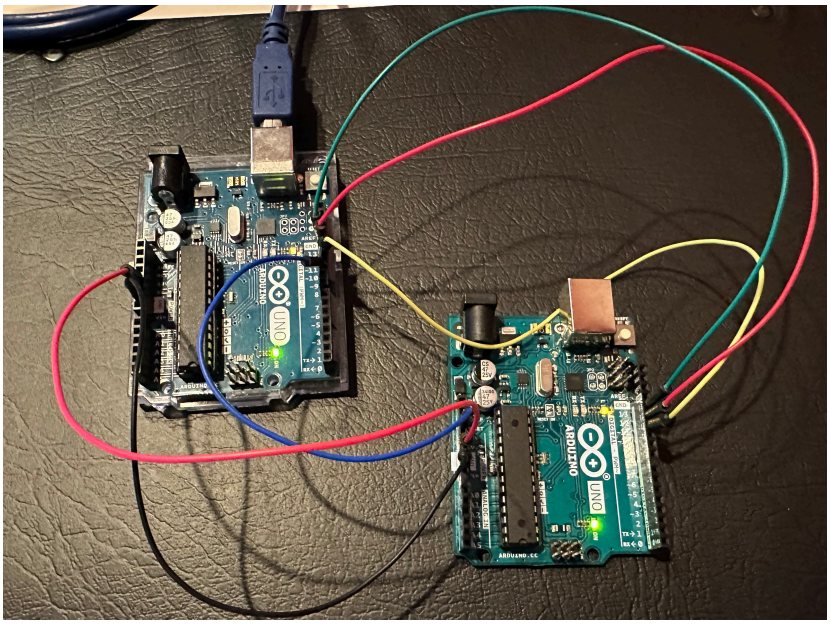


Figure 3. Using one Arduino as the programmer and a second Arduino to hold the target chip.

[4] The “M” and the “S” in the original ICSP naming convention stood for “master” and “slave” — terminology that we will discard.

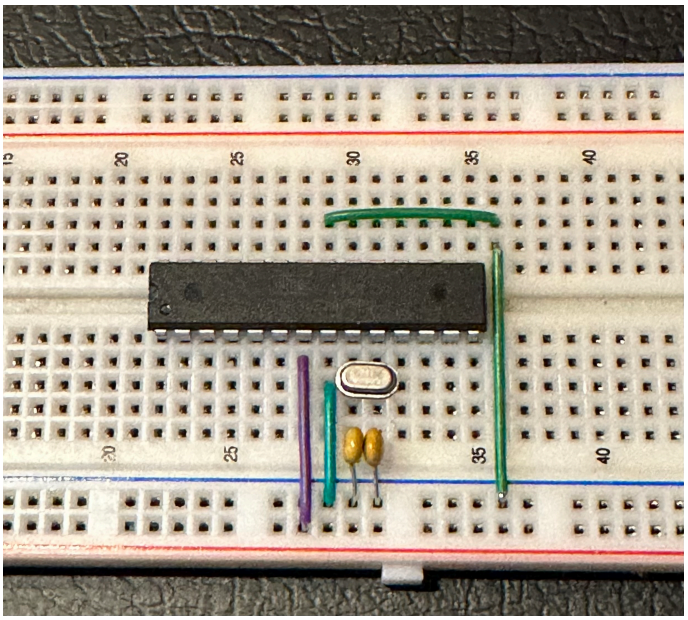
Using two Arduinos

1. Choose one Arduino to be the programmer. The other will obviously hold the target chip.
2. Connect programmer Arduino to the computer using the USB connection.
3. Under the Tools menu, select "Arduino Uno" as the board:
   Tools → Board → Arduino AVR boards → Arduino Uno.
   (It was probably already selected.)
4. Use the file menu to open the ArduinoISP program:
   File → Examples → 11.ArduinoISP → ArduinoISP.
   Upload the ISP program to the programmer Arduino.
5. Under the Tools menu, choose "Arduino as ISP" to be the programmer:
   Tools → Programmer → Arduino as ISP.
   (Not ArduinoISP! This is a bit confusing, but choosing ArduinoISP will not work.)
6. Remove the Atmega chip from the socket of the target board.
7. Install the bare (no bootloader) Atmega328P chip into the socket of the target board.
8. Make the following connections:
   - Connect a GND pin of the programmer Arduino to a GND pin of the target Arduino.
   - Pin 11 of the programmer to pin 11 of the target. (MOSI)
   - Pin 12 of the programmer to pin 12 of the target. (MISO)
   - Pin 13 of the programmer to pin 13 of the target. (SCK)
   - Pin 10 of the programmer to the RESET pin of the target.
   - The 5 V pin of the programmer to the 5 V pin of the target. The "power on" LED of the target should light, indicating that the target is ready to go.
9. Back in the IDE on the computer, choose Arduino Uno as the board. (Same as step 3, except that now it is referring to the target board. Again, we are assuming that only Uno boards in this exercise, so the board setting should not have changed.)
10. From the Tools menu, choose "Burn Bootloader". It should take a second or two to transfer the boot-loader code to the target. There should be some furious LED blinking on the boards, we should receive the "Done burning bootloader" message in the IDE output window.

If there is a failure, recheck each of the steps above. In particular, make certain that the programmer code is loaded into the programmer board, "Arduino as ISP" is chosen as the programmer, and all of the connections are correct as described in step 8.

Using one Arduino and a bread board

If we have only one Arduino, we will need to cobble together a board to hold the target chip — easy to do if we have read the description for building Cyduino. The target board must have a oscillator — required a crystal and the associated capacitors — and all the connections to the programmer board as described above.



a. Insert the bare Atmega328P chip into the bread board. Connect a 16 MHz oscillator crystal between pins 9 and 10 and connect 22-pF capacitors between pin 9 and ground and pin 10 and ground. Connect pin 8 to the 5-V rail. Connect pins 9 and 22 to the ground rail.

b. Repeat steps 2 through 5 from the "two-Arudinos" instructions above.

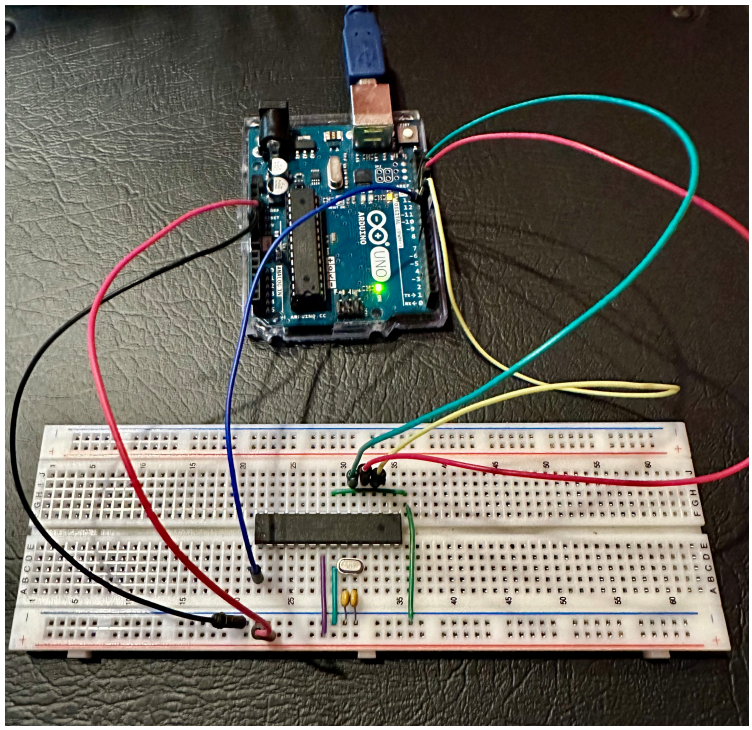
c. Make the following connections:

- GND pin of the programmer Arduino to the ground rail of the bread board.
- Pin 11 of the programmer to pin 17 of the target chip. (MOSI)
- Pin 12 of the programmer to pin 18 of the target chip. (MISO)
- Pin 13 of the programmer to pin 19 of the target chip. (SCK)
- Pin 10 of the programmer to the pin 1 of the target chip. (RESET)
- The 5 V pin of the programmer to the 5-V rail of the bread board. Power should be applied to the target and the chip should be "running". (We can check the voltages with a multimeter to ensure that connections are correct.)

d. Repeat steps 9 and 10 from the "two-Arduino" instructions above.

If all went well, we should see the "Done burning bootloader" message in the IDE output window. Now the chip with the freshly loaded boot loader can inserted back into an Arduino board, connected to the computer, and code for the desired application can be uploaded as usual.

Figure 4. Using an Ardino as the programmer and a bread-board Cyduino as the target.



Programming the chip directly

By using a programmer, we are able to upload boot-loader code onto a bare Atmega chip, which can then be inserted into Arduino board to be programmed in the usual fashion. But that brings up an obvious question: Since the boot loader is just a piece of code loaded into the chip's memory, can we upload other code to the chip? Could we upload the code for whatever application we are developing directly to the chip. The answer is Yes!

The programmer will upload whatever we want from the Arduino IDE. The process for uploading other code is nearly identical to the that for uploading the boot loader. First, make certain that the code to be uploaded is in the front-most active window in the Arduino IDE. Then, instead of burning the boot loader code as the final step, we instead choose the menu item: Sketch → Upload Using Programmer. The code is sent to the target via the Arduino serving as an ISP programmer.

Note: Do not choose "Sketch → Upload" or use the upload arrow in the tool bar of the IDE window. Those commands will upload the code to the Arduino, not to the target. Of course, if that happens, the "ArduinoISP" code that was in the Arduino is overwritten. It will be need to re-installed into the Arduino before it can be used as a programmer again. It is annoying when we make this mistake, but not catastrophic.

Programming the chip directly provides a couple of distinct advantages:

- No boot loader is required. The boot loader is only needed in order to interface the Arduino board to the computer using the USB interface. If we program the chip directly on the PCB (or bread board) there is no need to use the Arduino to write the program to the chip. By omitting the boot loader, we gain the memory that was used to store it. (The boot loader does not require much memory, but for some larger applications, every little bit can help.)

- There is no more infernal “chip swapping”. We can install the Atmega chip into its intended place and leave it there. Swapping chips takes time and poses the risk of damage — to the chip itself and to our fingers.

Note: If new code is uploaded directly to an Atmega chip, any previously stored code is overwritten, including any boot loader code. So if we upload code to a chip, and then want to transfer the chip back to an Arduino, we will need to re-install the boot loader *before* moving the chip to the Arduino.

And now we see why it is pointless to upload boot loader code to a chip on a Cyduino board. Why bother with it when the desired code can be sent directly?

Using a dedicated programmer

The idea of programming a microcontroller chip directly is very appealing, so much so that we may want to make a dedicated programmer. Now that we know more about how a programmer works, we could consider assigning one Arduino to exclusive duty as programmer and rig up some sort of wiring harness to make the connections easier to handle.

Or we could buy a dedicated programmer. There are many available — every microcontroller manufacturer provide some type of hardware interface so that their own chips can be programmed. A simple programmer that works with many types of microcontrollers is the USBtinyISP, which started as an open-source project. It is made to work with all AVR microcontrollers and it is supported by the Arduino IDE. Versions of the hardware are available from many places, DigiKey, Amazon, DFRobot, eBay, etc. Adafruit sells a kit to be soldered together, much like an Arduino Club project[5].

The USBtinyISP has a USB interface to connect to the computer, and it comes with a ribbon-cable for making direct connections to the ICSP header on the target. The microcontroller is an Atmel ATtiny, which is a small AVR controller that interfaces completely with the Arduino IDE. The programmer code should come be pre-installed on the ATtiny, unless the programmer is being built form scratch.

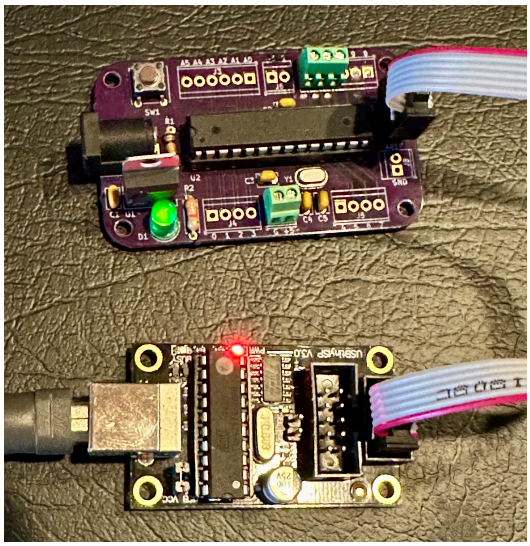


Figure 5. USBtinyISP (bottom) as the programmer uploading application code directly to the target on Cyduino board.

[5] In fact, an excellent Arduino Club project would be to come up with our own design for a USBtinyISP. If you take it on, let GT know when you would like to present it to the club!

The USBtinyISP is very easy to use.

1. Start up the Arduino IDE on the computer and get the code to be transferred ready to go.
2. Connect the USBtinyISP to the computer with the usual USB cable.
3. Connect the USPtinyISP to the target (Arduino or Cyduino) using the six-wire ribbon cable. Be sure to properly orient the connector on the target side — there is usually some indication of to match up pin 1 — a mark on the connector body itself, or a red wire on end of the ribbon cable. If the connection is made properly, the target chip should power up — look for the power-on LED or use a voltmeter to check pin voltages if there is no LED.
4. Make sure that the board is set to Arduino Uno using the menu item: Tools→Board→Arduino AVR Boards→Arduino Uno.
5. On the computer, choose the USBtinyISP as the programmer, using the menu item: Tools → progrmmer → USBtinyISP.
6. Upload the code: Sketch → Upload Using Programmer. Or if we want to upload the boot loader code use: Tools → Burn Bootloader.

That's it.

Once we start using a dedicated programmer, we probably won't using anything else with non-Arduino boards.