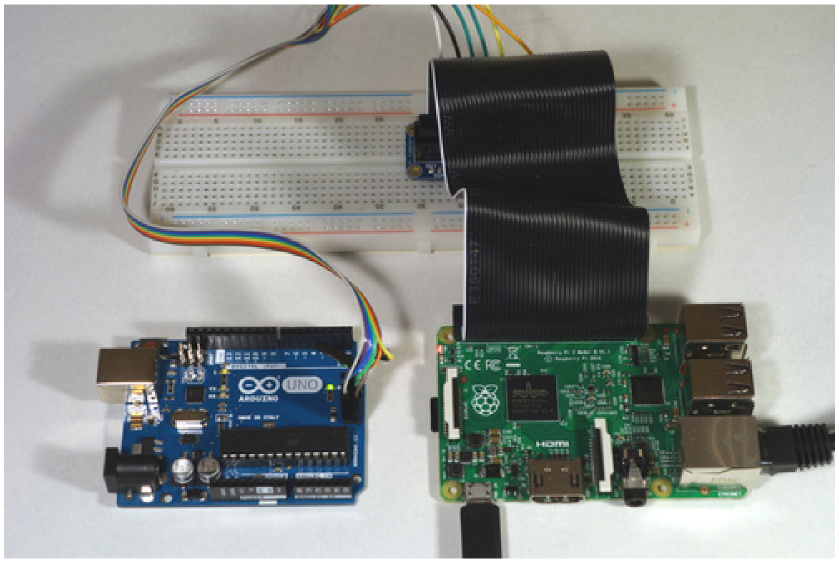
adafruit learning system

Program an AVR or Arduino Using Raspberry Pi GPIO

Created by Tony DiCola



Last updated on 2015-06-01 04:20:07 PM EDT

Guide Contents

Overview

This is a short guide to explain how to program an AVR microcontroller (like what powers an Arduino Uno) directly from the GPIO pins on a Raspberry Pi. Why would you want to program an AVR from a Pi? If you're building up an AVR-based board or product from scratch you'll need some way to program code onto the chip. Typically dedicated tools like the USBtinyISP (http://adafru.it/46) are used to program an AVR through its in-circuit serial programming (ISP or ICSP) pins, however with the latest version of the avrdude programming tool you can actually use Linux GPIO pins to program an AVR directly--no dedicated programmer required! This guide will walk you through how to install and use avrdude to program an AVR microchip or Arduino through its ISP pins with a Raspberry Pi.

Note that this tutorial is somewhat advanced and targeted at people who are familiar with programming an AVR chip directly instead of using friendlier tools like the Arduino IDE. It will help to be familiar with setting up a breadboard Arduino (http://adafru.it/fi4) and loading its bootloader through an ISP programmer. If you're a beginner or new to Arduino, stick with using the Arduino IDE to program your board (http://adafru.it/fi5).

To follow this guide you'll need the following hardware:

- **Raspberry Pi** - Any model will work, but note that you need 4 GPIO pins free for each AVR that will be programmed at a time.
- An **AVR chip & development board** or an **Arduino board** that exposes the ICSP pins (like an Arduino Uno).
- **Breadboard and wires** to connect the ICSP pins of the AVR/Arduino to GPIO pins on the Raspberry Pi.

Installation

You'll need to compile and install the latest version of the avrdude programming tool (http://adafru.it/fi6) on the Raspberry Pi. These instructions assume you're using a version of the Raspbian operating system (http://adafru.it/fi7) and are familiar with how to connect to the terminal on the Raspberry Pi (http://adafru.it/dZL).

Also note these instructions are written for **version 6.1** of avrdude. Earlier versions won't work because they don't have the required linuxgpio programmer type. Later versions might work, but they might require more dependencies or a different configuration. Stick with version 6.1 to follow these instructions.

Easy Install

To save some of the trouble of compiling avrdude a pre-compiled package for Raspbian has been added to Adafruit's Rasbian package repository. If you just want to get up and running as fast as possible follow these steps to install the package. If you run into an issue or would like to compile the code yourself, skip down to the compile avrdude section (http://adafru.it/fi8).

First follow the steps on this page to add Adafruit's package repository to your Raspberry Pi (http://adafru.it/fi9).

Once the Adafruit repository has been added and the package list refreshed, run the following command to install or upgrade the avrdude package:

```
sudo apt-get install avrdude
```

Answer yes to any question about installing packages, and after a moment the updated avrdude package should be installed. Skip down to the verify installation (http://adafru.it/fi8) section to continue.

Compile avrdude (Manual Install)

To manually compile avrdude follow the steps below. First run these commands to install some required dependencies:

```
sudo apt-get update
sudo apt-get install -y build-essential bison flex automake libelf-dev libusb-1.0-0-dev libusb-dev libftdi-dev libftc
```

Then download the source and configure avrdude for compilation by executing:

```
wget http://download.savannah.gnu.org/releases/avrdude/avrdude-6.1.tar.gz
tar xvfz avrdude-6.1.tar.gz
cd avrdude-6.1
./configure --enable-linuxgpio
```

After the configure script runs verify that it shows the line "ENABLED linuxgpio" like below (you can safely ignore the DISABLED and DON'T HAVE warnings):

```
Configuration summary:
----------------------
DO HAVE    libelf
DO HAVE    libusb
DO HAVE    libusb_1_0
DON'T HAVE libftdi1
DO HAVE    libftdi
DON'T HAVE libhid
DO HAVE    pthread
DISABLED   doc
DISABLED   parport
ENABLED    linuxgpio
```

Now build and install avrdude by running:

```
make
sudo make install
```

Verify Installation

Now that avrdude is compiled and installed you can verify the expected version 6.1 is installed. Run the 'avrdude -v' command (without quotes) and check that you see output like the following with version 6.1:

```
avrdude: Version 6.1, compiled on May 28 2015 at 18:41:48
         Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
         Copyright (c) 2007-2014 Joerg Wunsch

         System wide configuration file is "/usr/local/etc/avrdude.conf"
         User configuration file is "/home/pi/.avrduderc"


avrdude: no programmer has been specified on the command line or the config file
         Specify a programmer using the -c option and try again
```

Once avrdude 6.1 is compiled and installed, continue on to learn how to configure avrdude to program using the GPIO pins.

Configuration

Wiring

To program an AVR from the Pi you'll need to have 4 GPIO pins free on the Raspberry Pi. These pins will connect to the AVR's ISP/ICSP **MOSI**, **MISO**, **SCK**, and **RESET** pins. In addition you'll connect the Pi's 5 volt power and ground to the AVR to power it during programming.

If you're using an Arduino Uno you can access all the required pins from the small ICSP header on the far end of the board (http://adafru.it/fia).

As an example using an Arduino Uno here's one possible way to connect it to a Raspberry Pi:

- Arduino **ICSP VCC** to Raspberry Pi **5 volt pin**.
- Arduino **ICSP GND** to Raspberry Pi **ground pin**.
- Arduino **ICSP RESET** to Raspberry Pi **GPIO #12**.
- Arduino **ICSP SCK** to Raspberry Pi **GPIO #24**.
- Arduino **ICSP MOSI** to Raspberry Pi **GPIO #23**.
- Arduino **ICSP MISO** to Raspberry Pi **GPIO #18**.

Configuration

You'll need to create a custom avrdude configuration file to tell avrdude what GPIO pins to use for programming the AVR. It's easiest to copy the default avrdude.conf file and make the necessary changes to it.

The avrdude.conf file will be in one of two spots depending on how you installed avrdude in the previous section. If you did the **easy install method** to install through a package then avrdude.conf is in the **/etc/avrdude.conf** location. However if you did the **manual install method** to compile the code and install it then avrdude.conf is in the **/usr/local/etc/avrdude.conf** location.

Copy the avrdude.conf configuration to a new file called **avrdude_gpio.conf** in your home directory, and then edit it with the nano text editor by running the commands below. If you used the **easy install** method to install an avrdude package these commands are:

```
cp /etc/avrdude.conf ~/avrdude_gpio.conf
nano ~/avrdude_gpio.conf
```

However if you used the **manual install** method to compile the code then instead run these commands:

```
cp /usr/local/etc/avrdude.conf ~/avrdude_gpio.conf
nano ~/avrdude_gpio.conf
```

Once in the editor scroll all the way down to the very end of the file. You'll need to add a new section that configures the GPIO pin programming. Paste in the following section at the end of the file:

```
# Linux GPIO configuration for avrdude.
# Change the lines below to the GPIO pins connected to the AVR.
programmer
  id    = "pi_1";
  desc  = "Use the Linux sysfs interface to bitbang GPIO lines";
  type  = "linuxgpio";
  reset = 12;
  sck   = 24;
  mosi  = 23;
  miso  = 18;
;
```

This configuration will create a programmer with name 'pi_1' and assign the ICSP pins to the provided GPIO pin values. If you've hooked up the ICSP pins to different GPIO pins make sure to modify the configuration to match your setup.

Also note that you can create mulitple programmer sections if you're programming multiple AVR chips from the same Raspberry Pi. Each programmer section needs to have a unique name assigned with the **id = "name"** line. In addition each AVR needs a distinct set of 4 GPIO pins configured for its ICSP pins.

Once you've modified the configuration file save it and quit nano by pressing **Ctrl-O**, **enter**, and then **Ctrl-X**.

Continue on to learn how to use avrdude with the custom GPIO programming configuration.

Programming

Now you're ready to program the AVR using the Pi's GPIO pins! Make sure you've followed all the steps to this point and have compiled, installed, and configured avrdude for GPIO programming.

To program the chip you'll need a .hex file that has the compiled code you wish to run. An easy way to generate this file is to turn on the verbose compile and upload output in the Arduino IDE (under preferences). Configure Arduino for the board/chip you're using and compile the code. In the debug output look for the line at the end that calls avr-objcopy to prepare the .hex file, for example compiling a Blink example on my machine produced the following:

```
/home/tony/Programs/arduino-1.6.4/hardware/tools/avr/bin/avr-objcopy -O ihex -R .eeprom /tmp/build61546102
```

The file /tmp/build6154610255332504576.tmp/Blink.cpp.hex is the output .hex file that can be written directly to the AVR/Arduino using avrdude. Grab the .hex file and copy it to your Raspberry Pi.

Another way to generate a .hex file is to setup your own avr-gcc toolchain and compile code for the desired chip. This can be a somewhat daunting process so I recommend using Arduino's prebuilt toolchain. However if you're going at it yourself take a look at the instructions on setting up avr-gcc here (http://adafru.it/fib)--good luck!

Now from the Raspberry Pi run an avrdude command to verify it can connect to the AVR chip:

```
sudo avrdude -p atmega328p -C ~/avrdude_gpio.conf -c pi_1 -v
```

Notice the chip type is specified with **-p**, the path to the custom avrdude.conf is specified with **-C**, and the name of the programmer is specified with **-c**. If you're using a different chip, configuration file path, or programmer name be sure to change the values.

Once the command runs you should see output like the following with avrdude reading the memory & fuses of the AVR chip:

```
avrdude: Version 6.1, compiled on May 28 2015 at 18:41:48
         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                    : unknown
        Using Programmer              : pi_1
        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 : linuxgpio
        Description     : Use the Linux sysfs interface to bitbang GPIO lines
        Pin assignment  : /sys/class/gpio/gpio{n}
          RESET   =  12
          SCK     =  24
          MOSI    =  23
          MISO    =  18

avrdude: AVR device initialized and ready to accept instructions

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

avrdude: Device signature = 0x1e950f
avrdude: safemode: lfuse reads as FF
avrdude: safemode: hfuse reads as D6
avrdude: safemode: efuse reads as 5
```

```
avrdude: safemode: lfuse reads as FF
avrdude: safemode: hfuse reads as D6
avrdude: safemode: efuse reads as 5
avrdude: safemode: Fuses OK (E:05, H:D6, L:FF)

avrdude done.  Thank you.
```

If you see an error make sure you're running the command as root with sudo. Also go back and ensure the custom avrdude configuration exists in the expected location, the configuration has the specified programmer, and the programmer configuration is using the GPIO pins you have connected to the AVR.

Also note if you receive an error that a GPIO pin is in use try changing that pin to use a different one in the programmer configuration.

Once you've verified avrdude can talk to the chip you're ready to program it! You'll need the .hex file that was compiled, then run a command like the following to program the chip:

```
sudo avrdude -p atmega328p -C ~/avrdude_gpio.conf -c pi_1 -v -U flash:w:Blink.cpp.hex:i
```

Again the **-p** option configures the chip to program, the **-C** option points to your custom avrdude configuration, and the **-c** option specifies the custom programmer configuration block to use. The **-U flash...** option tells avrdude to wipe and program the flash memory of the AVR chip, and to use the provided **Blink.cpp.hex** file. Change the name and path to the hex file to the appropriate value for the code you're writing to the chip.

If avrdude successfully programs the chip you should see output like the following:

```
avrdude: Version 6.1, compiled on May 28 2015 at 18:41:48
         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                    : unknown
         Using Programmer              : pi_1
         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 : linuxgpio
        Description     : Use the Linux sysfs interface to bitbang GPIO lines
        Pin assignment  : /sys/class/gpio/gpio{n}
          RESET   =  12
          SCK     =  24
          MOSI    =  23
          MISO    =  18

avrdude: AVR device initialized and ready to accept instructions

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

avrdude: Device signature = 0x1e950f
avrdude: safemode: lfuse reads as FF
avrdude: safemode: hfuse reads as D6
avrdude: safemode: efuse reads as 5
avrdude: NOTE: "flash" memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "Blink.cpp.hex"
avrdude: writing flash (1030 bytes):
```

```
Writing | ################################################## | 100% 0.89s

avrdude: 1030 bytes of flash written
avrdude: verifying flash memory against Blink.cpp.hex:
avrdude: load data flash data from input file Blink.cpp.hex:
avrdude: input file Blink.cpp.hex contains 1030 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.87s

avrdude: verifying ...
avrdude: 1030 bytes of flash verified

avrdude: safemode: lfuse reads as FF
avrdude: safemode: hfuse reads as D6
avrdude: safemode: efuse reads as 5
avrdude: safemode: Fuses OK (E:05, H:D6, L:FF)

avrdude done.  Thank you.
```

Congratulations, you've successfully programmed an AVR chip using the GPIO pins on a Raspberry Pi!

Arduino Bootloader Note

If you program an Arduino board like the Uno using the method in this guide be aware that you might remove or overwrite the USB/serial bootloader. This can actually be useful if you need just a little more space in your sketch (the bootloader takes a few kilobytes of memory at the end of flash), however it means that you won't be able to program the Arduino using the Arduino IDE's normal USB/serial programming interface. Don't worry though you can burn the original Arduino bootloader back on your Arduino board to have it work with the Arduino IDE again.

Check out this tutorial for information on burning an Arduino bootloader from a .hex file using avrdude (http://adafru.it/fii). If you download the Arduino IDE look for a firmware .hex file in the **hardware/arduino/avr/bootloaders/optiboot** directory beneath the IDE's installation. For an Arduino Uno the **optiboot_atmega328.hex** file is the bootloader file to use. Program this hex file to the Arduino using avrdude on the Raspberry Pi to restore the Arduino to its normal state and make programming work with the Arduino IDE again.