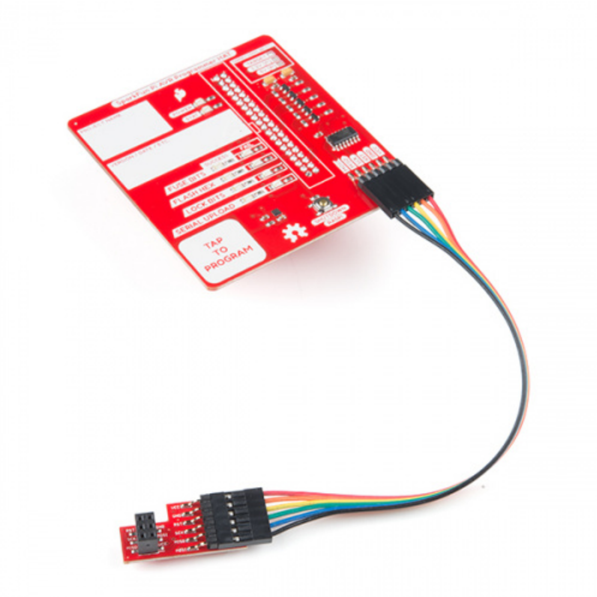
Pi AVR Programmer HAT Hookup Guide

Introduction

In this tutorial we will use a Pi AVR Programmer HAT and a Raspberry Pi 3B+ to program an ATMega328P target IC (RedBoard). We will program the Arduino bootloader over SPI using the capsense pad to engage, and then do some programming using avrdude in the command line. We will also cover how the Pi AVR Programmer HAT hardware works in conjunction with Python, avrdude, and shell command files.



SparkFun Pi AVR Programmer HAT
DEV-14747

The Pi AVR Programmer HAT makes it easy to program AVRs directly from the SPI hardware pins on any Raspberry Pi. It was originally designed as an in-house solution for SparkFun production, but now is offered as a robust programming tool for anyone to purchase. It is by far one of the fastest, most reliable, and hack-able (fully open sourced) AVR programming solutions available. It can be used directly from the command line using avrdude commands, or with some simple setup steps, it can function as a stand-alone programmer with capsense-pad engage and status leds!

All of the design files, firmware, and example programming files can be found here:

SPARKFUN PI AVR PROGRAMMER GITHUB REPOSITORY

There are many reasons for programming your AVR via an in-system programmer (ISP):

- If your AVR doesn't have a bootloader on it, it's probably the only way to load code.
- ISP provides a faster and more reliable code upload.
- If your project requires that your hardware UART pins (RX/TX) be connected to another device, this can conflict with serial uploading of code. ISP programming does not require using these pins. Instead, it uses MOSI/MISO/SCK/RESET – Arduino pins D11/D12/D13/RESET. This means you can leave RX and TX connected during development and re-program your AVR with new code without un-plugging and re-plugging any of your programming lines.
- With ISP, you can overwrite the bootloader and squeeze out some extra flash space.
- ISP allows you to poke at the fuse bits to change many settings, including the brown-out voltage.

Covered In This Tutorial

In this tutorial we will introduce you to all of the important aspects of the Pi AVR Programmer HAT. It's split into a series of sections, which cover:

- Board Overview – A look at the hardware components that make up the Pi AVR Programmer HAT.
- Raspberry Pi Setup – To get up and programming, there is a little bit of setup work to do on your Raspberry Pi. Here, we will show what settings to change and what files need to be modified and/or copied.
- Closer Look at the Repository Files – Here you'll find a more detailed description of each file necessary to setup your Pi.
- Hardware Hookup – How to hookup the Pi AVR Programmer from your Raspi to the example Target AVR.
- ISP Programming: Stand-Alone – How to use the HAT in stand-alone mode and program an arduino bootloader (or any hex file) via SPI onto an ATMega328p target IC (the Redboard).
- ISP Programming: Command Line – A more advanced, command-line-based approach to using the Pi AVR Programmer HAT.
- ISP Programming: Within the Arduino IDE – How to use the HAT from within the Arduino IDE. A nice one-click solution for programming your Arduino with the speed of ISP.
- Speed Test – We take the HAT for a speed test and see just how fast it goes!

Required Materials

Note The wishlist is our list of suggested parts to complete this tutorial. You can use any other Raspberry Pi Model you'd like (0, 2, 3, etc.). In this tutorial we use the 3B+ model. Another good option is the Raspberry Pi 3 Starter Kit, which includes *most* everything you need to get up and running with your Pi. You can choose a different target Arduino board. We use the Redboard in this tutorial.

To complete this tutorial, you will need the following hardware. You may not need everything though depending on what you have. Add it to your cart, read through the guide, and adjust the cart as necessary.

Pi AVR Programmer HAT Hookup Guide Wishlist SparkFun Wish List

SparkFun Pi AVR Programmer HAT
DEV-14747

Raspberry Pi 3 B+
DEV-14643
Everyone knows and loves Raspberry Pi, but what if you didn't need additional peripherals to make it wireless. The…

microSD Card with Adapter - 16GB (Class 10)
COM-13833
This is a class 10 16GB microSD memory card, perfect for housing operating systems for single board computers a…

Wall Adapter Power Supply - 5.1V DC 2.5A (USB Micro-B)
TOL-13831
This is a high-quality switching 'wall wart' AC to DC 5.1V 2,500mA USB Micro-B wall power supply manufactured s…

Pi Tin for the Raspberry Pi - Clear
PRT-13103
The Raspberry Pi is everybody's favorite little computer, the last thing that you'd want is for something bad to happ…

Multimedia Wireless Keyboard
WIG-14271
With Single-Board Computers (SBCs) on the rise, it is a good idea to have an easy way to interface with them. Ope…

SparkFun RedBoard - Programmed with Arduino
DEV-13975
At SparkFun we use many Arduinos and we're always looking for the simplest, most stable one. Each board is a bit…

SparkFun USB Mini-B Cable - 6 Foot
CAB-11301
This is a USB 2.0 type A to Mini-B 5-pin cable. You know, the mini-B connector that usually comes with USB Hubs, …

Suggested Reading

To read *even more* about the history behind this project and design choices, check out the full tutorial/write up here:

Raspberry Pi Stand-Alone Programmer

MARCH 8, 2018

This tutorial will show you how to use a headless Raspberry Pi to flash hex files onto AVR microcontrollers as a stand-alone programmer. It also tells the story about production programming challenges, how SparkFun came to this solution, and all the lessons learned along the way.

Whether you're a beginner or experienced electronics enthusiast, the Pi AVR Programmer HAT should be easy to get up-and-running. If you've programmed an Arduino before, you'll be well-prepared for the next step. Here are some addtional tutorials we'd recommend reading before continuing on with this one.

Serial Peripheral Interface (SPI)

SPI is commonly used to connect microcontrollers to peripherals such as sensors, shift registers, and SD cards.

Installing an Arduino Bootloader

This tutorial will teach you what a bootloader is and why you would need to install or reinstall it. We will also go over the process of burning a bootloader by flashing a hex file to an Arduino microcontroller.

What is an Arduino?

What is this 'Arduino' thing anyway?

Installing Arduino IDE

A step-by-step guide to installing and testing the Arduino software on Windows, Mac, and Linux.

Logic Levels

Learn the difference between 3.3V and 5V devices and logic levels.

Serial Terminal Basics

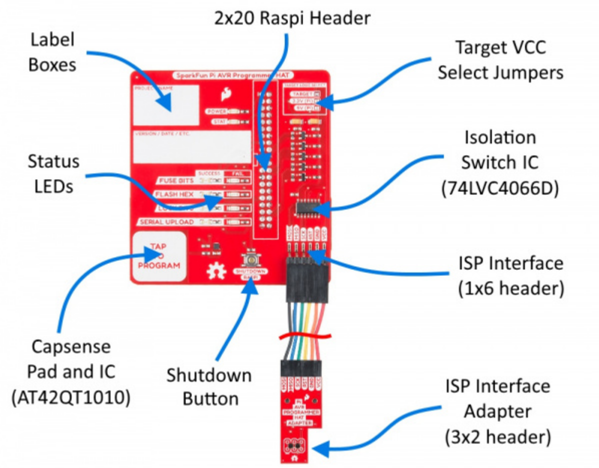
This tutorial will show you how to communicate with your serial devices using a variety of terminal emulator applications.

Raspberry Pi 3 Starter Kit Hookup Guide

Guide for getting going with the Raspberry Pi 3 Model B and Raspberry Pi 3 Model B+ starter kit.

Board Overview

Before we get into using our Pi AVR Programmer HAT, let’s quickly overview what components fill the board out:


2x20 Raspi Header
Label Boxes
Target VCC Select Jumpers
Isolation Switch IC (74LVC4066D)
Status LEDs
ISP Interface (1x6 header)
Capsense Pad and IC (AT42QT1010)
Shutdown Button
ISP Interface Adapter (3x2 header)

- **2x20 Raspi Header** – This female header will connect down to the GPIO pins on your Raspberry Pi. When pushed all the way down into the GPIO pins, the HAT will sit flush with the top of your enclosure.
- **Target VCC Select** – Jumpers to choose what logic level you’d like to talk to your target. Options include “**TARGET**”, **3.3V,** and **5V**.

  > **Note:** The “TARGET” option means that the target IC will provide the logic level reference voltage. “TARGET” is the default setting for this jumper. Also note that the 3.3V and 5V options are provided

from those voltages on the Raspi 2x20 headers. Please only close one of these jumpers at a time. Closing more than one jumper will damage your Raspi.

- **Isolation Switch IC** – The multiplexer IC that disconnects the SPI lines from your Pi to the Target.
- **SPI Interface (1x6 header)** – These headers provide optional direct access to the SPI lines. This is in a 1x6 straight header format, so it will mate nicely with your 6-pin Jumper wire. Most programmers use ribbon cables, which will fail after so many insertions. This 1x6 cable and adapter solution is much more robust. You can also opt to remove the supplied jumper cable, and wire up to your target with your own jumper wires or custom adapter.
- **SPI Interface Adapter (2x3 header)** – This adapter mates easily with the standard ISP header found on most AVR boards.
- **Shutdown Button** – Hold this down for 6 seconds to shut down your Pi.

  Note: This is simply a momentary button on a GPIO pin. It could be used for other things, but the Python module running the Pi AVR Programmer HAT (test.py) is watching this line to engage a full shutdown of the Pi. After you engage a shut down, please watch the "active" LED on the Raspberry Pi. It is located inside the enclosure next to the power LED. Wait for it to stop blinking, then you can remove power.

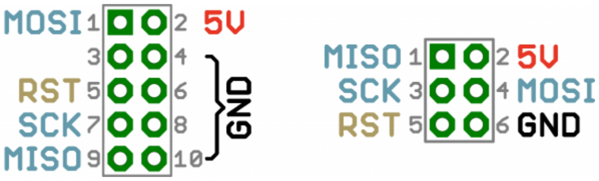
- **Capsense Pad** – Use this to engage programming. Simply tap with your finger and that will trigger the onboard capsense ATQ421010 IC which is connected to a GPIO on the Pi.
- **Status LEDS** – Used for indicating power, success and/or fail for various stages of programming.
- **Label Boxes** – These blank white silk-screen boxes are handy for labeling your Pi with a project name and version of your loaded hex file.

AVR ISP Pinouts

AVRs are programmed through an SPI interface. There are six unique signals required for communication between ISP and AVR:

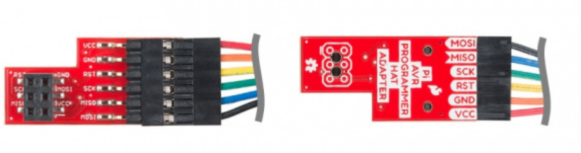
- VCC
- GND
- Reset
- MOSI
- MISO
- SCK

To route those signals between devices, there are two standardized connectors – one 10-pin, 2x5 connector and another 6-pin, 2x3 connector:



AVR ISP Pinouts – Top View.

The Pi AVR Programmer HAT includes a 1x6 jumper cable and adapter which terminates with a 2x3 connector. There are labels on the bottom side near each leg on the SMD header:



ISP Adapter – Top and Bottom View.

If your target device has the 2x5 pinout option, then you could use some of our F/F jumper cables to connect directly from the right-angled 1x6 male header on the HAT PCB to your target.

Jumper Wires Premium 6" F/F Pack of 10
◉ PRT-08430

Raspberry Pi Setup

Before we can do any programming, we need to do some setup work on our Raspberry Pi. If you are fairly familiar with Raspberry Pi, then you should check out the quick setup list for advanced users below. If you are fairly new to Raspberry Pi, then we recommend checking out the following tutorial: The Raspberry Pi 3 Starter Kit Hookup Guide. This offers a great walkthrough to setting up your Raspberry Pi with NOOBS (Raspberry Pi's easy-to-use graphical OS installer).

Raspberry Pi 3 Starter Kit Hookup Guide
APRIL 11, 2016

Guide for getting going with the Raspberry Pi 3 Model B and Raspberry Pi 3 Model B+ starter kit.

Quick Setup List:

1. Install avrdude: Run the following command `sudo apt-get install avrdude` .
2. Put these four files in **/home/pi**:
   1. *test.py*
   2. *avrdude_gpio.conf*
   3. *pi_program.sh*
   4. *optiboot.hex*

   > **Note:** You can use any hex file you'd like to program, but for this example, we are going to show how to re-program a RedBoard with an optiboot bootloader hex file.

3. Put the modified version of *rc.local* in your **/etc** folder.
4. Enable SPI hardware: Run `sudo raspi-config` (as explained in the "Raspberry Pi SPI and I2C Tutorial").
5. Open permissions on these files (to avoid debug errors). Run the following command: `sudo chmod 777 test.py avrdude_gpio.conf pi_program.sh your_firmware.hex` .
6. Adjust *pi_program.h* to your desired programming settings. Here you can choose fuse bits, device ID (your target type), programming speed (`-b 125000` is the flag for setting this).
7. Launch *test.py*. Run `sudo python test.py` (or reboot and rc.local will launch it for you).
8. Tap PROGRAM!

Closer Look at Repository Files

There are a lot of files in the GitHub repository for this project.

SPARKFUN PI AVR PROGRAMMER GITHUB REPOSITORY

However, you don't actually need *all* of them to get up and running. There are quite a few that are extra examples and will show you some more of the advanced features of this programmer. Let's take a closer look at the most important ones.

test.py

The ***test.py*** is the Python module that handles all of the operations of the Pi AVR Programmer HAT. It is launched at bootup from inside *rc.local*. Once it is up and running, it blinks the "STAT" LED to show that it is alive. It then listens to the capsense pad, finds your firmware hex file, engages programming, parses output from avrdude, and ultimately blinks status LEDs.

It also has some functionality to watch for media drives and pull in any new hex files that live on them. This is an easy way to update your programming hex file on your Pi, but beware, it will overwrite your existing hex file, so be careful. It's usually a good idea to stop test.py before plugging in any thumbdrives (until you're *absolutely* sure you want test.py to pull in the hex file). To stop test.py (and any other background Python modules), simply use this command:

```
sudo killall python
```

It’s also not a bad idea to back up your hex files elsewhere (outside of **/home/pi**).

pi_program.sh

The ***pi_program.sh*** shell file contains the actual programming calls to avrdude. Test.py launches this shell to engage programming. It is very similar to the types of programming we do in SparkFun production. It allows you to adjust some variables at the top for device and fuse bits. Fuse bits can be a little tricky. If you need to adjust these and want some help, please check out our fuse bits tutorial.

```
# DEVICE
# Here are some commonly used "part no"s in avrdude at SFE
# UN-comment the one you wish to use, or plug in something different
# Use "avrdude -p?" for a list of supported devices
DEVICE=atmega328p
#DEVICE=m32u4
#DEVICE=t2313

#FUSE BITS
HIGH_FUSE=0xD8
LOW_FUSE=0xFF
EXT_FUSE=0xFD # due to masking, 0x05 = 0xFD, 0x07 = 0xFF, ***Note on an ATMEGA2560, ext fuse writes and verifies as 0xFD
LOCK=0x0F # due to masking, 0x0F = 0xCF
```

Note that it also has a nice little trick at the top to search for a hex file within **/home/pi** and use that for programming.

```
# get firmware file name
firmware=$(find /home/pi/*.hex)
$firmware .= "/home/pi/$firmware"
```

Beware, that if you have two hex files in this directory, it will use the first one it finds. You can modify *pi_program.sh* to call a specific hex file by changing the `$firmware` variable.

The last thing to note about this file is the manual GPIO control in the middle of calls. In between fuse bits programming and flashing of the hex, there is a manual flip of the GPIO controlling the reset line. After using this programmer for thousands of programming cycles, we found that in order to truly reset the target in between calls to avrdude, a more reliable approach is to quickly toggle that GPIO from the shell file. Without this, the second call to avrdude can occasionally fail to reset the target. But with this toggle, we’ve had 100% success!

```
sudo gpio -g mode 26 output
sudo gpio -g write 26 0
sleep 0.1
sudo gpio -g write 26 1
sleep 0.1
```

avrdude_gpio.conf

The provided ***avrdude_gpio.conf*** configuration file is very similar to the default configuration file. The only modification necessary is adding in the linuxspi programmer definition (Note that this also defines your RESET pin). On the Pi AVR Programmer HAT, we have this hard wired to GPIO PIN 26. If you are working from a different *.conf* file, then this little block of text below needs to be added to the very bottom of the *avrdude.conf* file.

```
programmer
  id = "linuxspi";
  desc = "Use the Linux SPI device in /dev/spidev*";
  type = "linuxspi";
  reset = 26;
;
```

rc.local

The provided ***rc.local*** file is very similar to the default rc.local file you get with most raspberry Pis. This file runs at bootup, so it is handy if you'd like to have some commands run every time you power up your Pi (very useful if you plan to run it headless). The only addtional command added to the default rc.local is as follows. It simply calls Python to launch our test.py module.

```
python /home/pi/test.py &
```

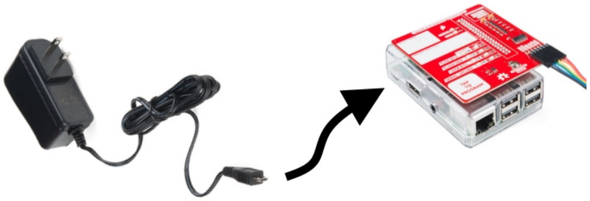
Note the “ `&` ” is very important here. It allows your Pi to continue on and run the GUI. If it wasn't there, then your Pi would just wake up and run the Python module, and then do *nothing else*, this is dangerous, because you wouldn't be able to modify your Pi from here on out. It's kind of like “bricking” your Pi, so please don't forget that very important “ `&` ”.

Hardware Hookup

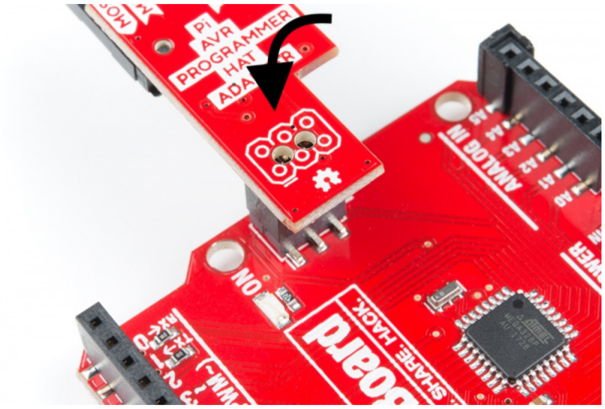
Once you have completed the software setup to your Raspi, then it's time to plug in some hardware!

- If you haven't already, put your Pi into an enclosure. Note that this is optional; it will work just fine without.
- Plug in the HAT on top of your enclosed Pi.

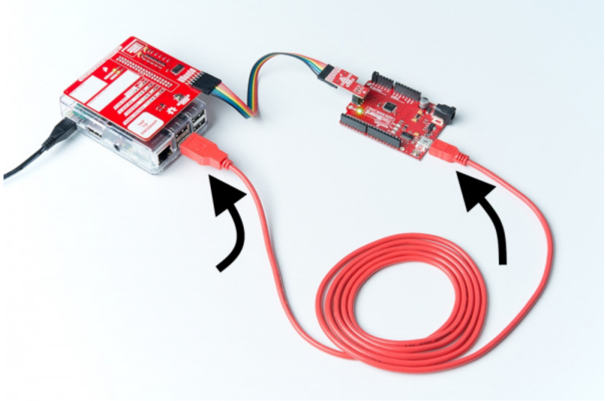
- Power your Raspi using a micro-B power adapter.



- Plug in the programming cable into your target IC. In this example, we are plugging into the 2x3 SPI header on the RedBoard. Note that this is polarized. The small white line indicates “pin 1”. Make sure to align those up!



- Plug in a USB cable from the Pi to the RedBoard. This will provide power to the RedBoard. You could also opt to power the board via the barrel jack if you wish.



ISP Programming: Stand-Alone (aka Headless)

In this section, we are going to program using the Raspberry Pi “headless” (aka *no* monitor, mouse or keyboard). We will use the built in capacitive touch pad to engage programming and status LEDs to indicate success or failure. For this example, we will program the Arduino optiboot bootloader onto the RedBoard.

After you have setup your raspi for use with the Pi AVR Programmer HAT, then there are two important things to consider while using it as a stand-alone programmer:

- **Your hex file** – This should live in the ‘**/home/pi**’ directory.
- ***pi_program.sh*** – This shell file contains all the settings for programming and can be edited with any text editor.

Note: If you followed along with the setup instructions above, then you should already have the correct hex file and settings. And if you followed along with the hardware hookup above, then you should be set to program your RedBoard.

If you are plugging into the 2x3 header on a different AVR target IC, then we do recommend double checking that you have the polarity correct. The _pin 1_ is usually marked with a line of silk (as on the adapter), but it can also be indicated by a different style pad (usually square vs round).

After your setup and powered, programming is a simple as 1-2-3:

1. To engage programming in stand-alone use, simply tap the capacitive touch pad labeled, “TAP TO PROGRAM”.
2. The STAT LED will go solid during programming. This can take a couple seconds – depending on the size of your hex file.
3. Finally, the SUCCESS and FAIL LEDs will indicated the status at each stage of programming

> **Note:** Because of the way we have set this up to only program the bootloader, the “Serial Upload” LEDs will not light up. This is okay for now. If you’d like to do some serial uploading in stand alone mode, you can try out the serial upload example from the repo here.

To see more about the results of your programming (debug messages from avrdude), you will need to access some files on your raspi. During each cycle of programming, the output from avrdude is saved to some text files that live within **/home/pi**.

- ***/home/pi/fuse_results.txt***
- ***/home/pi/flash_results.txt***
- ***/home/pi/SERIAL_UPLOAD/serial_upload_results.txt*** – Note that this will only be updated if you did any serial uploading

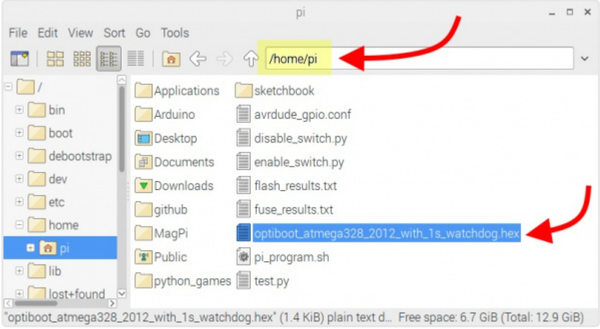
ISP Programming: Command Line

In this section, we are going to program the Target AVR IC (Redboard) using calls to avrdude in the command line. This will require a monitor, keyboard and mouse hooked up to your Pi

Download the bootloader hex file for the RedBoard here:

DOWNLOAD OPTIBOOT HEX FILE (HEX)

Save it in the correct directory (i.e. **/home/pi/**):



> **Note:** Make sure this is the only hex file in that directory, because *pi_program.sh* is going to find it and use if for programming.

Sanity Check – Device Signature Verification

Before we get into any actual programming, it's a good idea to ensure our connections are all correct and avrdude is working properly. To do this, we are going to simply call avrdude and ping for the device ID.

The Pi AVR Programmer HAT has an isolation switch between the raspi GPIO pins and the target. This serves as both protection of your rapsi and as a means to "free up" your target once you're done programming. You can use some handy Python modules to quickly set this switch: enable_switch.py and disable_switch.py. For easy access, it's a good idea to save them in the */home/pi* directory.

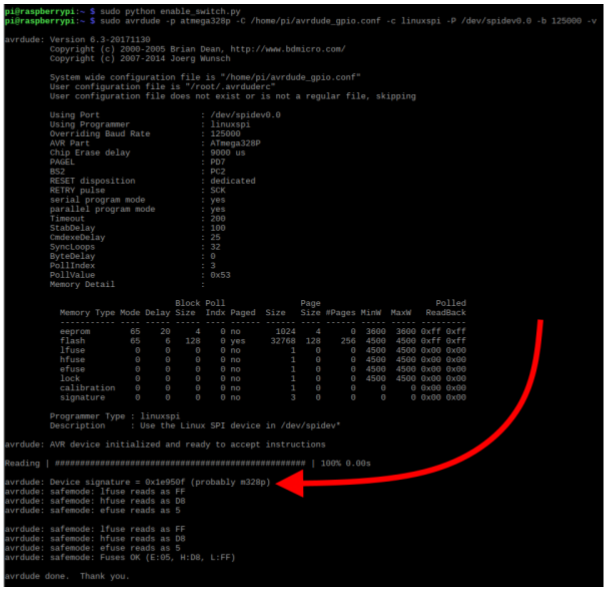
Make sure to enable the programming switch by running a quick python module named *enable_switch.py*.

```
sudo python enable_switch.py
```

Then run your avrdude command. Here is our first call to avrdude to check the device ID.

```
sudo avrdude -p atmega328p -C /home/pi/avrdude_gpio.conf -c linuxspi -P /dev/spidev0.0 -b 125000
 -v
```

Your readout should look like this:



```
pi@raspberrypi:~ $ sudo python enable_switch.py
pi@raspberrypi:~ $ sudo avrdude -p atmega328p -C /home/pi/avrdude_gpio.conf -c linuxspi -P /dev/spidev0.0 -b 125000 -v

avrdude: Version 6.3-20171130
         Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
         Copyright (c) 2007-2014 Joerg Wunsch

         System wide configuration file is "/home/pi/avrdude_gpio.conf"
         User configuration file is "/root/.avrduderc"
         User configuration file does not exist or is not a regular file, skipping

         Using Port                    : /dev/spidev0.0
         Using Programmer              : linuxspi
         Overriding Baud Rate          : 125000
         AVR Part                      : ATmega328P
         Chip Erase delay              : 9000 us
         PAGEL                         : PD7
         BS2                           : PC2
         RESET disposition             : dedicated
         RETRY pulse                   : SCK
         serial program mode           : yes
         parallel program mode         : yes
         Timeout                       : 200
         StabDelay                     : 100
         CmdexeDelay                   : 25
         SyncLoops                     : 32
         ByteDelay                     : 0
         PollIndex                     : 3
         PollValue                     : 0x53
         Memory Detail                 :

                                  Block Poll               Page                       Polled
           Memory Type Mode Delay Size  Indx Paged  Size   Size #Pages MinW  MaxW   ReadBack
           ----------- ---- ----- ----- ---- ------ ------ ---- ------ ----- ----- ---------
           eeprom        65    20     4    0 no       1024    4      0  3600  3600 0xff 0xff
           flash         65     6   128    0 yes     32768  128    256  4500  4500 0xff 0xff
           lfuse          0     0     0    0 no          1    0      0  4500  4500 0x00 0x00
           hfuse          0     0     0    0 no          1    0      0  4500  4500 0x00 0x00
           efuse          0     0     0    0 no          1    0      0  4500  4500 0x00 0x00
           lock           0     0     0    0 no          1    0      0  4500  4500 0x00 0x00
           calibration    0     0     0    0 no          1    0      0     0     0 0x00 0x00
           signature      0     0     0    0 no          3    0      0     0     0 0x00 0x00

         Programmer Type : linuxspi
         Description     : Use the Linux SPI device in /dev/spidev*

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: safemode: lfuse reads as FF
avrdude: safemode: hfuse reads as D8
avrdude: safemode: efuse reads as 5

avrdude: safemode: lfuse reads as FF
avrdude: safemode: hfuse reads as D8
avrdude: safemode: efuse reads as 5
avrdude: safemode: Fuses OK (E:05, H:D8, L:FF)

avrdude done.  Thank you.
```

Click the image for a closer look.

This basic command defines the programmer type you're using and the AVR it's talking to. AVRDUDE will attempt to read the **Device Signature** from your AVR, which is different for each AVR type out there. Every ATmega328P should have a device signature of `0x1E950F`.

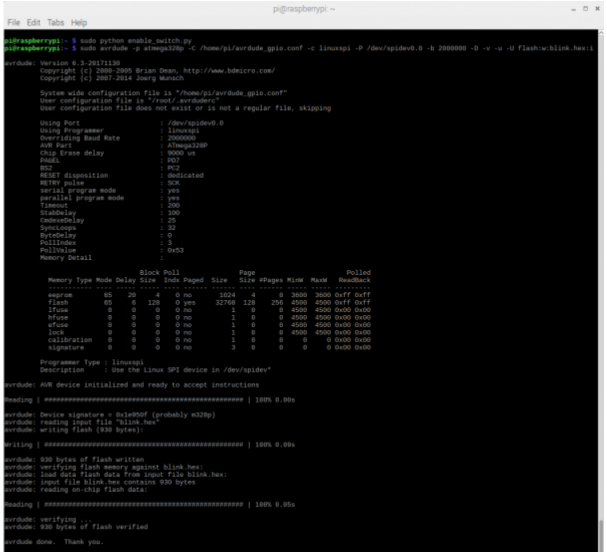
Flash Programming

Now that you've verified that everything is in working order, you can do all sorts of memory reading and writing with AVRDUDE. The main piece of memory you probably want to write is flash – the non-volatile memory where the programs are stored.

This command will perform a basic write to flash (using this HEX file as an example):

```
sudo avrdude -p atmega328p -C /home/pi/avrdude_gpio.conf -c linuxspi -P /dev/spidev0.0 -b 200000
0 -D -v -u -U flash:w:blink.hex:i
```

Writing to flash can sometimes take a little longer than a device ID ping. With the built-in ISP hardware pins and the Pi AVR Programmer HAT, you can use up to 2MHz reliably. For *blink.hex*, it only takes 0.09 seconds! You'll see a text status bar scroll by as the device is read, written to, and verified.



```
pi@raspberrypi:~ $ sudo python enable_switch.py
pi@raspberrypi:~ $ sudo avrdude -p atmega328p -C /home/pi/avrdude_gpio.conf -c linuxspi -P /dev/spidev0.0 -b 2000000 -D -v -u -U flash:w:blink.hex:i

avrdude: Version 6.3-20171130
         Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
         Copyright (c) 2007-2014 Joerg Wunsch

         System wide configuration file is "/home/pi/avrdude_gpio.conf"
         User configuration file is "/root/.avrduderc"
         User configuration file does not exist or is not a regular file, skipping

         Using Port                    : /dev/spidev0.0
         Using Programmer              : linuxspi
         Overriding Baud Rate          : 2000000
         AVR Part                      : ATmega328P
         Chip Erase delay              : 9000 us
         PAGEL                         : PD7
         BS2                           : PC2
         RESET disposition             : dedicated
         RETRY pulse                   : SCK
         serial program mode           : yes
         parallel program mode         : yes
         Timeout                       : 200
         StabDelay                     : 100
         CmdexeDelay                   : 25
         SyncLoops                     : 32
         ByteDelay                     : 0
         PollIndex                     : 3
         PollValue                     : 0x53
         Memory Detail                 :

                                  Block Poll               Page                       Polled
           Memory Type Mode Delay Size  Indx Paged  Size   Size #Pages MinW  MaxW   ReadBack
           ----------- ---- ----- ----- ---- ------ ------ ---- ------ ----- ----- ---------
           eeprom        65    20     4    0 no       1024    4      0  3600  3600 0xff 0xff
           flash         65     6   128    0 yes     32768  128    256  4500  4500 0xff 0xff
           lfuse          0     0     0    0 no          1    0      0  4500  4500 0x00 0x00
           hfuse          0     0     0    0 no          1    0      0  4500  4500 0x00 0x00
           efuse          0     0     0    0 no          1    0      0  4500  4500 0x00 0x00
           lock           0     0     0    0 no          1    0      0  4500  4500 0x00 0x00
           calibration    0     0     0    0 no          1    0      0     0     0 0x00 0x00
           signature      0     0     0    0 no          3    0      0     0     0 0x00 0x00

         Programmer Type : linuxspi
         Description     : Use the Linux SPI device in /dev/spidev*

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: reading input file "blink.hex"
avrdude: writing flash (930 bytes):

Writing | ################################################## | 100% 0.09s

avrdude: 930 bytes of flash written
avrdude: verifying flash memory against blink.hex:
avrdude: load data flash data from input file blink.hex:
avrdude: input file blink.hex contains 930 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.05s

avrdude: verifying ...
avrdude: 930 bytes of flash verified

avrdude done.  Thank you.
```

Click the image for a closer look.

The `-U` option command handles all of the memory reads and writes. We tell it we want to work with `flash` memory, do a write with `w`, and then tell it the location of the hex file we want to write.

Useful Options

Here are just a few last AVRDUDE tips and tricks before we turn you loose on the AVR world.

Specify AVR Device

Two options required for using AVRDUDE are the **programmer type** and **AVR device** specification. The programmer definition, assuming you're using the Pi AVR Programmer HAT, will be `-c linuxspi`. Note that you will also need to specify the port (because there are two SPI ports on the raspi. This is done with the `-P /dev/spidev0.0` portion of the call. If you need to use a different programmer check out this page and `CTRL + F` to "`-c programmer-id`".

The AVR device type is defined with the `-p` option. We've shown a few examples with the ATmega328P, but what if you're using an ATtiny85? In that case, you'll want to put `-p t85` instead. Check out the top of this page for an exhaustive list of compatible AVR device types.

Verbose Output

Adding one or more `-v` 's to your AVRDUDE command will enable various levels of verbosity to the action. This is handy if you need a summary of your configuration options, or an in-depth view into what data is being sent to your AVR.

There's plenty more where that came from. Check out the AVRDUDE Option Documentation for the entire list of commands.

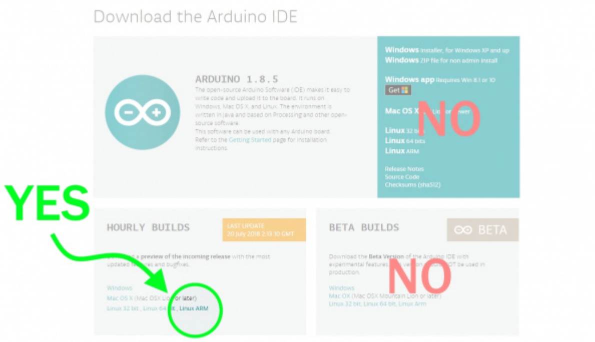
AVRDUDE OPTION DOCUMENTATION

ISP Programming: Within the Arduino IDE

Here, we are going to show how to use the Pi AVR Programmer HAT directly from the Arduino IDE. Note that this functionality is not completely fleshed out. We are currently developing support for this feature and working out any bugs right now. Read all about it on the GitHub pull request here:

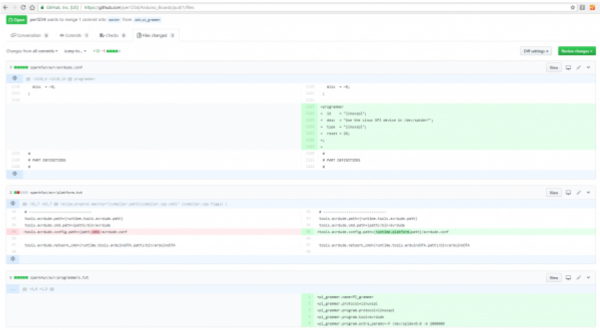
SPARKFUN ARDUINO BOARDS GITHUB BOARDS PULL REQUEST

It is very important that you have the correct version of Arduino. The version of avrdude that comes with Arduino on your Raspi is probably not capable of using the Pi AVR Programmer HAT. You need Arduino Nightly Version because it comes with the latest version of avrdude – *version 6.3-20171130*. For this tutorial, we are using **Arduino 1.8.6 Hourly Build 2018/07/19**.

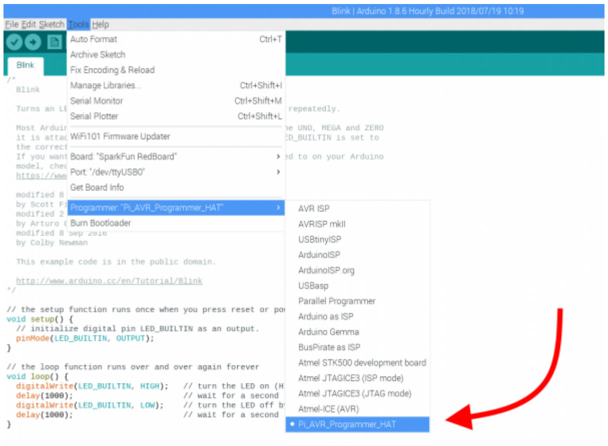

Download the Arduino IDE
ARDUINO 1.8.5
Windows Installer, for Windows XP and up
Windows ZIP file for non admin install
Windows app Requires Win 8.1 or 10
Get
Mac OS X
Linux 32 bits
Linux 64 bits
Linux ARM
Release Notes
Source Code
Checksums (sha512)
NO
YES
HOURLY BUILDS
LAST UPDATE 20 July 2018 2:13:10 GMT
Windows
Mac OS X (Mac OSX Lion or later)
Linux 32 bit , Linux 64 bit , Linux ARM
BETA BUILDS
BETA
Windows
Mac OX (Mac OSX Mountain Lion or later)
Linux 32 bit, Linux 64 bit, Linux Arm
NO

You need to install the SparkFun Arduino Boards. If you are comfortable using the boards manager, then you can install them directly there. For more help, check out the instructions in the GitHub repository's README.md, and this write up for Installing Custom Boards into Arduino.

Modify the *programmers.txt*, *avrdude.conf* and *platform.txt* files (that is, unless it's already pulled into the repo by now). You can view it on GitHub at the open pull request for SparkFun Arduino Boards here. For reference, the following screen-shot also shows the differences:

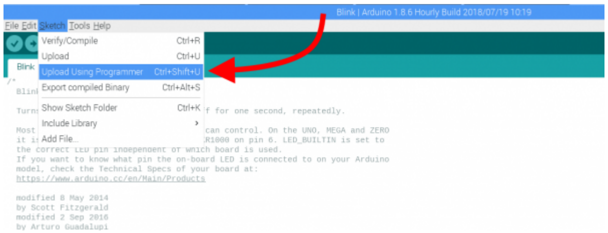


Select “*Pi_AVR_Programmer_HAT*” from the drop down menu in the Arduino IDE. If you don’t see it in your list of options, then try closing and re-opening Arduino.


Blink | Arduino 1.8.6 Hourly Build 2018/07/19 10:19
File Edit Sketch Tools Help
Blink
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "SparkFun RedBoard"
Port: "/dev/ttyUSB0"
Get Board Info
Programmer: "Pi_AVR_Programmer_HAT"
Burn Bootloader
AVR ISP
AVRISP mkII
USBtinyISP
ArduinoISP
ArduinoISP.org
USBasp
Parallel Programmer
Arduino as ISP
Arduino Gemma
BusPirate as ISP
Atmel STK500 development board
Atmel JTAGICE3 (ISP mode)
Atmel JTAGICE3 (JTAG mode)
Atmel-ICE (AVR)
• Pi_AVR_Programmer_HAT

```
/*
  Blink

  Turns an LE                                        repeatedly.

  Most Arduir                                       he UNO, MEGA and ZERO
  it is attac                                       ED_BUILTIN is set to
  the correct
  If you want                                       ed to on your Arduino
  model, chec
  https://www

  modified 8
  by Scott Fi
  modified 2
  by Arturo (
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or pow
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (H
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off b
  delay(1000);                       // wait for a second
}
```

Now, instead of using the standard upload button, you can select “Upload Using Programmer”. `Ctrl + Shift + U` is also pretty handy.


Blink | Arduino 1.8.6 Hourly Build 2018/07/19 10:19
File Edit Sketch Tools Help
Blink
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...

```
/*
  Blinl

  Turns                              f for one second, repeatedly.

  Most                               can control. On the UNO, MEGA and ZERO
  it is                              R1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
```

Turn on verbose output to see what’s going on better.

> **Note:** You can only use SparkFun boards for this to work right now (*SparkFun Redboard*, *SparkFun Makey Makey*, etc.). If it does get accepted into the arduino avr boards package, then it will work with all other AVRs.

Speed Test - How Fast Can This Thing Go?

For some fun, and to demonstrate the difference in programming speeds, we created a massive Arduino sketch, and recorded some of the programming speeds. Once compiled, this sketch used up a whopping 20440 bytes of memory – this is about 63% of the ATmega328P’s memory. For comparison, blink is only 930 bytes – or 2% of memory.

This test was done with a 5V Arduino running on a 16MHz Crystal (the SparkFun RedBoard), and so that allowed us to program reliably at 2MHz. If you are using a 3.3V Arduino at 8MHz, it would be smart to bring that down to 1MHz. Our experience in production programming at SparkFun has shown us that in fact 1/8 of oscillator speed is the best way to go for nearly 100% reliability.

Here were the speed test results:

- *Serial upload via Arduino IDE:*
    - 6.32s upload
    - 6.89s verify
    - **13.21s total**
- *ISP program via Arduino IDE and Pi AVR Programmer HAT:*

  - 4.04s write
  - 3.40s read
  - **7.44s total**
- *ISP program via Pi AVR Programmer HAT in stand alone mode – using test.py and pi_program.sh:*
  - 1.57s write
  - 0.74s read
  - **2.31s total**

As you can see from the results above, the Pi AVR Programmer HAT used directly from avrdude was much faster than the other options! Almost 11 seconds faster than the traditional serial upload. Dang!

Resources and Going Further

Now that you've successfully got your Pi AVR Programmer Hat up and running, it's time to incorporate it into your own project!

For more information about the SparkFun Pi AVR Programmer Hat, check out the resources below:

- Schematic
  - SparkFun Pi AVR Programmer HAT (PDF)
  - SparkFun Pi AVR Programmer Adapter (PDF)
- Eagle Files
  - SparkFun Pi AVR Programmer HAT
  - SparkFun Pi AVR Programmer Adapter
- AVRDUDE Manual
- Raspberry Pi Foundation: RPi Hardware SPI
- GitHub Repo: Pi AVR Programmer HAT – Here you'll find everything from PCB design files, and firmware to example programming setups.
- SFE Product Showcase: SparkFun Pi AVR Programmer HAT

We also recommend looking at the following resources:

- SparkFun
  - Raspberry Pi GPIO: Python RPi GPIO Example
  - SparkFun Bi-Directional Logic Level Converter
  - SparkFun AT42QT101X Capacitive Touch Breakout Hookup Guide
- Python for Beginners:File Handling Cheat Sheet in Python
- Ridge Solutions: Raspberry Pi – Restart / Shutdown Your Pi from Python Code

Want to check out some other SparkFun production processes or information about burning bootloaders on microcontrollers? Check out some of these related tutorials:

Installing an Arduino Bootloader

This tutorial will teach you what a bootloader is and why you would need to install or reinstall it. We will also go over the process of burning a bootloader by flashing a hex file to an Arduino microcontroller.

Using the Arduino Pro Mini 3.3V

This tutorial is your guide to all things Arduino Pro Mini. It explains what it is, what it's not, and how to get started using it.

Wireless Arduino Programming with Electric Imp

Reprogram your Arduino from anywhere in the world using the Tomatoless Boots wireless bootloader with the Electric Imp.

Pocket AVR Programmer Hookup Guide

Skip the bootloader and load your program directly onto an AVR with the AVR Pocket Programmer.

Raspberry Pi SPI and I2C Tutorial

How to use the serial buses on your Raspberry Pi.

Choosing an Arduino for Your Project

Examining the diverse world of Arduino boards and understanding the differences between them before choosing one for a project.