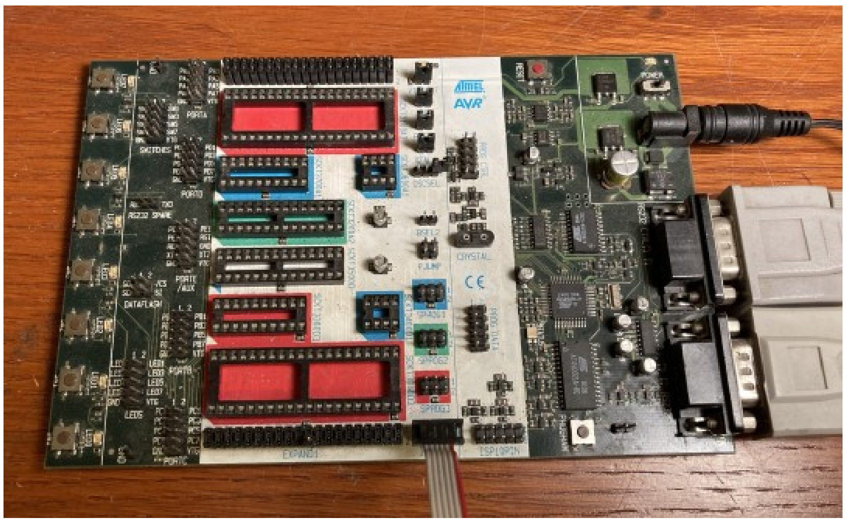
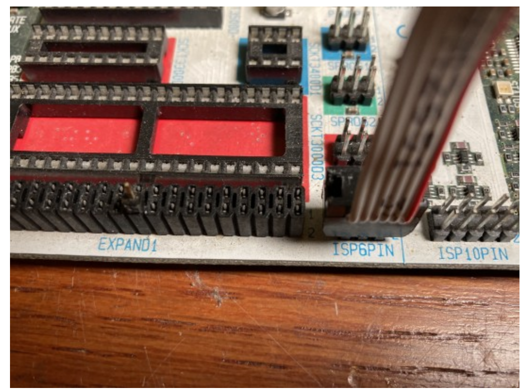
Instructions to setup Arduino UNO rev3 for ErgWare

I had several questions about how to use ErgWare with an Arduino. Below are the instructions on how to do it. I have tested everything including the chopper (see step 15 for testing the chopper part).

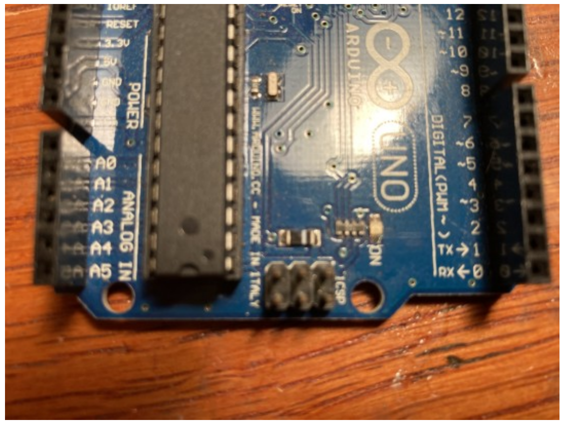
Step 1. Identify your programming board. I use the stk500. You might have the usbasp, or PonyProg or something else. The important thing is that it has to have an ISP programming interface. Some of these programmers are 10-pin, some are 6-pin. If you have the 10-pin type you'll have to get or make an adapter to 6-pin because the arduino uses 6-pin. My programming board has both.



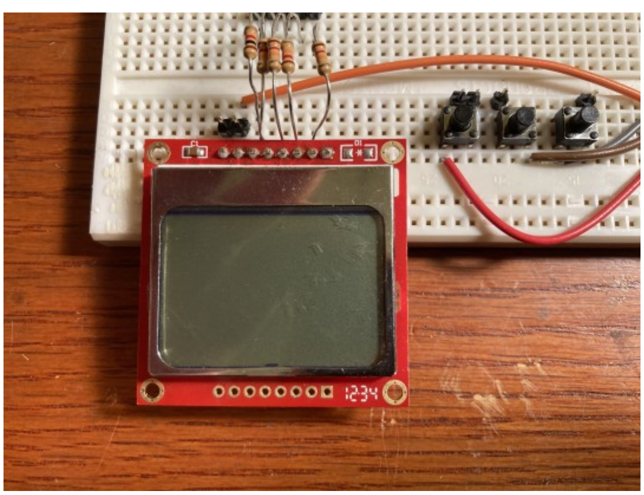


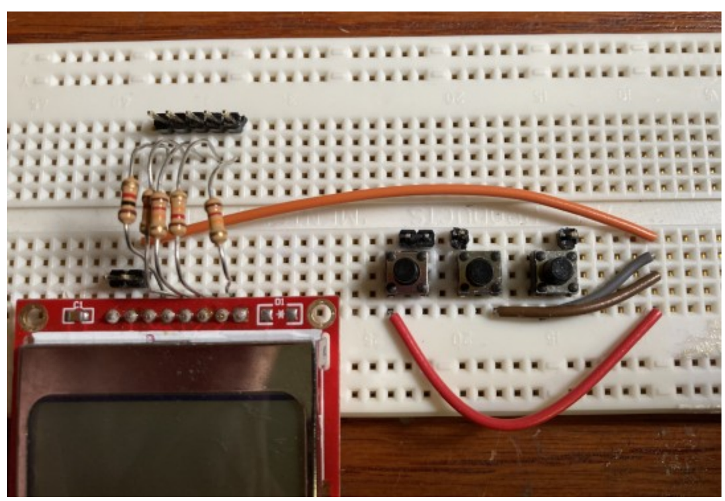
Step 2. Go get your Arduino.

Step 3. Identify the right ISP header on your Arduino. It is the one by the end of the board. It was labeled ICSP on mine.

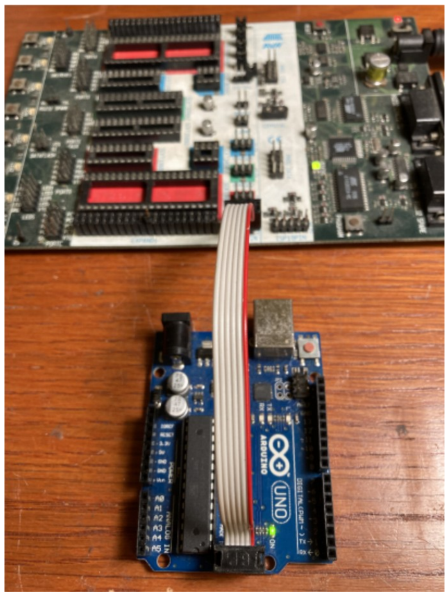


Step 4. Get the other parts and breadboard them. The Nokia 5110 LCD and 3 switches & a few wires and posts and a small breadboard. Yes, this step always takes time so make yourself a coffee. Note the **Nokia 5110 LCD must be powered (Vcc) off of the 3.3V** on the Arduino, otherwise it might get damaged. I am showing resistors in-line for the other pins, **but in the end I did not use them**. You don't need them, though they might protect the screen a bit longer. Use them if you want. **You can actually do this step whenever you want before Step 13 since we aren't yet going to hook anything up to the Arduino … we'll do that in Step 13.** Also, I did not wire up the backlight for this test but in Step 13 I explain how to do that. You might also need to solder some stuff, like pins for the LCD.

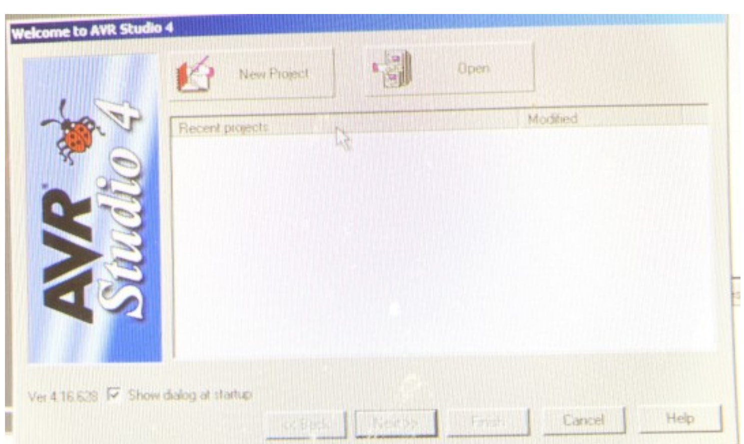




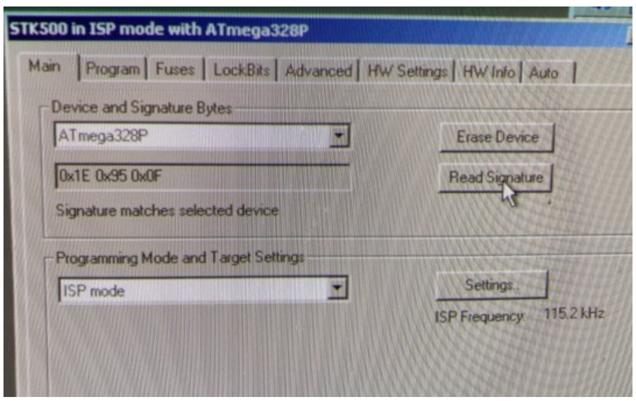
Step 5. Hook up the ISP conector on the arduino to your programmer. Note, you have to get it the right way around or it won't work. You will find out soon if you have it right. Here is what mine looked like. **Note that I am not using any external power to the Arduino.** No USB. No power cable. Nothing. All of the power is coming in over the ISP cable. Note also the voltage level **must be** 5V on your programmer, not 3.3V because the Atmega328 on the Arduino Uno runs at 16MHz and requires 5V to run at that speed.

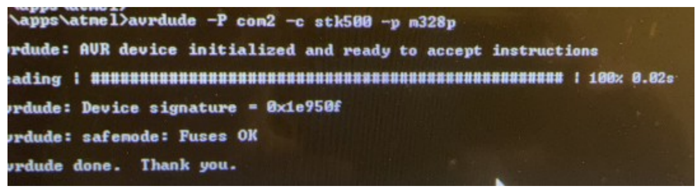


Step 6. Open up your programming environment. **You will NOT be using the standard Arduino environment you might be used to (though there might be a way to do it, I am not going to spend the time to figure that out right now).** I use Atmel Studio 4, version 4.16.628. It is very old but in works for me. I am not sure it would be compatible with some of the newer programmers like usbasp. You can also just use the avrdude command line. You can also use programs like AVRDUDESS & a bunch of others on the web. All of these do the same thing (allow you to program the chips with a nicer interface compared to the avrdude command line!) and will work. They allow you to access the core device with a nice GUI which is what we want to do.

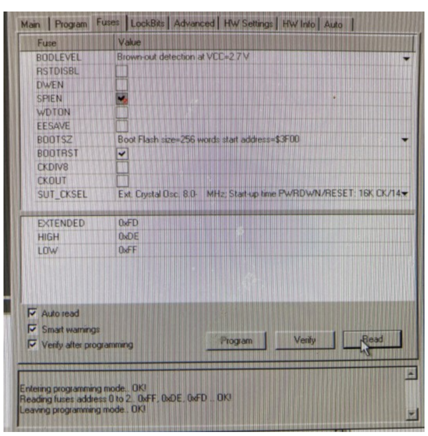


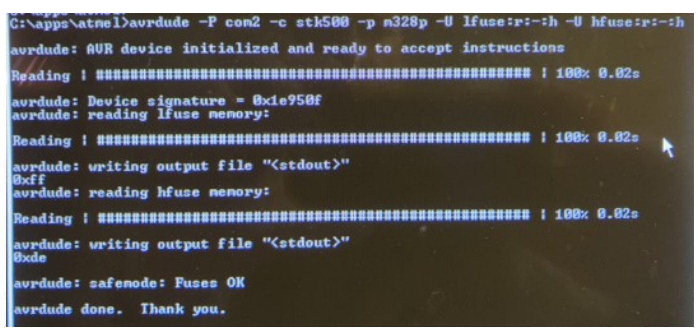
Step 7. This is the first really important step. Try to read the “device signature”. Below you can see screenshots of success in Atmel Studio 4 and on the avrdude command line. **Pay special attention to what you write on the avrdude command line. If you just type something randomly that you found on the web you might screw things up & “brick” your device.** The signature is 0x1E950F of this device. If you don’t get this to work, **STOP AND FIGURE IT OUT.** Something is wrong either with your programmer, or your wiring or something. This step needs to work, don’t try to force something with -F or anything in avrdude. If you think you have to “force” it with -F, something is wrong.





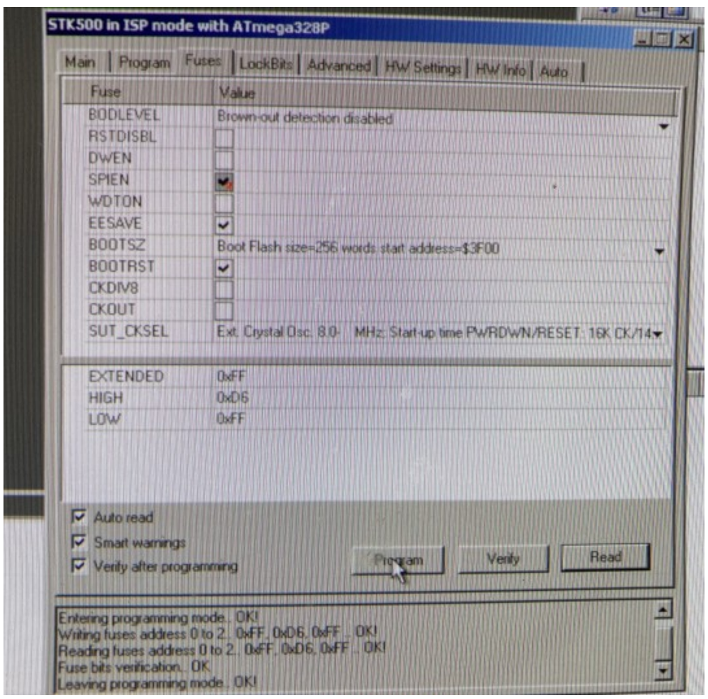
Step 8. We want to read the fuses (do **not** write them). This is just to ensure they can be read. My high fuse (Hfuse) read 0xDE, my low fuse (Lfuse) read 0xFF and my Extended fuse read 0xFD. This is what an “out of the box” Arduino reads, I think.





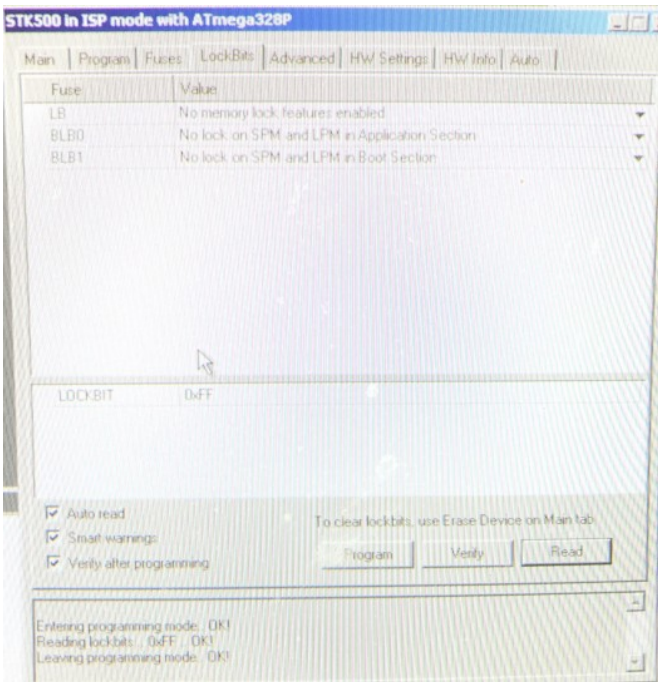
Step 9. We are now (sort of) at the point of no return. After steps 9 and 10 your Arduino will no longer function “like an Arduino”. What this means is we are going to “overwrite” the Arduino bootloader so it will no longer work with the standard Arduino programming interface. So don’t proceed unless you really want to do this and/or have confidence about your skills .... there is a way to recover the bootloader & full Arduino functionality if you want, but I won’t go into that procedure right now. We are also going to set the EESave fuse since we will be using/saving the Eeprom. I also disabled brown-out detection. Now we need to program the fuses. Double, triple, quadruple check at this point before you hit “program” in your program or “enter” on the avrdude command line. The fuse settings need to be:

Extended: 0xFF
High: 0xD6
Low: 0xFF



Step 10. We also need to re-program the “lockbits”. In Atmel Studio it says “to Clear Lockbits, use Erase device on Main tab”. So I went to the “main” tab and erased the device. It worked fine. There is also an avrdude command to erase the device
For my setup it would be:
avrdude -P com2 -c stk500 -p m328p -e

Your command might look a bit different in the -P and -c parts, but the -p m328p and -e should be the same. I am sure other programs have similar capability. The final lockbit setting we want is **0xFF.** After this is done, the arduino board is now ready to be programmed.



Step 11. The next step is to fill up the EEPROM with the fonts. We are going to flash some code to flash memory to run this instead of just flashing the eeprom. Why? Only because I thought some people might want to change the code themselves so I gave the source code as well for the font table. In any case, go to the github

https://github.com/dvernooy/ErgWare/tree/master/v0.4

Find “main.hex” in the “eeprom_write” folder & flash it to the chip. **To be crystal clear, even though the purpose of this program “main.hex” is to fill up the EEPROM, you <u>do not</u> want to write main.hex to the EEPROM, you want to write main.hex to flash memory.** Below is what it looks like from the avrdude commands. In Atmel studio 4 you can also program the flash with a *.hex file on the program tab. Either way will work. **The final step to do at this point is to unpower the board, and then power it up again (or just hit the reset button on the Arduino) to ensure the EEPROM is now written.**

[optional] *If you want to compile the source code yourself and flash that instead, go ahead. You can find the source code & makefile on the github. I won’t talk through how to do that here. If you look carefully at the screenshot below, you’ll actually notice I was flashing this from my programming environment using the makefile I supplied and “make program”. The programming environment I use is called “Programmer’s Notepad” (uses WinAVR-20100110… i.e. avr-gcc and avrdude). It is a bare bones simple C (and other) programming environment that allows editing, compiling (“make”), cleaning up files after compiling (“make clean”) and programming (“make program”) directly from it. It uses makefiles which themselves call avrdude … so you still need all that stuff behind the scenes. You can find my makefiles on the github page. Yes, this environment is really old, but I like it, so I still use it.*

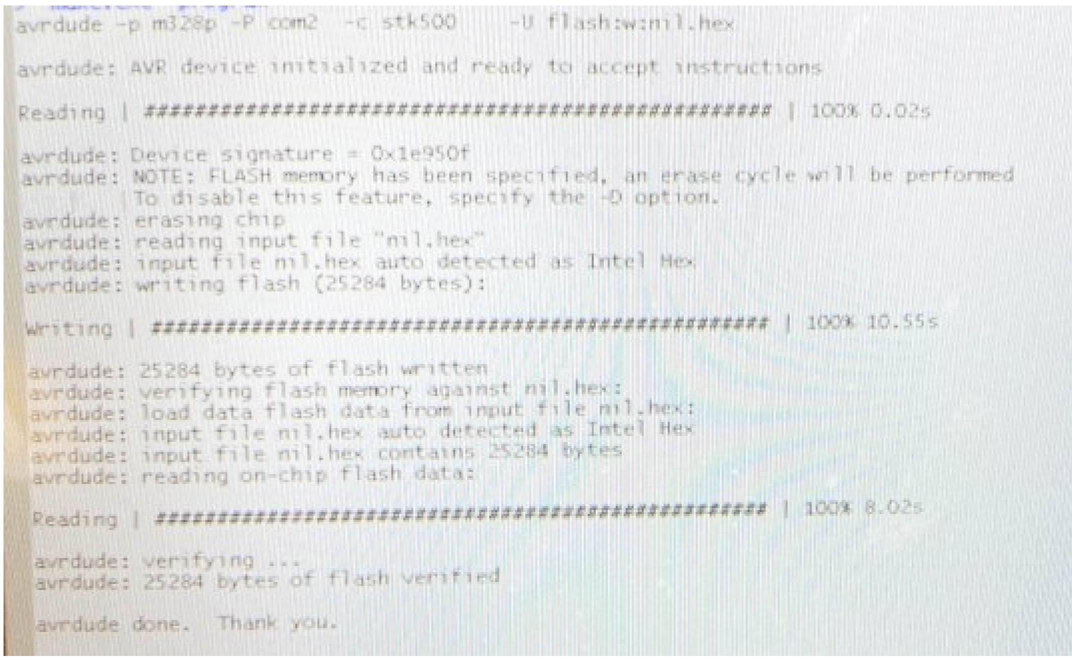
```
avrdude -p atmega328p -P com2     -c stk500    -U flash:w:main.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.02s

avrdude: Device signature = 0x1e950f
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 "main.hex"
avrdude: input file main.hex auto detected as Intel Hex
avrdude: writing flash (1556 bytes):

Writing | ################################################## | 100% 0.66s

avrdude: 1556 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex auto detected as Intel Hex
avrdude: input file main.hex contains 1556 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.50s

avrdude: verifying ...
avrdude: 1556 bytes of flash verified

avrdude done.  Thank you.
```

Step 12. We are now ready to flash the final program. Go to the github

https://github.com/dvernooy/ErgWare/tree/master/v0.4

Find “nil.hex” in the “source” folder & flash it to the chip. Use a similar method as Step 11. You can see the avrdude command below.

[optional] *Again, all the source code is in my repo if you want to make changes & recompile them to suit your own needs.*



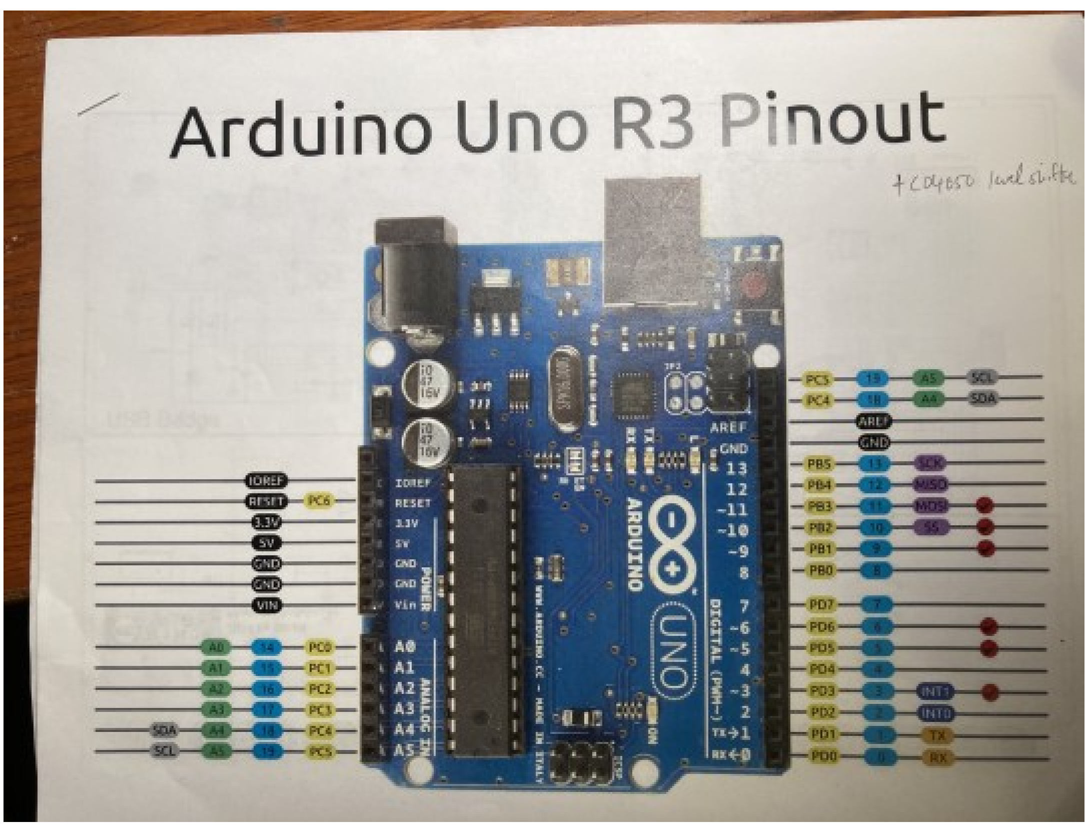
```
avrdude -p m328p -P com2  -c stk500    -U flash:w:nil.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.02s

avrdude: Device signature = 0x1e950f
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 "nil.hex"
avrdude: input file nil.hex auto detected as Intel Hex
avrdude: writing flash (25284 bytes):

Writing | ################################################## | 100% 10.55s

avrdude: 25284 bytes of flash written
avrdude: verifying flash memory against nil.hex:
avrdude: load data flash data from input file nil.hex:
avrdude: input file nil.hex auto detected as Intel Hex
avrdude: input file nil.hex contains 25284 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 8.02s

avrdude: verifying ...
avrdude: 25284 bytes of flash verified

avrdude done.  Thank you.
```

Step 13. Time to wire up the board fully. Turn off the power (unplug the programming cable) while you do this. You can use this pinout diagram I found on the web that was really useful for me. Here are the correct pinouts to use (assuming Nokia 5510 LCD)

LCD_Vcc → Arduino 3.3V (very important, DO NOT use the Arduino 5V)
LCD_GND → Arduino GND
LCD_SCE (or LCD_CE) → Arduino PC1 or Arduino A1
LCD_RST → Arduino PC0 or Arduino A0
LCD_D/C → Arduino PC2 or Arduino A2
LCD_MOSI (or LCD_Data) → Arduino PC3 or Arduino A3
LCD_SCLK (or LCD_CLK) → Arduino PC4 or Arduin A4
LCD_Backlight → 1kohm in series with Arduino 3.3V. Don't wire the backlight directly to 3.3V. Put a 1kohm resistor in series with the 3.3V line.

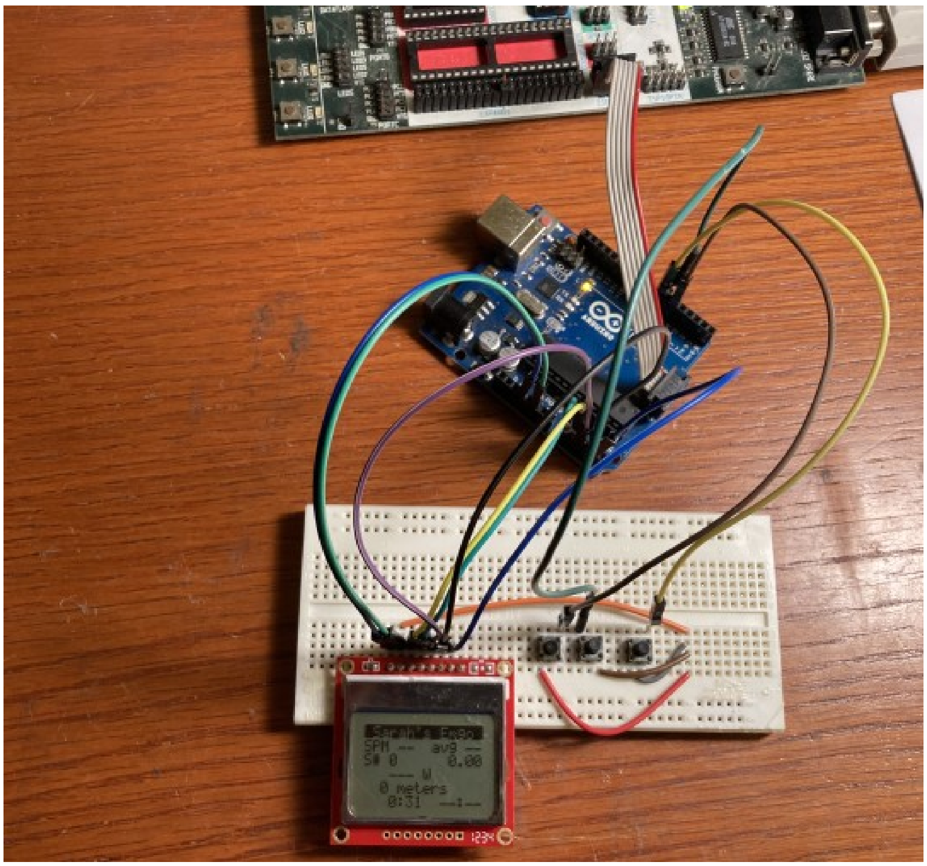
Switch 1 high → Arduino PD7 or Arduino 7
Switch 1 low → Arduino GND
Switch 2 high → Arduino PD6 or Arduino ~6
Switch 2 low → Arduino GND
Switch 3 high →Arduino PD5 or Arduino ~5
Switch 3 low → Arduino GND

*** You will also need to wire the chopper input to Arduino PD2 or Arduino 2 to really fully test this on an erg, but I did not test that in this tutorial. However I did write the code so that will work (see Step 15) ***



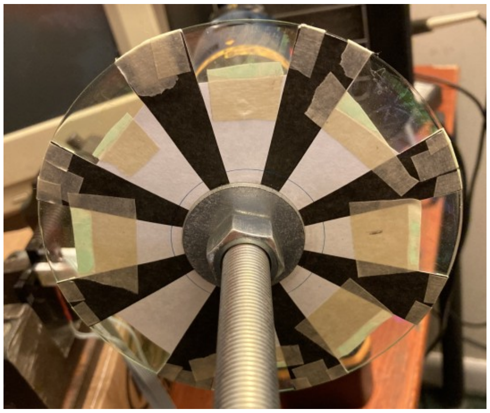
Step 14. Once it is all wired up correctly, you can re-apply power and you should see something on the LCD. You can press any one of the buttons to escape from the splash screen and you should see the screen below. You can press the buttons to move around the menu. **Note, at this point you can use anything to power the board … the programming cable, the usb port or the external power port. They will all work, I did test that. So you no longer need the programmer at this point unless you want to flash new firmware in the future.**

Also – remember this is an 8-bit microcontroller, and not a modern smartphone, so the interface is a “bit” simpler than an iPhone.

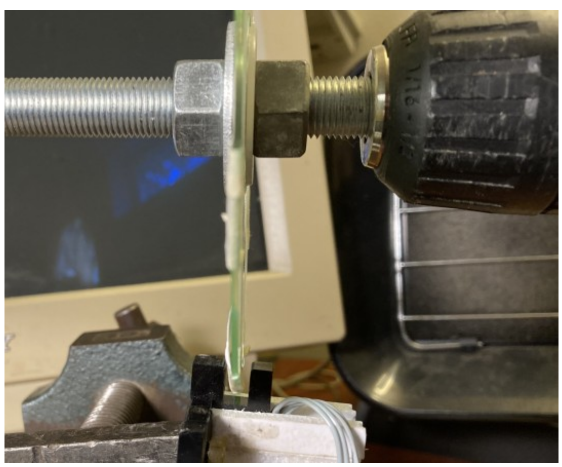


Take a nice break at this point! If you got this far, you deserve it.

Step 15. Now it is time to get the chopper working. The first step is to build a chopper wheel and something to "simulate" your erg. I took an old CD and made an 8-spoke pattern. It doesn't matter too much how much you block and how much you don't in the pattern, but aim for about 50% blocked and 50% clear. It does matter though that you use the 8-spoke pattern, otherwise you will need to adjust and re-compile the code. I just printed a pattern on my printer, then cut out the clear parts with scissors then just taped it on the CD. I'm sure you can do it more neatly than I did.



Then I found an old threaded rod that fit in my drill (to simulate the erg) and bolted the CD onto the rod so it would spin when I ran the drill. You can figure something out for yourself. You can also see the CD positioned in the slot in the photo-interrupter. I am holding the photo-interrupter in a small hobby vise to make things simple. I also have a little foam holder for the photo-interrupter. It came from my junk parts bin so I have no idea why the foam is there. Your part will probably look brand new. Mine looked like it had been through some abuse.



Here is another picture of the drill. And you can clearly see the photo-interrupter held in the vise.

I also made a really long ribbon cable for it to go over to the circuit board to “simulate” the long length you will need for your erg. The photointerrupter needs to get mounted near the flywheel of the erg and you will probably want the electronics up near your face so you can see it while rowing.

The next step is to look at the signal levels with the photo-interrupter clear and with it blocked. Here is where life will either get really easy or really hard for you. In my case, it got really hard, probably because the photo-interrupter part was probably not really working to begin with. You can avoid much of the craziness I had to go through below by just buying a brand new part.

In any case, I’ll tell you what I did to get my part working. **Don’t get intimidated by all of this**, I think it will be a lot easier with a new part & you can probably avoid 90% of the stuff below. I’ll update this document with information on using a new part if I ever get one.

In any case, here is what I did to make it work. The part I had was marked OPB3738 and I could not find anything about it on the internet. As I said before though, it was the only part I had lying around so I just kept going.

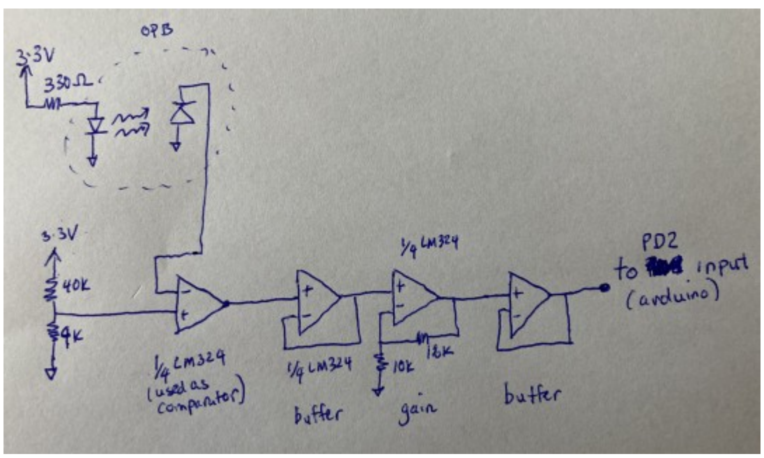
The first thing was to find the anode and cathode of the LED emitter/transmitter side of the photo-interrupter. You can probably find this information on the datasheet from the web. I hooked up the cathode of the LED to ground and the anode of the LED to 3.3V with a 330 ohm resistor in series with the anode to limit the current. The next thing was to see if any signal was coming out of the photodiode/phototransistor/receiver side. There were 3 wires coming out and I was hoping to get a nice 3.3V when the coupler was clear and 0V or so when blocked. On a new part with a readily available data sheet, this would be easy to figure out. On this part, every test I did was weird. It did not behave like either a direct logic output nor an open collector part, no matter what I tried.

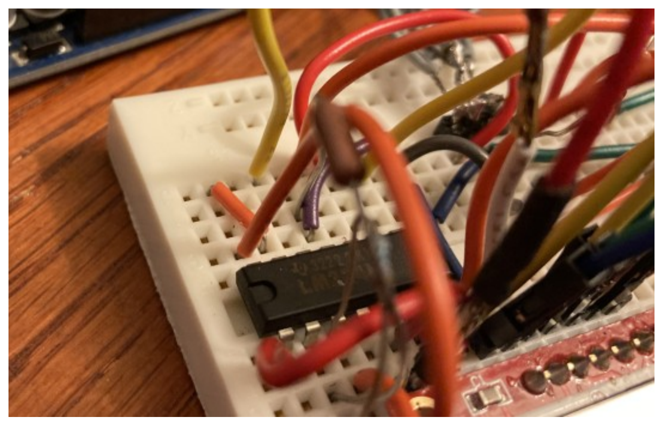
Eventually, I had to settle for the fact that the voltage across 2 of the wires measured 206mV when unblocked and -33mV when blocked. It is nice to have a voltmeter to do these measurements and see what is happening. But at least that is something I could work with.

For a brand new photo-interrupter, these levels should be more like 3.3V when unblocked and 0V when blocked. In fact, if you get readings like 3.3V and 0V, you can probably just stick the output into PD2 /ARDUINO 2 of the arduino and be done with it.

In my case, I needed to amplify these voltages to get levels the Arduino board can deal with. So I built the circuit below that uses an LM324 first as a comparator and then a few buffer and gain stages. I hope you do not have to do any of this nonsense if your part works to begin with. I powered the LM324 off of 5V to maximize the signal levels.



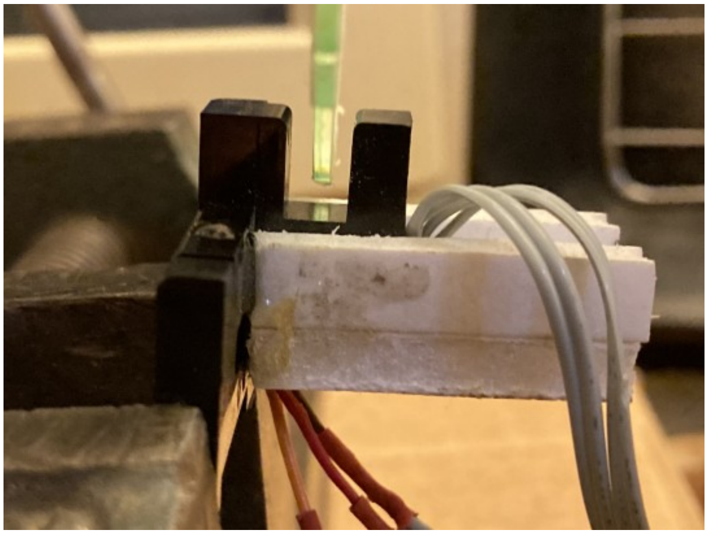


After building this circuit, I managed to get signal levels of 509 mV (0.509V) when unblocked and 2.67V when blocked. It “inverted” due to the comparator, but the software won’t care. Also, these levels will work well enough for the Arduino to interpret the 2.67V level as a “one” and the 0.509 V level as a “zero”.

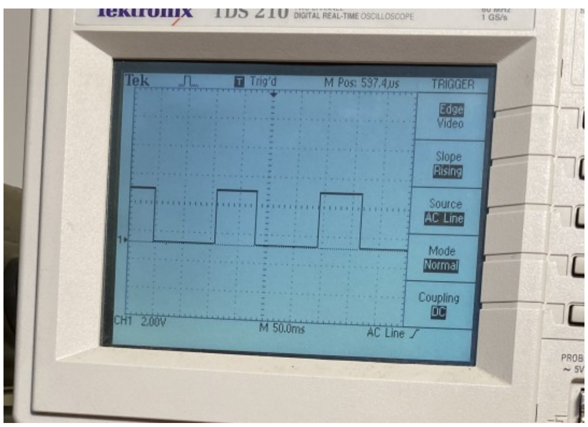
 

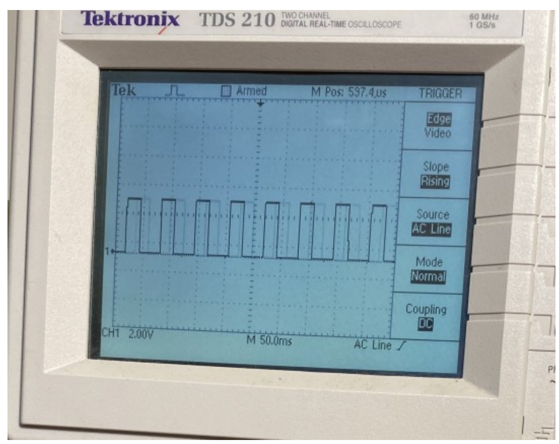
Again, I don’t think you will have to go through any of this hassle to amplify the signals if you have a brand new OPB-XXX part. It might be good to just use one buffer stage of the

LM324 though. Ok, now for another quick test with the actually chopper. First, I made sure the CD sat squarely in the gap.



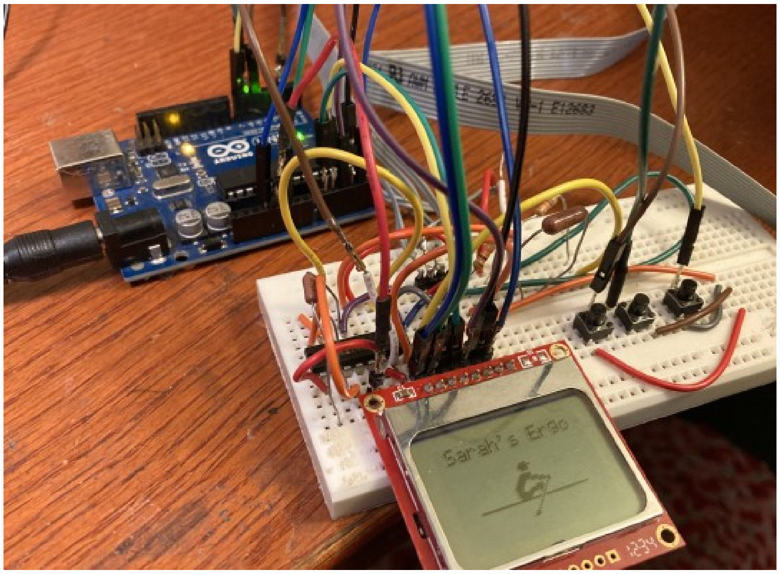
Then I turned on the drill and looked at the signals on my oscilloscope when I ran the drill slow and fast. You might not have an oscilloscope – that's fine. In my case, it was just another check to make sure the chopper was making a nice square wave output. It was.





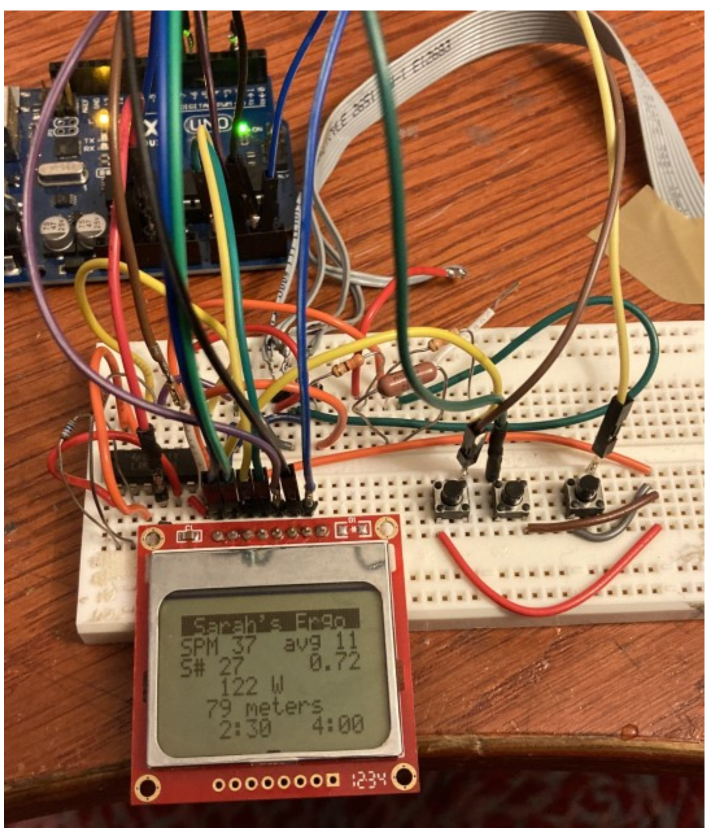
Ok, at long last time to check and see if it works when wired into the Arduino board. Below is what my setup now looks like with everything wired up. Remember to connect the chopper output (or the amplifier output if you needed one) to PD2 or Arduino 2.

Then start up the software.



Navigate to the Row! Screen. Finally, I just held my hand on the drill trigger and ran it in cycles slow-fast-slow-fast-slow-fast … etc… to simulate rowing strokes and I watched the S# on the display continuous count up every cycle (below it shows S# 27 because I did 27 slow-fast cycles). The S# tells you the stroke count on your erg.

In other words, **ITS WORKING !!!!!!**



One final word of caution … you really need to make sure not to use a “cheap” power supply when powering the arduino board using the external power jack. If so, you might notice the stroke count and distance count continously move up even if you aren’t doing anything. That is “noise” from a really cheap, noisy power supply causing problems. I saw this in my tests and could fix it by using a better power supply. I had no problems when I used a higher quality power supply and also no problems just powering with the USB jack.

The final step is to package this all up nicely and attach it to your erg so you can see it while you are rowing. Remember also you might want to hook up the backlight as I suggested in Step 13 to make the display easier to see.

I hope this write-up helps, let me know if you run into any problems and I can help.

-Dave