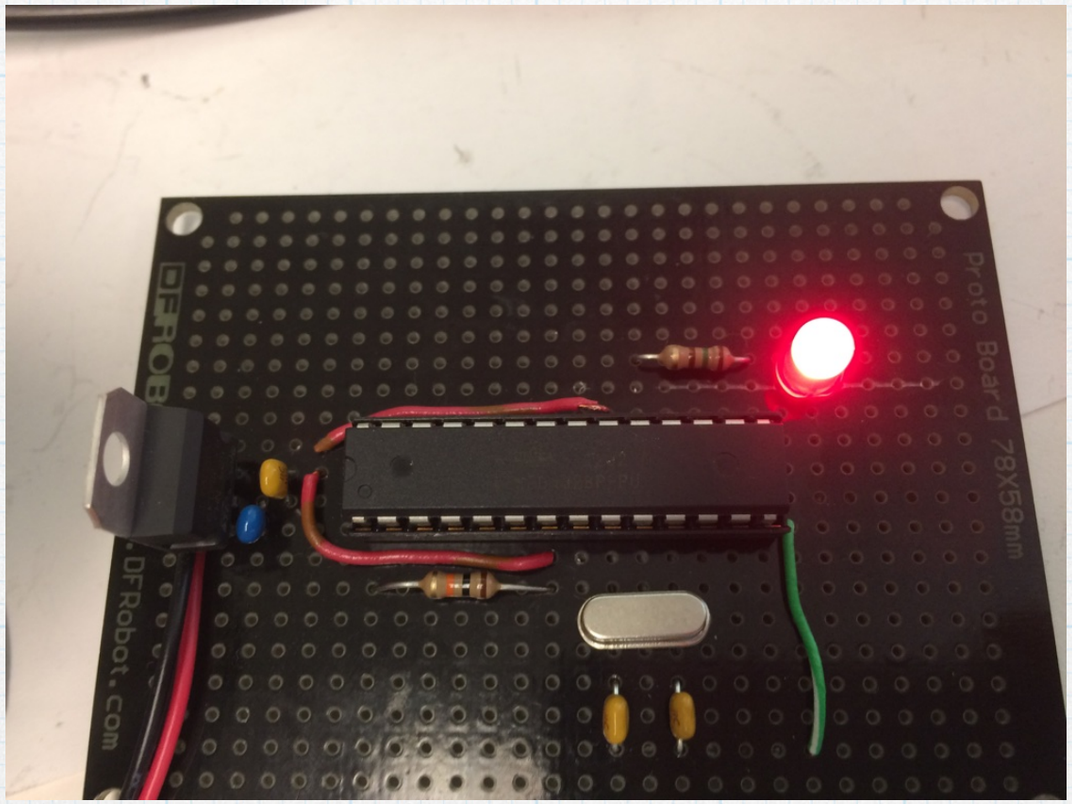
Bareduino

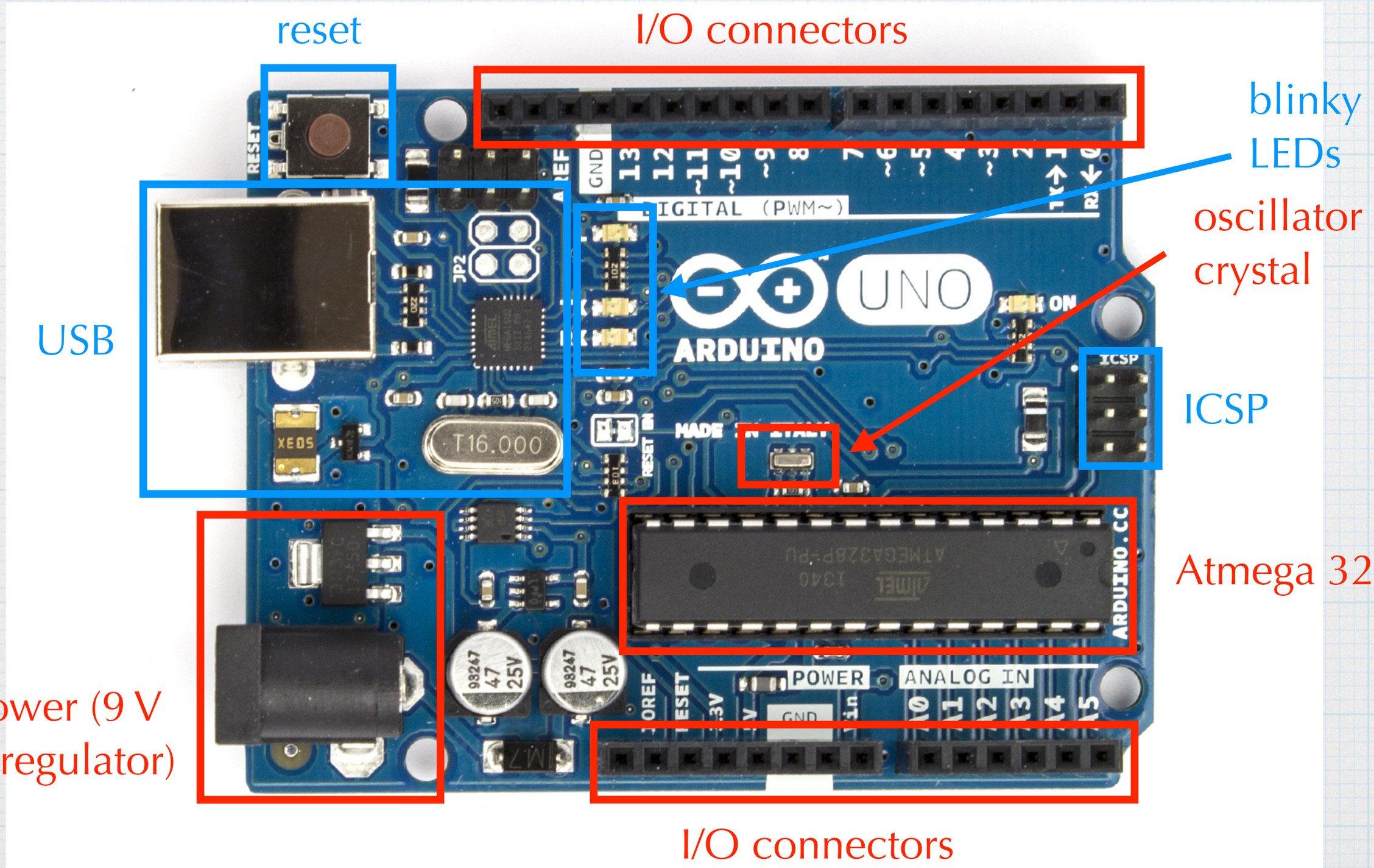
As a first step towards incorporating an Atmega328 controller into a project, let's prototype a bare-bones version of an Arduino.



Saves a little money. You can configure the hardware exactly the way that you want. Plus, it's fun!

Arduino Uno

Recall the basic layout of the hardware. The items with red outlines are essential. (The reset might be considered essential and we may choose to add the ICSP interface.)



Bareduino

Boil the hardware down to the bare minimum.

- Atmega 328
- voltage regulator (to run off a 9-V battery)
- quartz crystal and capacitors for the clock
- pull-up resistor on the reset pin.

Everything else is gone, although we might consider adding some optional items

- Momentary switch for reset.
- ICSP header for programming
- LED power indicator
- Header pins, terminal blocks, or other types of wiring connencors.

Bill of Materials

Part	Digi-Key	Price	Comment
proto-board	1738-1000-ND	1.52	cheap
Atmega 328	ATMEGA328P-PU-PN	1.91	not programmed
28-pin socket	ED3050-5-ND	0.315	good enough
7805 regulator	497-1443-5-ND	0.426	5-V linear
0.33 µF capacitor	445-5263-ND	0.223	for regulator
0.1 µF capacitor	BC2665CT-ND	0.187	for regulator
crystal	CTX1085-ND	0.30	16 MHz
22 pF capacitors	BC1005CT-ND	0.348	need 2 for crystal
10 kΩ resistor	CF14JT10K0CT-ND	0.029	for re-set pin
red LED	160-1853-ND	0.24	for blinking!
150 Ω resistor	CF14JT150RCT-ND	0.029	for LED
battery strap	36-84-4-ND	0.49	for 9-V battery
		6.02	

Note the you will also need a 9-V battery to serve as the DC voltage source. These are the prices if buying in large quantities. Buying one or two at time will be about 10% more expensive.

Programming

Building microcontroller hardware doesn't have much purpose without having software loaded on the microcontroller to provide the functionality.

A primary consideration in deciding the best path to loading software onto the controller is the question of how often the software will be changed.

If the project is "one and done", meaning that the software will never be changed once it is loaded onto the controller, then it is probably easiest to program the chip *before* installing it onto the board. The programming can be with the chip installed in the Arduino board. Once the software is installed, the programmed chip can be transferred to the socket on the bareduino.

If the software is under continued development, then frequently swapping the chip back and forth between the Arduino and bareduino is probably not practical. In that case, it would be better to include an ICSP header on the bareduino and use a programmer interface to load the software. This approach is described more fully in a companion set of slides describing the programming process.

For our initial operation of the bareduino, we will start by using the ATmega328 chip from the Arduino bard and make use the “program and swap” method.

Caution: Swapping chips from board to board greatly increases the likelihood of something bad happening. Removing the chip from the socket is a bit tricky, and requires some gentle but firm prying with a screwdriver. If you are not careful, you risk bending pins or poking holes in your finger tips. Also, there is risk of wrecking the chip with electrostatic discharge or bad wiring. Be careful!

So the initial step is write a simple Arduino program on the that blinks an LED (or has some simple external function). We can use the “blink” program that flashes on LED on pin 13. Or write our own the flashes on LED somewhere.

On the next slide is a nearly identical program for flashing an LED on any pin.

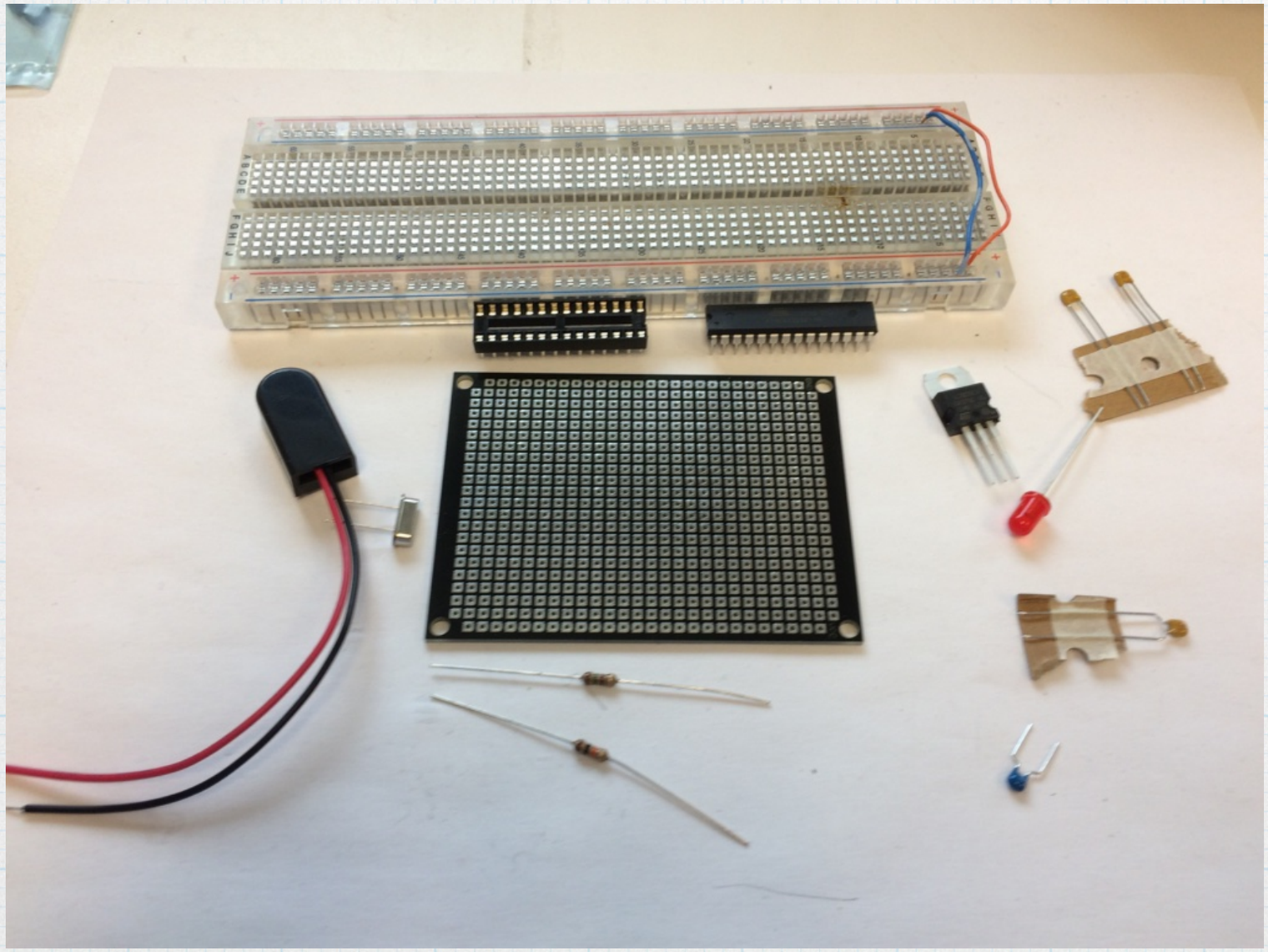
```
int high_time = 1000;
int low_time = 1000;
int led_pin = 6;

void setup() {
  pinMode( led_pin, OUTPUT );     // initialize the output pin
}

void loop() {
  digitalWrite( led_pin, HIGH );   // turn on the LED
  delay( high_time);               // wait
  digitalWrite( led_pin, LOW );    // turn off the LED
  delay(low_time;                  // wait
}
```

Load the desired program into Arduino. And then test it to make certain that the blinking thing blinks!!

We will move the ATmega chip from the board later in the process.

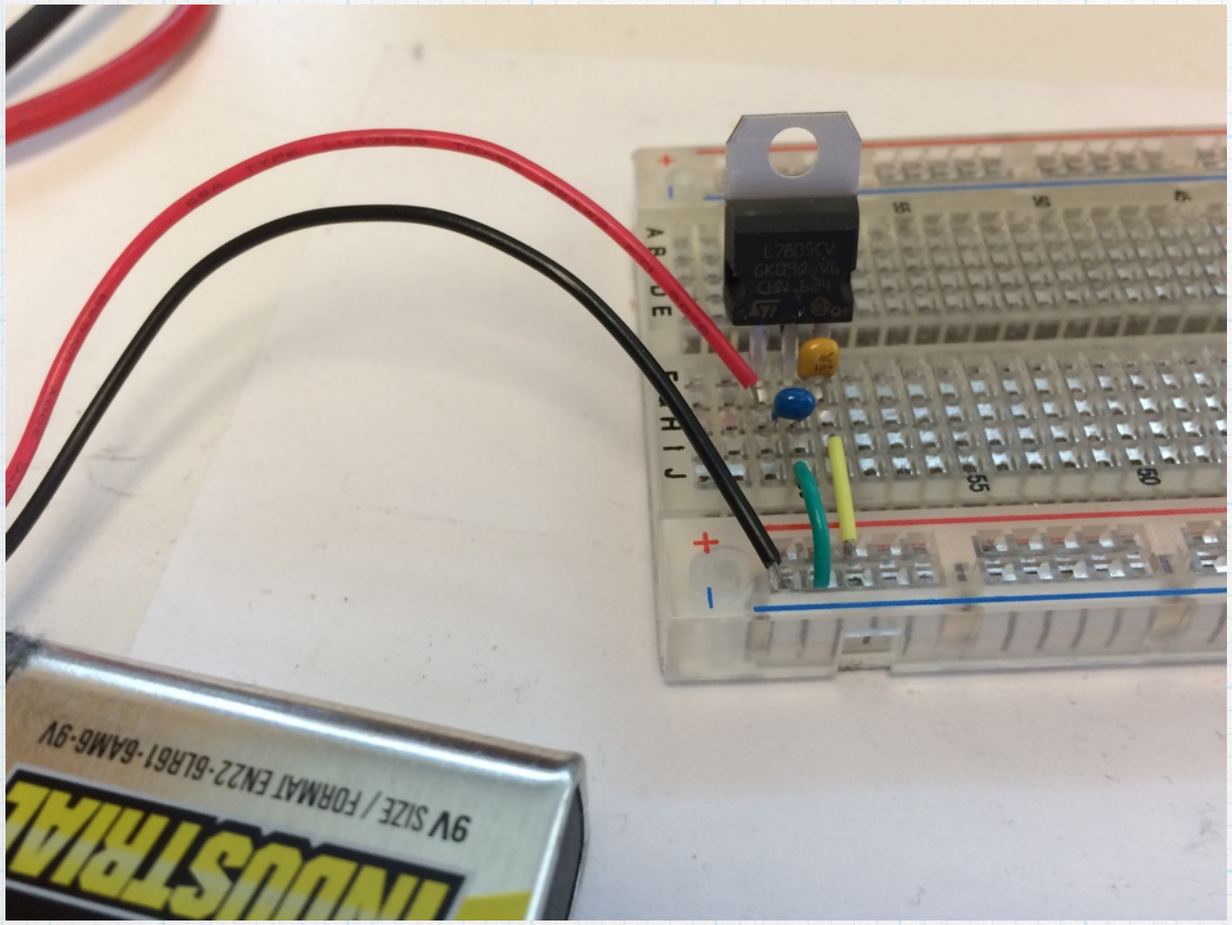
Bread-boarding

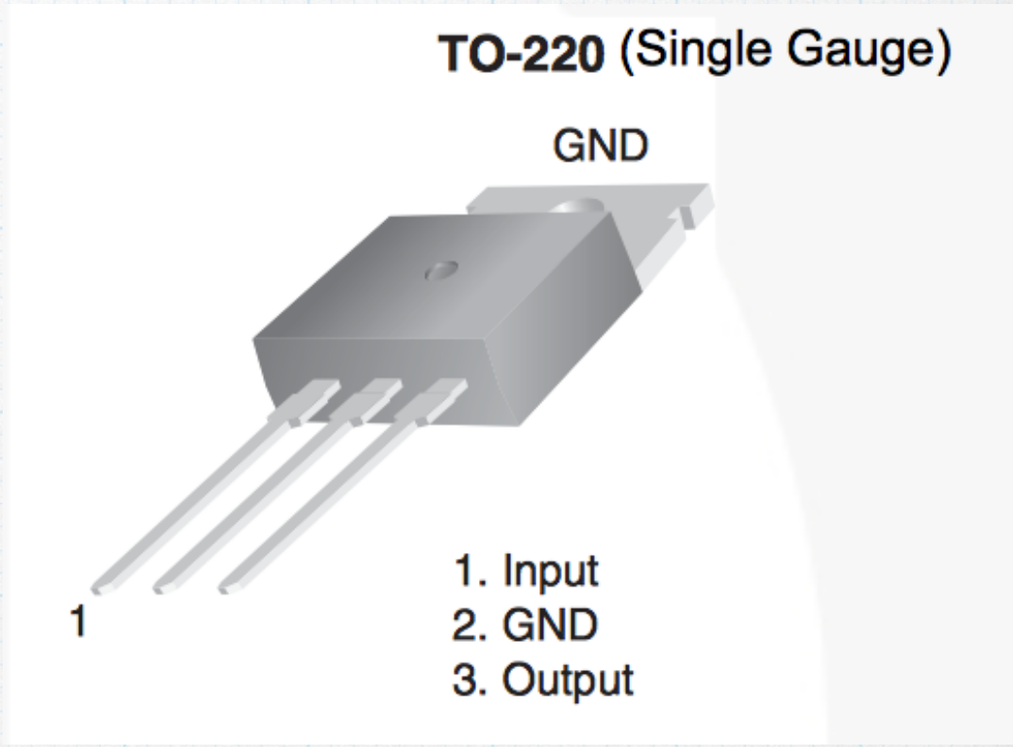
It's always a good idea to bread-board your hardware before soldering together a prototype. It's much easier to de-bug a bread board than a soldered circuit.



Gather your parts.

Set up the power. The battery provides approximately 9 V and the 7805 regulator in turn gives a steady 5 V. The input is 9 V and the output is 5 V.

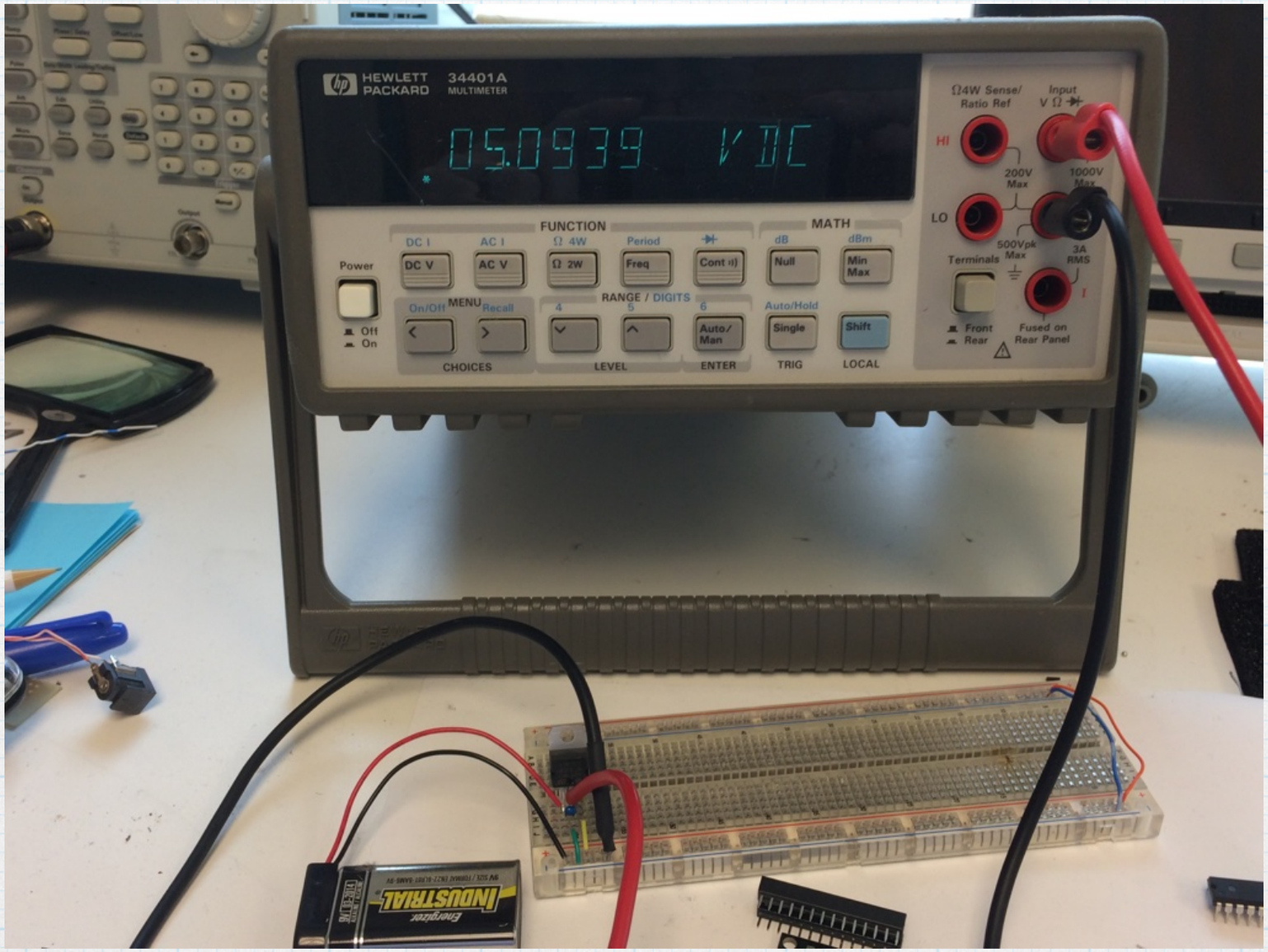




The capacitors provide stability — 0.33 µF between the input and ground and 0.1µF between the output and ground.

The red lead of the battery connector is the positive input voltage and the black lead goes to ground

It's a good idea to test the regulator output before going too far.



Add the controller. We need a pin mapping so we know how "Arduino" translates into Atmega328.

Atmega168 Pin Mapping

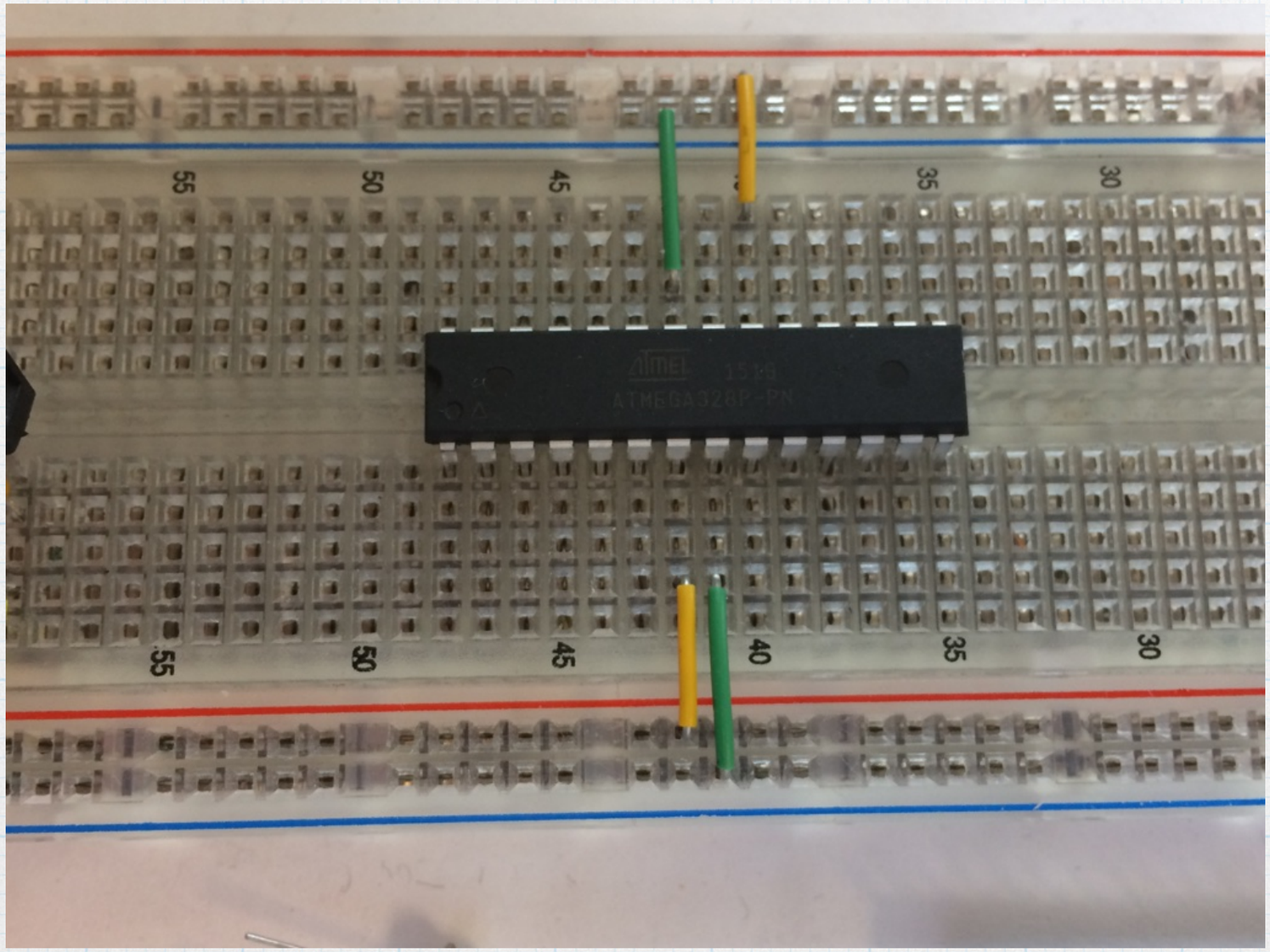
Arduino function					Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11(PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

Note: Atmega328 has same pin configuration as Atmega 168.

Basic power connections

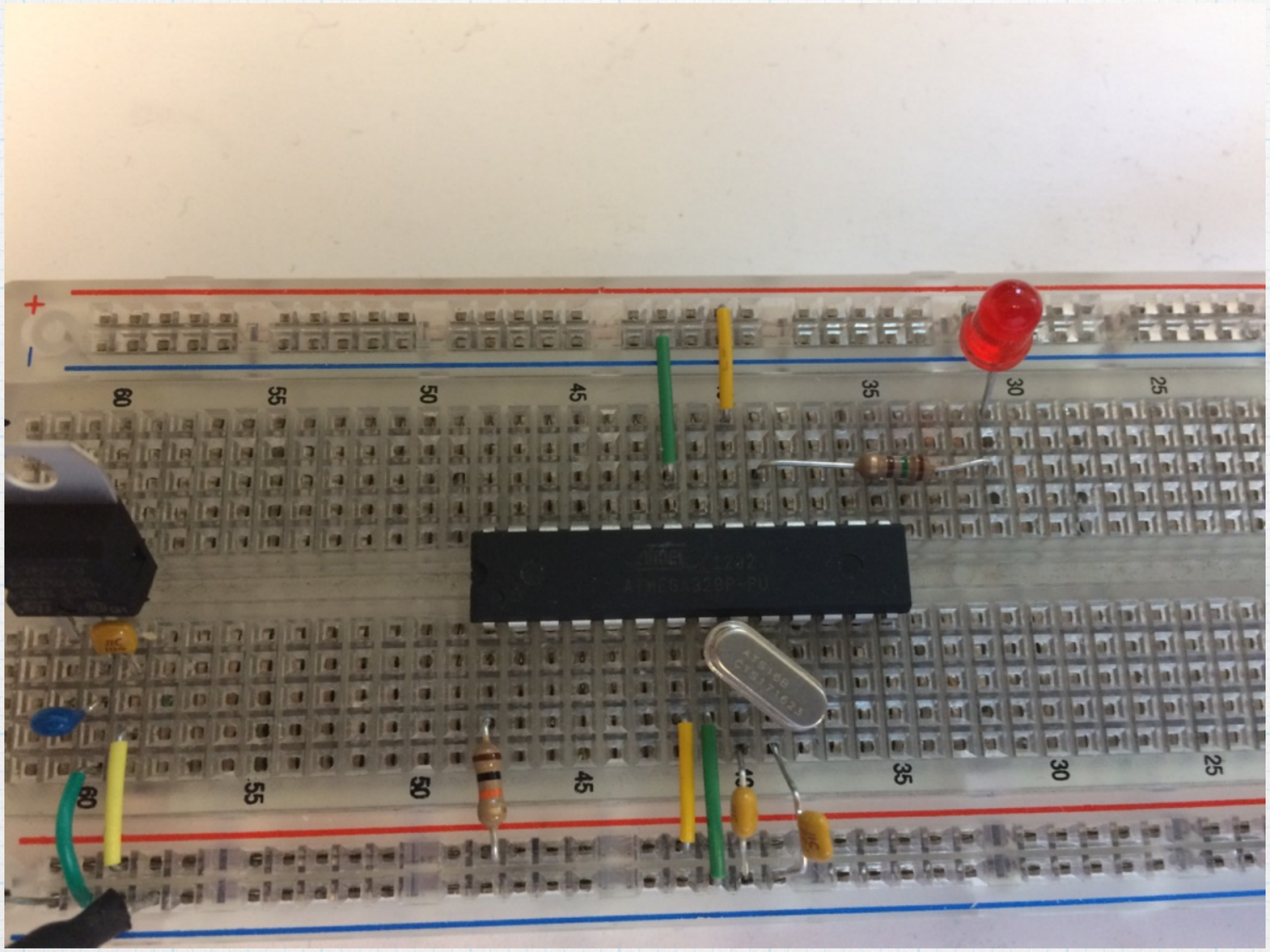
Move the Atmega328. Make sure the power is off and *carefully* pry the chip out of Arduino socket. Install it into the proto board and hook up the power pins.



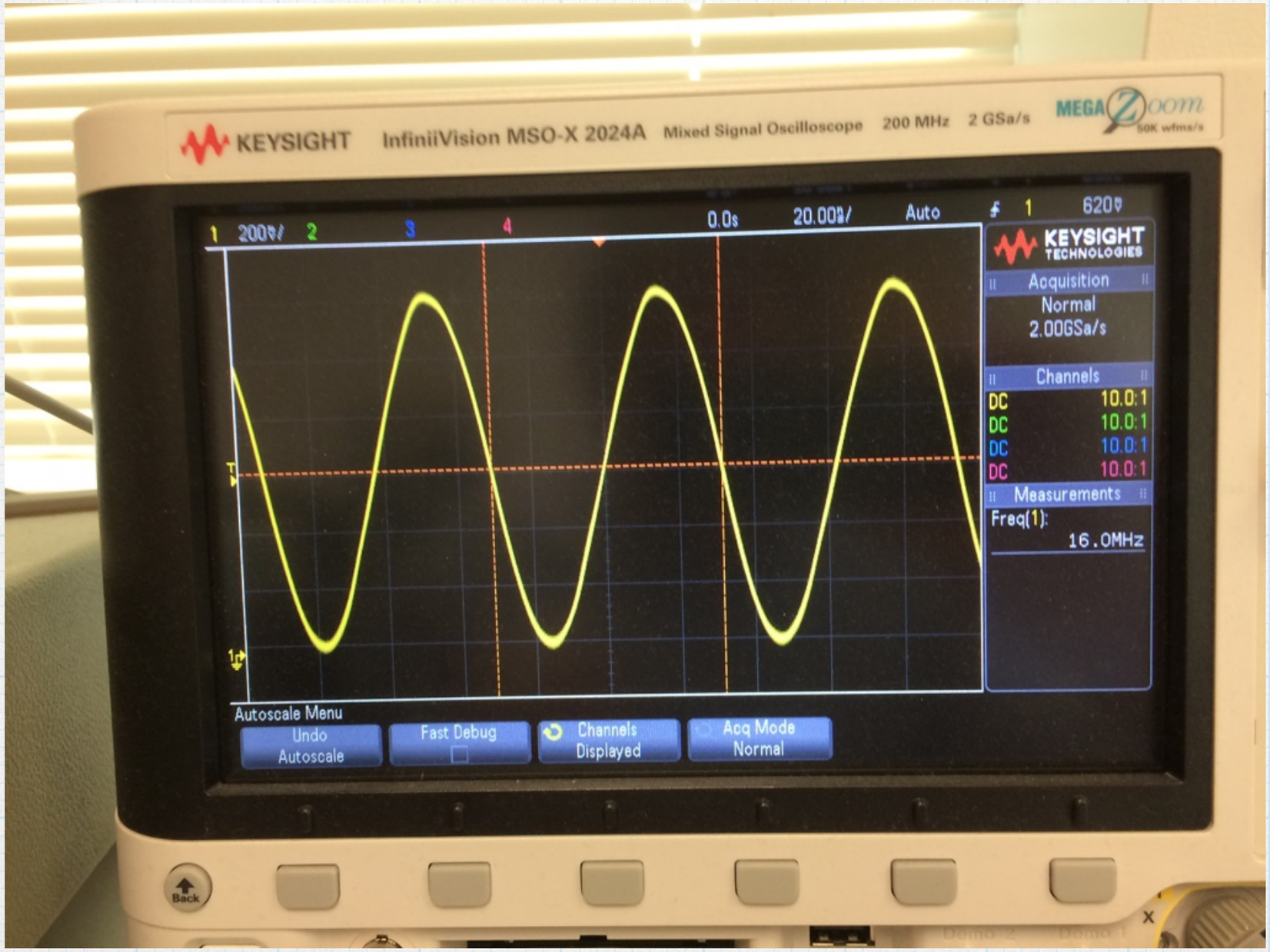
Pins 8 & 22 → ground (green wires)
Pins 7 & 20 → 5 V (yellow wires)
Pin 1 → 5 V through a 10 kΩ resistor.

Oscillator crystal and the rest

Add the oscillator crystal and capacitors.

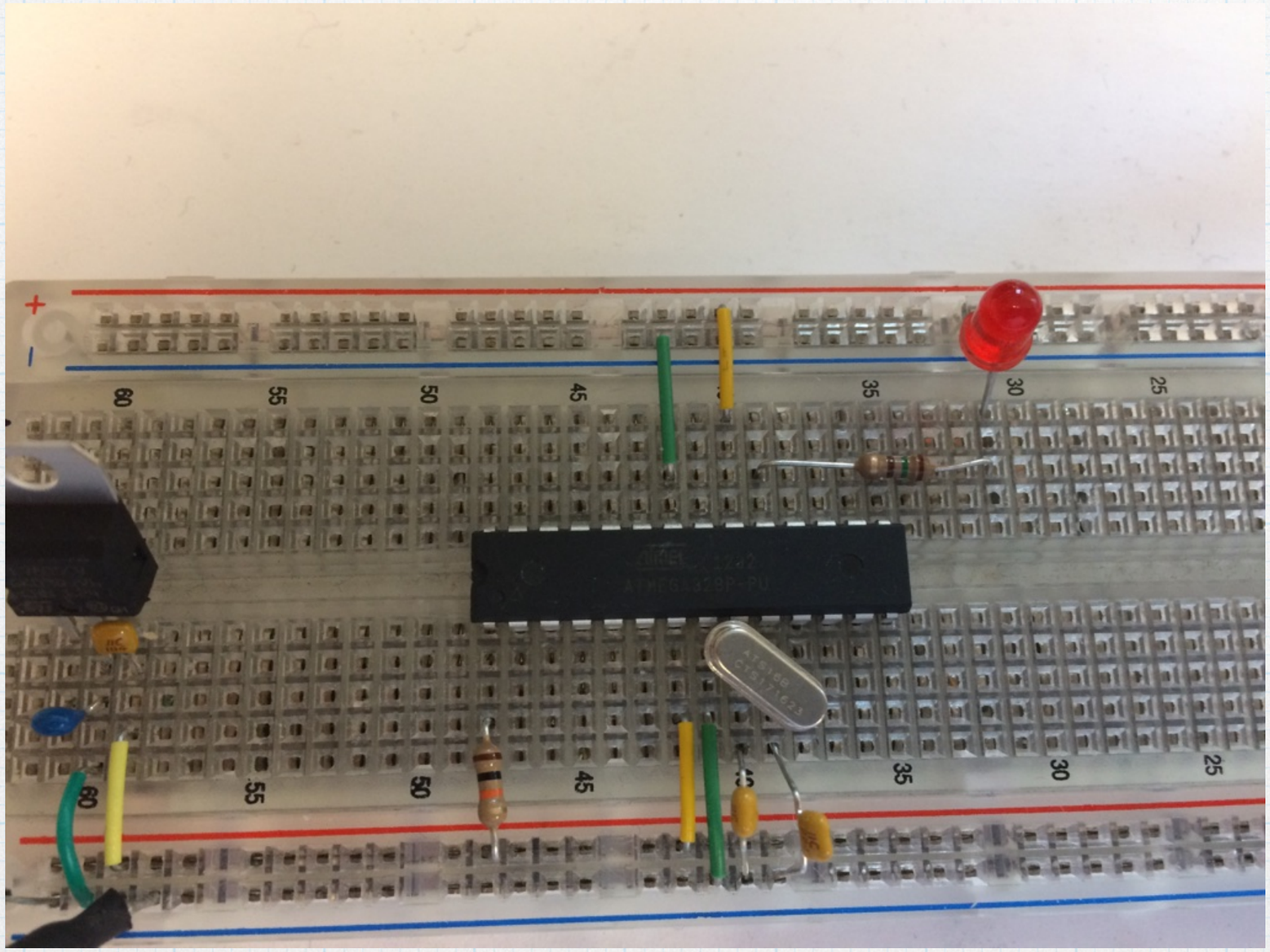


The crystal goes between pins 9 & 10, and there are two 22 µF capacitors – one from each pin to ground.



Apply power and check the oscillator waveform. Note that the frequency is 16 MHz, as expected for the crystal used.

And the LED and the pull-up resistor.

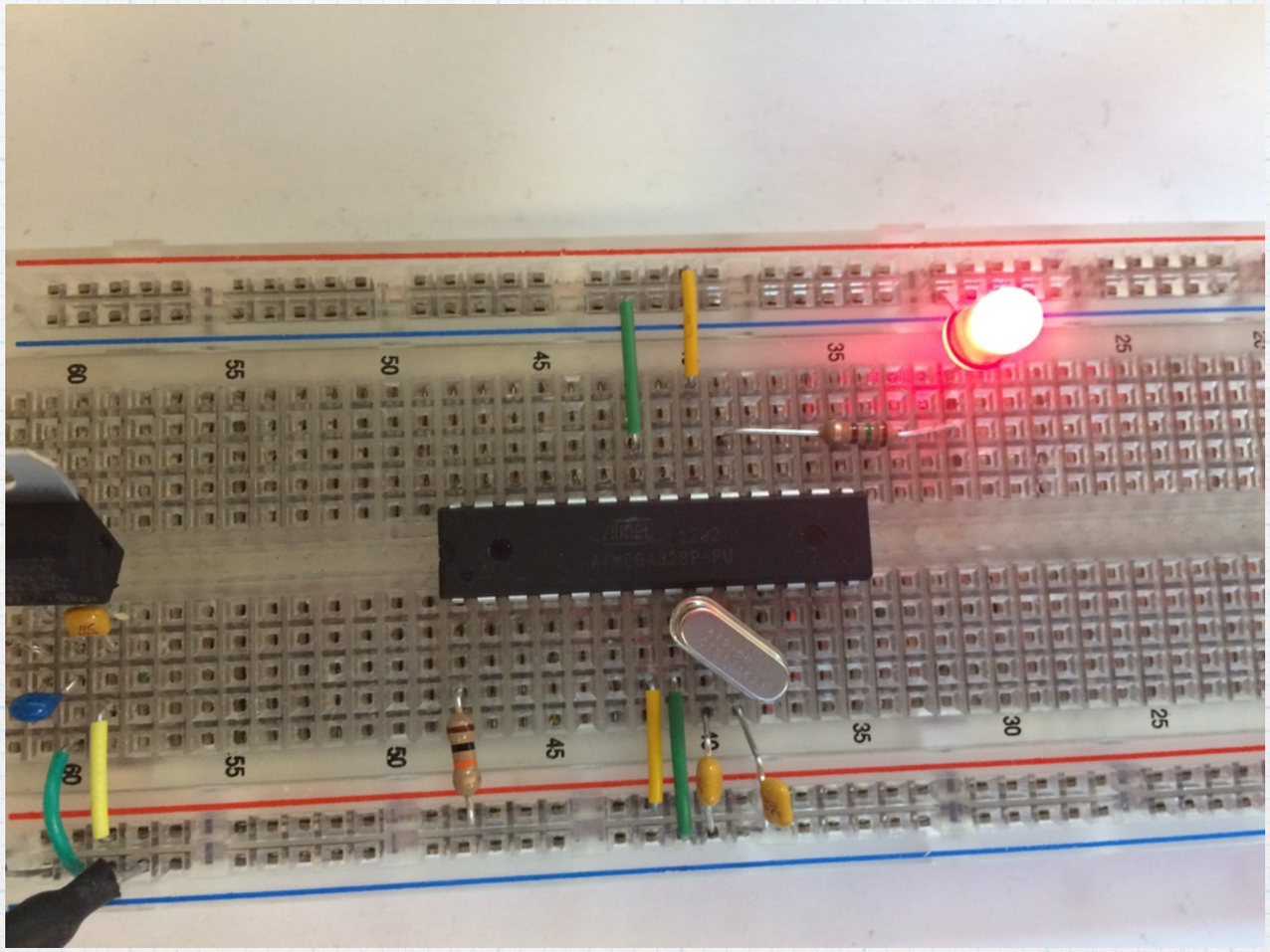


LED (pick your favorite color) connected to chip pin 19 (Arduino output 13) through a limiting resistor.

Fire it up!

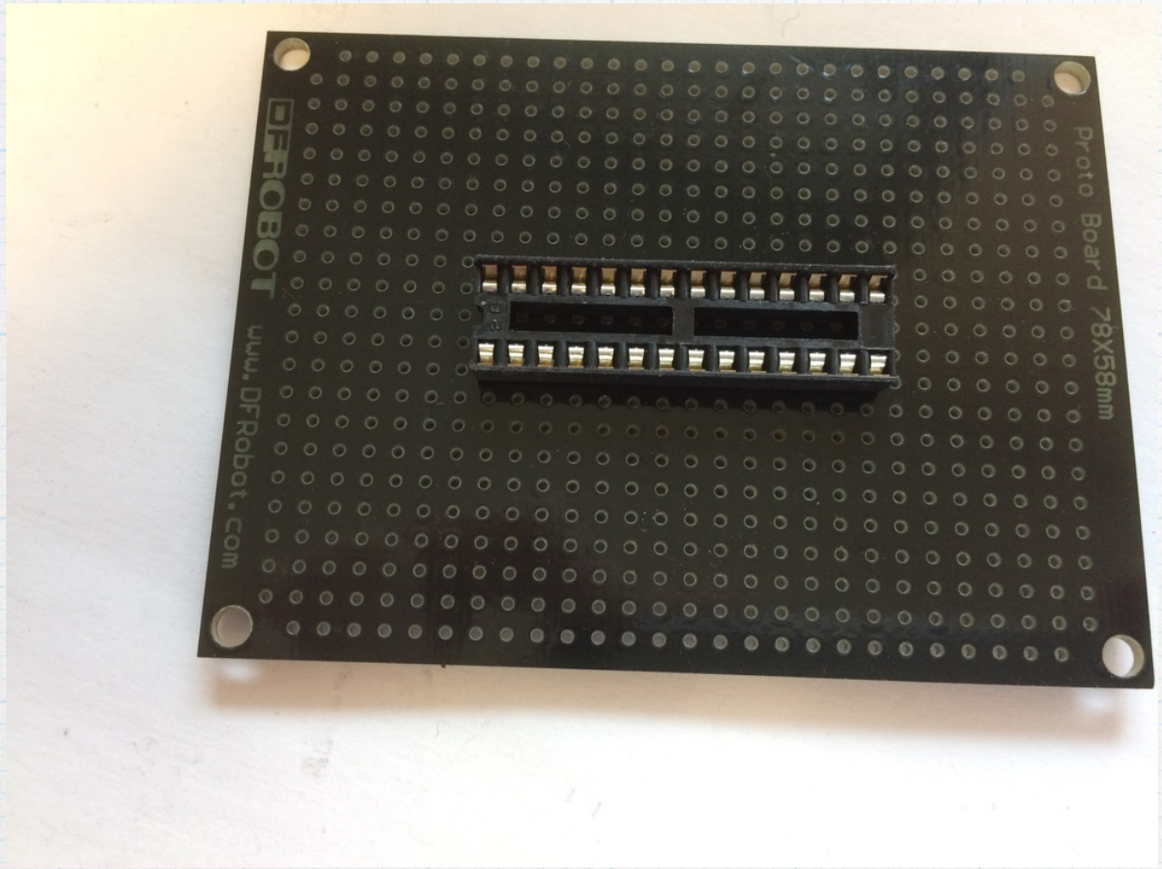
Apply the power. (Attach the battery lead.) If everything was connected properly, the sketch should start up and the light should blink.

“Bareduino!!”

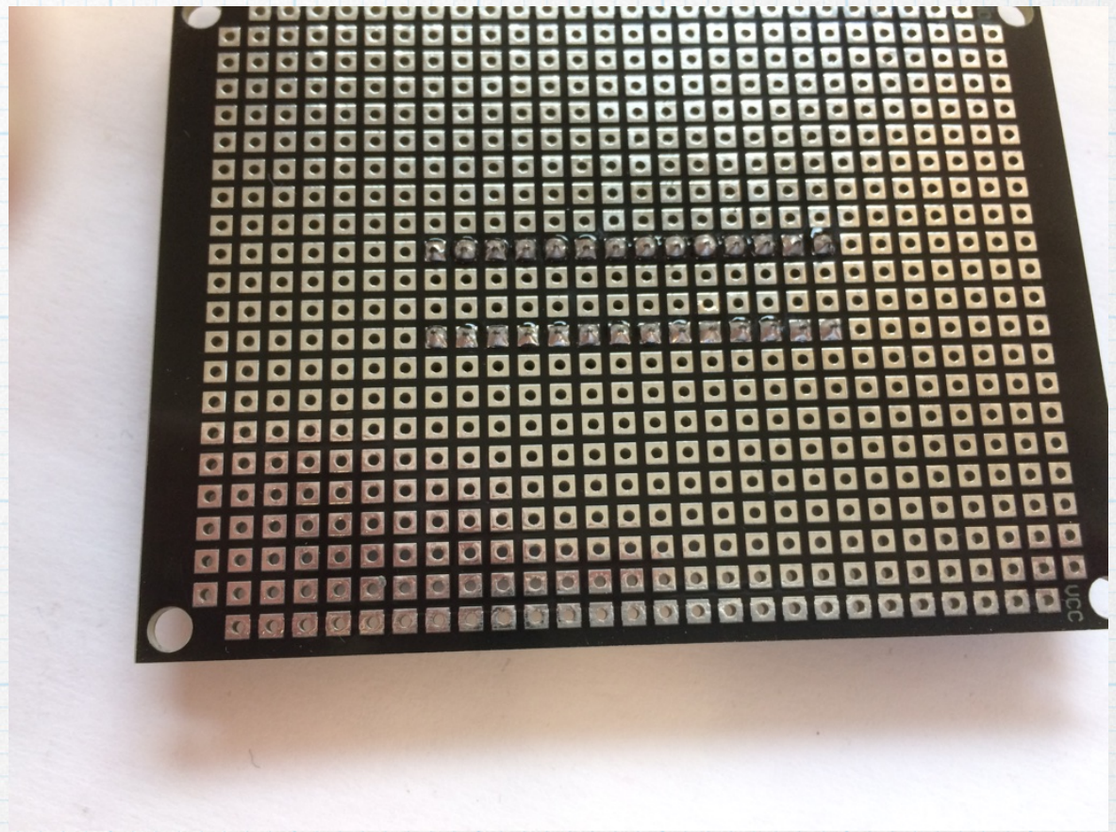


It blinks!

Perf board version - step by step



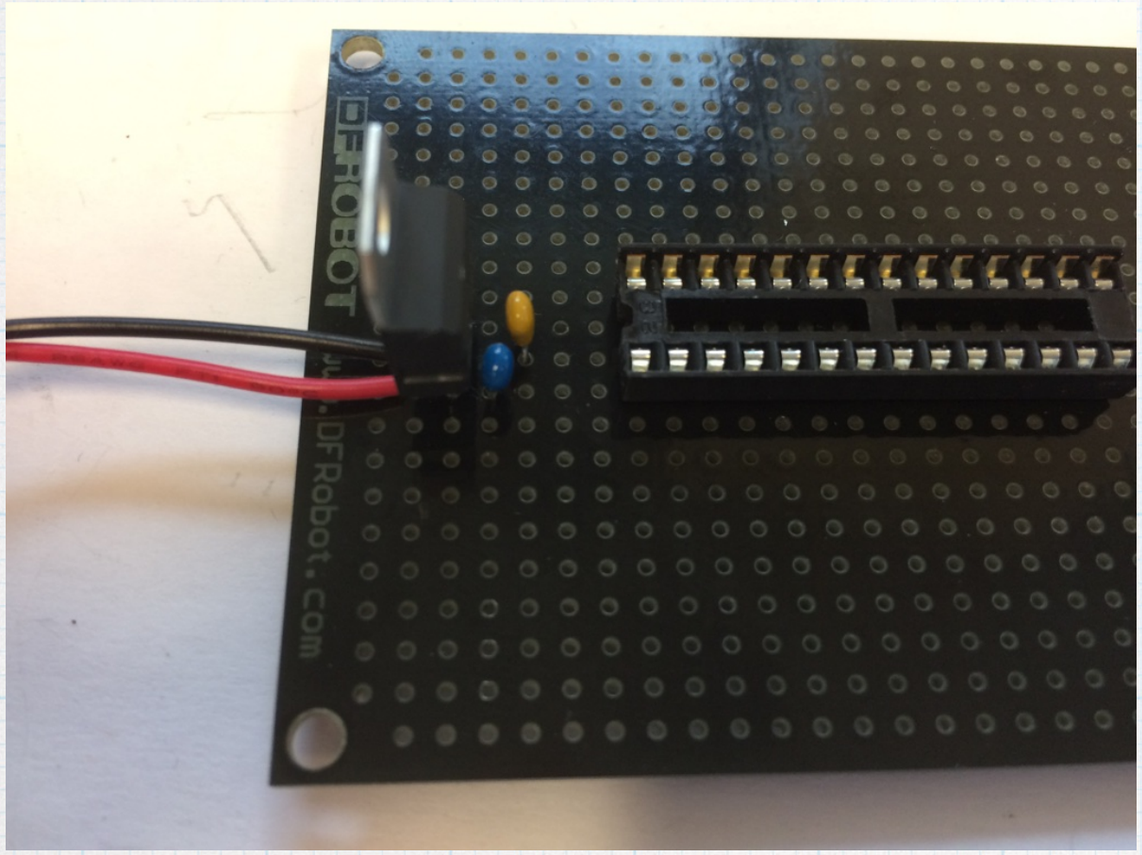
Front (component side)

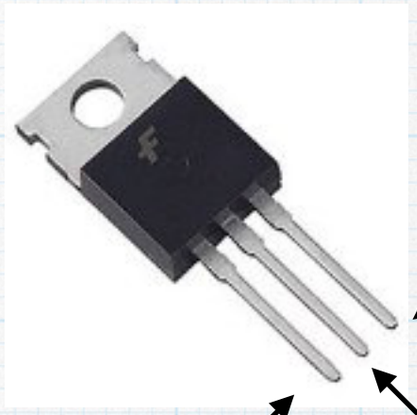


Back (solder side)

Solder the 28-pin socket to the perfboard. It can be placed anywhere on the board, depending on the other components that you might be connecting. For this simple application, placing it in the center is just fine.

Front





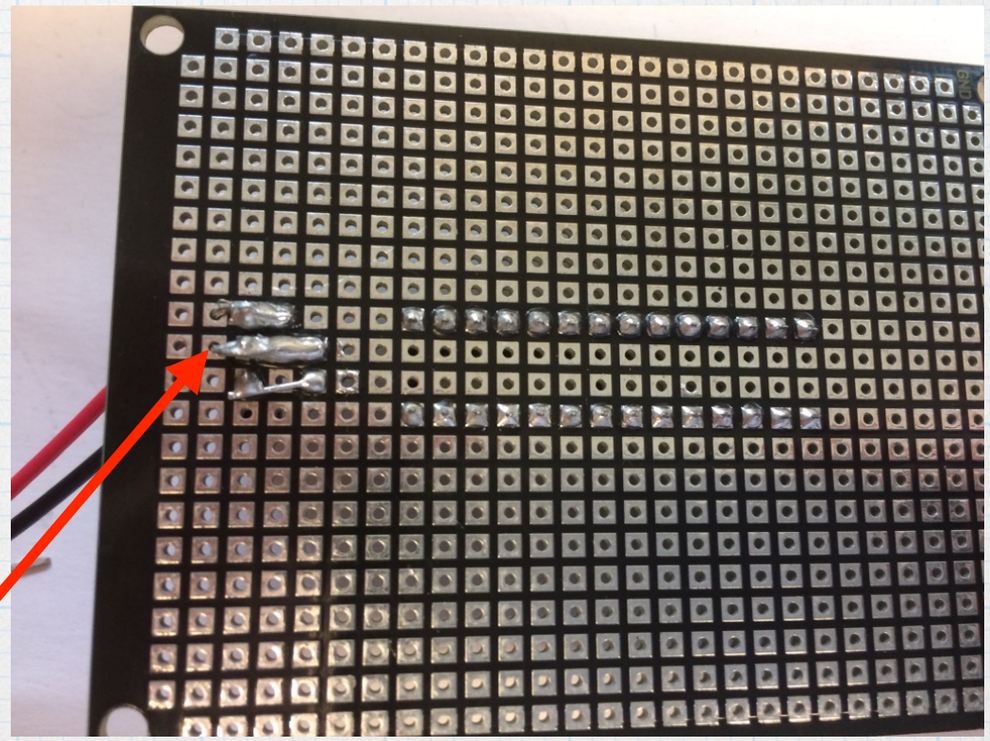
Out (5 V)

In (9 V)

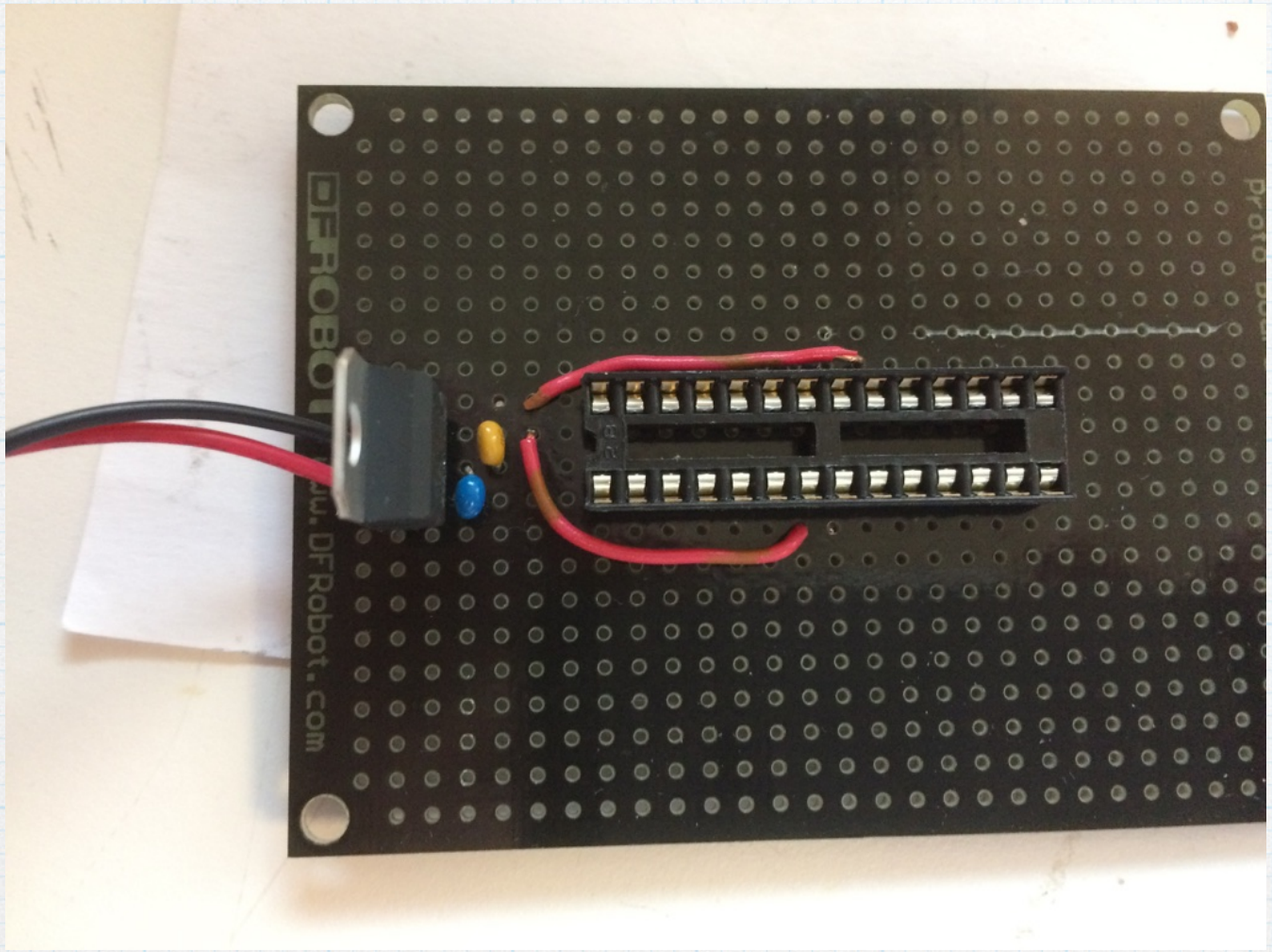
ground

Solder in the 5-V regulator (7805). It can go anywhere. Again, if you have many other components that will be added, Then add the capacitors and battery strap.

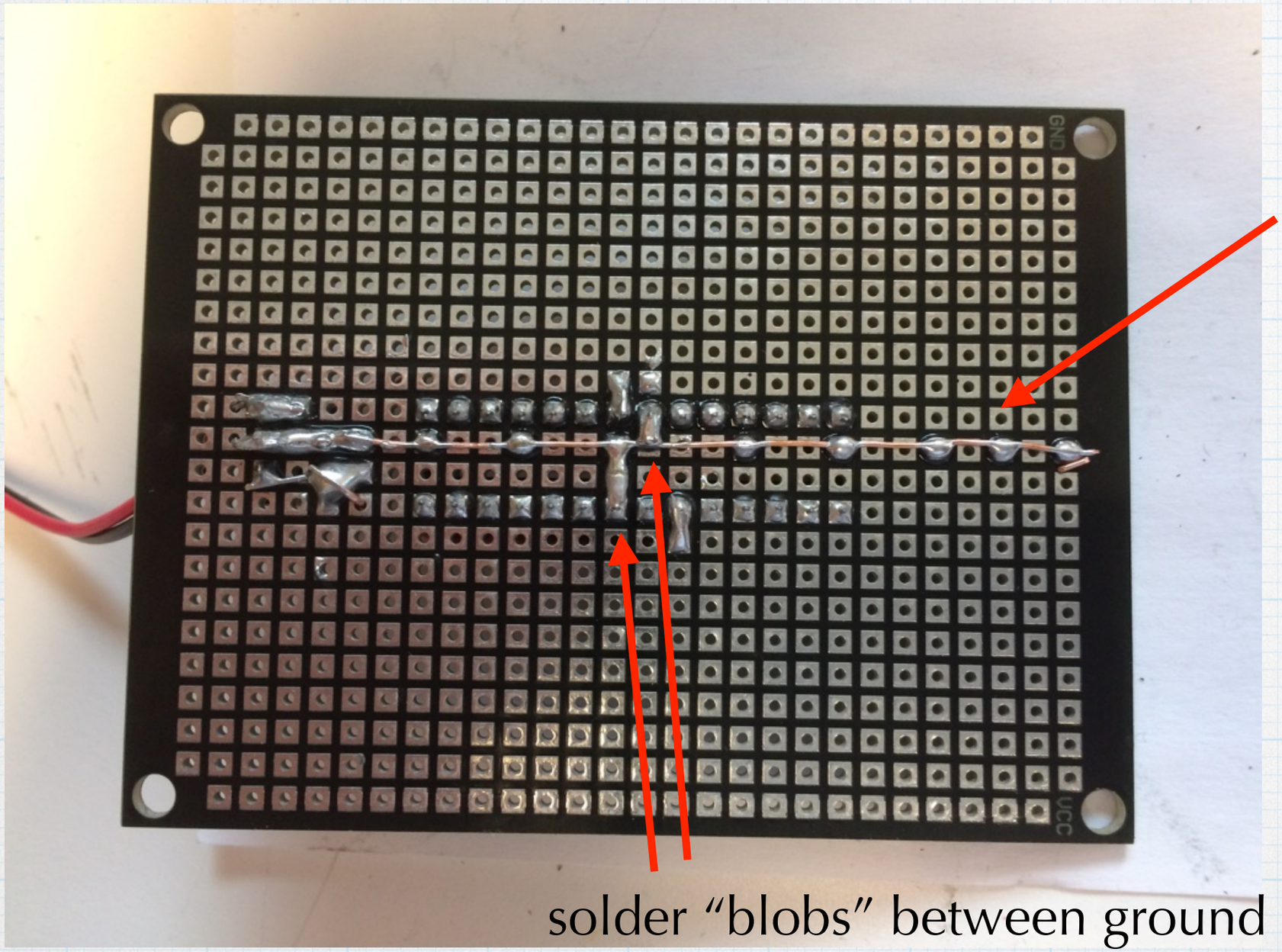
Solder connections for the regulator and capacitors.



Back



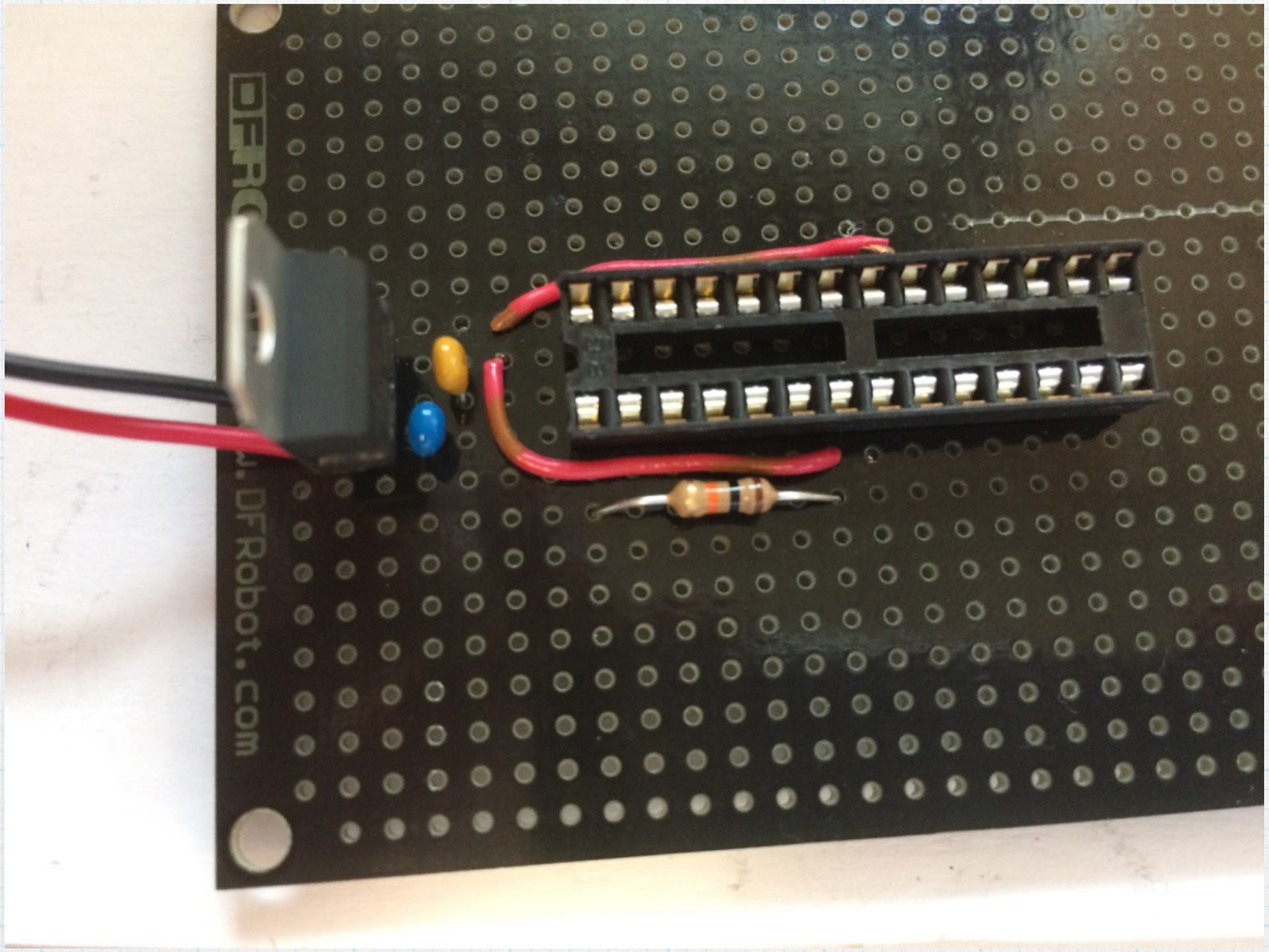
Connect pins 7 and 20 to 5 V for power. The wires are soldered to the pins on the back side.



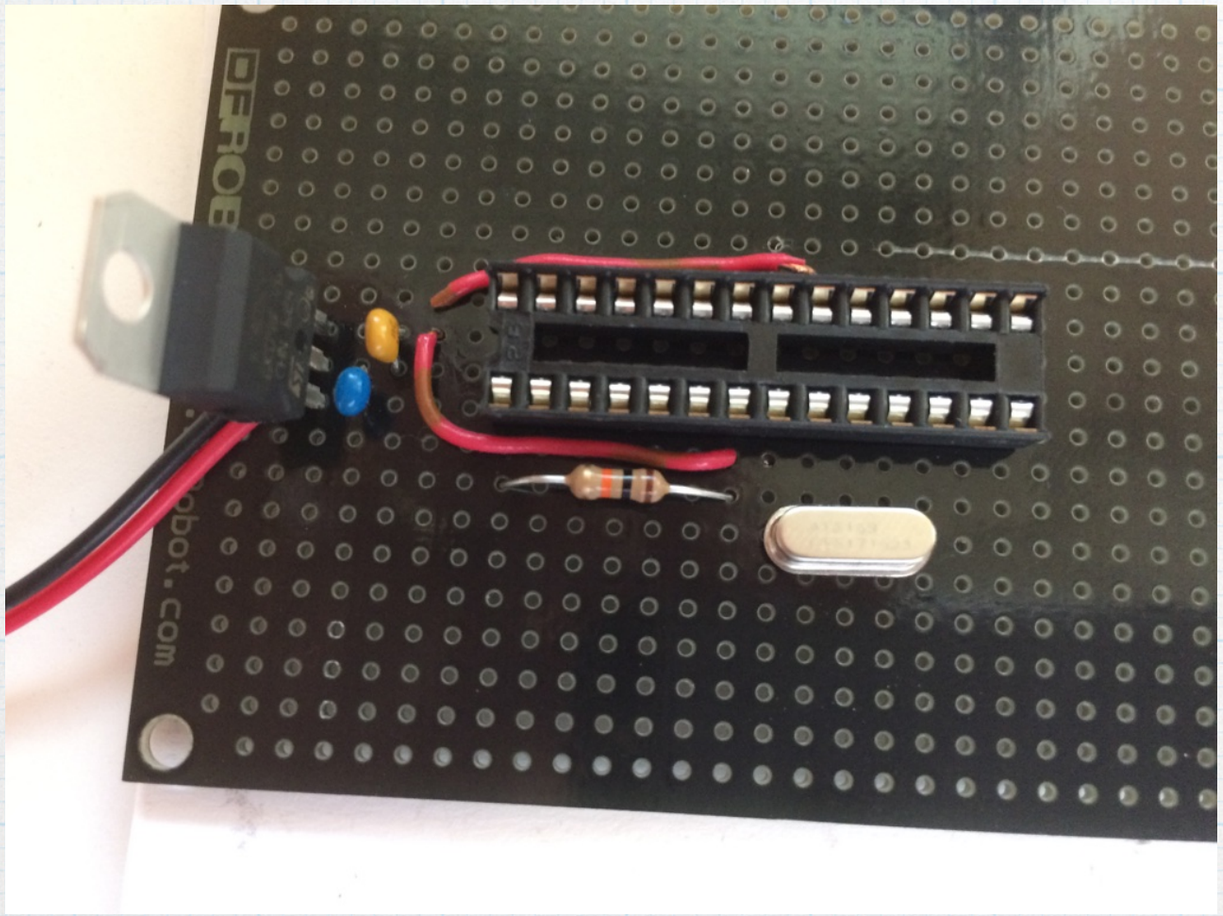
Ground line connected to regulator ground. Not necessary, but can be handy for connecting other components to ground.

solder “blobs” between ground line and chip ground connections

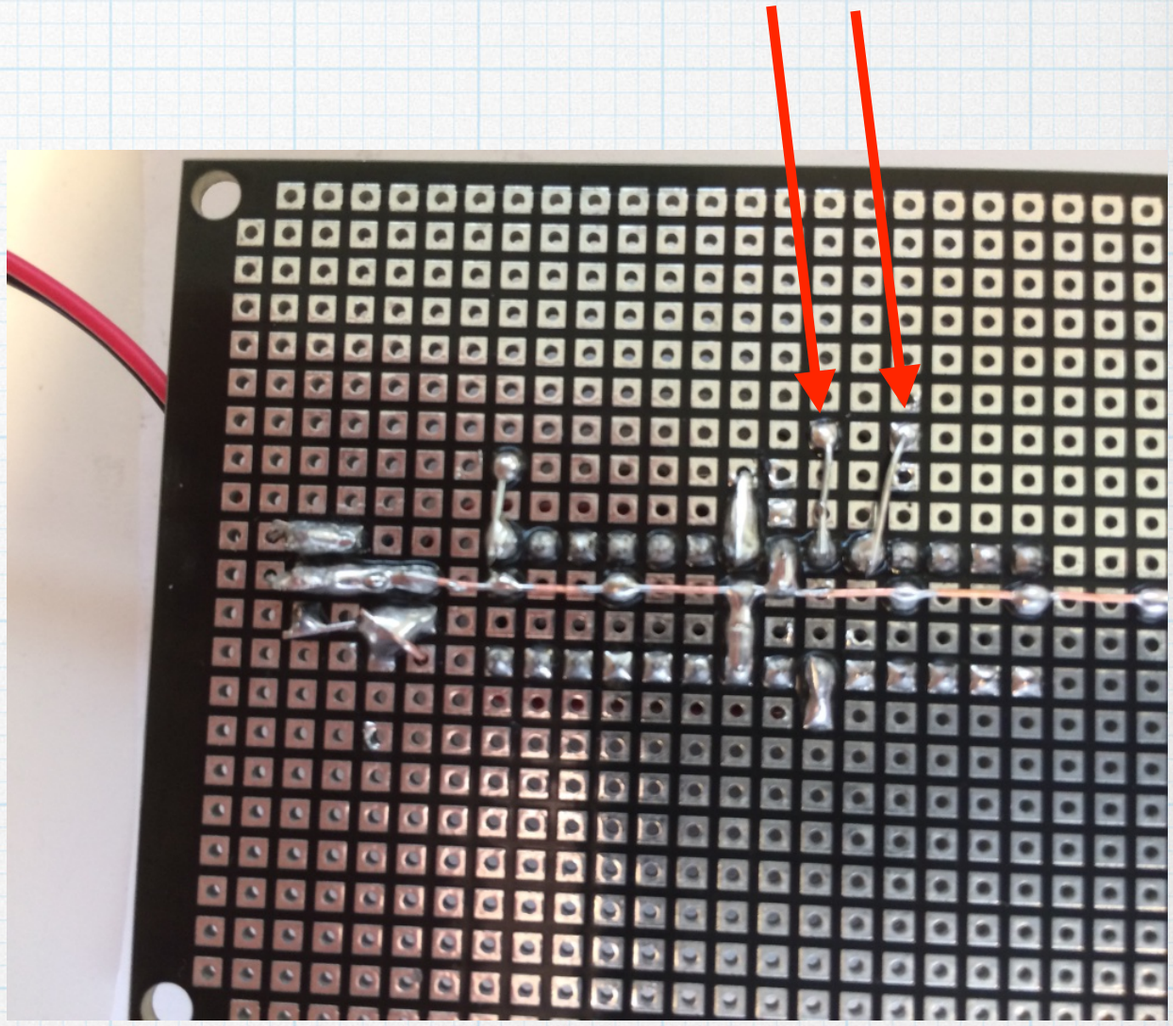
Connect pins 8 and 22 to ground. 8 is connected to 22 on the back side. I made a “ground line” running down the center. Then did “solder blobs” to the two ground connections on the chip.



Add the pull-up resistor for the reset — from pin 1 to 5 V. (Any 5 V node is fine.)



Front



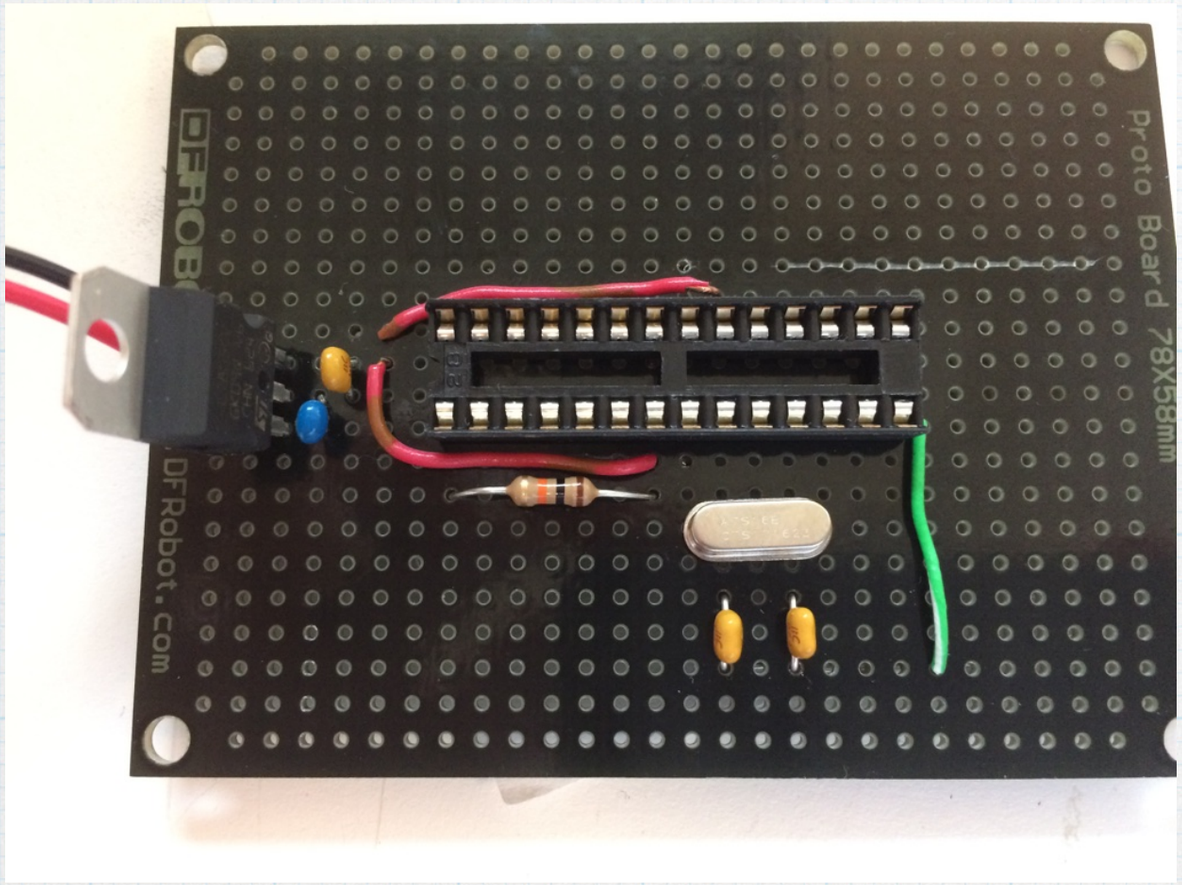
Back

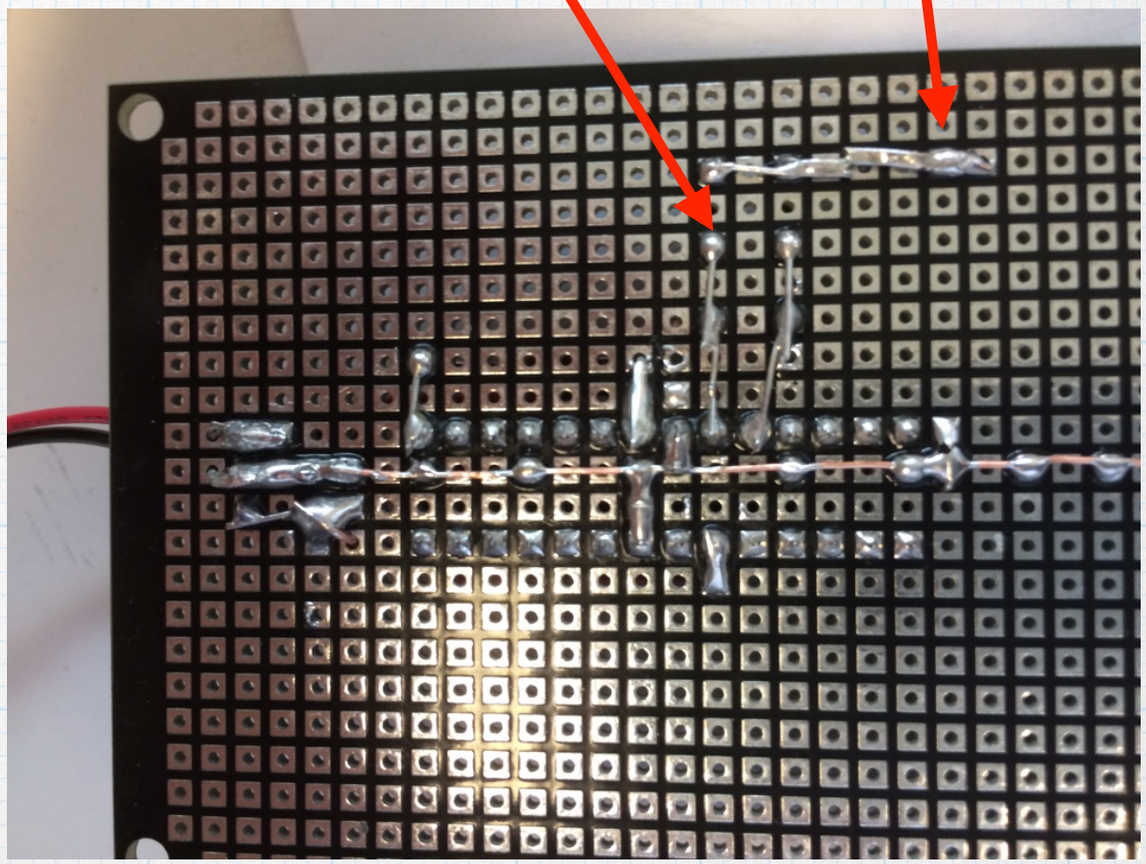
The crystal goes between pins 9 and 10.

ground via
green wire

Connections to crystal

Front





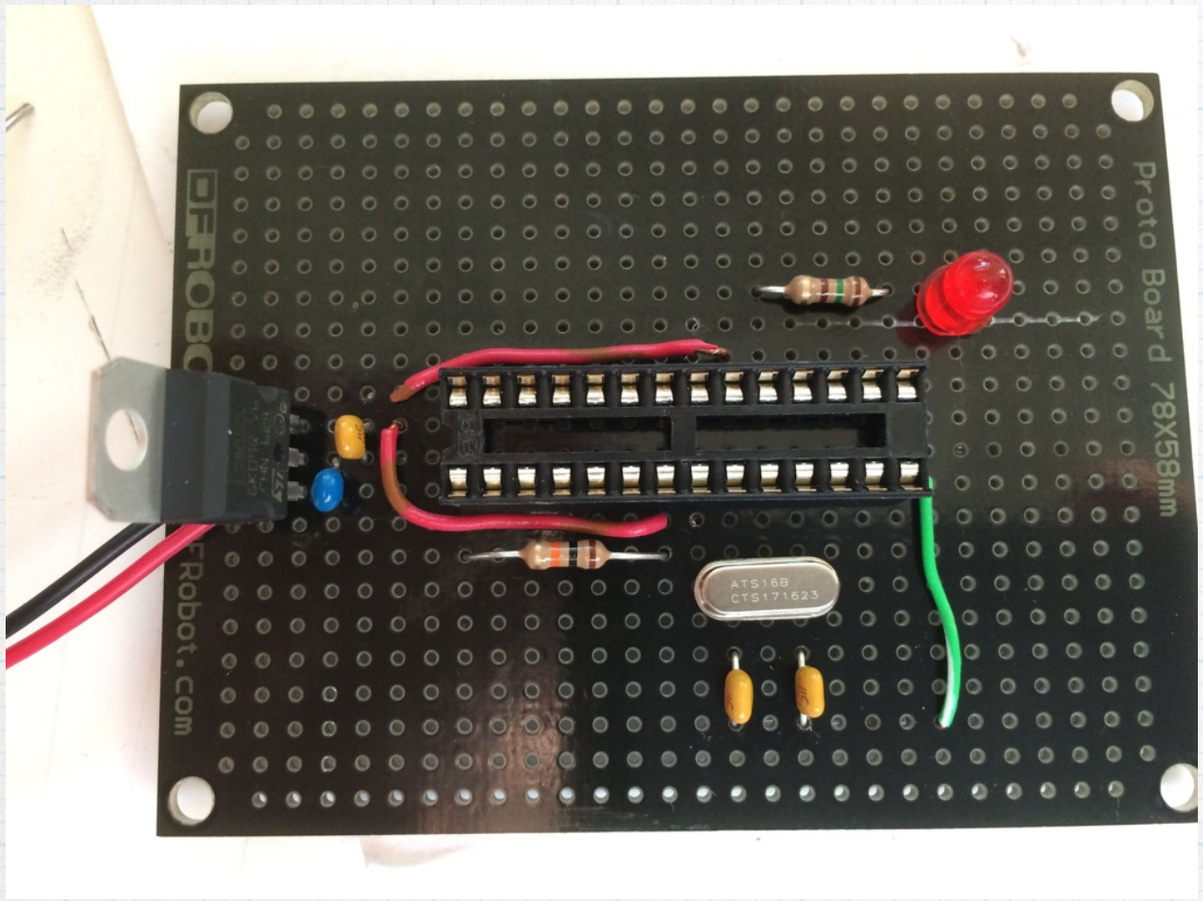
Back

And the two 22-pF capacitors — from pin 9 to ground and pin 10 to ground.

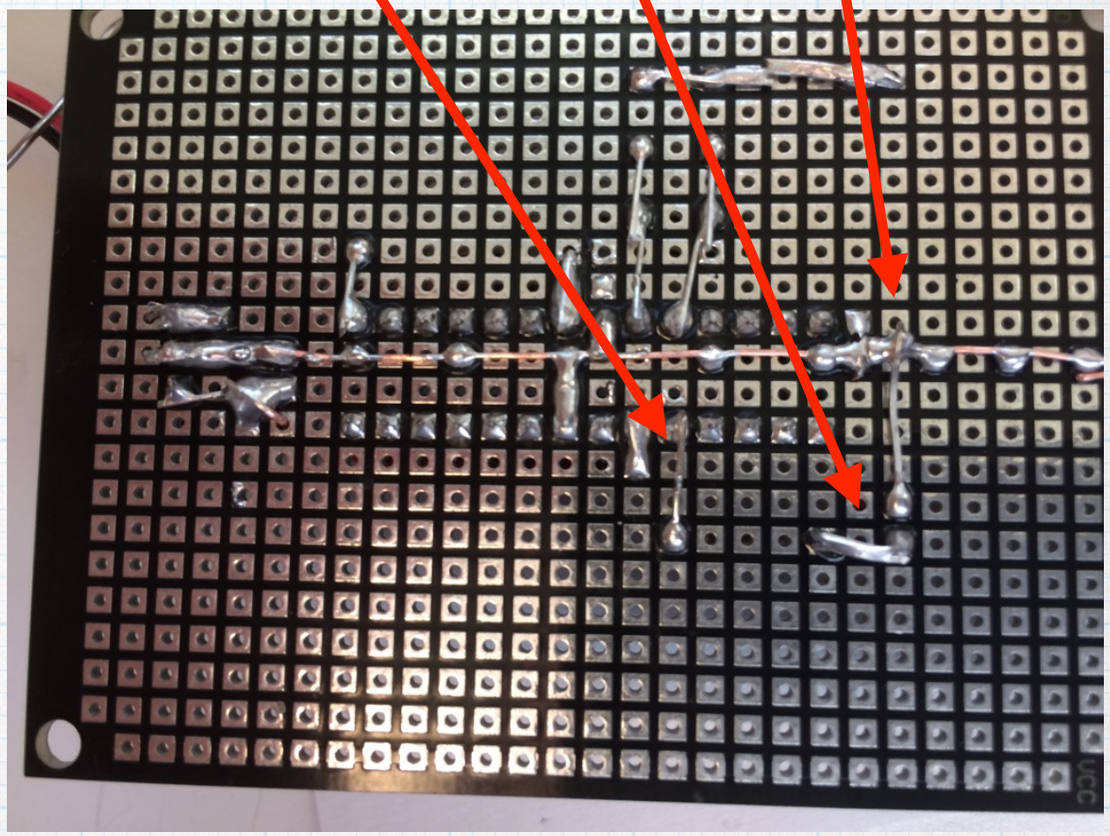
Resistor to pin 19

Resistor to LED

LED to ground line



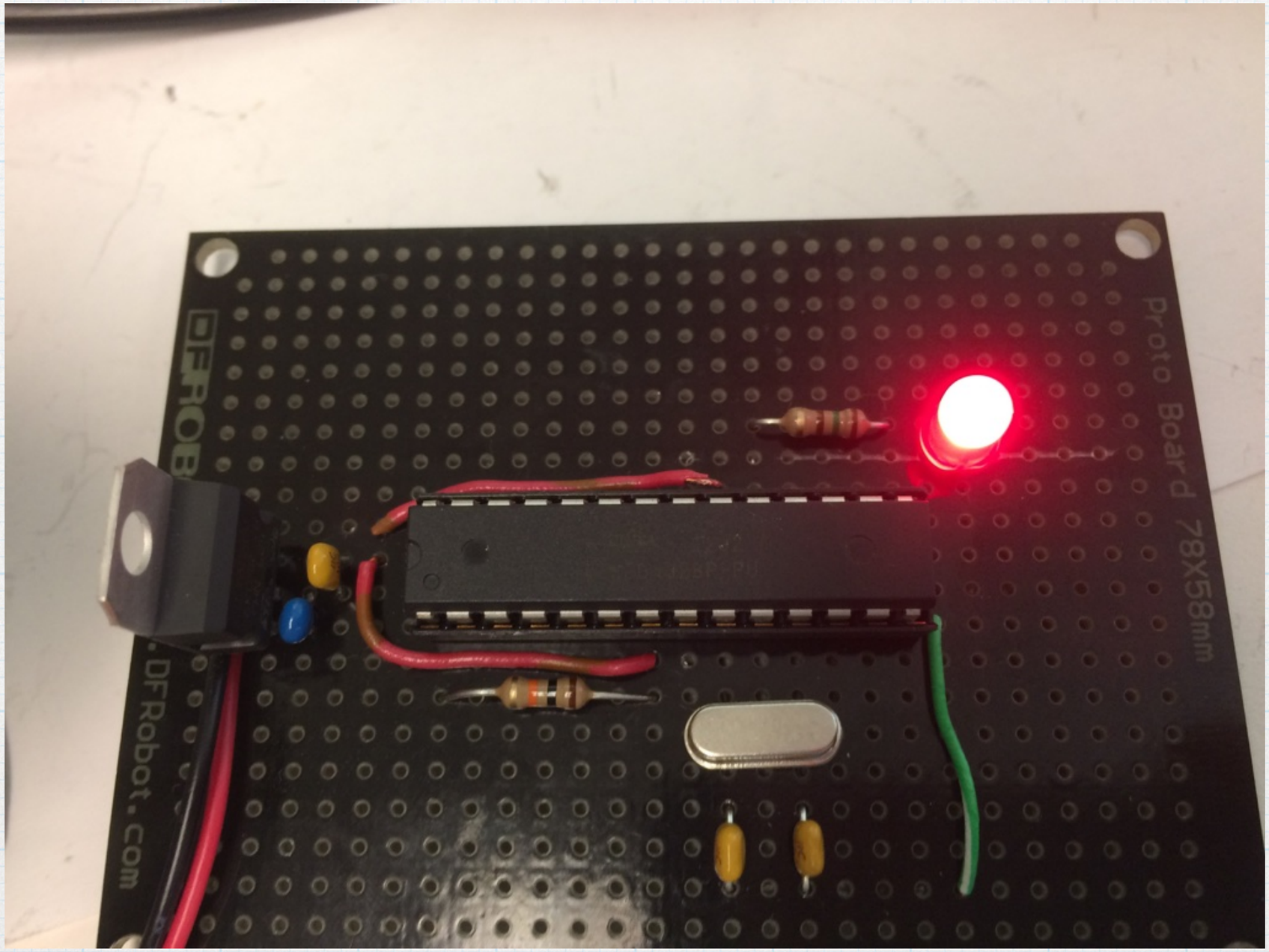
Front



Back

Finally, add the LED. For our overly simple application of a blinker (using the canned example program): a 150-Ω limiting resistor is connected to pin 19 (which is known as pin 13 in Arduino code) and the LED then connects from the resistor to ground. Get the LED polarity correct! (It's easily checked. Loosely connect the LED to the resistor — don't solder it. Then apply 5 V (or the 9 V battery) to the combination, positive on the resistor side and negative on the diode. If you lights up, you're good. If it stays dark, you have a problem.)

Plug in the programmed chip and hook up the battery. If all the connections are good, it should work!



A few extra thoughts

- It cannot be emphasized enough: prototype your system before soldering things together. Of course, the Arduino board itself is the ultimate prototyping tool. It's whole purpose is to make it easy change both hardware and software. But once you have settled on a design and want to make it permanent, you don't want to tie up your (relatively) expensive Arduino to do something mundane like measuring the temperature and operating a switch. Or blinking an LED on and off.
- You can buy Atmega328 chips that have a boot loader already installed, saving you the "trouble" of installing yourself. (Not that much trouble, really.) Many places offer them, but they tend to cost around $5 per chip, which is a pretty big markup. (Or so it seems to me.)
- Adafruit sells some nice stickers that can be affixed to the top of the chip to indicate the Arduino names for each of the pins. You can get 10 stickers for $3.00. https://www.adafruit.com/products/554 . Or just make your own.
- There are nicer perf-boards available that might make building the circuit easier. The ones used here were super-cheap. I can put together a list of perf-boards that I like to use.
- Two ways to extend on what we learned with the bare-duino exercise: (1) moving to other controller chips, either bigger or smaller and (2) switching to the Atmel programming tools rather than using the (somewhat limited) Arduino software "IDE".