## Setup lab: make sure R/pi hardware and its toolchain works.

This is a relatively trivial lab to make sure your r/pi (model 1 A+) and ARM toolchain works.

- **Make sure you do the prelab first!**
- In addition, there is now a POSTLAB that lists out many of the common solutions to problems that people ran into

There's a lot of fiddly little details in getting a pi working, and this lab is many people's first time working with hardware, so we break the lab down into many (perhaps too-many) small steps. Our theorem for this class: the smaller the step, the more obvious what the actual (distal) error is and the less time you have to spend debugging.

We'll use different variations of a blinky light using GPIO pin 20 (second from bottom on the right):

1. You'll turn on an LED manually;
2. then use a pre-compiled program loaded from the SD card (why not skip 1?);
3. then use a bootloader (why not skip 2?);
4. then install the r/pi tool chain, compile a given assembly version and use it (why not skip 3?);
5. then write your own blink program and compile: this is the fun part. It's also the longest (why not skip 4?).

The final sign off for the lab:

- Show you have a working `blink` compiled using the tool chain and started using the bootloader.
- Bonus: write up a short example problem you ran into (or someone you helped ran into) and how you solved them and post to the newsgroup so we can start accumulating examples.

Note: you obviously have to do your own work, but please help others if they get stuck or if you figure out something. This is the kind of class where talking over things will make everything a lot clearer, and where having multiple people debug an issue is an order of magnitude more effective than working alone. If you think of a way of explaining something that's clearer than the documentation or the README, post it in the newsgroup so others can benefit from your insight!

---

## Crucial: Life and death rules for pi hardware.

**Always**:

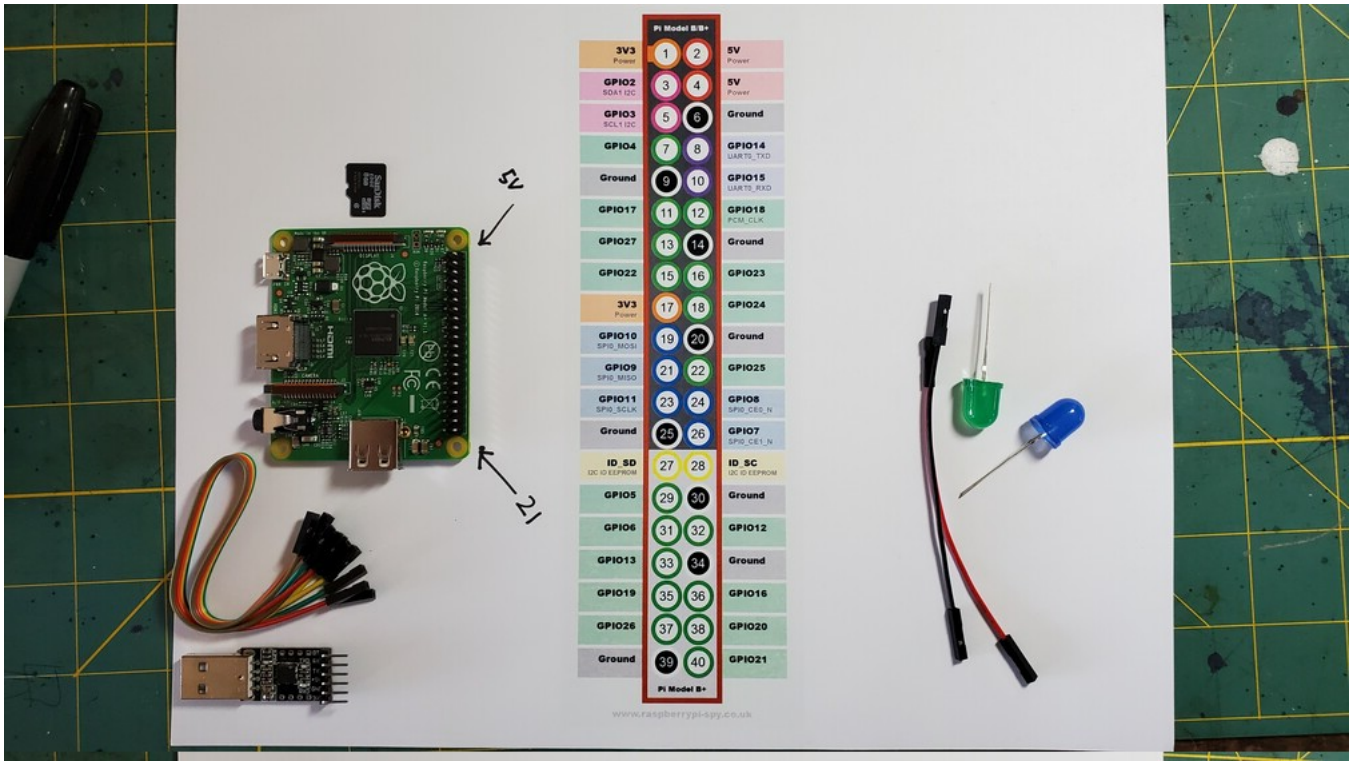
1. Whenever you make a hardware change --- messing with wires, pulling the SD card in/out --- ***make sure pi is disconnected from your laptop***. It's too easy to short something and fry your hardware. Also, pulling the SD card out while under power sometimes causes corruption when some bytes have been written out by your laptop and others have not.
2. If anything ever gets hot to the touch --- the serial device, the pi --- ***DISCONNECT***! Sometimes devices have manufacturing errors (welcome to hardware), sometimes you've made a mistake. Any of these can lead to frying the device or, in the worst case, your laptop. So don't assume you have a safety net: it's up to you to avert disaster.
3. While the pi has its own unique features, it's also like all other computers you've worked with: if it's not responding, "reboot" and retry by unplugging it (to remove power) and reconnect.

---

### 0. Make sure you have everything.

You should have:

1. two or more R/PI A+ (or Zero);
2. two or more microSD card and adapter;
3. two or more CP2102 USB-TTL adapter;
4. a bunch of LEDs
5. a bunch of female-female jumpers;
6. printout of the pi's pins (digital is okay, but a printout will be more convenient).



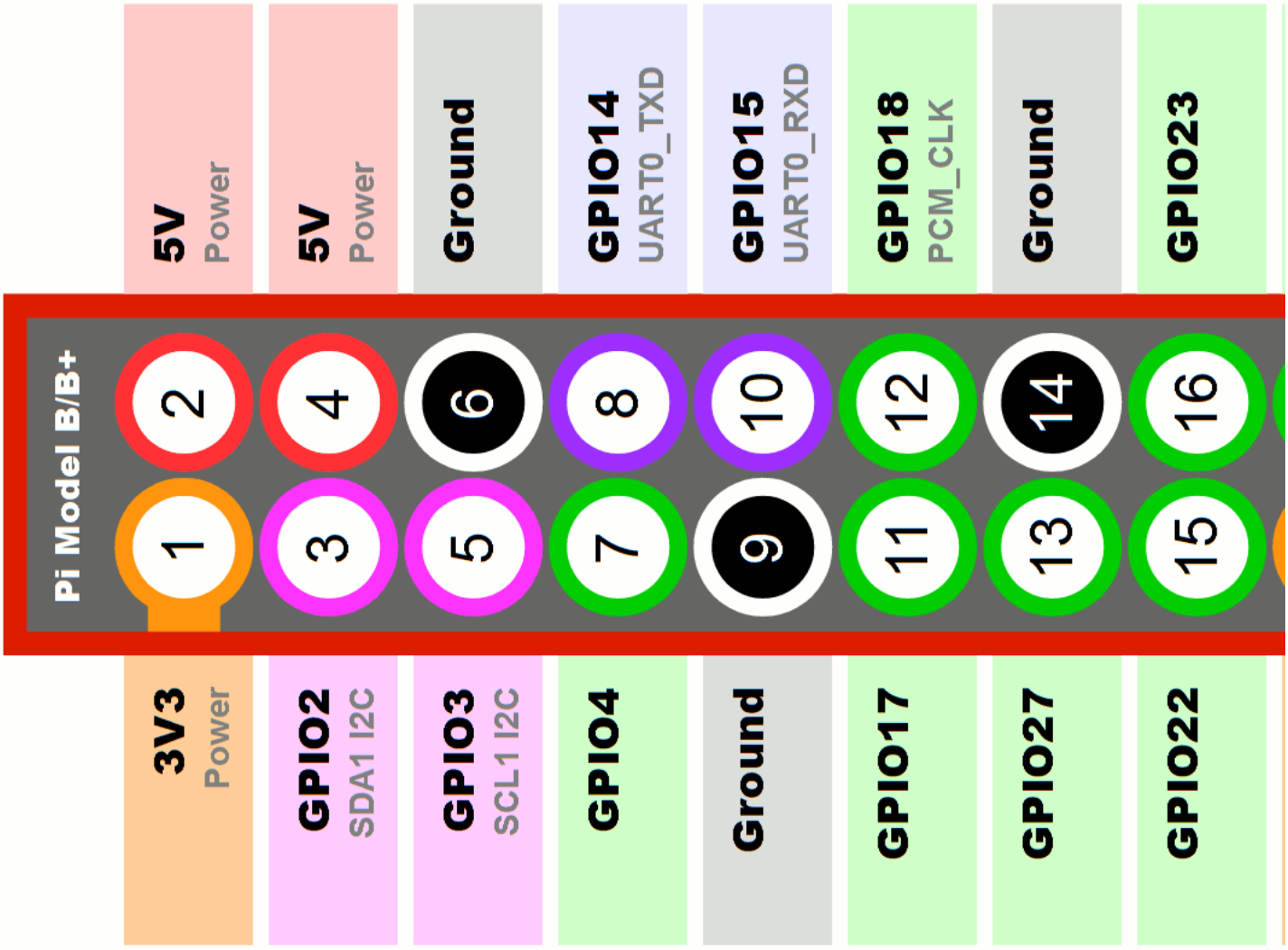

The pi is oriented with the two rows of pins on the right of the board. The two pins at the top of the rightmost row are 5v, the top left 3v. When connecting things I usually try to use the bottom left ground pin, and the bottom right GPIO 21 pin, since these are impossible to miscount.

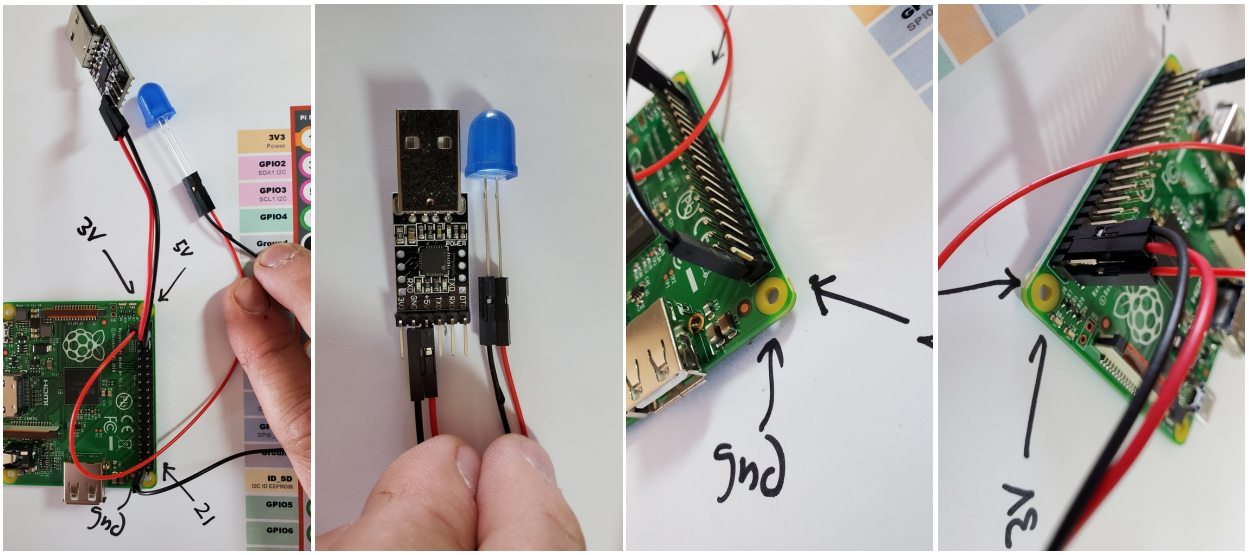
---

### 1. Make sure hardware is working:

Before we mess with software that will control hardware, we first make sure the hardware works: When doing something for the first time, you always want to isolate into small pieces.

So as our first step, we will use the USB-TTY to power the pi, and use the pi's power to directly turn on an LED. This tests some basic hardware and that you know how to wire.




Mechanically:

1. Connect the USB-to-TTL Serial cable's power (red) and ground (black) wires to a 5v pin (not the 3v pin) and ground pins on the pi that are next to each other (see your printout; upper right corner).
2. Plug the USB-TTY into your laptop's USB port.
3. Using a red (power) and black(ground) female jumper wires, connect your LED to an unused ground pin and the 3v power pin (there are several) to make sure the hardware components work, and you know how get them to.

If the LED doesn't go on, reverse its connections. You'll note that one leg of the LED is longer than the other. This is used to indicate which one is for power, which is for ground. You can empirically determine which is which.

~~If still doesn't go on, plug someone else's working version into your computer. If that doesn't work, ask.~~ If it still doesn't go on, try with your other Pi and/or another LED. If that doesn't work, ask. (Thanks to virtual classes, you are your own lab partner. Don't hesitate to ask others for help if something isn't working; normally you'd be side-by-side with a few dozen others who could help you.)

(Note: the color of the wire does not matter for electricity, but it makes it much easier to keep track of what goes where: please use red for power, black for ground.)

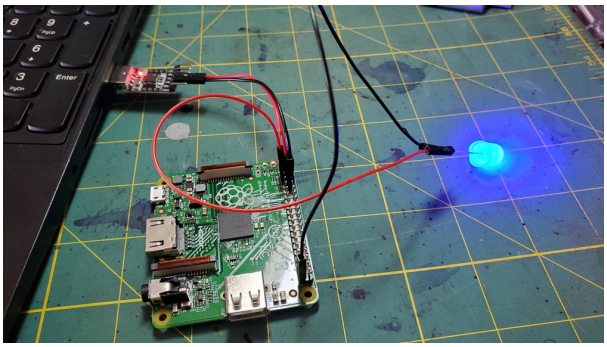
4. Try another LED and try some of the other ground and power pins.

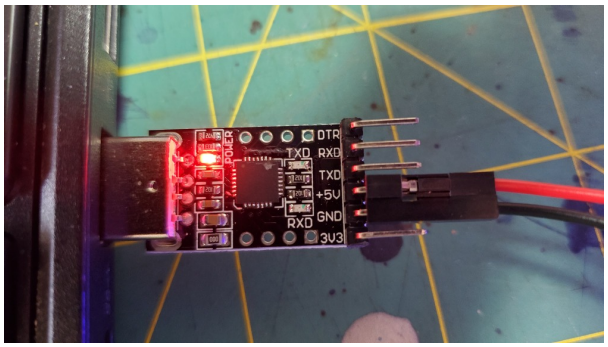
(EE folks: We don't use a breadboard b/c it's bulky; we don't use resistors for the same reason + the LED is big enough we generally don't fry it.)

What can go wrong:

1. If your USB-to-TTL starts heating up, disconnect! It appears that 1 in 8 is defective.
2. If your pi starts heating up, now or ever, disconnect! If you have a short, where a pin with power feeds right into ground, you'll fry it.
3. If you see smoke, disconnect! Smoke means something has fried, and the longer you leave it plugged in the more things will get destroyed.

Success looks like the following photos:






---

### 2. Make sure you're able to install firmware, etc:

Now you'll run a precompiled program ( `part1/blink-pin20.bin` ) on the pi and make sure it can blink pin 20.

Note: next week when you develop your own remote bootloader (see next step) you have to use this SD card method repeatedly to load new versions, so pay attention to how you do it on your computer!

Mechanically:

1. Unplug the USB-TTY.
2. Plug SD card into your computer and figure out where it's mounted.
3. As discussed in the `PRELAB` , copy all the files from class `firmware` directory onto the SD card. Then copy `part1/blink-pin20.bin` to the SD card as `kernel.img` . Then type `sync` and then eject the SD card (don't just pull it out! data may not be written out.)

   For me, this is:

   ```
   #  from the 0-blink directory
   % cp ../../firmware/* /media/engler/0330-444/
   % cp part1/blink-pin20.bin /media/engler/0330-444/kernel.img
   % sync
   ```

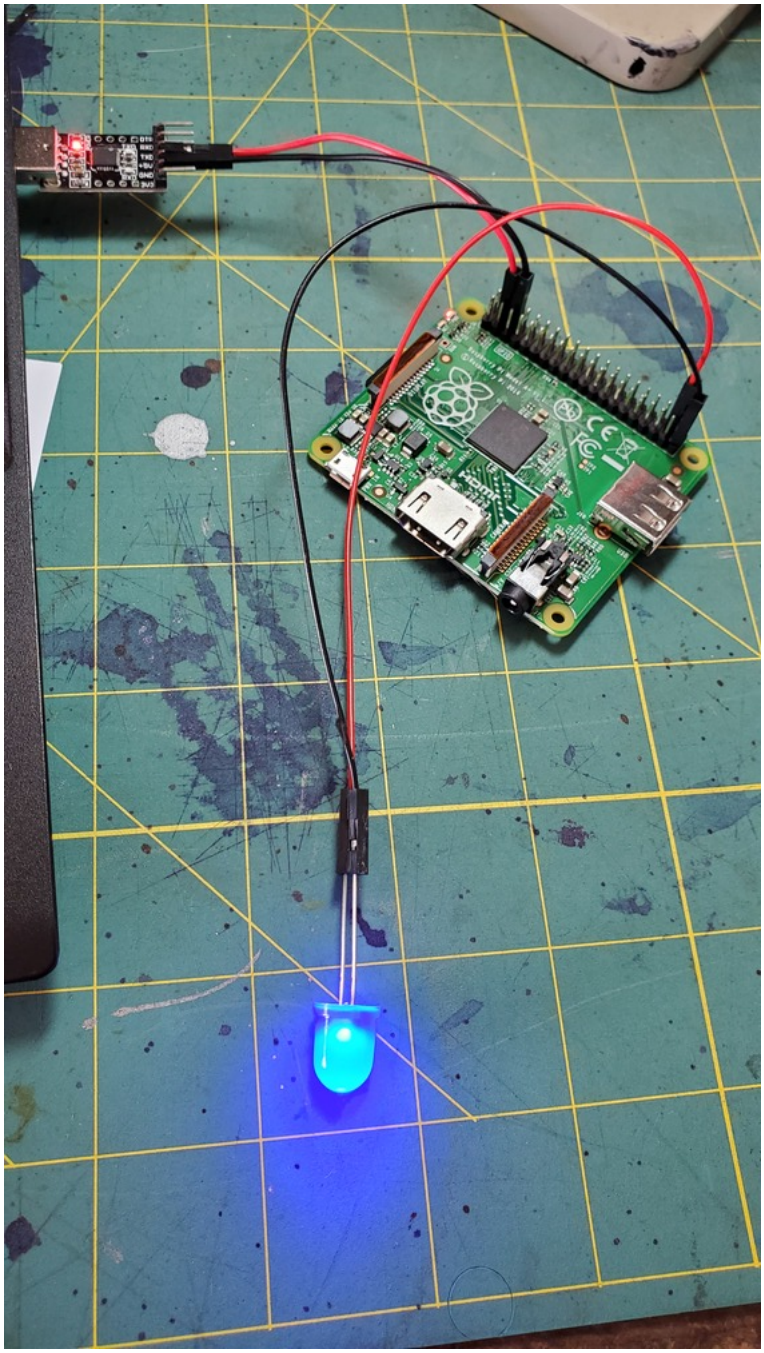
   On linux 20.04: the cards are often shipped with a corrupt FAT32 file system. Linux seems unable to repair this and will mount read-only. I had to mount it on a windows machine and format it.
4. With your pi disconnected from your laptop, reconnect the LED power from the 3v pin to gpio pin 20.
5. Plug the SD card into your pi -- you should feel a "click" when you push it in.
6. Plug in the USB-TTY to your USB to power the pi. The pi will jump to whatever code is in `kernel.img` .

The LED should be blinking. If you get this working, please help anyone else that is stuck so we all kind of stay about the same speed.

Troubleshooting:

0. First try to trouble shooting from part 1.
1. If it's not blinking, swap in someone else's card that is working.
2. If that works, compare their SD card to yours.
3. If that doesn't work, try your card in their rpi.

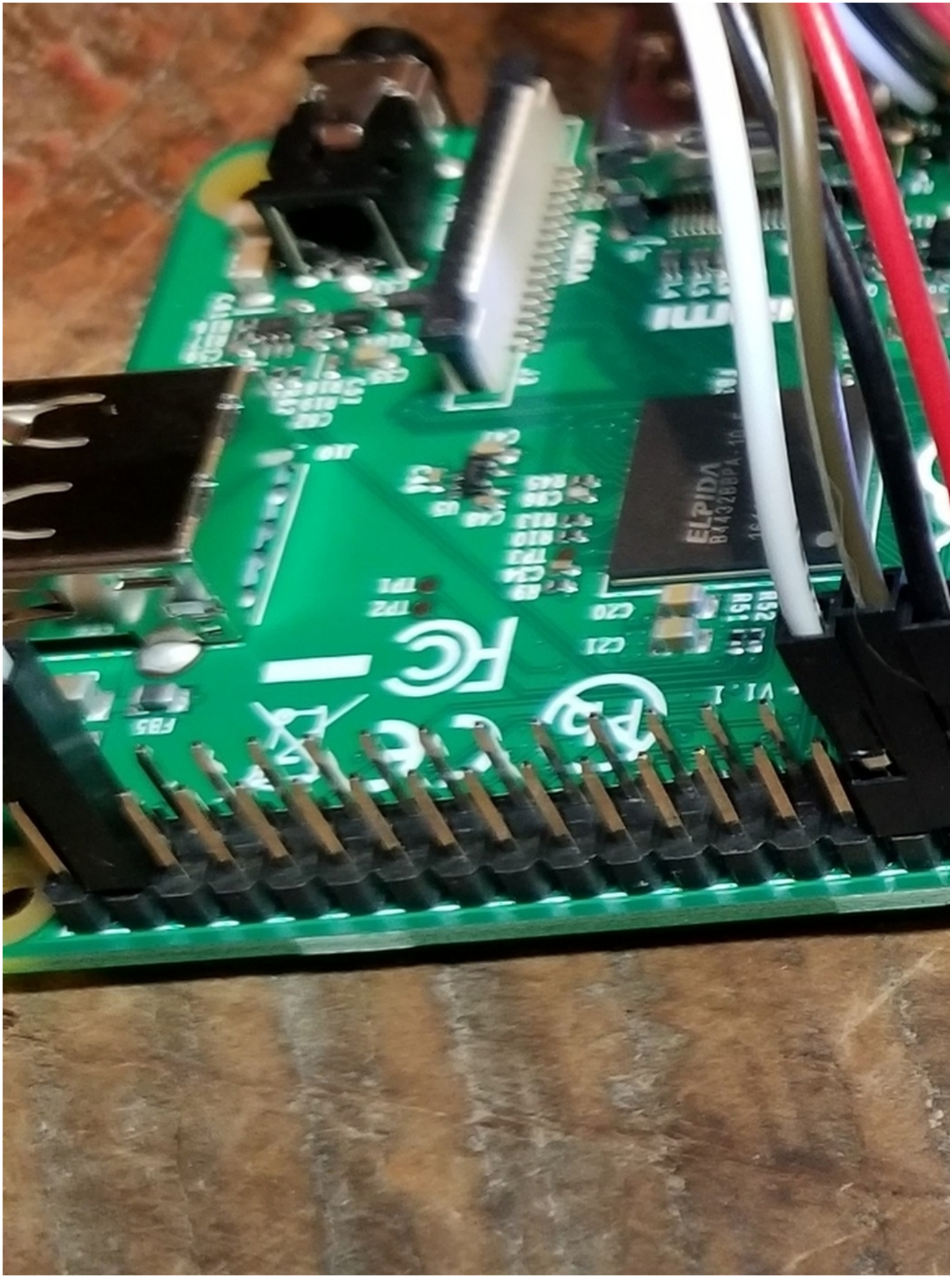
Success looks like:

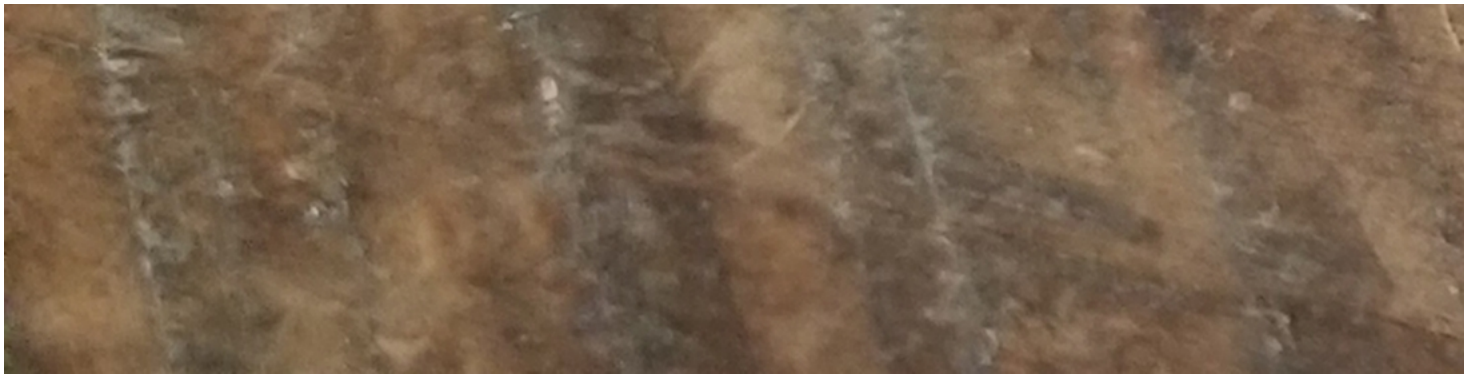


### 3. Send a new pi program from your computer rather than SD card.

As you've noticed, running new programs on the pi using the SD card method is tedious. We now set up your system so you can send programs directly from your computer to a plugged-in pi.

Method: install a program (which we somewhat inaccurately call a "bootloader") on the SD card (as before) that spins in a loop, waiting for your laptop to send a program over the UART-TTL. If the bootloader successfully receives a program, it copies it into pi memory, and then jumps to it. We currently give you a pre-compiled version ( `firmware/bootloader.bin` ). The next lab will have you implement your own.


ELPIDA



Mac:

- Download and install the drivers for a CP210x USB-to-UART driver as described in the cs107e docs.
- Make sure you reboot after doing so! (Go Apple!)

Linux:

- You shouldn't need drivers, however you may need to add yourself to the `dialout` group (or `serial`) depending on the distribution.

```
sudo adduser <your username> dialout
```

  If you do this, make sure you login and logout.

  If that still doesn't work, you may have to remove `modemmanager`:

```
sudo apt-get remove modemmanager
```

  In this case you may need to reboot.

Mechanically:

0. Unplug your pi. Don't modify your current wiring.
1. Did you unplug your pi?
2. Copy `firmware/bootloader.bin` on your SD card as `kernel.img` (see a pattern?), `sync`, and plug back into the pi.

```
# from the 0-blink directory
% cp ../../firmware/bootloader.bin /media/engler/0330-444/kernel.img
% sync
# now safe to unplug the SD card
```

3. Hook the TX and RX wires up to the pi. Do you TX/TX and RX/RX or switch them? (Hint: Think about the semantics of TX (transmit) and RX (receive).)
4. Copy `bin/pi-install.linux` or `bin/pi-install.*.macos` (`arm` or `x86_64`) to your local `bin/pi-install`. Make sure when you type `pi-install` something happens! If not, make sure your local `~/bin/` directory is in your path, and that you have sourced your shell startup file.

   On `tcsh` on my linux machine, after plugging the pi back in:

```
% cp ../../bin/pi-install.linux ~/bin/pi-install
% rehash  # so the shell refreshes its cache
% pi-install part1/blink-pin20.bin
# the pi is now blinking.
```

5. Your LED should be blinking, just like before.
6. If you unplug your pi and re-plug it in, you should be able to run a hello program:

```
%  pi-install part1/hello.bin
opened tty port </dev/ttyUSB0>.
pi-install: tty-usb=</dev/ttyUSB0> program=<part1/hello.bin>
...
listening on ttyusb=</dev/ttyUSB0>
hello world
DONE!!!
```

   It exits in such a way that you can rerun it multiple times.

Trouble shooting:

- if `pi-install` can't find the ttl-usb device, run `ls -lrt /dev` and look for a something with `USB` in its name towards the end. You can give the specific device as an argument:

```
% ls -lrt /dev
... lots of stuff ...
drwxr-xr-x  4 root root            80 Mar 31 19:03 serial
drwxr-xr-x  2 root root          4420 Mar 31 19:03 char
crw-rw----  1 root dialout 188,     0 Mar 31 19:07 ttyUSB0
```

```
drwxrwxrwt  2 root root            40 Mar 31 19:09 shm
crw-rw-rw-  1 root tty       5,     2 Mar 31 19:09 ptmx

# use ttyUSB0
%  pi-install /dev/ttyUSB0 part1/hello.bin
```

---

### 4. Make sure your r/pi toolchain is working.

For this class you need to compile bare-metal r/pi programs on your computer, which is most likely not a bare-metal r/pi itself. Thus we need to set up the tools needed to `cross-compile` r/pi programs on your computer and to r/pi binaries.

Install the toolchain:

- For a mac use the cs107e install notes. Note: do not install the python stuff.
- For ubuntu/linux, ARM recently changed their method for distributing the tool change. Now you must manually install. As of this lab, the following works:

```
wget https://developer.arm.com/-/media/Files/downloads/gnu-rm/10.3-2021.10/gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2

sudo tar xjf gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2 -C /usr/opt/
```

Then either add symlinks to these:

```
sudo ln -s /usr/opt/gcc-arm-none-eabi-10.3-2021.10/bin/* /usr/bin/
```

Or, cleaner, add `/usr/opt/gcc-arm-none-eabi-10.3-2021.10/bin` to your `path` variable in your shell configuration file (e.g., `.tchsrc` or `.bashrc`), save it, and `source` the configuration. When you run:

```
    arm-none-eabi-gcc
    arm-none-eabi-ar
    arm-none-eabi-objdump

You should not get a "Command not found" error.


You may also have to add your username to the `dialout` group.

If gcc can't find header files, try:

    sudo apt-get install libnewlib-arm-none-eabi
```

Now test that the toolchain works and produces a runnable binary:

1. Reset your pi: unplug the TTY-USB then plug it back in to your laptop.
2. Compile and bootload `part2/blink-pin20.s` using the shell script.

```
% cd part2
% sh make.sh     # compile blink-pin20.s to blink-pin20.bin
% ls
blink-pin20.bin  blink-pin20.s    make.sh  README.md
% pi-install blink-pin20.bin    # send it to the pi.
```

3. If everything worked, your LED light should be blinking. Congratulations!

---

### 5. Extension: Run two pi's at once

Because of supply chain issues, we don't have enough pi's (yet?) to give out two. However, if you are working in a group, it's worth getting two pi's running at the same time --- this will clarify issues and also make it easier to do networking.

Configuring your second pi is a great way to re-enforce the steps above. Also you're going to want two working systems at all times. It will make it much easier to isolate what a problem is by swapping pi's and then (if needed) swapping components (e.g., switching SD cards). Finally, you will need two pi's for the networking labs.

The steps:

1. Configure your second pi as above. The microSD will likely have a different name.

```
# from the 0-blink directory
% cp ../../firmware/* /media/engler/sdd1/
% cp ../../firmware/bootloader.bin  /media/engler/sdd1/kernel.img
% sync
# unplug the microSD from your laptop and plug into your pi

# as a quick test: run hello since it doesn't need wiring
% pi-install part1/hello.bin
find-ttyusb.c:find_ttyusb:55:FOUND: </dev/ttyUSB0>
opened tty port </dev/ttyUSB0>.
... stuff ...
hello world
DONE!!!
```

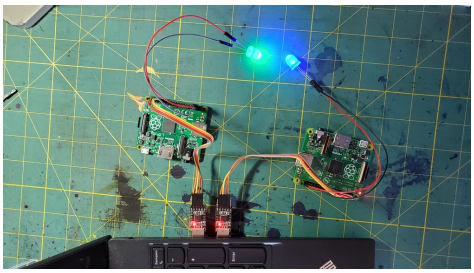
2. In two different terminal windows, run `blink-pin20.bin` on each. You will need to specify each distinct TTL-USB device on the `pi-install` command line, otherwise `pi-install` will try to load the same one (the last mounted device).

   For me (again: Linux. MacOS will have a different device mount point.):

   ```
   # in one terminal
   % pi-install /dev/ttyUSB0 part1/blink-pin20.bin

   # in another terminal
   % pi-install /dev/ttyUSB1 part1/blink-pin20.bin
   ```

   Success looks like:



Trouble shooting:

1. If running blink does not work try running `part1/hello.bin` (an LED wire might have come loose.)
2. If that doesn't fix it, check that a small red light is blinking when you plug in the pi. If not, the bootloader isn't running. Unplug/plug-in the pi and see if that fixes it. If not, unplug it and check the microSD is all the way in. If not, check the TTY connections on both the device and the pi.

To see where the devices are loaded, you can do `ls -lrt /dev/` as above. On many Unix systems you can also look at the end of the kernel log. For example, on Linux:

```
% tail -f /var/log/kern.log
# plug in the device
[105660.736891] usb 1-2: new full-speed USB device number 22
... lots of stuff ...
[105660.933050] cp210x 1-2:1.0: cp210x converter detected
[105660.935793] usb 1-2: cp210x converter now attached to ttyUSB0
```

We can see it's connected to `ttyUSB0`. Using `tail -f` lets us see immediately as the log file changes (e.g., when you plug in or pull out your TTL-usb).

Your system likely has an even better way; so it's worth searching online.