Overview | Stand-alone programming AVRs using CircuitPython

Originalartikel

Backup

<html> <div class=„page-content all-page-view-content“ readability=„54“> <div class=„row-fluid build-image“> <div class=„row-fluid build-text“ readability=„44“> If you've ever wanted a stand alone AVR programmer, that is super easy to use, you've come to the right place! This guide will show you how to turn any CircuitPython powered board with 4+ GPIO pins into an AVR progammer all on its own. No software like avrdude is needed, this software will program the chip all on its own, just drag the HEX file onto the CircuitPython disk drive. Perfect to putting bootloaders on empty chips, or field-reprogramming a project! <div class=„row-fluid build-text“ readability=„44“> In theory, any and all AVR chips with SPI-programming interfaces are supported. However, we only have examples for ATmega328P chips (used in Arduino compatibles), ATtiny85 (used in original Trinket/Gemma), and ATmega2560 (Arduino Mega compatibles) To program other chips, you'll need to find out the signature, size of the flash, and the flash-page size. You can find this in the datasheet or in

```
avrdude.conf
```

  This code only supports SPI-based programming, not JTAG, SWD or parallel! <div class=„page-content all-page-view-content“ readability=„64“> <div class=„row-fluid build-text“ readability=„47“> Nearly all AVRs have a 'serial' programming interface, that's what we'll be using to program them. If your chip requires SWD, JTAG or parallel, this software won't work! In this example we'll show how to wire up an existing Arduino 328P compatible or raw 328P chip to a Feather M0 for programming For other chips, the wiring is similar, but you'll need to look up which pins are Power, Ground, Reset, and SCK/MOSI/MISO <div class=„row-fluid build-text“ readability=„31“> <h2>Power Pins</h2> Do these pins first because they're easy to forget! If connecting to a Arduino-compatible: connect GND on the Arduino to GND on the Feather. Then either plug the Arduino into USB, or connect the Arduino 5V to Feather USB If connecting to a bare chip: connect both GND pins together and to the Feather GND. Connect AVCC to VCC to the Feather 3V pin If you're

breadboarding a bare ATMega328 chip, don't forget there are *two* power pins and *two* ground pins <div class=„row-fluid build-text“ readability=„38“> Connect the CircuitPython SCK pin to the target SCK (on Uno/Atmega328 this is also known as Digital #13) Connect the CircuitPython MISO pin to the target MISO (on Uno/Atmega328 this is also known as Digital #12) Connect the CircuitPython MOSI pin to the target MOSI (on Uno/Atmega328 this is also known as Digital #11) Connect CircuitPython D5 (or any digital pin, as long as you change the code too) to the target RESET If you are breadboarding a chip, it may need a clock or crystal and it needs to be there to program the chip! If your board has a crystal or oscillator already, skip this. If you're programming a 'raw' ATmega328, you'll want to add it: Connect CircuitPython D9 (or any digital pin with PWM out, as long as you change the code to) to the target XTAL1 <div class=„row-fluid build-image“> Fritzing for diagram <table class=„build-table“ readability=„1“><tr class=„build-row“ readability=„3“><td class=„side-images“> <li class=„medium-side“> </td> <td class=„side-text“ readability=„28“> VCC lines are Red Ground/GND lines are Black SCK is green MOSI is blue MISO is yellow RESET is purple XTAL is grey Notice that the notch on the chip is to the right - away from the Feather! </td> </tr></table> <div class=„row-fluid build-image“><img class=„50000-asset img-responsive“ srcset=„https://cdn-learn.adafruit.com/assets/assets/000/050/000/medium260/hacks_unoprog.png?1515719126 260w, https://cdn-learn.adafruit.com/assets/assets/000/050/000/medium640/hacks_unoprog.png?1515719126 640w, https://cdn-learn.adafruit.com/assets/assets/000/050/000/medium800/hacks_unoprog.png?1515719126 800w, https://cdn-learn.adafruit.com/assets/assets/000/050/000/large1024/hacks_unoprog.png?1515719126 1024w“ sizes=„(max-width: 768px) 100vw, (max-width: 1024px) 65vw, (max-width: 1365px) 47vw,

750px“
src=„https://cdn-learn.adafruit.com/assets/assets/000/050/000/medium800/hacks_unoprog.png?1515719126“ alt=„hacks_unoprog.png“/> Fritzing for diagram <div class=„row-fluid build-text“ readability=„40“> For Arduino UNO and compatibles, we recommend powering from USB or DC power. Then connect GND pins together, and wire up Reset, SCK, MOSI, and MISO as seen above. XTAL pin is not required, Arduinos have on-board crystals. <div class=„page-content all-page-view-content“ readability=„80“> <div class=„row-fluid build-text“ readability=„35“> To use the AVR programming library you'll need to install the Adafruit CircuitPython AVRprog library on your CircuitPython board. First make sure you are running the latest version of Adafruit CircuitPython for your board. Next you'll need to install the necessary libraries to use the hardware–carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle. Our introduction guide has a great page on how to install the library bundle for both express and non-express boards. Remember for non-express boards like the, you'll need to manually install the necessary library from the bundle: adafruit_avrprog.mpy You can also download the adafruit_avrprog.mpy from its releases page on Github. Before continuing make sure your board's lib folder or root filesystem has the adafruit_avrprog.mpy file copied over. <div class=„row-fluid build-image“><img class=„49927-asset img-responsive“
srcset=„https://cdn-learn.adafruit.com/assets/assets/000/049/927/medium260/hacks_lib.png?1515373870 260w,
https://cdn-learn.adafruit.com/assets/assets/000/049/927/medium640/hacks_lib.png?1515373870 640w,
https://cdn-learn.adafruit.com/assets/assets/000/049/927/medium800/hacks_lib.png?1515373870 800w,
https://cdn-learn.adafruit.com/assets/assets/000/049/927/large1024/hacks_lib.png?1515373870 1024w“ sizes=„(max-width: 768px) 100vw, (max-width: 1024px) 65vw, (max-width: 1365px) 47vw, 750px“
src=„https://cdn-learn.adafruit.com/assets/assets/000/049/927/medium800/hacks_lib.png?1515373870“ alt=„hacks_lib.png“/> <div class=„row-fluid build-text“ readability=„18“> Next connect to the board's serial REPL so you are at the CircuitPython >>> prompt. For this simple example, we're assuming you don't need a clock-driving pin here, if you do, see the full example at the end of the page! <div class=„row-fluid build-text“ readability=„31“> <h2>Imports</h2> You'll need to import a few libraries

```
board
```

- for assigning hardware pins

```
busio
```

- we use SPI bus to talk to the target device

```
adafruit_avrprog
```

- the library that we're using! <div class=„build-code code-element“ readability=„7“> <pre class=„code-text-only c4“> >>> import board >>> import busio >>> import adafruit_avrprog </pre> <pre class=„prettyprint linenums“> >>> import board >>> import busio >>> import adafruit_avrprog </pre> <div class=„row-fluid build-image“> <div class=„row-fluid build-text“ readability=„36“> <h2>Initialize hardware</h2> Next, create the hardware interface, you'll need an SPI port and one extra pin for the reset line. We'll use

```
board.D5
```

to match our diagrams on the previous page, but it can be any pin you like! <div class=„build-code code-element“ readability=„15“> <pre class=„code-text-only c4“> >>> spi = busio.SPI(board.SCK, board.MOSI, board.MISO) >>> avrprog = adafruit_avrprog.AVRprog() >>> avrprog.init(spi, board.D5) </pre> <pre class=„prettyprint linenums“> >>> spi = busio.SPI(board.SCK, board.MOSI, board.MISO) >>> avrprog = adafruit_avrprog.AVRprog() >>> avrprog.init(spi, board.D5) </pre> <div class=„row-fluid build-text“ readability=„34“> <h2>Communication / Signature Check</h2> Next we'll verify that we can talk to the chip, once that works we are best off crafting our programmer into a full main.py project but at least we can quickly determine if things worked out. Start by initializing the programming interface with

```
avrprog.begin()
```

which will pull the reset line low and send some commands to get the chip to listen. Then read the signature, you'll get an array of numbers - its probably best to turn this into hex values before printing since they're referred to as hex values in datasheets. Finally, call

```
avrprog.end()
```

  <div class=„build-code code-element“ readability=„13“> <pre class=„code-text-

only c4“> >>> avrprog.begin() >>> [hex(i) for i in avrprog.read_signature()] ['0x1e', '0x95', '0xf'] >>> avrprog.end() </pre> <pre class=„prettyprint linenums“> >>> avrprog.begin() >>> [hex(i) for i in avrprog.read_signature()] ['0x1e', '0x95', '0xf'] >>> avrprog.end() </pre> <div class=„row-fluid build-image“> <div class=„row-fluid build-text“ readability=„32“> You can see here we have a 0x1E950F chip attached, also known at an ATmega328P <div class=„row-fluid build-text“ readability=„34“> You can save this code to main.py and use the REPL to see the signature data, it also includes the code for setting up the crystal-driving PWM output <div class=„build-code code-element“ readability=„29“> <pre class=„code-text-only c4“> „“„ Read Signature Test - All this does is read the signature from the chip to check connectivity! “„“ import board import busio import pulseio import adafruit_avrprog spi = busio.SPI(board.SCK, board.MOSI, board.MISO) avrprog = adafruit_avrprog.AVRprog() avrprog.init(spi, board.D5) # we can generate an 6 MHz clock for driving bare chips too! clock_pwm = pulseio.PWMOut(board.D9, frequency=6000000, duty_cycle=655362) avrprog.begin() print(„Signature bytes: “, [hex(i) for i in avrprog.read_signature()]) avrprog.end() </pre> <pre class=„prettyprint linenums“> „“„ Read Signature Test - All this does is read the signature from the chip to check connectivity! “„“ import board import busio import pulseio import adafruit_avrprog spi = busio.SPI(board.SCK, board.MOSI, board.MISO) avrprog = adafruit_avrprog.AVRprog() avrprog.init(spi, board.D5) # we can generate an 6 MHz clock for driving bare chips too! clock_pwm = pulseio.PWMOut(board.D9, frequency=6000000, duty_cycle=655362) avrprog.begin() print(„Signature bytes: “, [hex(i) for i in avrprog.read_signature()]) avrprog.end() </pre> <div class=„row-fluid build-text“ readability=„36“> <h2>SPI / Wiring Errors</h2> If something went wrong, you'll get an

```
SPI transaction failed
```

exception. Check your wiring! Also, sometimes the chip doesn't quite hear us, try connecting again. Common problems: The target isn't powered - make sure it is powered via USB or via the CircuitPython board. A shared Ground wire is required Make sure you have the reset pin on the target connected to whatever pin you setup when you created the

```
avrprog
```

object On ATmega2560, MOSI and MISO are connected opposite than the way you think. Either way, its OK to try swapping those two wires, see if that helps! The target is expecting a crystal but you don't have one, for example the UNO bootloader requires that the chip have a crystal or oscillator connected up, it's not optional! <div class=„page-content all-page-view-content“ readability=„52“> <div class=„row-fluid build-text“ readability=„28“> OK now that you've read the signature, you can write some code!

We have a few examples available you can use 'out of the box' - all are available here. You can download the library zip to get all the files. For each programming demo, we also have a matching 'hex' file, that's a requirement - it's the file you'll be programming into the chip! Copy the programming sketch into main.py and also grab the matching hex file. For example: <div class=„build-code code-element“ readability=„23“> <pre class=„code-text-only c4“> „“„ UNO Optiboot programming example, be sure you have the UNO wired up so:

```
UNO Ground to CircuitPython GND
UNO 5V to CircuitPython USB or make sure the UNO is powered by USB
UNO Pin 13 -&gt; CircuitPython SCK
UNO Pin 12 -&gt; CircuitPython MISO
UNO Pin 11 -&gt; CircuitPython MOSI
UNO RESET  -&gt; CircuitPython D5 (or change the init() below to change it!)
```

Drag „optiboot_atmega328.hex“ onto the CircuitPython disk drive, then open REPL! “„“ </pre> <pre class=„prettyprint linenums“> „“„ UNO Optiboot programming example, be sure you have the UNO wired up so:


```
UNO Ground to CircuitPython GND
UNO 5V to CircuitPython USB or make sure the UNO is powered by USB
UNO Pin 13 -&gt; CircuitPython SCK
UNO Pin 12 -&gt; CircuitPython MISO
UNO Pin 11 -&gt; CircuitPython MOSI
UNO RESET  -&gt; CircuitPython D5 (or change the init() below to change it!)
```


Drag „optiboot_atmega328.hex“ onto the CircuitPython disk drive, then open REPL! “„“ </pre> <div class=„row-fluid build-text“ readability=„31“> Indicates you need optiboot_atmega328.hex <div class=„row-fluid build-image“> <div class=„row-fluid build-text“ readability=„38“> Then run the REPL and look for the

```
Ready to GO, type 'G' here to start &gt;
```

prompt and type the letter G into the REPL. You should see the code begin by checking the identity of the chip (the signature), erasing the chip, then programming it. <div class=„row-fluid build-image“><img class=„50001-asset img-

responsive“
srcset=„https://cdn-learn.adafruit.com/assets/assets/000/050/001/medium260/hacks_flash_start.png?1515720290 260w,
https://cdn-learn.adafruit.com/assets/assets/000/050/001/medium640/hacks_flash_start.png?1515720290 640w,
https://cdn-learn.adafruit.com/assets/assets/000/050/001/medium800/hacks_flash_start.png?1515720290 800w,
https://cdn-learn.adafruit.com/assets/assets/000/050/001/large1024/hacks_flash_start.png?1515720290 1024w“ sizes=„(max-width: 768px) 100vw, (max-width: 1024px) 65vw, (max-width: 1365px) 47vw, 750px“
src=„https://cdn-learn.adafruit.com/assets/assets/000/050/001/medium800/hacks_flash_start.png?1515720290“ alt=„hacks_flash_start.png“/> <div class=„row-fluid build-text“ readability=„33“> It will skip most of the flash 'pages' because they're empty. At the end you'll get to the pages that are flashed and verified: <div class=„row-fluid build-image“><img class=„50002-asset img-responsive“
srcset=„https://cdn-learn.adafruit.com/assets/assets/000/050/002/medium260/hacks_flashdone_verified.png?1515720332 260w,
https://cdn-learn.adafruit.com/assets/assets/000/050/002/medium640/hacks_flashdone_verified.png?1515720332 640w,
https://cdn-learn.adafruit.com/assets/assets/000/050/002/medium800/hacks_flashdone_verified.png?1515720332 800w,
https://cdn-learn.adafruit.com/assets/assets/000/050/002/large1024/hacks_flashdone_verified.png?1515720332 1024w“ sizes=„(max-width: 768px) 100vw, (max-width: 1024px) 65vw, (max-width: 1365px) 47vw, 750px“
src=„https://cdn-learn.adafruit.com/assets/assets/000/050/002/medium800/hacks_flashdone_verified.png?1515720332“ alt=„hacks_flashdone_verified.png“/> <div class=„row-fluid build-text“ readability=„33“> It's very very rare for something to go wrong during verification. But if it does you'll see something like this. Just start over by hitting ^C and ^D in the REPL to begin again. <div class=„row-fluid build-image“><img class=„50003-asset img-responsive“
srcset=„https://cdn-learn.adafruit.com/assets/assets/000/050/003/medium260/hacks_verifyfail.png?1515720410 260w,
https://cdn-learn.adafruit.com/assets/assets/000/050/003/medium640/hacks_verifyfail.png?1515720410 640w,
https://cdn-learn.adafruit.com/assets/assets/000/050/003/medium800/hacks_verifyfail.png?1515720410 800w,
https://cdn-learn.adafruit.com/assets/assets/000/050/003/large1024/hacks_verifyfail.png?1515720410 1024w“ sizes=„(max-width: 768px) 100vw, (max-width: 1024px) 65vw, (max-width: 1365px) 47vw, 750px“
src=„https://cdn-learn.adafruit.com/assets/assets/000/050/003/medium800/hacks_verifyfail.png?1515720410“ alt=„hacks_verifyfail.png“/> <div class=„row-fluid build-text“ readability=„32“> That's it! You've programmed the chip. For more details, keep reading. <div class=„page-content all-page-view-content“ readability=„97“> <div class=„row-fluid build-text“ readability=„55“> Before you can really do anything you need to tell AVRprog library what the chip is. We'll use a python dict for that. Define name (that's for your information and printing errors), sig - a list of the three-byte signature, flash_size - the size of the flash memory in bytes, page_size - the size of each flash memory page in bytes, and fuse_mask - a list of the four fuses in a list

```
[low, high, ext, lock]
```

 Fuse mask is the oddest one, but basically it defines which bits are actually used in each fuse. For example, the ext fuse is often only the bottom three bits, so its 0x07. If you're not sure, you can set all four to 0xFF and then when you burn fuses, set all the high bits to 1. Here are some chip examples: <div class=„build-code code-element“ readability=„19“> <pre class=„code-text-only c4“> attiny85 = {'name': „ATtiny85“} attiny85['sig'] = [0x1E, 0x93, 0x0B] attiny85['flash_size'] = 8192 attiny85['page_size'] = 64 attiny85['fuse_mask'] = (0xFF, 0xFF, 0x07, 0x3F) </pre> <pre class=„prettyprint linenums“> attiny85 = {'name': „ATtiny85“} attiny85['sig'] = [0x1E, 0x93, 0x0B] attiny85['flash_size'] = 8192 attiny85['page_size'] = 64 attiny85['fuse_mask'] = (0xFF, 0xFF, 0x07, 0x3F) </pre> <div class=„build-code code-element“ readability=„19“> <pre class=„code-text-only c4“> atmega328p = {'name': „ATmega328P“} atmega328p['sig'] = [0x1E, 0x95, 0x0F] atmega328p['flash_size'] = 32768 atmega328p['page_size'] = 128 atmega328p['fuse_mask'] = (0xFF, 0xFF, 0x07, 0x3F) </pre> <pre class=„prettyprint linenums“> atmega328p = {'name': „ATmega328P“} atmega328p['sig'] = [0x1E, 0x95, 0x0F] atmega328p['flash_size'] = 32768 atmega328p['page_size'] = 128 atmega328p['fuse_mask'] = (0xFF, 0xFF, 0x07, 0x3F) </pre> <div class=„build-code code-element“ readability=„19“> <pre class=„code-text-only c4“> atmega2560 = {'name': „ATmega2560“} atmega2560['sig'] = [0x1E, 0x98, 0x01] atmega2560['flash_size'] = 262144 atmega2560['page_size'] = 256 atmega2560['fuse_mask'] = (0xFF, 0xFF, 0x07, 0x3F) </pre> <pre class=„prettyprint linenums“> atmega2560 = {'name': „ATmega2560“} atmega2560['sig'] = [0x1E, 0x98, 0x01] atmega2560['flash_size'] = 262144 atmega2560['page_size'] = 256 atmega2560['fuse_mask'] = (0xFF, 0xFF, 0x07, 0x3F) </pre> <div class=„row-fluid build-text“ readability=„37“>

```
avrprog.verify_sig(chip_dict, verbose=True)
```

 We suggest calling this first, you can call it whenever you like, and it will return True/False.

```
chip_dict
```

is that dictionary you made above <div class=„row-fluid build-text“ readability=„39“> This one is easy, just call

```
avrprog.erase_chip()
```

- the chip erase command is the same for all chips. It may take a second on bigger chips. You must do this before programming new firmware! Also, if your chip has the lock-firmware-fuse set, you may have to erase the flash before you can change the lock fuse. <div class=„row-fluid build-text“ readability=„53“> You can read, write and verify fuses. Read fuses with

```
avrprog.read_fuses(chip_dict)
```

 Which will return a list of the four fuses [low, high, ext, lock] Write fuses with

```
avrprog.write_fuses(chip_dict, low=0xll, high=0xhh, ext=0xee, lock=0xkk)
```

 Only arguments that are passed in will be written, so you can choose to write one fuse, or all 4. Verify fuses with

```
avrprog.verify_fuses(chip_dict, low=0xll, high=0xhh, ext=0xee, lock=0xkk)
```

 Only arguments that are passed in will be verified, so you can choose to verify one fuse, or all 4. <div class=„row-fluid build-text“ readability=„60“> OK this is the good part, here's how you can write and verify flash memory. Reading memory to disk is not supported yet!

```
avrprog.program_file(chip_dict, "filename.hex", verbose=True, verify=True)
```

 This function does all the work really, give it the chip information dictionary, and the name of a file (full path is OK). If

```
verify
```

is True, it will verify each page manually after writing. This is way faster than writing the whole file and then verifying the whole file so we recommend it. If you really want, you can also verify against a file with:

```
verify_file(chip_dict, "filename.hex", verbose=True)
```

 But it will check every single byte of the flash chip, so for example, if its a sparse hex file, like most bootloaders are where only a small portion of flash is data and the rest is empty, the empty parts are still checked. So it's very slow! <div class=„row-fluid build-text“ readability=„31“> Not supported at this time!   </html>