ARDUINO PLAYGROUND

GEEKY PROJECTS FOR THE *CURIOUS MAKER*

WARREN ANDREWS

EARLY ACCESS

no starch press

NO STARCH PRESS EARLY ACCESS PROGRAM: FEEDBACK WELCOME!

Welcome to the Early Access edition of the as yet unpublished *Arduino Playground* by Warren Andrews! As a prepublication title, this book may be incomplete and some chapters may not have been proofread.

Our goal is always to make the best books possible, and we look forward to hearing your thoughts. If you have any comments or questions, email us at **earlyaccess@nostarch.com**. If you have specific feedback for us, please include the page number, book title, and edition date in your note, and we'll be sure to review it. We appreciate your help and support!

We'll email you as new chapters become available. In the meantime, enjoy!

ARDUINO PLAYGROUND

WARREN ANDREWS

Early Access edition, 9/8/16

ISBN-10: 1-59327-744-X
ISBN-13: 978-1-59327-744-4

Publisher: William Pollock
Production Editor: Laurel Chun
Cover Illustration: Josh Ellingson
Developmental Editor: Jennifer Griffith-Delgado
Technical Reviewer: Scott Collier
Copyeditor: Julie Jigour

BRIEF CONTENTS

Chapter 1: The Reaction-Time Machine 1

Chapter 2: An Automated Agitator for PCB Etching 17

Chapter 3: The Regulated Power Supply 45

Chapter 4: A Watch Winder 65

Chapter 5: The Garage Sentry Parking Assistant 105

Chapter 6: The Battery Saver 133

Chapter 7: A Custom pH Meter

Chapter 8: The Ballistic Chronograph

Chapter 9: The Square Wave Generator

Chapter 10: The Rainbow Thermometer

Appendix

CONTENTS IN DETAIL

1
THE REACTION-TIME MACHINE **1**

Parts List . . . 2
Downloads . . . 3
Reaction vs. Reflex . . . 3
How Does the Game Work? . . . 3
Measuring Time with the Arduino Nano . . . 4
Expected Speed Ranges . . . 5
The Schematic . . . 5
The Breadboard . . . 6
The Arduino Sketch . . . 7
Customized Reaction Commentary . . . 10
What Happens in the Loop . . . 12
Construction . . . 12
Preparing a Sturdy Case . . . 13
Mounting the Hardware . . . 14
Ideas for Customization . . . 15

2
AN AUTOMATED AGITATOR FOR PCB ETCHING **17**

Special Tools . . . 19
Parts List . . . 19
Downloads . . . 20
How Automatic Motor Reversal Works . . . 20
The Schematic . . . 21
Determining the Reversal Threshold . . . 22
Using an H-Bridge . . . 25
The Breadboard . . . 26
The Sketch . . . 29
The Shield . . . 33
PCB Layout . . . 33
Shield Design Notes . . . 34
Construction . . . 37
The Limit Wires . . . 38
The Crank Bushing . . . 39
Packaging . . . 41
The Etching Process . . . 42

3
THE REGULATED POWER SUPPLY **45**

Parts List . . . 46
Required Tools . . . 47
Downloads . . . 47
A Flexible Voltage Regulator Circuit . . . 48
The Schematic . . . 49
How the Circuit Works . . . 51

The Breadboard 53
Preparing the Arduino Pro Mini and LCD 53
Building the Breadboard 53
The Sketch 56
The Shield 57
Construction 59
Preparing the Enclosure 60
Mounting the Circuit Board 61

4
A WATCH WINDER 65

Why a Watch Winder? 66
Required Tools 69
Parts List 69
Acrylic 69
Other Hardware and Circuit Components 70
Downloads 71
Basic Watch Winder Requirements 71
Using an Arduino to Control Winder Revolutions 72
Using a Hall Effect Sensor to Monitor Rotations 72
The Schematic 73
The Breadboard 74
The Sketch 78
The Shield 84
Overview of the Motor Assembly 86
Construction 87
Preparing the Motor Plate and Bearing Box Acrylic 87
Bonding the Acrylic for the Bearing Box 90
The Stand 91
Preparing the Motor and the Driveshaft 92
Making the Watch Basket 96
Adding the LEDs 98
Leaving the Components on Display 99
Keeping the Watches in the Basket 100
Design Notes 100
Total Rotation Adjustment 100
How Many LEDs to Use and Where to Put Them 101
Motor Voltage 102
How Many Rotations Does the Watch Winder Make? 102
Closing Thoughts 103

5
THE GARAGE SENTRY PARKING ASSISTANT 105

Required Tools 106
Parts List 106
Optional Parts 108
Downloads 108
Basics of Calculating Distance 108
How the Garage Sentry Works 109
The Schematic 111

The Breadboard 112
The Sketch 114
Inside the setup() Function 116
Inside the loop() Function 116
Determining Distance 117
Triggering the Alarm 118
Construction 119
Drilling Holes for the Electronics 119
Mounting Options 121
Soldering the Transistors and Current-Limiting Resistors 123
Wiring the Pieces Together 123
The Deluxe Garage Sentry 124
The Deluxe Schematic 125
A Bigger Box 126
The Shield 127
The Sketch for the Deluxe Garage Sentry 128

6
THE BATTERY SAVER 133

Boats, Tractors, and Other Vehicles 135
Parts List 137
Special Tool Requirements 139
Downloads 139
The Schematic 139
How the Battery Saver Prevents Draining 141
Arduino to the Rescue 142
The Breadboard 142
The Sketch 145
The Shield 147
The PCB Layout 147
Preparing the Shield and Pro Mini Controller 148
Construction 148
Preparing the Enclosure 149
The Contact Support 150
Preparing the Copper Contact Assembly 151
Mounting Supplies for the Solenoid 152
Preparing the Release Rod, Springs, and E-Clip 153
Making the Release Lever and Pylon 154
Assembling All the Parts 156
Installing the Battery Saver into a Vehicle 159
Operating the Battery Saver 159
Normal Operation 159
Setting the Threshold Voltage 160
Protection from the Environment 160
Applying Cool Amp 161

1

THE REACTION-TIME MACHINE

In this chapter, I will show you how to build a time machine—that is, a Reaction-Time Machine. I'd love to say that this project will bring you "back to the future," but alas, it won't. The "time" it's looking at is the time it takes you to react to a stimulus, which makes for a fun game. This project is designed to accurately measure an individual's reaction time and provide an area for comments on the level of the individual's performance (see Figure 1-1). There is also plenty of room to personalize the game to make it even more fun for you, your friends, and your family.

Figure 1-1: Completed Reaction-Time Machine

Parts List

This project has one of the smallest parts counts of all the projects in this book, but don't let that attenuate its value for you. My family and friends have enjoyed playing the game repeatedly, and it's portable, so you can take it with you to get-togethers and other events.

Here's what you'll need:

- One Arduino Nano or clone
- Two SPST momentary switches (preferably one with a red button and one with a button of a different color)
- One SPST toggle switch
- One red LED
- Two 10-kilohm resistors
- One 470-ohm resistor
- (Optional) One audible annunciator, Mallory Sonalert or similar
- One 4 × 20 LCD display
- One I^2C adapter, if not included with the display (see "LCD Displays" on page XX)
- One 9V battery

- One 9V battery clip
- One 3.5 mm jack (if remote switch is used)
- One Hammond 1591 BTCL enclosure
- 28–30 gauge hookup wire
- Solder

Downloads

Before you start this project, check the following resource files for this book at *https://www.nostarch.com/arduinoplayground/*:

- Sketch file: *Reaction.ino*
- Drilling template for case: *Filename*

Reaction vs. Reflex

People often confuse reactions and reflexes, so I will start by defining both. *Reflexes* are involuntary, automatic responses to a stimulus. In a reflex action, the stimulus bypasses the brain and travels from the source of the stimulus to the spinal cord and back to the receptor that controls the response, without any cognitive acknowledgment. (Though I know many people for whom almost all stimuli—and information—seem to bypass the brain, often just getting lost instead.) Think of the doctor hitting your knee with a patellar hammer to trigger your knee-jerk reflex.

Reactions, on the other hand, take the stimulus to the brain to be processed, and then a return reaction travels to a receptor to result in some motor action. This process takes somewhat longer than a typical reflex, though some athletes are said to have reaction times so fast that it's possible their response is more similar to a reflex than a reaction.

NOTE Sports Illustrated *has done interesting work in this area, with eye-opening articles on baseball players and other athletes who have what appear to be exceptional reaction times.*

How Does the Game Work?

The Reaction-Time Machine game measures how long it takes an individual to press a button in response to a visual stimulus—in this case an LED. With a minor modification, you can add an auditory stimulus to the game: simply replace the LED with an audible annunciator, such as a Mallory Sonalert. Reaction time is measured in milliseconds or seconds (your choice), and it is the time between the moment the stimulus is activated and the moment the participant presses the button.

HISTORY OF REACTION-TIME DEVICES

Over the years, there have been many devices to measure reaction time. One of the simplest I remember from years ago required you to keep your fingers on either side of a ruler held by another person in mid-air. When the ruler was dropped, you would see how far it traveled before you could grasp it. The distance was translated to time using the algebraic equation

$$S = \frac{1}{2} AT^2,$$

where S is the distance traveled, A is the acceleration due to gravity, and T is the reaction time. After you build this project, try both the ruler test and the Reaction-Time Machine to see how close your times are between devices.

Measuring Time with the Arduino Nano

While there are many ways to measure elapsed time, this project takes advantage of the Arduino Nano's ability to keep accurate time. Microcontrollers keep time exceptionally well, and they measure the time that elapses between one input and another with a minimum latency. In addition to timing your reactions, the Nano shows the result on an LCD display.

The Nano does almost all of the work in this project; the other components are basically passive. After testing some early builds, I added features to the sketch to make the game more interesting and accurate. For example, I initially used a simple push button to reset the Nano and start a counter. The participant would press the red stop button as soon as the LCD display indicated so, and the Nano measured the time between pressing the reset and stop buttons. I found, however, that the player could anticipate the reset button being pushed and come up with some amazing reaction times.

To prevent the player from anticipating when the stimulus is about to occur, I had the Nano start the timer on a delay instead. The version in this book generates a random delay from when the reset button is depressed, activates the stimulus after the random delay, and counts the time from the stimulus to the moment the participant responds by depressing the stop button. That solved one problem.

Then, one of the participants tried to jump the gun and get an early start by holding down the stop button. I solved this problem by setting a minimum reaction time in the sketch. Any time under that minimum throws an error, and the LCD displays "Jumped the Gun" to indicate that the player pressed the button too soon.

I used a relatively large display—4 lines with 20 characters each—so there would be enough room to display the reflex time and some commentary on the relative prowess of the player. You can make your commentary as funny or serious as you want, but it must not exceed 60 characters in length—that is, three lines of 20 characters each. While I leave the commentary up to you, the sketch for this project includes some ideas that I used when putting it

together. You can always edit the commentary and reload the sketch to show comments specific to a set of users, like friends or relatives.

Expected Speed Ranges

Most individuals' reaction times seem to vary greatly, based on the small sample I tested. Interestingly, age doesn't seem to be a factor. The average reaction time was around 200 ms, and that is the average reaction time identified by many researchers.

The fastest response of anyone I sampled was 105 ms; however, the individual was not able to repeat that performance. Several individuals scored between 105 and 125 ms, but not consistently. Significantly lower reaction times may well be anomalous or the result of an individual actually anticipating the stimulus. My players' failure to repeat extremely fast reaction times would tend to bolster that idea. (I wouldn't want to accuse anyone of successfully pre-guessing the release moment.)

The Schematic

While the display could have been wired directly, using the I^2C interconnect made it a lot simpler and reduced the interface to only four wires: positive, ground, data, and clock (see Figure 1-2).

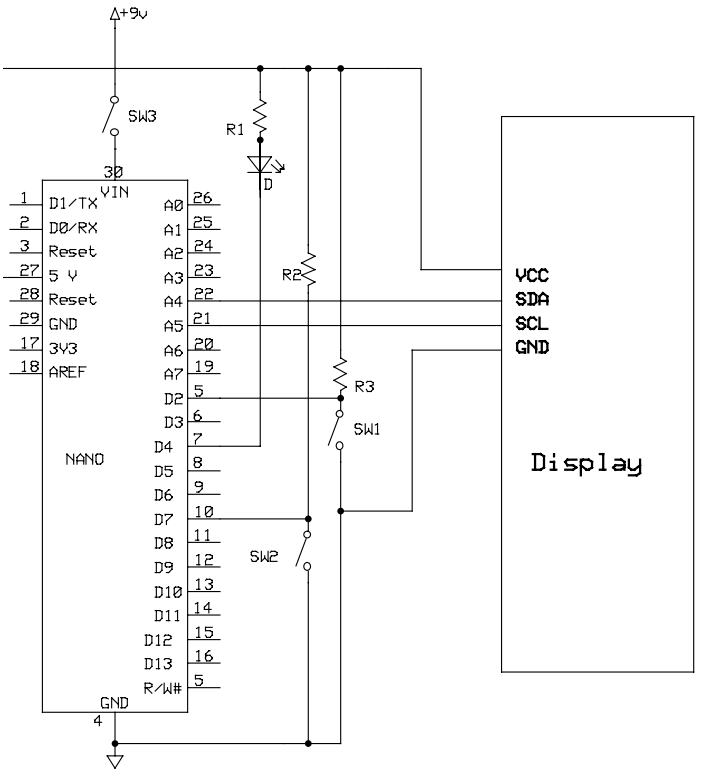


Figure 1-2: Schematic diagram of the Reaction-Time Machine

The only components needed are the Nano, three switches (one toggle switch for power and two momentary push-button switches for activate and reset), an LED, the display, and three resistors. Despite the relatively sparse parts count, the project performs elegantly.

The Breadboard

As is the case for most of my Arduino projects, the first step is to prepare a breadboard to prove the concept and test the sketch. Here's how to wire up the breadboard:

1. Connect the red strips (5V) on the breadboard together.
2. Connect the blue strips (GND) on the breadboard together.
3. Insert the Arduino Nano (or clone) in the breadboard, leaving two rows on one side and three on the other. (If the Nano does not come with stakes soldered in, prepare the board according to "Preparing the Nano and Pro Mini for Use" on page XX.
4. Connect the 5V terminal on the Nano to one of the red strips on the breadboard.
5. Connect the GND terminal on the Nano to one of the blue strips on the breadboard.
6. Connect the negative wire from the battery connector to the blue strip (GND). Remember that the breadboard has no switch, so you must disconnect the battery to turn it off.
7. Connect the positive lead from the battery connector to VIN on the Nano. (Do not connect the positive terminal of the battery to the red strip—it could permanently damage the Nano.)
8. Attach 5-inch wires to two normally open momentary push-button switches. (I use #22 solid conductor wire so it can plug in to the breadboard directly.)
9. Prepare a wire harness for the LCD display (see "Preparing LCD Displays" on page XX).
10. Connect the red wire from the LCD display to the red strip on the breadboard (5V) and the black wire from the LCD display to the blue strip (GND).
11. Insert the yellow wire from the display (SDA) to pin A4 on the Nano.
12. Insert the green wire from the display (SCL) to pin A5 on the Nano.
13. Connect one side of each push-button switch to the blue strip (GND).
14. Connect the other side of the red reaction switch (SW2) to pin D7 on the Nano.
15. Connect the other side of the yellow reset switch (SW1) to pin D2 on the Nano.

16. Connect a 10-kilohm resistor from pin D7 on the Nano to the red strip (5V).
17. Connect a 10-kilohm resistor from pin D2 on the Nano to the red strip (5V).
18. Connect the anode side of the LED (the longer leg) to the red strip on the breadboard (5V) and the cathode side to an empty row on the breadboard.
19. Connect a 470-ohm resistor from the cathode side of the LED to pin D4 on the Nano.

Upload the *Reaction_19.ino* sketch to the Arduino Nano (see "Uploading a Sketch to the Arduino" on page XX), and you should now be ready to go. Figure 1-3 shows the breadboard laid out with the switches dangling from their wires.

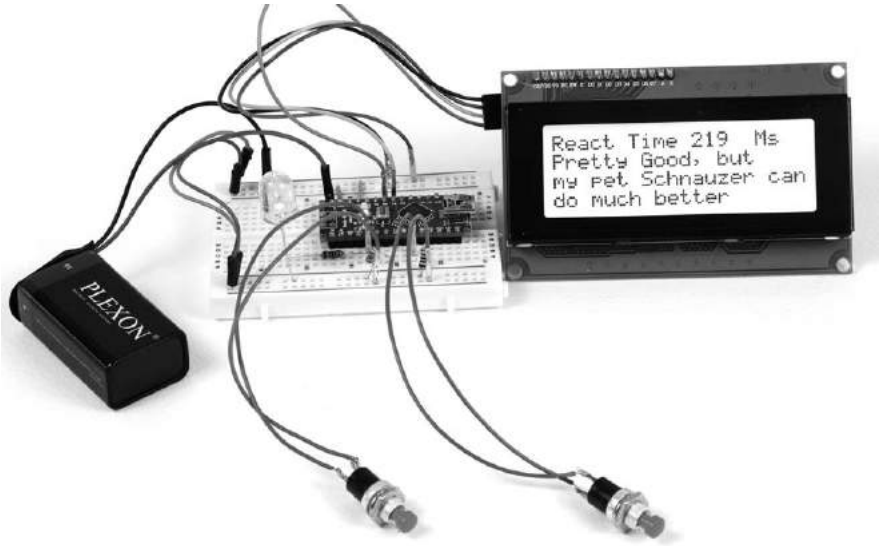


Figure 1-3: The breadboard setup for the Reaction-Time Machine. Because there is no on/off switch, you have to disconnect the battery to shut it off.

The Arduino Sketch

The sketch is the actual computer program that tells the Arduino what to do and when to do it. It is written in a language of its own that comprises structures, variables, arrays, functions, and so on, which represent a recipe for the microcontroller to follow. This language is converted into a sequence of zeros and ones that are routed to various parts of the controller and can perform storage, timing, comparison, arithmetic functions, and more.

The process of converting a computer language to a sequence of zeros and ones is called *compiling*. The compiling routine in the Arduino Integrated Development Environment (IDE) is activated when you click the Verify and Compile buttons in the upper-left side of the Sketch window.

The sketch gets pretty long because of all the comments that can be inserted when it checks the score; however, the basic operation uses only a handful of code lines. You can use the scoring function as is, modify it, or copy and paste it to make a new scoring function. As you'll see in my comment options, I've had fun with it.

The following code has been truncated to minimize the number of lines. However, you can simply go to *https://www.nostarch.com/arduinoplayground/* to download the entire sketch, which includes a number of comments.

```
/*
Includes score function, random number generation, false start
"jump the gun" indicator, and multiple comments spaced 10 ms apart

Mod for "jump the gun" gives response if time <70 ms
*/

#include <Wire.h> //Libraries included
#include <LiquidCrystal_I2C.h>

int start_time = 0;
int stop_time = 0;
int reacttime = 0;
int x;
int R;
int randnumber1;
int z;

LiquidCrystal_I2C lcd (0x3F, 20, 4); //Initiate LCD

void setup() {
  Serial.begin (9600);
  pinMode(2, INPUT);
  pinMode(4, OUTPUT);
  pinMode(7, INPUT);
  lcd.init();
  lcd.backlight();
}
//Begin function "score"
void score() {
  lcd.clear();
  lcd.print("Reaction Time ");
  lcd.print(reacttime);
  lcd.print(" ms");
  lcd.setCursor (0, 1);

  if((reacttime >= 105) && (reacttime < 135)) {
    lcd.print("Approaching Superman");
    lcd.setCursor(0, 2);
    lcd.print("but you can still do");
```

```
    lcd.setCursor(0, 3);
    lcd.print("a lot better");
  }

  if((reacttime >= 135) && (reacttime < 180)) {
    lcd.print("Superhero Status");
    lcd.setCursor(0, 2);
    lcd.print("but not yet");
    lcd.setCursor(0, 3);
    lcd.print("Superman");
}

  if((reacttime >= 180) && (reacttime < 225)) {
    lcd.print("You are trying ??");
    lcd.setCursor(0, 2);
    lcd.print("but not hard enough");
    lcd.setCursor(0, 3);
    lcd.print("still a loser");
  }

  if(reacttime > 225) {
    lcd.print("Lost your touch");
    lcd.setCursor(0, 2);
    lcd.print("If you ever had it");
    lcd.setCursor(0, 3);
    lcd.print("on the border of wimpy");
  }
}

//Begin main program
void loop() {
  digitalWrite(4, HIGH);
  lcd.clear();
  lcd.print("System is Armed");
  delay(1000);
  lcd.setCursor(0, 1);
  lcd.print("      READY   ");
  lcd.setCursor(0, 2);
  lcd.print(" Push Red Button");
  lcd.setCursor(0, 3);
  lcd.print("When Red lamp lights");

  randnumber1 = random(5, 25); //Generate random number between 5 and 25
  R = randnumber1;
    for(x = 0; x < R; x++);
  delay(5000);
  if(x == R) {

    digitalWrite(4, LOW); //Turn on start lamp
    start_time = millis(); //Initiate timer
    lcd.clear();
    lcd.print("Mash React Button");
    lcd.setCursor(0, 1);
    lcd.print("          ");
    lcd.setCursor(0, 2);
```

```
    lcd.print("          ");
    lcd.setCursor(0, 3);
    lcd.print("          ");

    while(digitalRead(7) == 1); //Wait for response

    stop_time = millis(); //Complete timing cycle
  }

  reacttime = stop_time - start_time;

  if(reacttime < 70) { //Jump the gun indicator
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Too anxious. You");
    lcd.setCursor(0, 1);
    lcd.print("(Jumped the Gun)");
    lcd.setCursor(0, 3);
    lcd.print("Could be Fatal!");
  }
score();
Halt:
 while(digitalRead(2) == 1);
}
```

The `#include` lines initiate the libraries: the I^2C library, *Wire.h*, establishes the rules for I^2C communications, and the LiquidCrystal library allows the Arduino to control LCDs. Then, we define the seven variables used to calculate reaction time. Next, `setup ()` sets up the serial communication—in case you want to adjust the code and view it on the serial monitor—and defines various pins as inputs and outputs. Inputs are required for the reset and stop buttons, and an output pin is defined for the LED that tells the player when to press the stop button.

Customized Reaction Commentary

One of the most entertaining aspects of this project is the chance to get creative when displaying the player's reaction time. After `setup()`, the sketch shows a function called `score()`, which lists different comments that could be displayed on the LCD based on the participant's response speed. A function may not necessarily be the most efficient approach (a look-up table or other approach could also have been used), but it works well enough. I used only a single scoring function in this iteration; however, you could easily define as many as you like and change your sketch to select one. For example, you might write a second function called `score1()` that could include a different set of comments and timing. Then, to switch from one function to the other, you'd have to change only the line that calls `score()` to call `score1()` instead.

To customize the sketch to include comments that could refer to your own friends or family members, you can simply enter your comments in place of the ones that are in my sketch. Don't forget to keep the text you want to print to the LCD in quotes so the Arduino recognizes the printable characters.

A word on the reaction time itself: each comment is for a range of reaction times of either 5 or 10 ms. I selected these ranges arbitrarily. After you play with the Reaction-Time Machine for a bit, you may wish to change these ranges based on the fact that users' responses may cluster around a particular area, such as from 195 to 225 ms. I found that many reaction times were in the 190 to 250 ms range, but your friends and family may be different. In that case, you can separate the comments by as little as 1 or 2 ms so players don't keep getting the same comment.

You can add as many comments as you wish, up to one comment per millisecond. If you accidentally overlap the times, the sketch may not compile.

NOTE *You can find reaction-time measurement tools on the Web if you want to see how your game's measurements compare. However, their accuracy is suspect because of the latency in the PC itself.*

ON WRITING CODE TO SET UP LCDS

There are a few points to note about the setup of the LCD display. The sketch uses a LiquidCrystal library, *LiquidCrystal_I2C.h*. If this library is not included in your Arduino IDE, you can easily download it using the instructions provided in the reference section on the Arduino website (*http://www.arduino.cc/reference/*).

In addition, each I^2C device comes with its own I^2C address. This allows several I^2C devices to be used on a single serial line. Usually the device documentation provides the address—in the case of the I^2C LCD I used, the address was 0x3F. Thus, when the sketch initiates the LCD, the code looks like this:

```
LiquidCrystal_I2C lcd (0x3F, 20, 4);
```

However, different displays come with different addresses. If you have an I^2C device that you do not have an address for, you can easily find the address by hooking up the device to an Arduino, downloading a scanner sketch from *http://playground.arduino.cc/Main/i2cScanner/*, and running the sketch. The scanner sketch should display the I^2C address on the serial monitor.

Many projects in this book use similar code to work with an LCD, so refer to this box any time you need a refresher on how that code works.

What Happens in the Loop

Now let's look at the sketch's loop. After `void loop ()` initiates the start of the program, the program calls `digitalWrite (4, HIGH)` to turn off the active light. Then, the LCD screen is cleared, and text is written to the LCD to indicate that the system is armed and ready for a player to push the reaction button as soon as the red LED illuminates, as shown in Figure 1-3.

Next, a random number between 5 and 25 is generated, and the program calls `delay(5000)` to count every five seconds from zero to the random number. As soon as the random number is reached, three things happen: first, the annunciator lamp illuminates; second, an internal timer is started in the Nano; and third, the display then changes to read "Mash the React Button."

NOTE *A wider range of random numbers might make this game even more interesting for players. You can easily experiment by changing the random number count, the delay, or both.*

The Nano is then instructed by `while (digitalRead (7) == 1);` to wait until the reaction button is depressed. After the button is depressed, the Nano calculates the reaction time with `reacttime = stop_time - start_time`. This time will be displayed on the LCD and used to select the appropriate comment in the `score()` function. Also, if the player's reaction time is less than 70 ms at this point, then the conditional statement looking for a participant to be "jumping the gun" displays appropriate wording for the LCD. The system is then halted and ready to be reset.

Otherwise, the serial print block is included in case you want to adjust the code and view it on a serial monitor. It also helps for debugging purposes.

Finally, the `score()` function is invoked, followed by the `Halt` command, and the system is ready to have the reset button depressed.

Construction

Building the Reaction-Time Machine can be as simple or as complex as you want. Initially, I placed all the components in the vinyl package that a flexible wrist brace came in. I cut a hole for the display connectors with an X-ACTO knife and punched the holes for the switches and LED with a paper punch, followed by a tapered reamer. The result was somewhat crude, as shown in Figure 1-4.

Figure 1-4: This was the Reaction-Time Machine's original, primitive package, which worked but turned out to be too flimsy. The vinyl was only 0.018 inches thick.

Preparing a Sturdy Case

Of course, a real case makes the game much sturdier, which is important when you have competitive players mashing those buttons. To keep things as simple as possible, I employed one of the clear ABS plastic cases from Hammond (1591 STCL). The clear top of the case allowed me to place the display behind the cover rather than machining out a hole for the display to protrude through. To mount the components, I simply drilled holes in the cover according to the drawing in Figure 1-5.

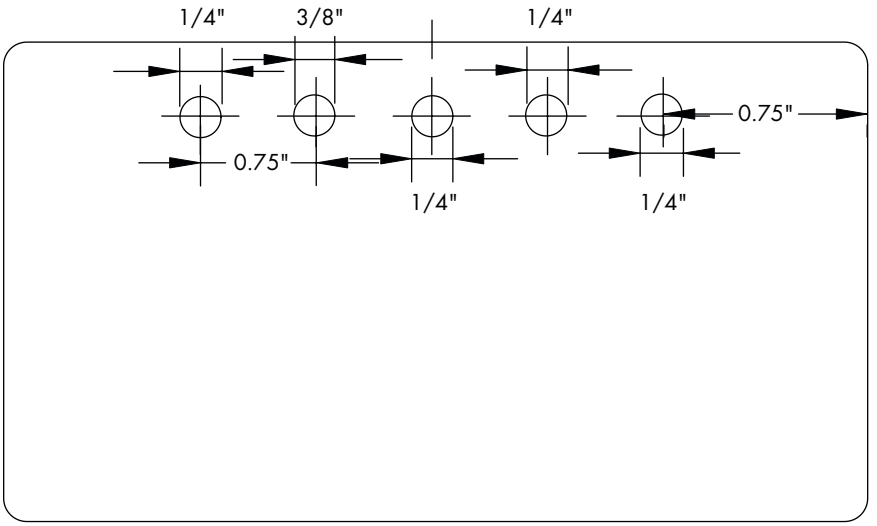


Figure 1-5: Drilling template for the Reaction-Time Machine

Quarter-inch holes work well for the momentary push-button switches, as well as for the toggle switch and 3.5 mm jack. For the 10 mm LED, I used a 3/8-inch drill and then reamed the hole out to make a tight fit. No other mounting hardware for the LED was necessary.

NOTE *The 3.5 mm jack is wired in parallel to the execute switch. If you want to use an external stand-alone switch, it can simply plug in to the jack. I abandoned the effort, however, as most participants preferred to hold the box in their hands.*

Mounting the Hardware

To mount the display to the case, I used two-sided 3M Indoor/Outdoor Super Heavy Duty mounting tape. I cut two sections the size of the LCD display's end bezel sections and bonded the display directly to the cover. The tape is difficult to remove, so make sure to place it right the first time. I used the same tape to mount the Nano and the battery holder to the back of the display. When mounting the display, I also used wire cutters to carefully cut off the corners of the display circuit board so it would fit far enough into the case without hitting the cover mounting pylons. See Figure 1-6 for the finished product, viewed from the underside.

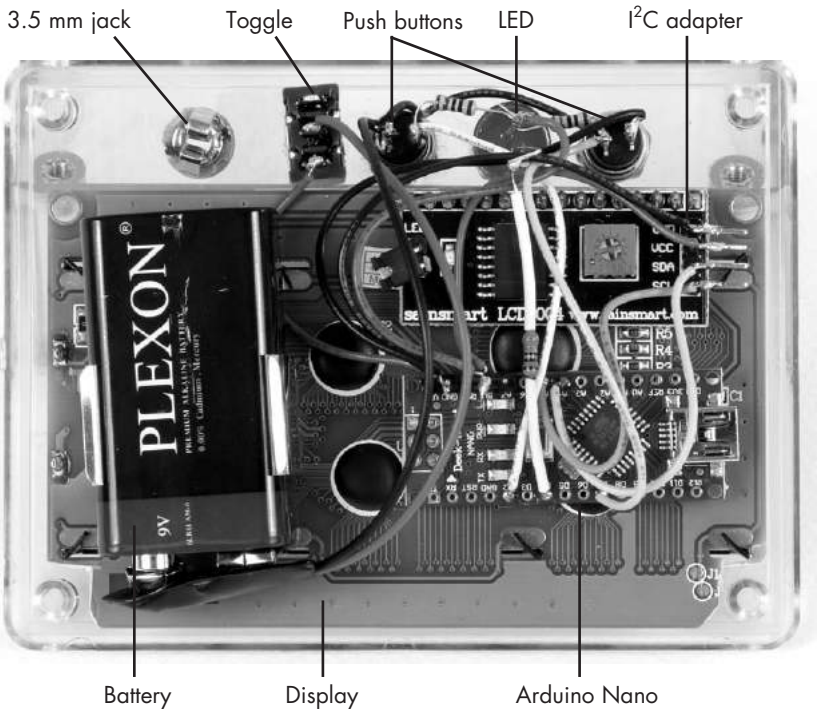


Figure 1-6: This is the rear of the unit mounted in the ABS plastic enclosure. Notice that the corners of the display (lower left and right) have been clipped off to fit around the top mounting pylons. The 3.5 mm jack is not wired, as I decided not to use it in this implementation.

Once all the components are in place, all that remains is to solder the components together, inserting the resistors where required. Take particular note of the I^2C adapter, which is the black paddleboard just below the switches and LED. While I could have bent the connectors and used a header to wire that up, the case may not have closed, depending on how carefully I crimped the connectors. Instead, I elected to solder the wires directly. It was only four wires, and it worked without much trouble. Finally, I printed out and attached labels from a Brother label maker. Figure 1-7 shows the completed unit.

Figure 1-7: The completed Reaction-Time Machine mounted in the Hammond 1591 STCL clear plastic enclosure

Ideas for Customization

There are many variations you could implement to increase the versatility and enjoyment of the Reaction-Time Machine. For example, as I developed it, I connected a Hall effect switch to one of the analog inputs and modified the sketch to automatically decrease the reaction time by a percentage when the Hall effect switch is activated. Then, I taped a small magnet to my finger that sat opposite the Hall effect switch so as I grabbed the box, it activated the switch. When I played, my reaction time was reduced by around 20 percent, while others had an actual reading. Far be it from me to suggest that readers try to hoodwink their adversaries, of course!

There are other modifications that can be made, such as incorporating a tone sound, or beep, as the sketch counts up to the random number. This can easily be accomplished with the addition of an annunciator and a few lines of code. If you're ingenious, there are other sound effects you could add, such as a vulgar sound that plays when poor scores are achieved.

You can also exercise your brain and add code to the sketch that will average scores after, for example, three tries before you reset it. I experimented with many variations as I played with the device, but I would caution that you can spend a great deal of time for minimal advantage. Put the game together and enjoy.

2

AN AUTOMATED AGITATOR FOR PCB ETCHING

This project uses the Arduino microcontroller to sense change in a motor's current drain and then reverse the direction of the motor. There are numerous applications for the measurement and use of current drain, and this project provides an example method that can prove useful in the development of future electronics projects.

"Designing and Building Your Own Circuit Boards" on page XX illustrates different ways to design and make circuit boards at home for a very modest cost using readily available and environmentally safe household products. Part of this process includes etching the copper off a clad board. The process is more efficient when the board is agitated in the etching solution, resulting in a laminar flow of liquid across the surface of the board in both directions. Depending on the chemical activity of the etchant and thickness of copper to be etched, this process can take anywhere from

10 or 15 minutes to well over half an hour! Standing there stirring the pot is pretty boring, but you can create a device that dunks the board in and out of the solution for you (see Figure 2-1).

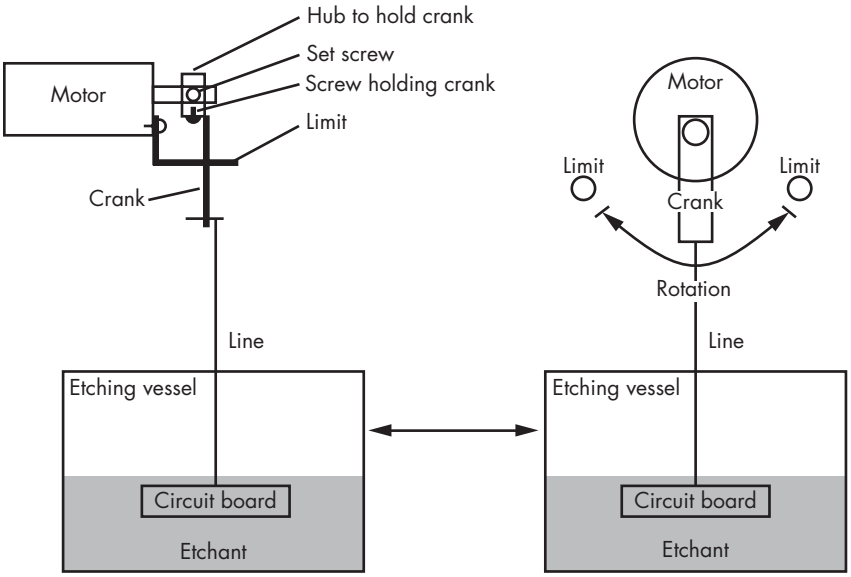


Figure 2-1: Illustration of the motor, crank, and etching vessels set up to dip a circuit board in and out of the etchant. While there are many ways to agitate a circuit board, dipping it into and out of the etching solution works well, especially for small boards.

In this project, the Arduino microcontroller, under the control of a sketch, waits to get real-world information from a system. The microcontroller then processes that information and uses it to make something happen.

INSPIRATION BEHIND THE AUTOMATIC MOTOR REVERSAL PROJECT

This project has its roots in a problem my friend had with a model train set accessory. The accessory included a tramway to take make-believe skiers up and down a miniature mountain. The original mechanism failed, so I created a little circuit to drive a DC motor that moved the skiers up and down. My idea was that when the tramcar reached either the top or bottom of its run, the motor would slow down or stall, resulting in an increase in current drain. That excessive current drain would reverse the motor by changing the polarity and thereby send the car back the other way. To date, the skiers are still at the bottom of the mountain because my friend and I never installed the board, but the core circuit works well and promises other interesting applications.

The ability to receive an input, process the information, and produce an output is *the* fundamental function of any microcontroller. In this case, the Arduino starts the motor turning, waits until it detects the motor drawing more current than usual, and then reverses the motor's rotational direction. This simple function has a number of different applications: you could use the voltage drop to provide a safety turn-off for an overloaded motor, create a system to limit motion, and more.

Special Tools

There are only a couple of special tools you will need. One is a 6-32 tap you can buy at Ace Hardware for a little over $1. But if you want a complete tap and die set for future projects—which is probably a good idea, as they're also just handy to have around the house—you can pick up a set at Harbor Freight (*http://www.harborfreight.com/*) for under $10 (item #69679). Other vendors offer similar items.

A handful of drill bits are also required. You can purchase drill bits individually, or you might think about getting an entire set. Once again, our friends at Harbor Freight have drill sets starting at under $4. It's always useful to have some drill bits around, so if you can afford it, a numbered drill set—#1 through #60—would be a good investment.

Of course, a drill is also useful—and necessary. See "Tools" on page XX.

Parts List

- One Arduino Nano (Available from Newegg, Mouser Electronics, and other retailers. You could also purchase a clone on eBay for just under $6.)
- One SN754410 quad H-bridge IC, with socket if desired (Available from eBay, Newark Electronics, Mouser Electronics, Digi-Key Electronics, and other suppliers. Note that if you use the socket, you lose whatever value the PCB offers as a heat sink.)
- One printed circuit board (PCB) or perf board
- One current-limiting resistor (You should have a selection available for experimentation, from 1 ohm to 10 ohm. You can buy resistors from many suppliers, including Jameco, MCM, Mouser, Digi-Key, and Newark, for under $0.05. A 1/8 W resistor will work for smaller motors, but get a 1/4 or 1/2 W resistor for larger loads.)
- Two 330-ohm, 1/8 W resistors (Available through previously listed suppliers)
- Two LEDs, one red, one green (Available on eBay and elsewhere for under $0.10 each. Consider buying an assortment, as we'll use them in upcoming projects.)

- One LM7805 voltage regulator (Many suppliers—including eBay, Amazon, Jameco, MCM, Mouser, Digi-Key, Newark, and others—will have this part.)
- One plastic box (I recommend the Hammond 1591 XXATBU available from Newark Electronics, Mouser Electronics, and Digi-Key.)
- Two 2-pin female headers to connect the motor to the shield (Pololu Robotics & Electronics item #1012)
- Four 4-pin female headers to plug the Nano into (Pololu Robotics & Electronics item #1014)
- One small solder lug
- One 3.5 mm, 2-conductor jack and plug for power supply (Any supplier listed previously should carry this.)
- One SPST toggle switch (Any supplier listed previously should carry this.)
- One plug-in wall adapter with an output of 5 to 12V at 200 mA or better (Try eBay, Jameco, MCM, or Electronic Goldmine. Refer to "The Voltage Regulator" on page XX for some important considerations before buying an adapter.)
- One gear head motor (I used a 6V motor from Amazon, the Amico 20 RPM 6VDC, and I've seen similar motors on eBay for about $6 and change. Later projects use the same motor, so you might buy a few. You can find motors as low as 8 RPM and on in the same price range. There are many places to buy motors with 10 to 50 RPM and 5 to 15V on the Web, including surplus houses like Electronic Goldmine.)

Downloads

Before you start this project, check the following resource files for this book at *https://www.nostarch.com/arduinoplayground/*:

- Sketch: *Reverse_IV_3.ino*
- Shield (PCB): *Reverse.pcb*
- Template: *Motormount.pdf*

How Automatic Motor Reversal Works

The Arduino is perfect for this project because it can control the whole system, and it simplifies the problem of accommodating different motors with different current requirements. Implementing the project in discrete components would require several more components than the equivalent Arduino circuit. Further, changing values for different motors or different reversal thresholds would mean changing a lot of hardware, but with Arduino, you just have to make a simple program change. The Arduino also provides the flexibility to add delays at each end of the run if desired.

The motor circuit you'll connect to the Arduino uses a resistor between the power supply and the motor (see Figure 2-2). When the motor slows or stalls, the current increases, creating a voltage drop across the resistor.

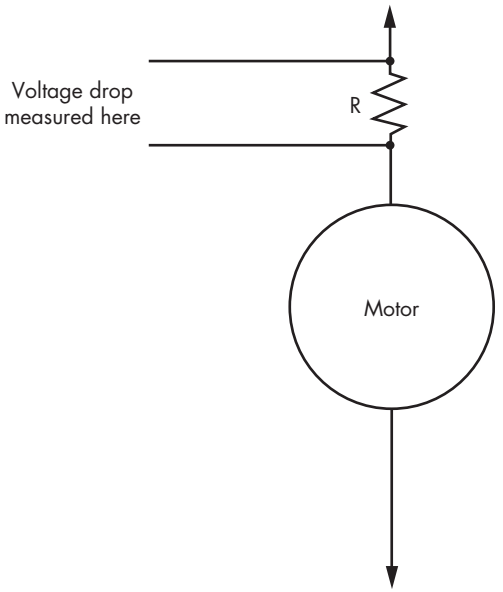


Figure 2-2: A voltage is created across the resistor between the positive supply and the input to the motor. It is this voltage that triggers the operation of the circuit.

The voltage drop across resistor R is the real-world input to the microcontroller. In this project, that voltage drop is fed to the Arduino Nano's two analog input pins that straddle the dropping resistor. The microcontroller digests this input and creates an output designated by your program.

NOTE *You could implement the circuit with only a single analog input, but that would curtail some of the flexibility of the circuit—particularly if you use motors that run at different voltages.*

The Schematic

The agitator circuit feeds the voltage that appears across resistor R1 into two of the Arduino's analog input pins, A0 and A1, setting up the real-world input (see Figure 2-3).

All grounds in this circuit are connected together, and the voltage across pins A0 and A1 is the voltage your program will use to decide when to reverse the motor's direction. Note that this voltage is not referenced to either the positive or negative rail, but it must be between 0 and 5V to prevent damage to the microcontroller. If you get stuck on wiring the H-Bridge, see "Using an H-Bridge" on page 25.

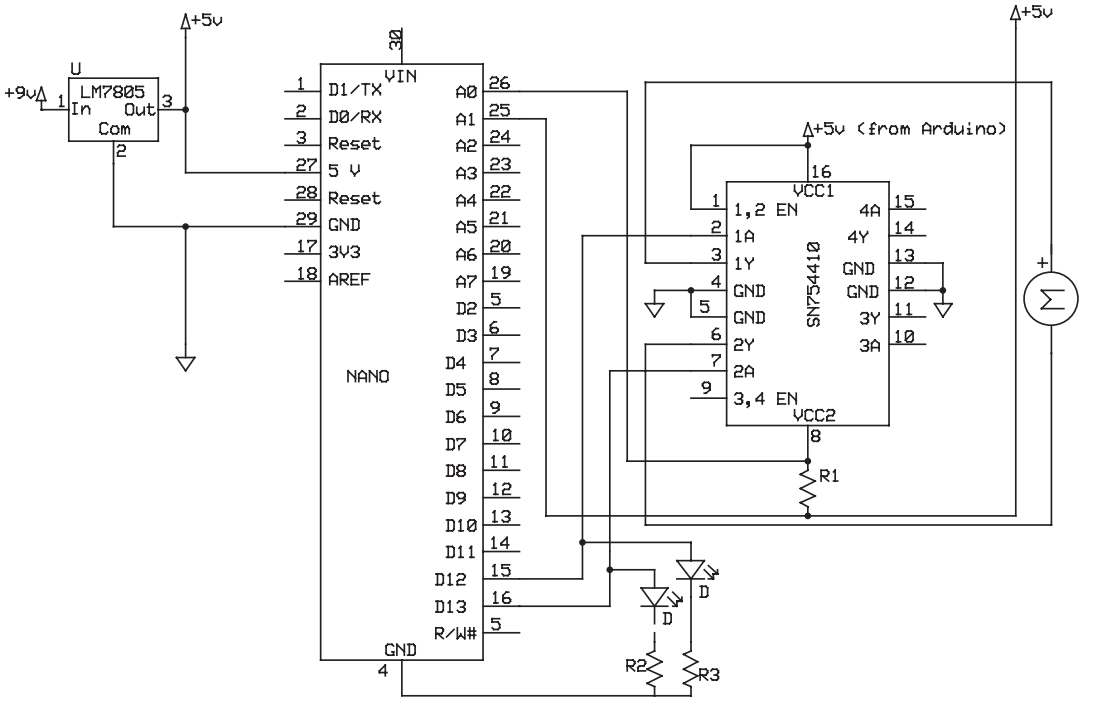


Figure 2-3: The completed schematic for this project shows the 5.6-ohm voltage-drop resistor (R1), the two LEDs (D), the 330-ohm current-limiting resistors (R2 and R3), and the quad H-bridge (SN754410), of which half is used.

The analog-to-digital converter (ADC) behind each analog pin provides 10 bits of resolution, which means the converter can deliver up to 1,024—that is, 210—different values, from 0 to 1,023, depending on the input.

Thus, if the power supply is 5V, each increment is roughly

$$5\text{V} \div 1{,}023 \approx 0.0048\text{V} .$$

Determining the Reversal Threshold

In order to write a program that tells the Arduino when to reverse your motor, you have to determine that point yourself, with some math and a little bit of faith.

First, determine the current drain of the motor you're using. It's usually printed on the motor's label. The motor I used has a current drain of about 40 milliamps (mA), or 40 thousandths of an ampere (see Figure 2-4). Now we get into the heavy math. You're going to have to use a formula known as *Ohm's law* to determine the voltage threshold to set in the sketch.

Figure 2-4: I used an Asian import motor, shown here with one limit pin installed, that has demonstrated reliability and performance. The screws are M3 – 0.05.

I used a 5.6-ohm resistor in series with my motor circuit. Using Ohm's law, which states that voltage equals current times resistance ($V = IR$, with voltage in volts, current in amperes, and resistance in ohms), we're able to calculate that 40 mA times the resistance of 5.6 ohm is about 0.224V:

$$\frac{40\text{A}}{1000} \times 5.6\Omega = 0.224\text{V}$$

Now, go back to the ADC. It has 1,024 units to represent 5V, so each unit represents 0.0049V. A little arithmetic reveals that the 0.224V dropped represents about 46 units out of the 1,024:

$$\frac{0.224\text{V}}{0.0049\text{V per unit}} = 45.85 \text{ units}$$

There are some estimates you have to take on faith—at least until you confirm with a test. This is one. As a motor is slowed or stalled, the current drain increases. Depending on the motor, the increase in current is typically somewhere between two and four times the normal current drain, but possibly more.

NOTE *With no load (or minimal load), current drain on the motor is minimal. With a usual running load, current can be four to five times the no-load current. With a heavy load, current can be as much as 10 times that, depending on the motor design.*

So according to our good-faith model, a good place to start setting the threshold for reversing the motor would be in the area of 90 to 100 units of the ADC's 1,024 units.

Alternatively, you could use a digital multimeter to measure the exact current drain first (see Figure 2-5). To use a multimeter to measure current drain, set its indicator to 200 mA to start; you may need to set it as high as 10A if the motor doesn't move when you build the circuit described here.

Figure 2-5: Multimeters are handy for many projects and useful to have around the house. They're available from a variety of sources at a range of prices. I use this cheap one from Electronic Goldmine, but if you plan to do high-voltage experiments, invest in a really good multimeter.

Build the circuit as shown in Figure 2-6, and then connect the red lead of the multimeter to the power supply. Connect the black lead of the multimeter to the motor lead to complete the circuit. If the reading is negative, reverse the red and black leads of the multimeter. Depending on your power supply voltage and the motor's voltage requirement, you may also need to connect the motor to power through a voltage regulator circuit, as described in "The Voltage Regulator" on page 34.

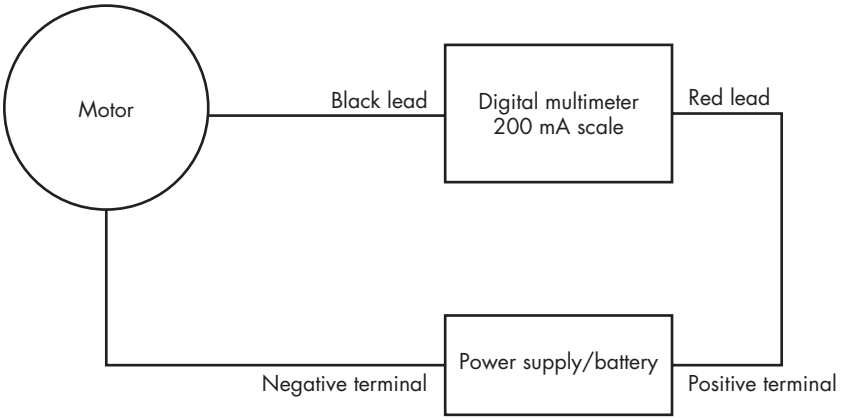


Figure 2-6: Connection diagram for measuring the current drain of the motor

To check the current drain, hold the shaft of the motor to slow it, and watch the readout on the multimeter. You can get an accurate indication of the number of ADC steps by plugging your readout in to Ohm's law, calculating the voltage, and converting into steps, as I did.

NOTE *In the sketch, I use a value of 100 as the threshold for reversing. You could also calculate the absolute value of the voltage drop by multiplying 100 by 0.0049V:*

100 steps 0.0049V per step = 0.49V

Remember, the exact threshold depends on the type of motor you use. Different motors will have different current capabilities and may even require a different value resistor. The description and model of the motor I used is included in the "Parts List" on page 19. Also, note that the value of current drain is not precise. The nature of permanent magnet motors is such that the current drain under load will be a range, not an exact number.

As the current increases, the voltage drop increases until it reaches the point where the microcontroller is instructed to do something. At that point, the difference in analog voltage that appears between A0 and A1 is above the preset threshold, which will set the Arduino into action. Once the threshold is reached, the Arduino tells the H-bridge to reverse the current to the motor.

Using an H-Bridge

You'll likely encounter an H-bridge driver in future projects because it's a very versatile part and can serve numerous functions. There is quite a selection of H-bridge chips available, but I've been using the Texas Instruments SN754410 quad H-bridge. It's popular because it operates over a wide voltage range and is extremely flexible—and inexpensive. The logic operates at a 5V level, while the drive can be as much as 36V with a continuous output of 1A (and a peak output of 2A), making it capable of driving a wide variety of hobby motors, solenoids, and even relays. It comes in a standard 16-pin dual inline package (DIP). The DIP package was a longtime standard but is slowly being replaced by newer types (see "Dealing with Small ICs" on page XX). It's the conventional centipede-looking circuit.

Figure 2-7 shows the pinout for the SN754410 H-bridge, and Table 2-1 shows its function table. You'll find more information in Texas Instruments' data sheet at *http://www.ti.com/lit/ds/slrs007b/slrs007b.pdf*.

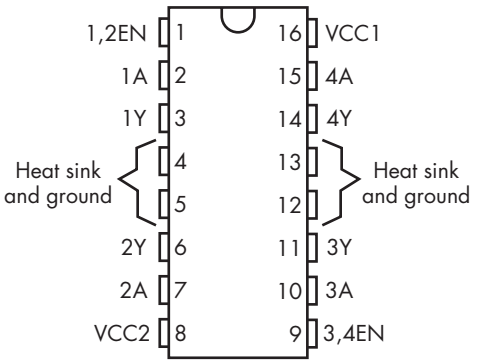


Figure 2-7: The pinout for the SN754410 quad H-bridge chip used in this project. Note that pin 1 is in the top-left corner of the chip when viewed from the top with the notch pointing up.

Table 2-1: Function Table for the SN754410

Inputs		Output (Y)
A	**EN**	
H	H	H
L	H	L
X	L	Z

According to the data sheet, in this function table, *H* stands for *high level*, *L* stands for *low level*, *X* means the level is irrelevant to the circuit behavior, and *Z* indicates high impedance, which turns the motor off.

The H-bridge is an elegant motor-control solution for several reasons. It allows you to reverse the polarity from a single supply, and it provides for different logic and control voltages. In addition, if both inputs of the dual H-bridge are either high or low, there will be no output. The sketch takes advantage of that in a function written to stop the motor. Other projects in this volume also use this capability.

The Breadboard

For most Arduino projects, I suggest building the circuit on a breadboard first to make sure you're going in the right direction and to prove your initial hypothesis. Use a standard breadboard and the plug-in wires that are sold as accessories for the breadboard (see Figure 2-8).

Figure 2-8: Typical small breadboard and plug-in wires

Before you begin building the circuit on the breadboard, look over your Arduino. Many Arduino boards come complete with the male headers already soldered in place. However, that's not always the case; some Asian suppliers include the headers loose with the processor board. If your board lacks headers, see "Preparing Arduino and LCD Boards" on page XX for complete instructions on attaching them.

Most breadboards include a red and blue strip on the entire length of each side of the board; the holes next to these strips are used for power (+) and ground (-), respectively. Before you hook up the circuit, use a wire to connect the red column on the right to the red column on the left. Connect the blue columns to each other, too.

WARNING *Do* not *connect the red column to the blue column! This will cause a short circuit and will burn out the electronics.*

Figure 2-9 shows my breadboard for this project, and the schematic from Figure 2-3 lays out the connections.

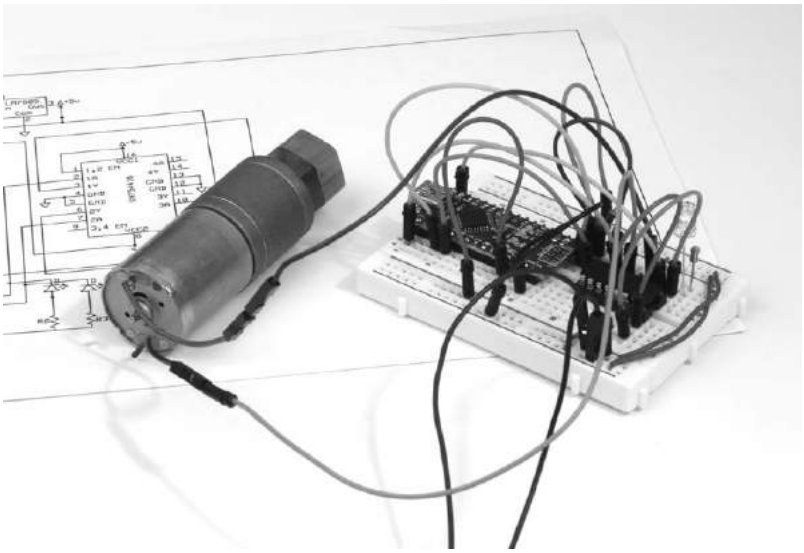


Figure 2-9: This is the breadboard I used as a proof-of-concept to make sure everything worked as anticipated.

WARNING *Don't plug the Arduino in to the computer while it is actually receiving power from the voltage regulator. This could burn out the Arduino.*

I suggest prototyping your circuit as follows:

1. Insert the Nano board into the breadboard, leaving a couple of rows of holes at one end.
2. Place a wire from the pin labeled *5V* on the Nano (pin 27) to the positive (red) strip on the breadboard.

3. Place a wire from GND on the Nano (pin 29) to the negative (blue) strip on the breadboard.
4. Find three consecutive holes on the board where they will not connect to anything and insert the three leads of the LM7805 into them.
5. The input lead of the LM7805 will go to the 9V power supply, the ground of the LM7805 will go to the blue negative rail, and the output of the chip will go to the red stripe. (See Figure 2-11 for the LM7805 pinout.)
6. Insert the H-bridge into the breadboard with the notch facing the Nano, and leave a couple of rows between the H-bridge and the Nano.
7. Use a wire to connect pin 1 and pin 16 of the H-bridge together (see Figure 2-7). Then, use another wire to connect pin 1 to the positive connection on the breadboard. This connection from pins 1 and 16 provides the voltage to run the logic on the H-bridge and also to enable the section of the H-bridge used.
8. Use a wire to connect pins 4 and 5 of the H-bridge, and then connect them to the negative terminal on the breadboard. Running a wire from either pin 4 or pin 5 to ground will do the trick.
9. Similarly, connect pins 12 and 13 of the H-bridge together, and connect them to ground.
10. Use a wire to connect one side of the motor (it doesn't matter which) to pin 3 of the H-bridge, and connect pin 6 of the H-bridge to the other side of the motor.
11. Connect digital pin D12 of the Nano to pin 2 of the H-bridge.
12. Connect digital pin D13 of the Nano to pin 7 of the H-bridge.
13. Connect one side of the 5.6-ohm resistor (R1) to pin 8 of the H-bridge.
14. Connect the other side of resistor R1 to the breadboard's positive strip.
15. Insert a wire from pin 8 of the H-bridge to analog pin A0 of the Nano.
16. Insert a wire from the positive (red) connector to analog pin A1 of the Nano.
17. Insert the positive side (long lead) of one LED to D12 of the Nano.
18. Insert the negative side of the LED into an empty row on the breadboard.
19. From that row with the negative side of the LED, connect a 300-ohm resistor (R2) to ground (blue strip).
20. Insert the positive side (long lead) of the second LED to D13 of the Nano.
21. Insert the negative side of the second LED into an empty row on the breadboard.
22. From that row with the negative side of the second LED, connect a 330-ohm resistor (R3) to ground.

The VCC2 supply drives the output to the motor. It goes from the positive side of the supply—the output pin of the regulator in the schematic—through resistor R1 to pin 8 of the H-bridge. VCC2 becomes the low-voltage side of resistor R1; it will have a lower voltage as the load on the motor increases because the other end of the resistor is attached to the positive of the power supply. The VCC2 supply voltage can be anywhere from the 5V that the logic uses to the 36V limit of the H-bridge. For this project, I simply tied the voltage-drop resistor directly to the 5V supply, which worked well with a 6V motor.

The Nano's D12 and D13 output pins drive the A inputs of the H-bridge, while A0 and A1 inputs straddle the voltage-drop resistor, R1. It's this voltage-drop value that tells the Arduino to change the outputs to instruct the H-bridge to reverse the motor. When output D13 is high and D12 is low, output pin 2Y on the H-bridge becomes positive while 1Y remains negative. When D12 is high and D13 is low, the reverse happens, and 1Y becomes positive while 2Y stays negative. When both pins have high or low output, they are at the same potential (or voltage), and the motor is not driven. (Refer to the function table in the H-bridge chip's data sheet, or see Table 2-1.)

The Sketch

The following sketch is written so that when the motor reaches its limits in one direction, both outputs go low, and when it reaches its limits in the other direction, both outputs go high. When both outputs are either high or low, there is no potential across the motor and it is stopped for a specified delay time. After the delay is satisfied, the motor starts in the other direction. Because LEDs are wired to pins D12 and D13, you'll also get a visual indication. Both LEDs are illuminated when the motor pauses in one direction, and both LEDs are off whe the motor pauses in the other direction.

```
/* Sketch for the Automatic Motor Reversal Project
*/

//Identify pins that will not change
const int ledPin1 = 12;   //LED1 in schematic
const int ledPin2 = 13;   //LED2 in schematic
const int analog0 = A0;
const int analog1 = A1;
int analogValue0 = 0;     //Identify variables for analog inputs
int analogValue1 = 0;
int analogdifference = 0;
int threshold = 100; //The threshold value calculated to stop the motor

int reading;
int state;
int previous = LOW;
```

```
int count = 0;
int numberstops = 250;
int time = 0;         //The last time the motor reversed

//Amount of time to wait to get rid of the jitters when the motor reverses
int debounce = 400;

❶ void setup() {  //This is the setup routine
//Initializes pins as input or output
  pinMode(analog0, INPUT);
  pinMode(analog1, INPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);

  Serial.begin(9600); //Was used in setting up the parameters
}

❷ void loop() {  //This begins the processing section
    //Enter an endless do-nothing loop after the counter reaches the limit
       while(count > numberstops) {
         digitalWrite(ledPin1, LOW);
         digitalWrite(ledPin2, LOW);
       }

  analogValue0 = (analogRead(analog0)); //Read the analog values
  analogValue1 = (analogRead(analog1));

❸ //Setting up the analog difference
  analogdifference = analogValue1 - analogValue0; //This is the voltage drop
//analogValue1 will be greater than analogValue0

 //These were added to view what was happening on the serial monitor
 Serial.print("count =    ");
 Serial.println(count);
 Serial.print("analogdifference =      ");
 Serial.println(analogdifference);
 Serial.println();
 Serial.print("numberstops =     ");
 Serial.println(numberstops);

//This comparator looks at the difference or drop across the resistor
❹ if(analogdifference > threshold) {
    reading = HIGH;
  }
  else {
    reading = LOW;
  }

   //Toggles the output and includes the debounce
❺ if(reading == HIGH && previous == LOW && millis() - time > debounce) {
    if(state == HIGH) {
      state = LOW;
    }
    else {
      state = HIGH;
```

```
    }
      //Increments the counter each time the motor reverses
❻     count++;
    time = millis();
  }

  //Writes the state to the output pins that drive the H-Bridge
  digitalWrite(ledPin1, state);
  digitalWrite(ledPin2, !state);

  previous = reading;

}
```

This sketch sets up human-understandable aliases for the pins the project uses and adds convenient constants and variables for referencing analog inputs and other key values. After the sketch defines and initializes the input and output pins at ❶, it starts the main loop at ❷.

Inside the main loop, the sketch finds the voltage drop across the resistor in terms of analog steps ❸. At ❹, the sketch determines whether the reading was high or low. Threshold values from 100 to 120 work reliably for the 6V, 20 RPM motor I used, but you may need to experiment to find the right value for your motor. See "Determining the Reversal Threshold" on page 22 for more on how to estimate the threshold value. The reading at ❺ dictates whether to reverse the motor.

THE DROPPING RESISTOR IS KEY TO SENSING CURRENT

I've tried this reversing circuit with several similar motors, and I've only ever needed to make a slight adjustment to the threshold value in the sketch. But for a motor with extremely high or low current drain, you may need to anticipate a much different value for `analogdifference` and/or use a different dropping resistor, which was `R1` in the schematic. You might need to reduce the value of the dropping resistor to something like 2.2 ohm, which then requires a reduction in the value you compare `analogdifference` to.

For most small motors, the lower the value of the dropping resistor—which is usually between 1 and 10 ohm—the better, as the analog difference tends to be more stable. For other motors, experiment to find the resistor value that works best.

When the sketch checks `reading` to see whether the motor needs reversing, it also uses the `debounce` value to assure that a high reading wasn't caused by electrical noise created by the motor's commutator or brushes during a legitimate reversal. I set `debounce` to 400, but you may have to adjust that for different motors. For larger motors specifically, this may need to be set a little higher.

This sketch also includes a few functions that aren't strictly necessary to reversing the motor but are helpful when using the motor as a PCB agitator. These aspects of the project may appeal to you in other applications, too, so let's look at them in more detail.

One of the things that I added was a counter to track the number of times that the motor reversed. In the sketch, the count increment appears at ❻ as `count++`. In the project, when a certain value of `count` is reached, the motor stops (if `count = numberstops`). If you wanted to set off an alarm, such as an audible noisemaker, to tell you it's finished, that can easily be accomplished by adding a line to write to one of the digital outputs. I set a maximum `count` value in the sketch, using `numberstops = 250`, so the motor will reverse 250 times and then stop. That provides a little more than 15 minutes of etching time with the motor I've selected running at 5V, which should be enough to etch most circuit boards.

The `stop` function is just a do-nothing loop: when the maximum count is reached, the sketch enters the `while` loop at the beginning, stopping the agitation. This basically stalls the processor, and you have to hit the power switch to restart, or reset, the agitator. The placement of this loop near the beginning of the software is just a reminder that it's there.

MOD: ADJUSTABLE STOP AMOUNT

If setting a fixed stop maximum in a sketch doesn't leave you satisfied, try connecting a potentiometer between power and ground with the adjust pin, which is usually the center pin on the potentiometer, to the A2 input pin of the Arduino. Then, set `numberstops` equal to the value of A2, which should range from 0 to 1,023, depending on the position of the potentiometer wiper.

Here's how the sketch would differ. First, change

```
numberstops=250
```

to

```
numberstops = setNumber;
```

Then, add the following:

```
int setNumber;
int analogPin2 = A2;
int analogValue2;
setNumber = analogRead (analogPin2);
```

Because the timing is relative, you could use a 270-degree rotation linear potentiometer and make some rough markings on the enclosure to indicate the number of counts.

The thinking behind the count, optional alarm, and stop capabilities is that a reminder to check on your board is helpful. If the board has completed etching, continued agitation would speed undercutting of the traces, which is not a good thing because it weakens (and can break!) small copper traces. On the other hand, if it fails to etch in a reasonable time, you might need to refresh the etchant.

The Shield

For this project, I recommend making a small PCB *shield*, which is basically a host board designed to plug into the Arduino Nano. With a shield, your motor reversal project can remain compact, and you can design and build it with a minimum of effort.

PCB Layout

You could just solder the parts for your project directly to a piece of perforated project board, but I believe creating and populating the shield takes less time than putting the parts on a perforated board and wiring them by hand. You'll also gain invaluable experience by preparing, etching, drilling, and assembling your own PCB. And in the end, some projects are complex enough that wiring by hand just won't be an attractive option. (See Figure 5-12 on page 69 for an example.)

To make my printed circuit layouts, I use a free software program called ExpressPCB. If you've never laid out a PCB before, check out "Making Your Own PCBs" on page XX to learn how to use ExpressPCB. Figure 2-10 shows my layout of the PCB.

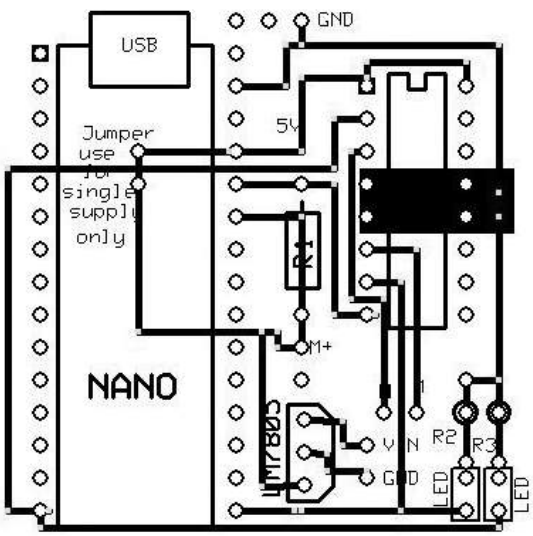


Figure 2-10: This is the actual PCB pattern I used in the project. The Arduino Nano can be soldered directly to the board or can plug in if you use header connectors.

If you don't want to lay out your own PCB but still want to make the board, download the *Reverse.pcb* file from *https://www.nostarch.com/arduinoplayground* and follow the directions in "Making Your Own PCBs" on page X. When you've made your PCB, just solder all the components to it in the right places, and you'll be done with the shield.

Shield Design Notes

If you lay out your own shield, there are a few design factors you should definitely keep in mind.

Analog Inputs

Be certain to connect the A1 and A0 inputs to the correct sides of resistor R1, according to the schematic in Figure 2-3. A1 should attach to the power supply side and A0 to the H-bridge side. In the sketch, to compare the analog values, we take the difference as `analogdifference = analogValue1 – analogValue0`, with `analogValue1` as the input at the high end of the resistor. In this case, `analogValue0` is A0, and `analogValue1` is A1.

Grounding and Heat Sink

Pins 4, 5, 12, and 13 are ground on the H-bridge, and they are also a heat sink to keep the chip from overheating. A small area on the proposed shield is included to increase the heat sink area. If you're using a relatively small motor—such as the 6V, 20 mA unit—no more heat sinking is required. If you're using a much larger motor or driving a heavy load, consider using the second side of the PCB as a heat sink.

The Voltage Regulator

This project uses its own 5V regulator to supply power to the Nano. A 9V, 200 mA plug-in wall adapter is connected to the voltage regulator LM7805 on the shield, which reduces the voltage from about 9V to 5V. An external regulator is included so a more powerful regulator than the one built into the Nano can be used. Make sure to connect the pins of the regulator correctly. Figure 2-11 shows the pinout of the regulator.

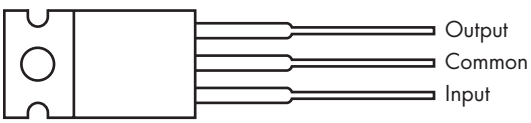


Figure 2-11: Pinout of LM7805 5V regulator

You could feed a 7.5V DC or 9V DC wall supply directly to the VIN pin of the Nano and use the onboard regulator, which worked with my motor. But if you use a larger motor—or higher-current LEDs—it might tax the onboard regulator and could conceivably burn it out.

The higher the voltage of the power supply, the more work the regulator has to do to bring it down to 5V. Overtaxing the regulator could cause

it to heat up and fail. For example, feeding the regulator 12V is probably at the high end for 5V regulation. A 9V input is better, and a 7.5V input is better yet. If the regulator chip gets warm, add a heat sink to the tab. A small piece of aluminum is often sufficient, but a regular heat sink can be used (see "The Mini Voltage Regulator" on page X). And while it's good to have the supply voltage as close to the output voltage as possible, remember that the regulator needs at least 1V above the regulated output to work, so it must be fed with at least 6V, which is a 5V-regulated output plus 1V. Input voltages above 12V are feasible, too, but just be sure not to exceed the limits of the device.

MOD: USING A HIGHER VOLTAGE

If you use a higher-voltage motor for this project, it will turn faster, have more torque, and so on. But you *can't* simply connect the higher voltage to the high end of the dropping resistor connected to pin 8 of the H-bridge. That would cause the voltage between both A0 and A1 and ground to exceed 5V, which is hazardous to the health of the ATmega328 microcontroller on the Arduino. (This is the only time that the voltage referenced to ground is important.) Thus, a modification is required. Look at R1 in the schematic in Figure 2-12. The supply first goes to resistor R2; R2 joins with resistor R3, which goes to ground.

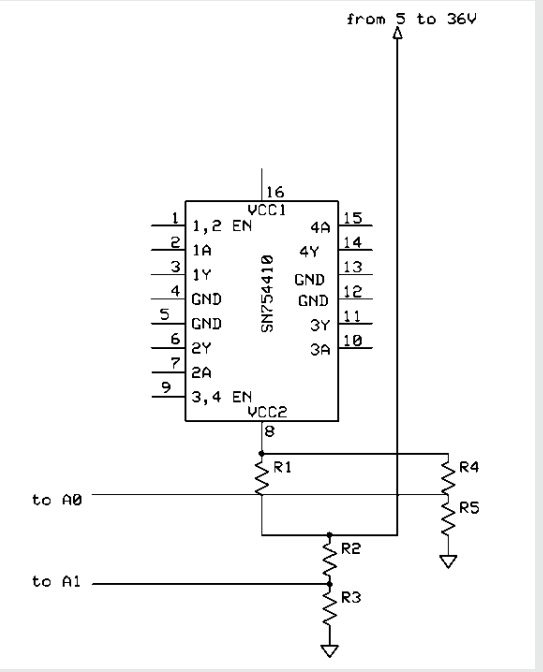


Figure 2-12: If you elect to use a higher voltage and drive a faster motor, you will have to modify the circuit by adding voltage dividers in front of both the A0 and A1 inputs.

(continued)

To avoid damage to the Nano processor, you will want to keep the voltage that appears at that joining point under 5V, referenced to ground. The easiest way to do this is to use a voltage divider. Two resistor pairs divide the higher voltage: the first pair is R2 and R3; the second is R4 and R5. The value of these resistors should be such that the output at the joining of each pair—R1 and R2, and R4 and R5—is somewhat less than 5V for whatever value of input voltage you use.

Use this formula:

$$V_{out} = V_{in} \times \frac{R2}{R1 + R2}$$

and the schematic in Figure 2-13 to determine the values of the resistors to use in a voltage-divider circuit.

For example, if you start with 9V and arbitrarily select a 10-kilohm resistor in series, you would have to shunt it with a 12.5-kilohm resistor to ground, according to the calculator. The closest resistor I had was 12 kilohm, and it worked fine. If you can't find a standard resistor to fit your needs, you can also combine two standard values in parallel to achieve the value you want with this formula:

$$R_{total} = \frac{R1 \quad R2}{R1 + R2}$$

If you don't want to do the algebra yourself, you could use one of the convenient online voltage-divider calculators such as *http://www.sengpielaudio.com/calculator-paralresist.htm* or *http://www.raltron.com/cust/tools/voltage_divider.asp*. SparkFun also has an excellent tutorial on voltage dividing, with a calculator of its own: *http://learn.sparkfun.com/tutorials/voltage-dividers/*.

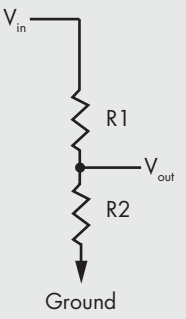


Figure 2-13: A basic voltage divider. To find the resistors you should use, plug the values from your own divider into the formula as if your divider were this circuit.

Directional LEDs

Of course, what Arduino project would be complete without blinking LEDs? As you'll see in the schematic and on the shield PCB, I included two LEDs: a red one for clockwise rotation and a green one for counterclockwise rotation. But which direction belongs to which LED is your choice: simply reverse the motor leads to change the LED status.

Construction

For this project, you'll use the motor-reverse technique to create an agitator that accelerates the etching of PCBs. To do this, you'll suspend a PCB from an Arduino-driven motor over etching solution, as shown in Figure 2-1. A small box will contain the Arduino Nano, the shield, the motor with limit wires, direction LEDs, a power switch, and the power jack.

After assembling the box, you just have to mount it somewhere above your etching setup and attach the reverser, either directly to the PCB or to a tray. I clamped my box to a cabinet door above my workspace, with a place for the etching vessel below (see Figure 2-14). The entire system can be assembled and disassembled quickly.

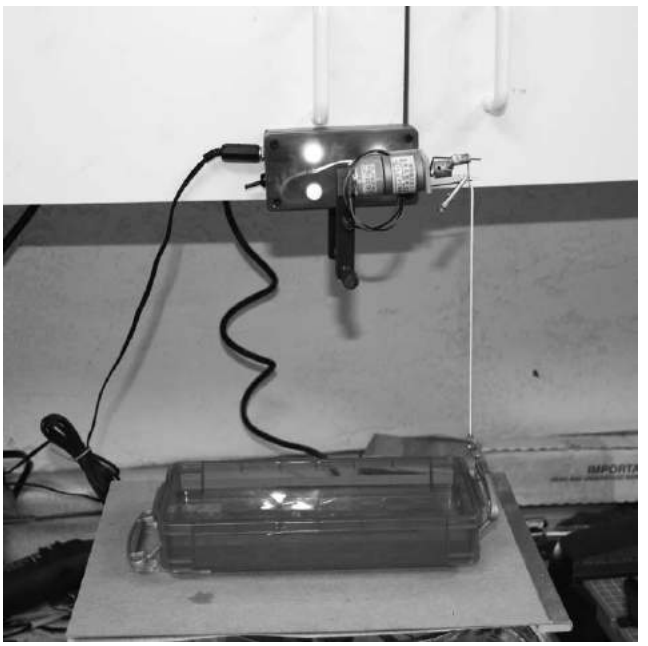


Figure 2-14: For larger PCBs, try etching in a tray for a more conventional approach. Just attach the motor reverser to your tray to agitate the board rather than using the reverser to dip the board in and out of the solution.

Construction of the rest of this project takes a little bit of patience and perhaps some ingenuity in scavenging some of the parts required. You will need a couple of M3 screws to mount the motor to the motor plate—in this case, a small aluminum L bracket—and some limit wires, preferably made of 0.039 piano or spring wire. You'll also need a small block of scrap brass or aluminum—round or rectangular, doesn't matter—to attach to

the motor shaft and crank, a long 4-40 or 6-32 screw to act as the crank, and an M3 standoff and solder lug to attach the agitator line to the crank. Figure 2-15 shows the nearly-finished, unmounted product.

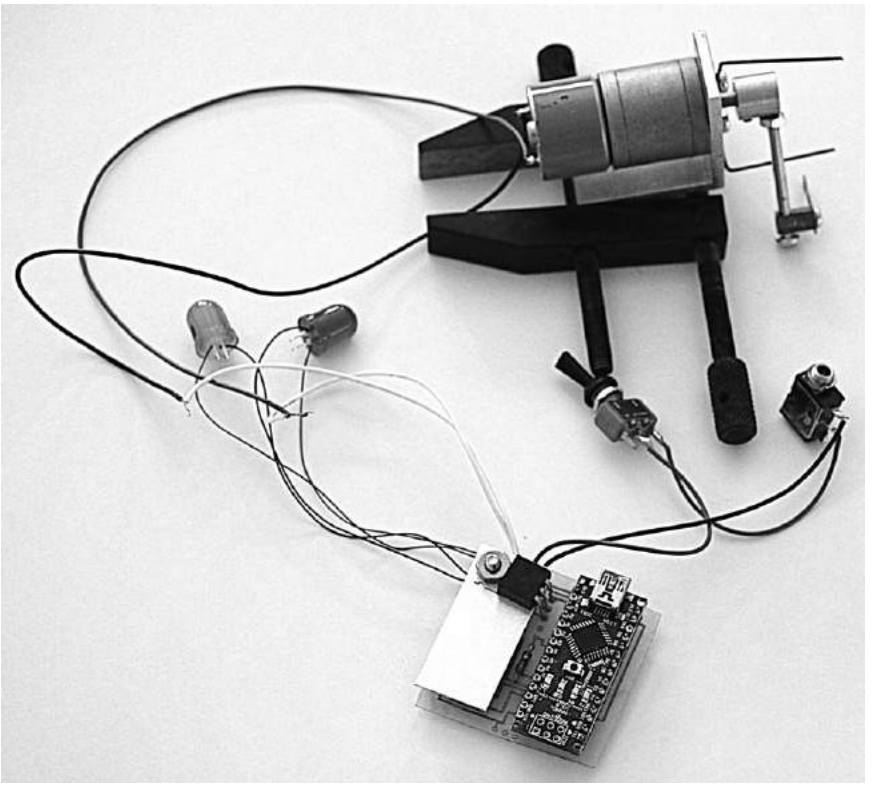


Figure 2-15: Wire up your components and lay them out for a final test before you put them in a box. For the test, I held the motor in a clamp so the crank was free to move. The regulator heat sink obscures much of the shield.

The Limit Wires

The limit wires will create resistance to the motor's rotation by essentially bumping into the motor crank. The point in the rotation where they strike the crank is the limit of rotation. When the crank runs up against the limit wire, the wires prevent the motor from turning and initiate the reversal.

I recommend piano or spring wire to provide a little spring as the crank hits it at the extent of rotation. Use a pair of needle-nose pliers to bend two pieces of the limit wire into shape (see Figure 2-16). These wires will fit on the motor mount screws outside of the motor mounting bracket. You can change the limit of rotation by loosening the screw and rotating the wire.

Figure 2-16: This is how the limit pins are formed. A good pair of needle-nose pliers does the trick.

The Crank Bushing

The crank bushing is simply what transfers the rotation of the motor to the crank. Figure 2-17 details the construction of the bushing, the standoff, and the solder lug.

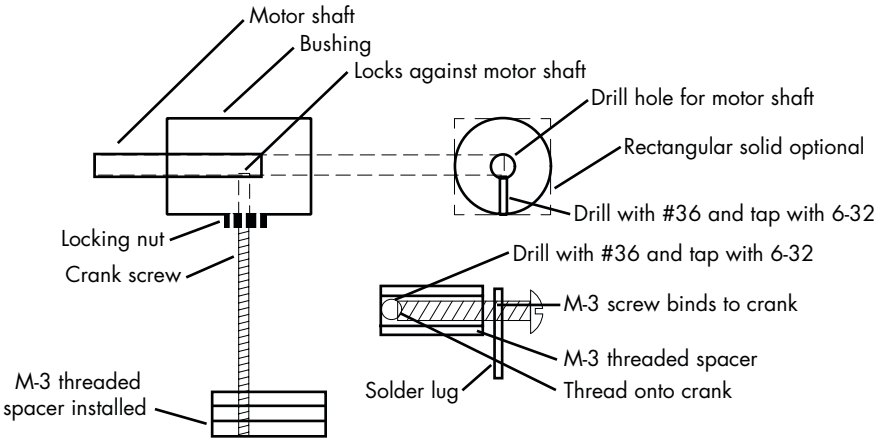


Figure 2-17: The detail of the drive mechanism that transfers the rotation of the motor to the lifting motion of the agitator

While there can be a number of different variations in your approach to assembling this part of the project, here's the sequence I used to put it together:

1. Drill a hole for the motor shaft through the center of the bushing, which can be a small piece of brass or aluminum round stock about 0.5 inches in diameter and 0.75 inches long. A rectangular piece will work just as well. Use a drill that is as close to the size of the motor shaft as possible. For example, if your motor shaft is 0.157 inches in diameter like the one I used, then a 11/64-inch drill bit is close enough. It isn't important to get the hole exactly on center—just close.

2. In the bushing, perpendicular to the motor shaft hole, use a #36 drill to drill a hole. Then, tap the hole you drilled so a long 6-32 screw can serve double duty as a setscrew and crank. You can also use a separate setscrew to move the crank farther from the motor, as I did in Figure 2-18.

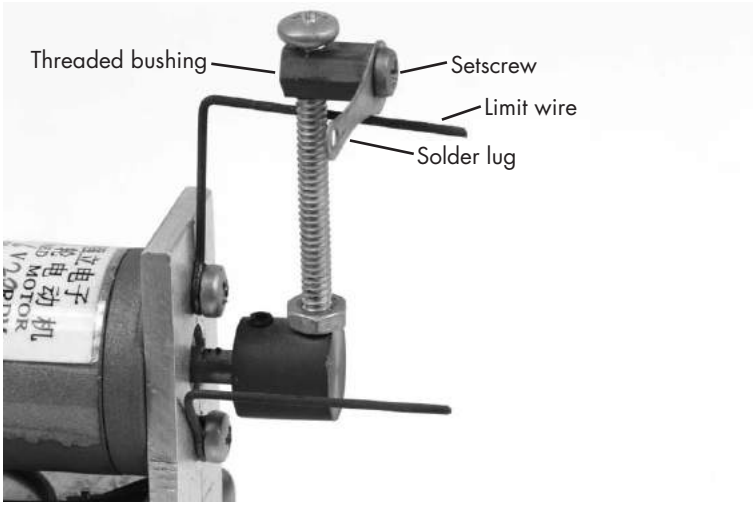


Figure 2-18: A photograph detailing the head of the crank. Note the solder lug used to hold the wire and the alligator stop clip on the left side.

3. Thread the crank screw into the bushing so it bears tightly against the motor shaft, and use a locking nut to hold the screw in place (see Figure 2-18).
4. At the end of the crank, you are ultimately going to attach the line that will pull the PCB in and out of the etchant. This fitting can be just a nut, or even an alligator clip, attached to the crank. However, in the detail, I used an M3 hex female-female standoff that was 7 mm long. I drilled clean through the standoff to one side, starting on one of the flat surfaces with the same #36 drill. I then tapped the hole with the 6-32 tap and threaded it onto the crank.
5. Take an M3 × 0.5 mm machine screw and put it through the solder lug (see Figure 2-19 for the lug itself and Figure 2-18 for the lug in place). Screw it into the standoff all the way so it binds on the crank screw.

Figure 2-19: The solder lug used to hold the wire that holds the etching board. If you can't purchase something similar, you can easily make one with a piece of scrap metal or plastic.

My local Ace Hardware store had all of the accessories I needed, with the exception of the M3 standoff, which I got from eBay. You should be able to find the same items at Home Depot or Lowe's.

Packaging

The shield and Nano fit in a standard plastic box (see Figure 2-20). Drill holes in the enclosure for the 3.5 mm power jack, the SPST switch that serves as a power switch and reset, the indicator LEDs, and the motor wires.

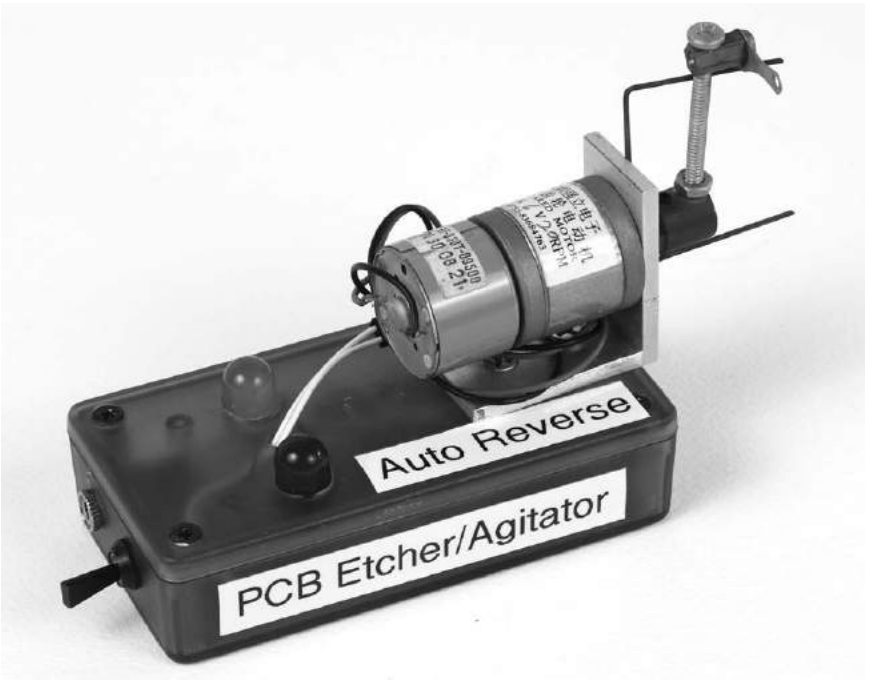


Figure 2-20: Completed box with motor, limit wires, direction LEDs, power switch (reset), and power jack. The LEDs light up, with one for each direction. When the motor pauses in one direction, both LEDs turn on; when it pauses in the other direction, both LEDs turn off.

Most 3.5 mm jacks use approximately a 1/4-inch hole, which is the same sized hole as the switch. If you want a tight fit, 15/64 inches is closer. Whether you use a 5 mm or 10 mm LED will dictate the size of the holes required for those. It's been my experience that different brands tend to have slightly different diameters, so you might want to try a smaller drill first and test whether the LED fits. The arbitrary English-sized drill bits for the 5 mm and 10 mm LEDs are 3/8 inches and 3/16 inches, respectively. If you have a set of tapered reamers, you can start with a smaller hole and ream it out to make a tight fit for the LEDs (see "Tools" on page XX).

Mount the motor on a small piece of aluminum angle, readily available at most hardware stores. I purchased a 1-inch section of 1.5 × 1.5–inch aluminum angle and cut it down to size with a hacksaw. If you're using the motor I use, you can copy the template in Figure 2-21 or download and print it from *https://www.nostarch.com/arduinoplayground/*, cut it out, tape it to the aluminum angle bracket, and carefully mark the hole positions on the

bracket with a center punch or nail. Now, drill the holes—1/8 inches for the motor mount and 5/16 inches for the center hole. If you use a different motor, you will have to measure and mark out the mounting holes.

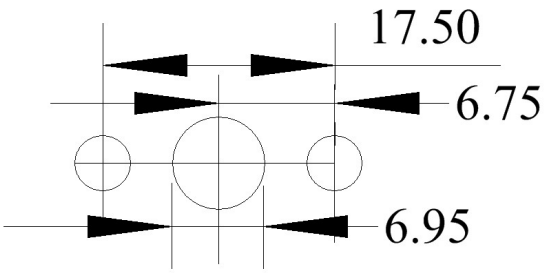


Figure 2-21: Template for motor mount

Just use some double-sided foam tape to secure the shield to the enclosure if you think you'll want to use it in another project. Otherwise, attach it to the inside with standoffs and screws in any size you like.

The Etching Process

There are a number of techniques for making PCBs. The most common is a subtractive approach, which involves starting with a copper clad board, or a copper foil bonded to an electrically insulating substrate, from which the copper is selectively removed to leave a pattern on the board. While the copper can be mechanically milled off, the most common approach is to selectively etch the pattern on the board chemically.

In the chemical etching process, the circuit pattern is printed on the blank board with a chemical resist so that the copper is removed by the etchant in the areas not treated with the resist. The etchant is a chemically active material that attacks the untreated copper on the clad board, leaving you with only the copper you need for your circuit. I describe how to etch circuits step-by-step in Chapter 0, and this project makes that process easier.

Our goal is to suspend an unetched circuit board over the etchant in the vessel and keep it in the etchant for the maximum time as the agitator goes up and down, resulting in a laminar flow of etchant across the surface of the circuit board. I suggest using a nylon cable tie to hold the circuit board during the etching process, as nylon is relatively impervious to the etchant. You could attach the tie, in turn, to the motor shaft with an alligator clip so the board is easy to remove (see Figure 2-22).

Figure 2-22: This Arduino-based etcher-agitator etches a board. The etchant should turn emerald as the copper is etched. The board is held by a wire tie that is attached to a wire by an alligator clip. The wire goes through a hole on the crank and is held in place with another alligator clip. One of the LEDs is lit.

I used a 250 mL beaker as an etching vessel. For very small boards, this works extremely well. For larger boards, I recommend a large measuring cup, such as a 2 qt Pyrex cup. A 600 mL beaker works for intermediate-sized boards. For even larger boards, you can use a tray, as illustrated in Figure 2-14.

The switch and power input are located on the left-hand side of the box. To hold the board being etched, I suspended a wire through the solder lug and attached that wire to the board with a small alligator clip. On the back of the lug, you can either tie a small knot in the wire or attach a clip of some sort to make sure the wire doesn't fall through the lug and into the acid. In my setup, a clamp (behind the motor in the photo) holds the box to an overhanging door.

Note that the etching vessel is sitting on a hot plate. Though etching will occur at room temperature, it's accelerated somewhat by heating. Be careful not to get the etchant too hot: if you set the hot plate on low to keep the liquid at about 100–120 degrees, it will speed etching without softening the resist.

3

THE REGULATED POWER SUPPLY

Whether you use a standard bench power supply or run your Arduino off the USB port of your computer, sooner or later you're going to need a stand-alone, regulated power supply capable of providing a variable voltage. This project shows you how to make exactly that, using only a handful of inexpensive parts. A variable power supply is one of the most frequently used tools in many workshops. This one is easy and fun to build, and you will find that you end up using it over and over again.

When set for 5V or 3.3V, the Regulated Power Supply can power most Arduino projects with ease. You can also use it to power some ancillary piece of equipment, to vary a particular voltage in a system while the main power is fixed, or simply to test a lamp circuit, LED, motor, or other device.

The circuit uses the extremely versatile LM317 regulator chip. If you ever find yourself in the need of a precision voltage regulator with some unusual demands, look up the LM317 on the web. The JavaScript Electronic Notebook has a particularly good article titled "LM 317 Voltage Regulator Designer" by Martin E. Meserve, which can be found at *http://www.k7mem.com/Electronic_Notebook/power_supplies/lm317.html*.

Parts List

The Regulated Power Supply is capable of providing an adjustable voltage from 1.25V to about 12V at up to 1.5A, depending on the fundamental power you use. It uses the LM317 single-chip voltage regulator to set the voltage. To build it, you will need the following parts:

- One Arduino Pro Mini (or clone)
- One LM317 voltage regulator IC
- One LM7805 5V voltage regulator IC
- Two 2.2 ohm 5 W resistors
- Three 10 kilohm 1/8 W, 1% tolerance resistors
- Three 6.8 kilohm 1/8 W, 1% tolerance resistors
- One 68 µF tantalum capacitor
- Three 0.1 µF ceramic capacitors
- One 1 µF tantalum capacitor
- One 16×2 LCD display
- One LCD-I2C adapter board
- Four 4-40 screws
- Eight 4-40 nuts
- One 5 mm LED (for power indicator)
- One SPST switch
- One 470 ohm resistor
- One 10 kilohm potentiometer
- One Hammond panel/case (#1456CE3WHBU)
- Two banana plug jacks
- One 3.5 mm jack
- One 12V 2A AC adapter
- One power adapter jack
- One PCB/shield
- Six 1 × 4 headers (Pololu, #1014)
- Four 1 × 4 housings (Pololu, #1903)
- Four 1 × 2 headers (Pololu, #1012)

- One heavy-duty TO-220 heat sink (available from many suppliers, including Amazon and Futurlec)
- One medium-duty TO-220 heat sink (available from many suppliers, including Amazon and Futurlec)
- Four male crimp connectors (Pololu, #1931; it's a good idea to get some spares of these)
- Four female crimp connectors (Pololu, #1930; it's a good idea to get some spares of these)
- 30-gauge hookup wire
- Solder
- One knob to cover the potentiometer shaft
- Double-sided foam tape

A note on heat-sink selection: there are a wide variety of heat sinks available. The one pictured in Figure 3-2 is the Futurlec TO220ST, which works okay but runs pretty warm as you approach the 1A range. A larger one that will still fit in an enclosure may be better. Futurlec TO220SMAL, the heat sink for the LM7805, is sufficient for the job.

Required Tools

- Soldering iron
- Drill
- Drill bits: 3/8 and 1/4 inches
- Hacksaw or keyhole saw (nibbler or other)
- Philips head screwdriver
- (Optional) Tapered reamer set
- (Optional) Crimping tool

Downloads

You will find the following files in this book's online resources to help you complete this project:

- Templates: *Power Supply Front.DXF, PS front bottom.DXF*
- Sketch: *vol reg 5-30.INO*
- Shield file: *Voltage Regulator.PCB*

WHAT THE REGULATED POWER SUPPLY IS AND ISN'T

The power supply in this project is not intended to replace a regular bench power supply. It does not provide any current limiting and is rated up to 1.5A, due to the current capacity of the LM317. However, for a wide variety of applications—including all of the projects in this book—it works very well. If you're new to Arduino, it will provide a solid power supply and save you a lot of batteries, if that's how you've been powering your projects. If you already have a full-sized bench supply, this project will be indispensable as a second supply. The Regulated Power Supply can be used for several projects in this book that require a secondary power supply.

I have used the Regulated Power Supply in other applications, and at the end of this project, I'll illustrate a more simplified version that can be used as a remote supply. There are times when you just don't have the proper voltage supply for a project, and this build fits the bill.

A Flexible Voltage Regulator Circuit

The basic LM317 regulator circuit in Figure 3-1 is the heart of this voltage regulator. Though it is relatively rudimentary, the chip's simplicity belies what a powerful and versatile tool it can be.

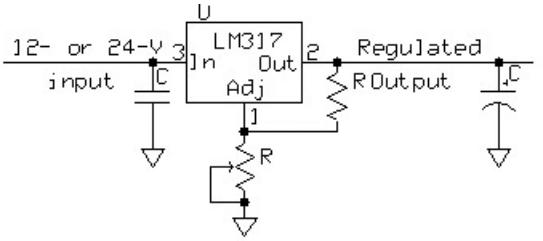


Figure 3-1: This is the schematic diagram for the regulator component of the Regulated Power Supply. The complete schematic with the display is shown in Figure 3-3.

I have used some variation of this circuit for many applications, from a stand-alone variable supply to an integrated part of a larger system, always with good results.

In the shop, I sometimes use kind of a "hair-wired" version, shown in Figure 3-2 (left), for testing LEDs and controlling things like motor speed

and lamp intensity. I've also used a breadboard version, shown in Figure 3-2 (right), to implement the regulator circuit. In both cases, I used a trimmer potentiometer, which requires an alignment tool or screwdriver to make adjustments.

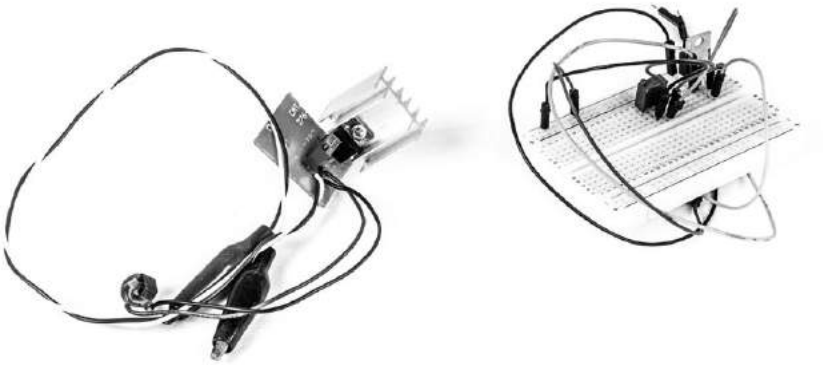
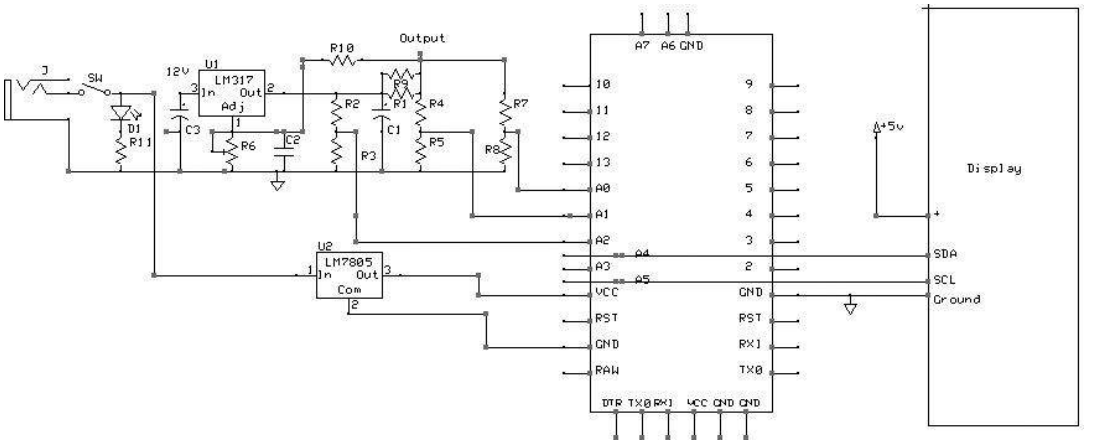


Figure 3-2: The "hair-wired" version of the voltage regulator circuit with a heat sink screwed to the LM317 (left) and a breadboard version of the regulator (right). I used this in an application with minimal power requirements and thus did not include the heat sink.

While those regulator circuits worked, one reason for using the more refined Regulated Power Supply format in Figure 3-1 was to eliminate the awkward trimmer potentiometer and instead use a standard 270 degree potentiometer and knob so that I could make adjustments quickly, easily, and repeatedly. But the main reason I built it was to have a secondary variable-voltage power supply with digital readout readily available on the bench.

The Schematic

While I wanted the Regulated Power Supply to be relatively robust, I didn't want it to be overly complex or hard to build. The hair-wired and breadboard versions did well in temporary or emergency applications when used with a digital multimeter (DMM), but I sought to build something more permanent that would have its own voltage and current readout, and that would stay on the workbench or sit on my desk as a regular addition to the tool set. Figure 3-3 shows the full schematic for the Regulated Power Supply.



Arduino Pro Mini
16-2 LCD/I2C display
U1: LM317
U2: LM7805
R1: 1.2 ohm 5 W resistor (two 2.2 ohm resistors in parallel)
R2, R4, R7: 10 kilohm 1/8 W resistor
R3, R5, R8: 6.8 kilohm 1/8 W resistor
R6: 470 ohm 1/8 W resistor
R7: 10 kilohm potentiometer
C1: 68 µF tantalum capacitor
C2: 1 µF tantalum capacitor
C3, C4, C5: 0.1 µF ceramic capacitor
SW: SPST switch
D1: LED
R11: 470 ohm resistor

Figure 3-3: Schematic for the Regulated Power Supply

For reference, Figures 3-4 and 3-5 show the pinouts of the LM317 and the LM7805, respectively.

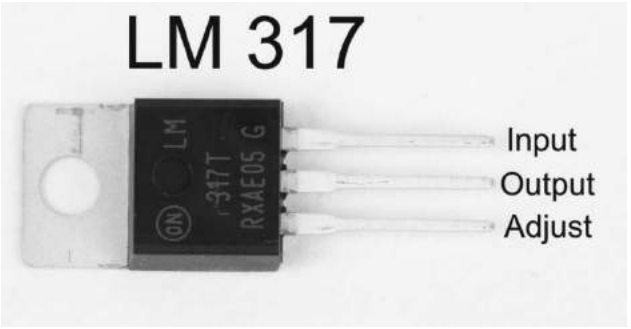


Figure 3-4: Pinout of the LM317 regulator

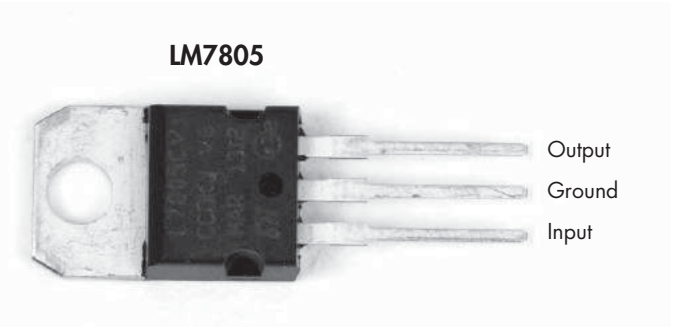


Figure 3-5: Pinout of the LM7805 5V regulator

How the Circuit Works

The circuit for this project is not overly complex. In essence, it measures the voltage at the output of the LM317 regulator using the onboard analog-to-digital converter (ADC) and compares it to an internal reference voltage. The result is sent to the LCD screen. However, the Arduino Pro Mini 5V version can accept a maximum of only 5V at the analog inputs. We therefore use a *voltage divider* to make sure that the voltage at the analog input pin doesn't exceed 5V. (Make sure voltage supplied to the LM317 is no greater than 12V.)

In this case, a voltage divider comprises two resistors connected in series, straddling the output of the LM317 and the ground rail of the breadboard (see the schematic in Figure 3-3). The voltage coming from the LM317 gets divided across the two resistors, R2 and R3. As you will note in the sketch, converting from the divided voltage back to the original levels for the display is simply a matter of reversing the arithmetic.

As shown in the schematic in Figure 3-3, the output of the LM317 connects to both the R2-R3 voltage divider and to resistor R1. Together, resistor R1 in parallel with R9 and the load, or whatever you want to power with the power supply, can be seen as another voltage divider. The R1-R9 voltage divider has a resistance of only 1.1 ohms, so the voltage drop across it is going to be relatively small. According to Ohm's Law (I = E/R) the voltage across R1 and R9 is going to be 1.65V for a maximum current drain of 1.5A (the maximum supported by the regulator IC): 1.5 A = 1.65V/1.1Ω.

This means that when the LM317 provides about 12V and the load draws 1.5A, there will be a 1.65V voltage drop across R1 and R9, leaving 10.45V at the power supply's output.

Looking at the circuit in Figure 3-3, we are comparing the voltage at analog inputs A0 and A2 of the Pro Mini. If you use the same values for the voltage divider for A0 and A1, you can eliminate one of the sets of resistors and simply connect A0 to A1.

VOLTAGE DIVIDER RESISTOR VALUES

The values of the resistors needed to achieve a certain voltage are determined using this formula:

$$V_{out} = V_{in} \times \frac{R_2}{R_1 + R_2}$$

A schematic is shown in Figure 3-6.

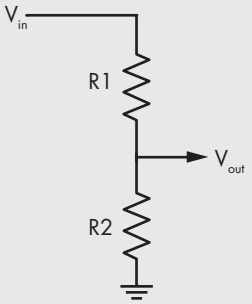


Figure 3-6: A typical voltage divider circuit

You can do the algebra if you'd like, but it's easiest to use an online calculator, such as the one at *http://www.daycounter.com/Calculators/Voltage-Divider-Calculator.phtml*. In this project, the objective is to achieve an output of around 5V. We'll start with a 12V input and a 10 kilohm resistor, represented by R_1 in the formula and marked R2 in the schematic. Fill in the calculator fields with this information, and the formula will give you a resistor value of 7.1 kilohm for R_2 (R3 in the schematic). The closest standard resistor value is 6.8 kilohm, so the project uses that along with the 10 kilohm resistor in its voltage divider.

But why start with a 10 kilohm resistor? The first reason is to avoid drawing too much current. Even if the entire 12V dropped across the 10 kilohm resistor, it would result only in a nominal drain of 1.2mA. Second, I have a lot of 10 kilohm resistors in the parts bin, and I am sure you do, too.

I use three sets of voltage dividers in this circuit. The first looks at the voltage at the output of the regulator, which is ultimately displayed on the LCD. The other two divide the voltage in front of and behind the voltage-dropping resistor so that the amperage can be measured according to the formula I = E/R, where E is the voltage drop across resistors R1and R9 and R is the value of those two resistors combined. Could I have eliminated one set of voltage dividers? Yes, by joining A0 and A1 together. I thought, however, that I might want to change those values at some point to increase the accuracy of the ammeter by bringing the value closer to the Arduino reference voltage, so I did not join them in my version of the project.

The Breadboard

As in all of my Arduino projects, I began with the standard breadboard. To make life easy, I used a standard potentiometer with pins that would fit into the 0.100-inch-spaced breadboard holes. With a little effort, a standard 16 mm rotary potentiometer (R7 in the schematic) with printed circuit board connectors will just about fit into every other hole in a breadboard.

Preparing the Arduino Pro Mini and LCD

The Arduino Pro Mini may or may not come with the male headers attached. If it doesn't, you'll have to solder them yourself. For more on preparing CPU boards, see the appendix on page XX. Make sure the number of header pins in your strip matches the corresponding holes in the Pro Mini; you may have to cut the strip to the proper number of pins if the included strip is too long. Trim two strips of headers to size and place the long ends of the two header strips into a breadboard, spaced so that the Pro Mini board will fit over them. Put the Pro Mini in place, and solder all the header pins. Then, take two header pins (use the surplus from the longer header or purchase these separately), insert them in the A4 and A5 holes in the Pro Mini, and solder. These are the pins used for the LCD display.

Finally, install five header pins on the edge of the board (at the TX0 and RXI end). Some boards come with straight headers, others with the long pins bent at a 90 degree angle. In most of the applications, I have found it easier to work with straight headers. You can use right-angle headers, but it may be more difficult to plug in the connector for programming the board, so I recommend replacing any right-angle pins with straight ones. You also might want to take a 1/2-inch length of #22 wire and solder it to the short end of the RST pin so it sticks up. A female header connector will connect to this during programming.

You will now have to get the LCD/I^2C assembly ready. If you purchased the display and adapter separately, you will have to assemble them. Go to "CPU Assembly," on page XX for instructions. If you purchased the display with the I^2C adapter, it's ready for assembly.

Building the Breadboard

Figure 3-7 shows an overhead view of the finished breadboard before you power it.

Here's the step-by-step guide to putting together the breadboard:

1. Insert wires to connect the two positive rails (red strips) together.
2. Insert wires to connect the two negative rails (blue strips) together.

WARNING *Be careful not to cross the two and connect the positive rails to negative rails. That could cause a short circuit and damage the hardware.*

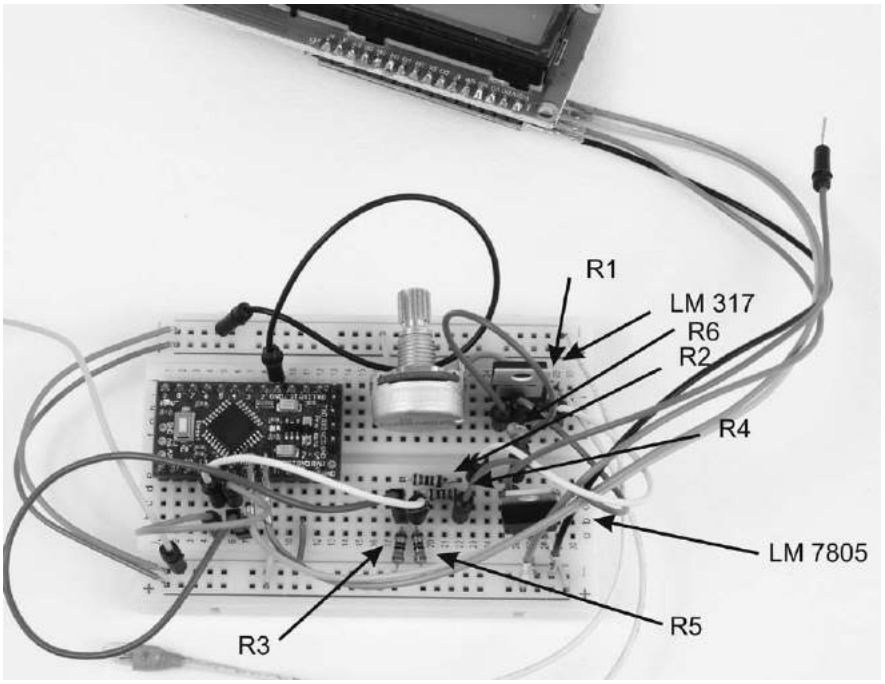


Figure 3-7: The breadboard for the Regulated Power Supply. The capacitors in the schematic—C1, C2, and C3—are not included in the breadboard but should be included in the completed unit.

3. Insert the 10 kilohm rotary potentiometer into the breadboard.
4. Insert the LM317, with or without heat sink attached, near the potentiometer, as shown in Figure 3-7. (See Figure 3-4 for the pinout of the LM317.)
5. With the potentiometer shaft facing you, connect the leftmost pin and center pin of the potentiometer together, and then connect both to the adjustment (ADJ) pin of the LM317.
6. With the potentiometer in the same orientation, connect the rightmost pin to the blue negative rail (ground).
7. Connect a 470 ohm resistor from the output pin to the ADJ pin on the LM317.
8. Connect a 1.2 ohm resistor (R1 in Figure 3-3—I used two 2.2 ohm resistors in parallel) from the output pin of the LM317 to a load of your choosing. For test purposes, I used a 1/8 W resistor and connected a 5V, 30mA incandescent indicator lamp for the load. (You may want to use the actual R1 and R9 resistors that you will use in the finished unit, so you can adjust the sketch before completing the unit.)
9. Connect the input pin of the LM317 to the 12V system input voltage. This is the wire going from the LM317 to the upper alligator clip in Figure 3-7.

10. Connect the blue negative rails to the negative side of the input power (probably a wall plug).
11. Insert the LM7805 into the breadboard, as shown in Figure 3-7. (See Figure 3-5 for the pinout of the LM7805.)
12. Connect the output pin of the LM7805 to the red positive rail of the breadboard.
13. Connect the input pin of the LM317 to the input pin of the LM7805. This is the point at which the input voltage from the power source will be connected.
14. Connect the ground pin of the LM7805 to the blue negative rail of the breadboard.
15. Insert a 6.8 kilohm, 1% tolerance resistor, and connect one side to the blue negative rail. This is resistor R3; the other side will connect to resistor R2. See Figure 3-8 for a top view of the breadboard.
16. Insert a 10 kilohm, 1% tolerance resistor (R2) into the breadboard with one side connected to the LM317 output pin and the other side connected to resistor R3 from Step 15.
17. Connect resistor R1 from Step 7 from the output pin of the LM317 to a blank hole in the breadboard. This row will be the output of the regulator.
18. Connect the voltage divider: first, insert a 10 kilohm, 1% tolerance resistor (R4) into the breadboard. Then, connect one side of resistor R4 to the same row as R1 (you'll have to use a jumper wire) and the other side to an empty row on the breadboard.
19. Insert a 6.8 kilohm, 1% tolerance resistor (R5) into the board with one side connected to the open side of R4 and the other side of R5 connected to the blue negative rail.
20. Insert the Arduino Pro Mini in the breadboard so that it straddles the center break, as shown in Figures 3-7 and 3-8.
21. Use a jumper to connect the VCC terminal of the Pro Mini to the red positive rail.
22. Use a jumper to connect the GND pins of the Arduino Pro Mini to the blue negative rail. (There are at least two to choose from—one is located between RST and D2, and the other is located between RAW and RST on the other side. Take your pick.)
23. Use a jumper wire to connect the joining point of R4 and R5 to pins A1 and A0 on the Arduino Pro Mini.
24. Find the junction point of R2 and R3, and use a jumper to connect that junction point to the A2 terminal on the Arduino Pro Mini.
25. Load the sketch onto the Arduino Pro Mini. (I often remove the Pro Mini from the circuit completely to program it. It's a little less confusing.)
26. Connect the LCD/I^2C display by connecting VCC and GND to the red positive and blue negative strips on the breadboard, respectively.

27. Connect the SDA to analog pin A4 on the Arduino Pro Mini and SCL to analog pin A5.
28. Connect the input of the LM7805 voltage regulator to some pin where the +12V will be attached.
29. Connect the output of the LM7805 to the red positive strip, and connect the ground to the blue negative strip.

Once all of those connections are in place, you're set to go. Upload the sketch and test the circuit.

The Sketch

The Regulated Power Supply sketch is about as simple as I could make it. The only difficulty is that although I use 1% tolerance resistors throughout, I've found some variation in resistance value. So be aware that you may need to make an adjustment to the sketch to accommodate for this.

Here is the sketch:

```
// Regulated Power Supply with volt and current read

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd (0x27, 16, 2); // Check your library for specific LCD
                                     // code both here and in setup.

float low_side_res = A0;
float volt_two;
float volt_three;
float volt_disp;
float low_side_res_2 = A1;
float hi_side_res = A2;
float volt_drop_1;

float amp;
float amp_3;
float amp_4;
float amp_disp;

void setup ()
{
  lcd.init ();
  lcd.backlight();
  // pinMode (amp_one, INPUT);
  // pinMode (amp_two, INPUT);
}
```

```
void loop ()
{
  volt_two = analogRead (low_side_res);
  volt_three = (volt_two*5)/1024.0;
  volt_disp = volt_three*(10000+6800)/6800; // Actual voltage reading
  amp_3 = analogRead (low_side_res_2);
  amp_4 = analogRead (hi_side_res);
  amp = amp_3 - amp_4;
  amp_disp = amp *5/1024*(10+6.8)/6.8/1.22*.9; // Calculation of amperage I=E/R
  // *0.9 = adjustment for random error in ref voltage in pro mini

  lcd.setCursor (1,0);
  lcd.print ("Volt    ");
  // lcd.print (amp);
  lcd.setCursor (12, 0);
  lcd.print (volt_disp);
  lcd.setCursor (1, 1);
  lcd.print ("mA     ");
  lcd. setCursor (11, 1);
  lcd.print (amp_disp*1000,2);

}
```

First, this sketch imports some libraries and sets up the LCD (see "On Writing Code to Set Up LCDs" on page XX). It then defines a series of variables, all floats, to use when setting the voltage, reading from the analog pins, calculating values to display on the LCD, and so on. The `setup()` loop is very short: it has only two lines for initializing the LCD. The main `loop()` reads the battery voltage and current, and performs the necessary calculations to display on the LCD. The `volt_disp` value is the voltage to be displayed on the LCD.

The Shield

While the circuitry is not overly complex, using a shield will simplify many of the connections for driving the LCD and constructing the voltage dividers, and it will make the assembly of the Regulated Power Supply easier than point-to-point wiring. Figure 3-8 shows the shield I designed, though you could also design your own, of course. The PCB file is available at *https://www.nostarch.com/arduinoplayground/*.

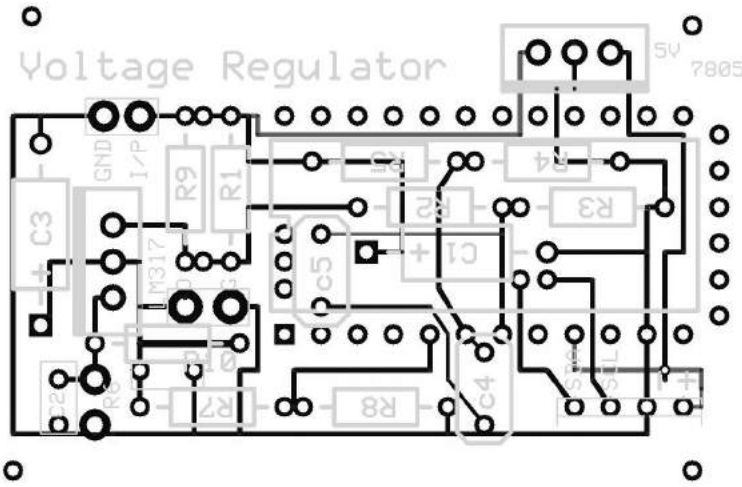


Figure 3-8: The PCB shield used in the Regulated Power Supply. Black is the top layer, dark gray is the bottom layer, and light gray is the silkscreen layer.

While I used two layers to construct the shield, with a little effort and a slightly larger board, the circuitry could be accommodated on a single layer.

The shield doesn't need to be populated in any particular sequence, but some components will be easier to fit before others. I suggest soldering in this order:

1. First, insert 2.2 ohm 5W, 1% resistors R1 and R9 into the PCB. These are voltage-dropping resistors that create the voltage for the ammeter (mA), which provide a total resistance (R1 in the schematic) of 1.1 ohms at 10 W. The resistors are a little longer than the configuration on the board, so you'll have to bend the leads to make them fit. When the Regulated Power Supply is running close to its maximum rating, expect these resistors—and the LM317 itself—to get a little warm.
2. Capacitor C1, a 68 µF tantalum capacitor, will be a tight fit for the holes. To make sure it doesn't interfere with the LM317, install the capacitor first. Then, install the LM317, making sure to leave room for the heat sink. Remember that the heat sink is likely to get pretty warm.
3. Make sure to install the LM317 with the pins correctly oriented according to the pinout in Figure 3-4 and the schematic in Figure 3-3. If you use the provided shield files, the thick line on the LM317 silkscreen corresponds to the metal tab on the IC. If you insert the part the wrong way, the system won't work, and the part could burn out. It would also be a pain to remove.
4. Install the LM7805 in the upper-right section of the PCB, and make sure it matches the pinout in Figure 3-5 and the schematic in Figure 3-3. You can use the heavy line in the silkscreen image of the PCB as a guide.

5. Try both voltage regulator ICs in the shield with the heat sinks installed (at least temporarily) to make sure they can fit without touching any active components. Remember that the heat sinks are active: the heat sink (tab) of the LM7805 is at ground potential, and the heat sink (tab) of the LM317 is at the output potential.
6. Install resistors R2, R3, R4, and R5. It's easier to place those before installing the female headers for the Arduino Pro Mini.
7. Then, solder in C1 and C2, and solder in the wires that will connect the Arduino Pro Mini to potentiometer R6, which you will mount on the chassis. You can leave the wires a little long and trim them when you install the shield in the enclosure. Install capacitors C4 and C5 as indicated in the schematic.
8. Next, solder the female headers that comprise the mount for the Pro Mini and the connector for the LCD display. The LCD connections are the SDA, SCL, –, and + connections in the bottom right of the PCB. I used male stakes and a female-to-female connector cable to connect from the LCD to the shield. (To learn how to make the custom connector, see the section on connectors on page XX.)

For the LCD and Arduino Pro Mini, I usually insert the headers into the board, solder just one pin, and then, with my finger on the top, heat that one pin and push on the connector to make sure it fits flush against the board. For the pins of the Pro Mini, I use only female headers for those that are active—that is, that have copper traces going to them. I also like to place one pin (I usually use a 1 × 4 pin header) right at the last pin on the Pro Mini to make alignment easy. This would translate to pins RAW, GND, RST, and VCC. In addition, I like to place at least two headers diagonally for mechanical stability. This would correspond to pins D8 and D9 on the Pro Mini. The male headers on the Pro Mini for A4 and A5 are located just above pins A2, A3, and VCC on the main row of connections.

Construction

When the Regulated Power Supply is all soldered together, you will need to prepare an enclosure and mount the circuit inside. I selected a nice-looking, powder-coated metal enclosure, approximately 2 1/4 × 3 1/4 × 4 3/4 inches.

Bear in mind, though, that while the case is not delicate, the paint is easy to scratch, so be careful. It's also a little pricey—coming in around $20—but as I will have it on my workbench all the time, I thought it was worth it. Of course, you could also use a different enclosure of your choosing and modify the templates provided with this book accordingly.

Preparing the Enclosure

The front panel of the enclosure is sloped, so you will need to put a piece of scrap wood behind the areas you need to center punch and drill to help hold it in place. Make sure to measure, center punch, and drill holes carefully.

The templates for this project are shown in Figures 3-9 and 3-10, and they can be downloaded from *https://www.nostarch.com/arduinoplayground/*.

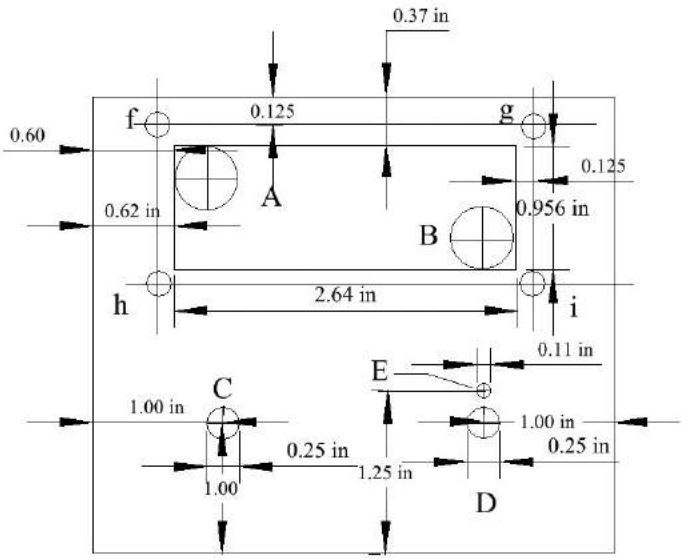


Figure 3-9: Sloping face of the Regulated Power Supply enclosure

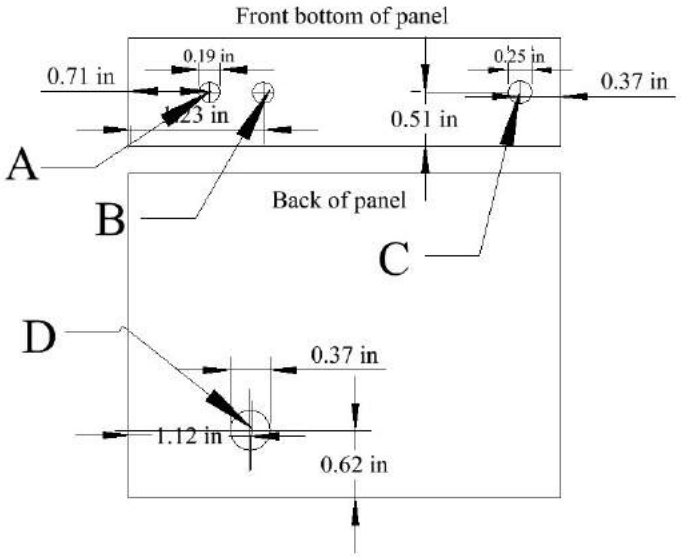


Figure 3-10: Front and bottom of the Regulated Power Supply enclosure

Here is how I suggest you prepare the enclosure:

1. Center punch and drill holes for the potentiometer, on-off switch (1/4 inches), and power indicator LED. See Figure 3-9 for the front panel dimensions.
2. Center punch and drill the hole for the power input jack in the rear of the panel (see Figure 3-9).
3. Drill holes for the output binders and 3.5 mm jack on the front of the case, as shown in Figure 3-10.
4. Carefully measure and mark the cut out for the LCD display, as shown in Figure 3-9. Center punch and drill 1/2-inch holes in the corners of the LCD screen area to help initiate saw cutting. You can eyeball this based on the diagram in Figure 3-9 or download a PDF file of the template. Either trace the image onto your enclosure with carbon paper or simply mark the corners with a center punch and connect the punch marks.
5. Carefully cut out a hole for the LCD display. There are a variety of tools you can use to do this. I first drilled holes A and B and then used a keyhole saw with a fine hacksaw blade (available at local hardware stores) to cut between the holes. Remember that the cutting occurs on the outward thrust, so you needn't keep pressure on the blade on the return stroke. You can clean up the burs with a file.
6. Carefully fit the display into the window, and file where necessary to get a secure fit. The backlight protrudes on one side of the display, so in order to avoid crushing the backlight, you can use nuts as spacers to keep it separated from the panel.
7. Drill holes f, g, h, and i, and fasten the display in place. As you do so, check carefully that the spacer nuts are wide enough (4-40 nuts come in different dimensions), and, if necessary, use two nuts or a nut and a washer to space out for the backlight.

Mounting the Circuit Board

Once you have assembled and tested the shield and mounted the LCD display, install the potentiometer, on-off switch, LED, and power jack. Then it's time to mount the shield in the enclosure. Originally, I drilled four holes in the board so that I could screw it into the enclosure, but I've found that 3M double-sided mounting tape also does the trick. I mounted the entire board, heat sinks and all, with a 1 1/4-inch length of the 3/4-inch wide double-sided adhesive. (The manufacturer claims it will hold 2 pounds.)

The adhesive is relatively aggressive, so before applying it, make sure you plan carefully where you want to mount the board so it will not be in the way of other components. I mounted the board upside down on the top section of the case so all components and connections were on the same platform (see Figure 3-11).

The final step is to connect the on-off switch, LED pilot lamp, LED current-limiting resistor, potentiometer, power input jack, and output connectors to the PCB according to the project schematic. I used #28 hookup wire to tie everything. The two binding posts / banana plug jacks and the 3.5 mm jacks are wired in parallel. Figure 3-11 illustrates how I mounted the shield. Note that I used small wire ties, which are optional, to keep the wires neat.

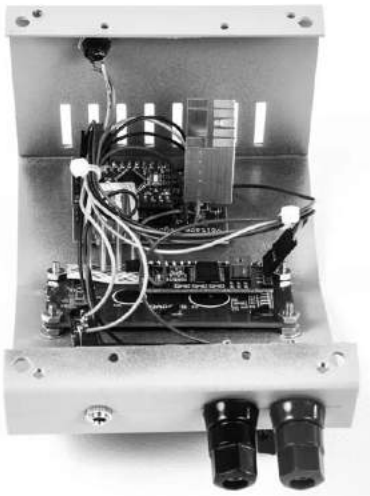


Figure 3-11: The completed (upside-down) assembly with the board, LCD display, output connectors, potentiometer, on-off switch, LED, and output power jack all mounted on the inside of the top of the enclosure.

Before closing up the enclosure, make sure there are no areas that might result in a short circuit. For example, I placed a small piece of insulating tape between the LCD screen and the shield. They might not be touching at the time of assembly but could touch when you close the enclosure. You can apply insulating tape to prevent this kind of short.

After testing for short circuits, all that remains is to put the two halves of the case together. The connections for the output—the binding terminal / banana jack and 3.5 mm jack—are in parallel. The final step is to screw the two pieces together, and you're off and running. I put a pointer knob on the potentiometer even though there are no markings on the case. You can add a dial if you want, but I find the digital readout sufficient.

And, the coup de grâce: remove the protective paper from the adhesive on some rubber feet, and install them on the bottom of the enclosure. Voilà! Figure 3-12 shows the completed unit driving a couple of incandescent panel lamps.

Figure 3-12: The completed Regulated Power Supply

The full Regulated Power Supply project will take up a decent amount of space on your workbench, but you will find the digital readout and banana jacks invaluable. If you're feeling adventurous, however, you can build a smaller version of the same circuit, minus the jacks and LCD, shown in Figure 3-13.

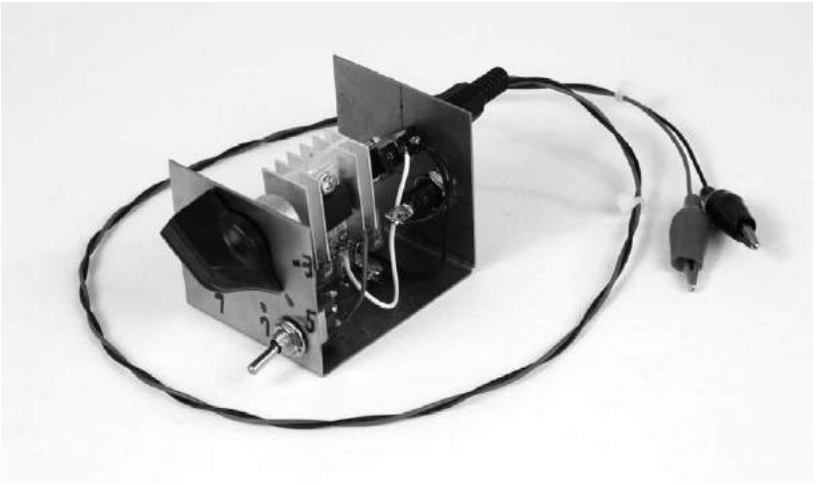


Figure 3-13: This is the quick-and-dirty power supply with the top off. I made it before biting the bullet and making the full-fledged Regulated Power Supply.

You can find instructions on how to build the Mini Regulated Power Supply at *https://www.nostarch.com/arduinoplayground/*.

4

A WATCH WINDER

If you're a collector of automatic, or self-winding, watches, you're probably familiar with watch winders and what they do. But, why have a watch winder in a book on Arduino microcontroller projects? The answer to that will become increasingly clear as we look at the technology in this project. Further, over the course of this project, we'll take a quick look at some automatic watch lore and how these seemingly anachronistic devices have survived and prospered in the digital age. Even if you do not collect such treasured timepieces, this project may just inspire you to start your own collection.

Why a Watch Winder?

Because, as a collector, you own more than a single automatic watch, you might want to think about keeping the watches that aren't currently on your wrist wound. If you read up on mechanical watches and winders, you will find many pros and cons (probably more pros) of using a watch winder. One big pro is that multifunction watches can take a long time to set if they run down. There are also arguments that if a mechanical watch sits in one position and doesn't run, the lubricant tends to migrate to a low point. Regular motion from a watch winder or from being worn keeps the lubricant distributed and in the bearings where it belongs. While many subscribe to this viewpoint, there is no real evidence either way.

There is yet another compelling argument for a watch winder. As a collector, it's nice to be able to display more of your collection than just one-at-a-time on your wrist. Many of the commercial winders available come inside exotic wood cases to show off the watches. But inexpensive winders tend not to be reliable, and the expensive ones are, well, expensive.

I took a chance and bought one of the more economical models and put two of my mechanical watches—a real and a faux Rolex—in it and figured I was done. But after less than six months, the winder failed. I took it apart, and it appeared to be very poorly designed and made. Even if I replaced the failed motor, the rest of the mechanism would probably not be reliable. While using the winder, the faux Rolex did not wind all the way and did not keep good time.

At that point, the question was whether to dig deep in my pockets for the $400 or $500 winder (there are even models that sell well in excess of $1,500, $2,000, or more) that promise reliability or to try to do better. So the gauntlet was, metaphorically, thrown down. The challenge was to design and build a reliable watch winder that would provide both a showcase for my watches and have the flexibility and control over timing that I wanted in a robust mechanical format. Arduino was the obvious choice for controlling the frequency of watch turns, and the mechanics went around that.

As you build your Watch Winder, you will find a lot of room for personalization both in the mechanical construction and the sketch. While a watch winder is a utilitarian device made to keep your watches wound, this version provides an elegant display platform for your timepieces—and it is itself a work of art, a kinetic sculpture. You can see the final result in Figure 4-2.

Because I selected Arduino as the logical timing element, I had to plan the other electronics and software around that. We will revisit the H-bridge circuit from Project 2 to drive the motor in both directions, and we'll use transistors for increased drive for the high-output LEDs. We'll also use a Hall effect sensor to measure the rotation of the watches.

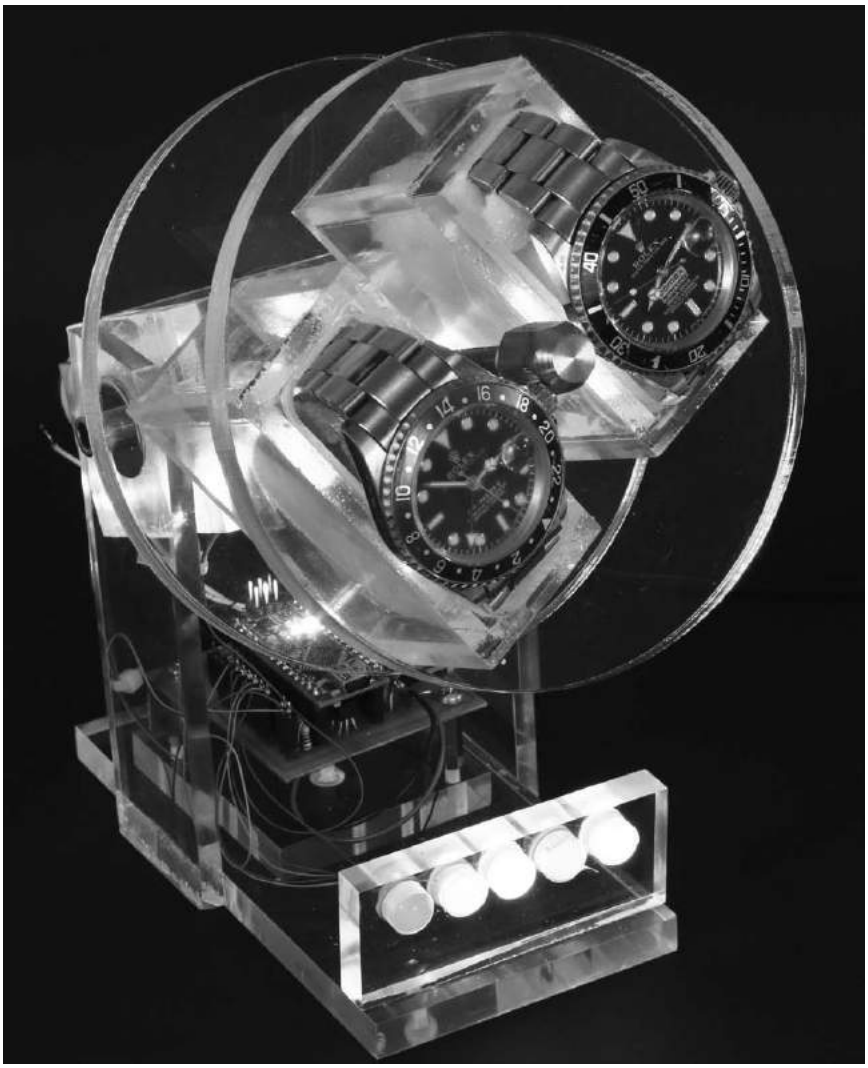


Figure 4-1: The finished Watch Winder. Unfortunately the black and white image doesn't do it justice: the brightly colored LEDs illuminate the device using the acrylic as a light guide to transport the various colored LEDs.

The sketch developed for this project uses functions and arrays to flash the LEDs in repeating patterns. The sketch also instructs the controller to read the state of the Hall effect sensor, which is either zero or one. Knowing this state allows the controller to decide when to wind the watches and to keep count of the number of turns to ensure that the watches don't get over- or underwound.

THE MYSTIQUE OF THE AUTOMATIC WATCH

The automatic watch was invented in the early 1920s and was commercialized several years later. Over the next several years, many improvements were made until it reached the level of sophistication of today's instruments. Automatic watches operate by using a pendulum attached to a ratchet assembly: the ratchet assembly winds the watch's mainspring as the pendulum swings. A built in slip-clutch mechanism prevents overwinding. See Figure 4-1 for a look inside one of these watches.

Figure 4-2: An automatic watch with the back removed, exposing the pendulum and the fulcrum (the screw in the center), which combine with a ratchet assembly to wind the watch's mainspring

Automatic watches from just about all watch manufacturers enjoyed broad success for several decades. However, in the early 1960s, Bulova developed its Accutron tuning-fork electronic watch, and the digital quartz electronic watch from Pulsar followed shortly after.[1]

Despite the influx of electronic watches (and now smart watches), leading makers of mechanical watches have survived—and even prospered—in this age. Today, automatic watches are sold anywhere from under $100 to tens or even hundreds of thousands of dollars.

Why would someone pay a premium for a watch that is not particularly accurate, is heavy, is often bulky, and has to be kept wound when not in use? I'm sure the answer is different for every collector, but I'd guess that they, like me, enjoy the elegance, prestige, sophistication, sense of history, and fine mechanical machinery that can't be achieved with its electronic counterparts—though the iWatch comes close in some respects. And like any collectible, one automatic watch is never enough—which brings us to the watch winder.

---

1. The transition from mechanical to electronic watches has been described as a prime example of Thomas Kuhn's concept of a paradigm shift, which he describes in his 1962 book *The Structure of Scientific Revolution*.

Required Tools

- A drill and set of standard bits
- A set of tapered reamers
- Assorted hand tools (See “Section name” on page XX.)
- Weld-On 4 and Weld-On 16 acrylic bonding fluid
- Assorted sandpaper, including grades from 220 to 600 and grade 1500 for final polish
- Jewelers rouge or other liquid plastic polish
- (Optional) Thread-locking fluid (You can find it in any auto supply store.)
- (Optional) Wire-wrap tool (These can be a little pricey. I bought a spare from OK Industries, but Techni-Tool also offers a relatively good price. Shop around but expect to pay upward of $15. This may well be valuable on other projects.)
- (Optional) Rotary tool (For example, you could get a Dremel with an abrasive cutoff wheel. Some tools are available for under $10.)

Parts List

If you want to build a Watch Winder like the one pictured, you will need several pieces of acrylic and some other hardware, which I detail in this section.

Acrylic

The following acrylic parts can easily be cut from a standard sheet of acrylic. Without the disks, which I recommend you purchase separately, everything can be cut from two 12×12-inch acrylic sheets (one 3/8-inch thick, one 1/4-inch thick). If you prefer, you can find vendors that will laser cut acrylic to your dimensions. (ZLazr, among many others, is equipped to do that.) It will cost a little more than doing it yourself but will make it both cutting and finishing easier.

- Four pieces with dimensions 1/4×2×1 1/2 inches (long sides of the watch basket; can be 3/8 inches)
- Four pieces with dimensions 1/4×1×2 inches (short sides of the watch basket; can be 3/8 inches)
- Two pieces with dimensions 3/8×3×2 inches (bearing holders of bearing box)
- Two pieces with dimensions 3/8×2×1 1/2 inches (mounting side of bearing box)
- One piece with dimensions 1/4×1×2 inches (motor mount)
- Two round pieces, 3/8 inches thick and 5 inches in diameter (watch basket ends)

- Two pieces with dimensions 1/4×1 5/8×5 inches (side supports for stand)
- One piece with dimensions 3/8 x 3×5 1/2 inches (base for stand)
- One piece with dimensions 3/8×3×1 inches (lightbar)
- One piece with dimensions 3/8×1 1/4×1/14 inches (shield mounting)
- Two 3.5 mm standoffs with M/F M3-05 threads (motor mounts)
- Three 1.5 mm standoffs with M/F M3-05 threads (shield mount)

There are several online vendors you could purchase the acrylic for this project from; just search for "acrylic sheet" on Google to find one near you. In the United States, *http://www.ZLazr.com/* seems to be good. At the time of this writing, I talked with the owner personally, and he said he can handle the kind of cutting required for this project with no problem.

Other Hardware and Circuit Components

- One Arduino Nano or clone
- One driveshaft, 8 inches long and 1/4 inches in diameter with 28 threads per inch (I suggest brass because it's easy to work. Amazon sells brass-threaded rods that are 24 inches long.)
- Two ball bearings (R4A-2RS, available from Amazon)
- Six jam nuts, 1/4-inch-28 (Buy these at your local hardware store.)
- Two decorative bolts, 1/4-inch-28, 1 inch long (I used chromed Allen bolts from a hardware store.)
- Ten 649 transistors (Try Digikey; I found them for a cheap price there.)
- One 754410 quad H-bridge (Available from Mouser, Digikey, SparkFun, eBay, and others. I recommend shopping around.)
- Ten 470-ohm resistors (Multiple sources, including Jameco, Digikey, Mouser, import stores, and so on, should sell resistors cheaply.)
- One 10-kilohm 1/8 W resistor
- One 0.1 µF ceramic capacitor C1
- One 10 µF tantalum capacitor C2
- One custom shield as described in "The Shield" on page 84, or perf board (If you use perf board, I recommend buying from Jameco.) You can also have the shield custom fabricated from ExpressPCB. (See "Making Your Own PCBs" on page XX.)
- One gear head motor (Surplus websites such as Electronic Goldmine should have this, though I used a 6V, 20 RPM motor from Amazon called the Amico 20 RPM 6VDC 0.45 A. I've seen the same motor, or one very similar, on eBay for only about $6 and change, too.)
- Fourteen LEDs, in assorted colors (You can get these from multiple sources; I'll discuss this in detail after the list.)

- Assorted hookup wire and wire-wrap wire (I buy these from OK Industries and Pololu. eBay also has a good selection of wire-wrap wire in different colors and at low prices.)
- Ten stakes for LED wire wrap or soldering (Try Pololu, item #966, or Electronic Goldmine, item #G19870, though other shops should have them. These are very inexpensive.)
- One length brass round that's 3/8 or 1/2 inches in diameter and approximately 3/4 inches long (I used one with a 3/8-inch diameter. Brass stock is readily available from many hobby shops and from Amazon in 6- and 12-inch lengths, which can be cut to size with a hacksaw.)
- One 6-inch length of piano wire that's 0.39 inches in diameter (This is readily available from any hardware store. A 36-inch length sells for under $1.)
- One niobium, or neodymium, magnet, approximately 3/8 inches round and 1/8 inches thick
- (Optional) One Amico H7EC-BCM counter (You can get this from Amazon.)(Optional) Eight 270-ohm resistors (Use these when you build the breadboard prototype if you choose to follow my exact instructions in "The Breadboard" on page 74.)

When you choose your LEDs for this project, you can pick any colors you like, so take this chance to personalize. I ordered a number of LED packages from eBay. Prices for a package of 10 to 20 LEDs ranged from $1.50 to $4. Many were high-output LEDs, and I purchased both clear and frosted versions. The higher-output units tended to be clear.

Downloads

Before you start building, go to *http://www.nostarch.com/arduinoplayground/*, download the resource files for this book, and look for the following files for Chapter 4:

- PDF of drilling templates
- PDF of mechanical drawings
- Express PCB layout of shield
- Sketch for Watch Winder

Basic Watch Winder Requirements

Some initial research suggested that a watch winder should rotate a watch between 600 and 1,200 revolutions per day to keep it in top shape. But that is not completely correct. I subsequently discovered that the range was actually much wider, and according to at least two websites of leading automatic watches, watches cannot be overwound because they have a

built-in protection system. I also learned that watches should be rotated in both directions, clockwise and counterclockwise to keep lubricant in the right places and to avoid possible uneven wear over a very long period of time. There is a wealth of information about this subject on the Web, both on sites for individual watch manufacturers as well as on sites for watch winders.

Apparently the total number of turns is the important part, not necessarily the sequencing of the turns or getting exactly the same number of turns in each direction. (There is a possible downside to winding, if a watch is wound too much over an extended period.) That doesn't sound so daunting, right? I thought so, too.

Using an Arduino to Control Winder Revolutions

A purely utilitarian watch winder just has to serve its function, rotating the watches so the pendulums swing. But it's more interesting to have a winder with extra features. As mentioned in "Why a Watch Winder?" on page 66, some winders are dressed up with fancy exotic wood boxes to display the watches.

However, this is an Arduino project, and extra technical features and LEDs should reflect the flexibility and versatility of the platform. In a developmental model, the original sketch instructed the electronics to turn a motor first in one direction and then the other, using delays to ensure that the requisite 650 to 1,000 revolutions occurred each day.

But it turns out that some watches need more than the minimum number of revolutions, and some can get away with less. The easiest way to change the number of revolutions is by adjusting the various delays in the sketch as needed. You could even add hardware to the circuit to allow you to adjust the number of turns per day with a potentiometer, as I describe in "Design Notes" on page 100.

To drive the motor itself, I used an H-bridge IC. It accepts control logic from the Arduino and lets you reverse the polarity to the motor from a single power supply to allow the motor to rotate in both directions.

NOTE *For more information on H-bridges, see "Using an H-Bridge" on page XX.*

Using a Hall Effect Sensor to Monitor Rotations

Then, there was the matter of how to meter the number of turns the device made to assist the timing and give some more information to the sketch. The number of turns per unit time is a function of the motor, and while the timing I provided for the motor specified could conceivably work, it might not be consistent for all motors.

For example, I sampled three motors of the same model at the same voltage, and each ran at a slightly different speed. Further, if you elect to substitute another motor with a different rotational speed, the rotation count would be different. And, in beta testing, one user experienced difficulty running a 6V motor on 5V. (See the note on page XX.) Because the number of turns per unit time is a function of the motor, these inconsistencies could present a problem if timing alone determined the total number of rotations; some mechanism to monitor the number of revolutions is needed.

To assure consistent timing, I decided to meter the number of turns the device made. Thus, I attached a small magnet to the rotating shaft that turns the watches and mounted a Hall effect device, or a sensor that detects a magnetic field, in line with the magnet. A small reed switch could be substituted for the Hall effect sensor if you wanted.

When the watch and the magnet rotate, the Hall effect switch turns on only when in close proximity to the magnet, causing the switch to turn on and off once per rotation. Each time the Hall effect switch changes state, the Arduino increments an internal counter. Combined with the sketch, this ensures the proper number of turns per day is made in all cases, regardless of the speed of the motor. Unlike the reed switch, the Hall effect switch does not require any buffering or debounce, as discussed in "The Sketch" on page 78, because a Schmitt Trigger is included in the device's circuit. If you elect to use a reed switch, you may have to add the debounce into the sketch.

When using a Hall effect switch with a permanent magnet, you just have to be careful how you move the magnet around. Some mechanical watches are damaged by close proximity to a strong magnetic field because the hairspring becomes magnetized, resulting in a change in physical characteristics that cause timing to be off. While the magnet specified is small and unlikely to cause a problem, I strongly recommend you keep any magnet at least an inch away from any watch—mechanical or electronic.

The Schematic

Figure 4-3 shows the schematic diagram of the circuit used for the Watch Winder. Notice that the output from the Hall effect device has a pull-up resistor tied to the positive supply. This holds the input to Arduino pin A0 high until the Hall effect switch, or reed switch, encounters a strong enough magnetic field, which closes the switch and brings the pin low. The Hall effect device uses what is essentially an open collector on its output, so without the pull-up resistor, the collector would be left floating and could give a false trigger.

The two capacitors prevent the LM7805 regulator from oscillating on its own and drawing excessive power. Although I looked at both the input and the output of the regulator with an oscilloscope and saw no oscillation, I decided to add the capacitors as a preventative measure. I selected them based on previous projects, and they work well.

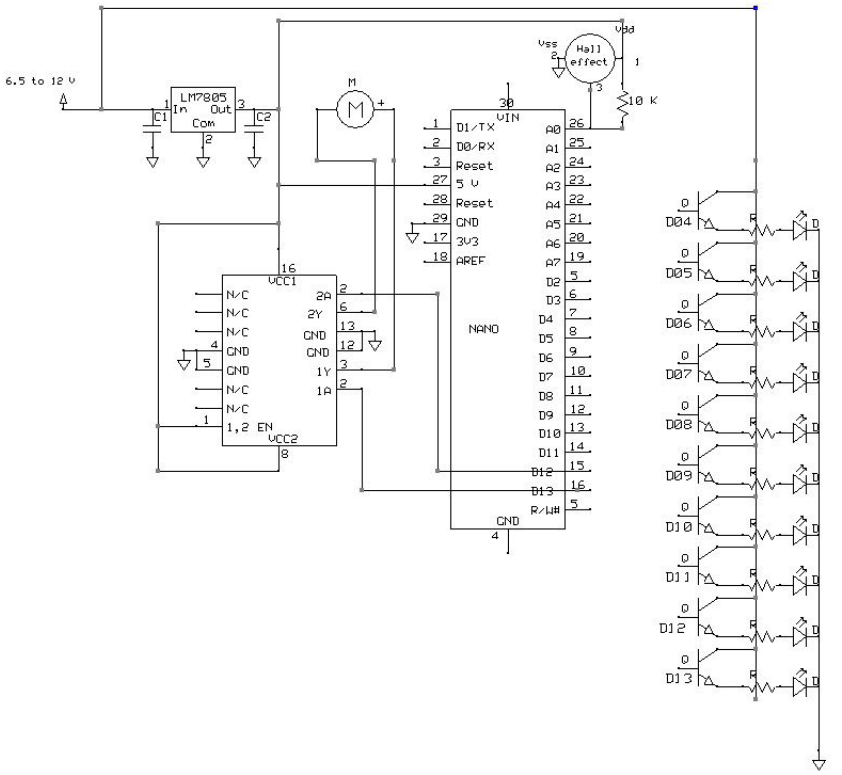


Figure 4-3: The Watch Winder circuit. The transistors connect to the digital outputs of the Nano, while A0 is tied high through the 10-kilohm resistor.

I was trying to develop a spectacular look for the Watch Winder, as befits some of the timepieces it holds, so I used higher-power LEDs, as described in the "Parts List" on page 69. These LEDs have a light output of as much as 100,000 to 200,000 or more millicandela (MCD). But that raised yet another problem. The Arduino Nano's processor chip, an ATmega328, can source or sink only 40 mA per output pin. Further, the entire chip is rated at only 200 to 300 mA for its entire current drain. Because the 100,000+ MCD LEDs draw around 30 to 60 mA each, something had to be done.

One 1 A transistor per LED is included in the schematic to pick up the load. The collectors of the NPN transistors—the positive side—go to V_{IN} rather than the 5V that powers the Nano and H-bridge, so the LEDs take no toll on the voltage regulator, even though the emitters follow the base and send 5V to the LEDs.

The Breadboard

Just like other projects we've discussed, the Watch Winder started out as a breadboard, shown in Figure 4-4. This allowed me to sound out the technology and do the preliminary tuning of the sketch.

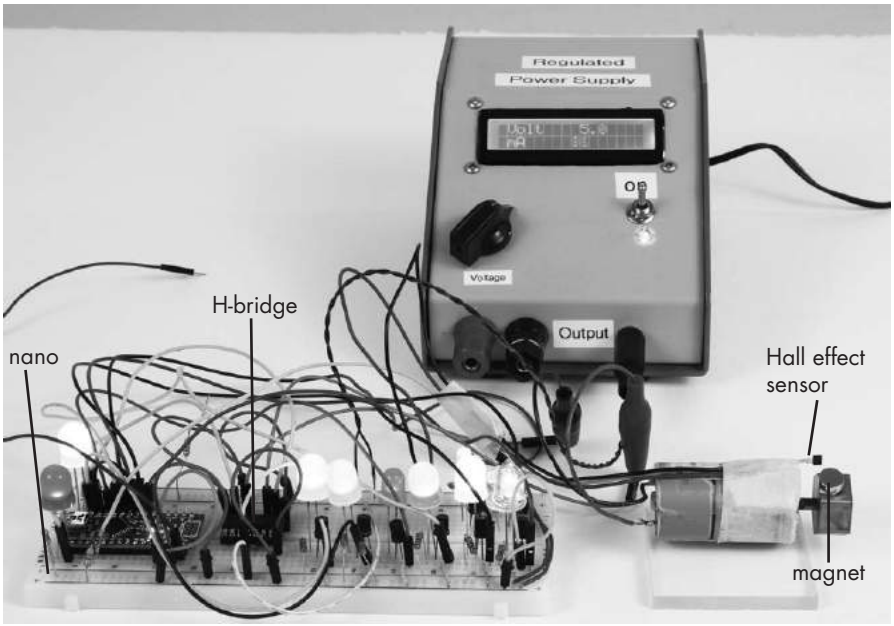


Figure 4-4: The Watch Winder breadboard was used as a proof-of-concept for the project. Here, I powered it with Project 4, the Regulated Power Supply.

I suggest building a breadboard for this project first so you can see where everything goes and why. With a breadboard, you also get to play with the sketch and LEDs without having to unsolder and resolder with each change. I used a 6.5-inch long breadboard to hold everything. I did take a couple of shortcuts on the breadboard, which are noted in the instructions; you can also just build straight from the schematic, instead.

To wire up the breadboard, take the following steps:

1. Connect the red strip on the right side of the breadboard to the corresponding red strip on the left. These are your positive rails.
2. Connect the blue strip on the right side of the breadboard to the corresponding blue strip on the left. These are your ground connections.
3. Insert the Arduino Nano at one end.
4. Connect the 5V pin of the Nano to the red strip.
5. Connect the GND pin of the Nano to the blue strip.
6. Insert the LM7805 regulator, and connect the output pin to the red strip.
7. Connect the ground terminal of the regulator to the blue strip.
8. The input terminal of the regulator will connect to a blank row in the breadboard, which will connect to the + 7.5V to 9V supply.
9. Connect capacitor C1 from the input of the regulator to ground.
10. Connect capacitor C2 from the output of the regulator to ground.
11. Insert the SN754410 H-bridge several rows away from the Nano, straddling the gutter in the middle of the breadboard.

12. Connect pins 4, 5, 12, and 13 of the H-bridge to ground.
13. Connect pins 8, 9, and 16 of the H-bridge to the positive rail.
14. Attach pins 14 and 11 of the H-bridge to the motor with leads at least 10 to 12 inches long. The connections to the motor will have to be soldered unless you use alligator clips or clip leads.
15. Attach approximately 8-inch wires to all three leads of the Hall effect sensor. Connect the leads attached to the positive and negative leads of the Hall effect sensor to the positive and negative, or red and blue, strips, respectively.
16. Connect the wire attached to the third pin of the Hall effect sensor to pin A0 on the Nano. The Hall effect sensor will be taped (I used masking tape) to the motor body (this works because the leads are insulated) in such a position that the active part of the device will be close to the magnet attached to the shaft as it goes around.
17. Connect a 10-kilohm resistor from pin A0 on the Nano to the red strip.
18. Connect pin 10 of the H-bridge to pin D13 on the Nano.
19. Connect pin 15 of the H-bridge to pin D12 on the Nano.
20. Insert five ZTX 649 transistors, each using three rows, on one side of the breadboard.
21. Connect the collector of each transistor to the red strip.

NOTE *These transistor connections differ from the final schematic, where the collectors will be tied to the 9V input voltage.*

22. Take five LEDs, and connect the positive terminal of an LED to the emitter of each transistor.
23. Connect the negative terminal of each LED to ground through a 270-ohm resistor.
24. Connect three additional transistors, LEDs, and 270-ohm resistors on the opposite side of the breadboard in a similar manner to the five groups connected in Steps 20–23.

NOTE *Steps 20–25 are different than the schematic, as the LEDs are driven directly from the Nano, for ease of experimentation. In the schematic, they are connected using a transistor. Using the 470-ohm resistor instead of the 270-ohm resistor limits the current to the Nano.*

25. Connect the positive terminal of another LED to pin D12 of the Nano.
26. Connect the negative terminal of the pin D12 LED to an empty row on the breadboard.
27. Connect one end of a 470-ohm resistor to the negative terminal of the pin D12 LED, and connect the other end of the resistor to ground.
28. Connect the positive terminal of another new LED to pin 13 of the Nano.

29. Connect the negative terminal of the pin 13 LED to an empty row on the breadboard.
30. Connect one end of a 470-ohm resistor to the negative terminal of the pin 13 LED, and connect the other end of the resistor to ground.
31. Connect pins D4, D5, D6, D7, D8, D9, D10, and D11 on the Nano to the bases of the transistors feeding the LEDs.

Because the breadboard is for illustration only, the order that the connections are made in doesn't matter unless you want to reprogram the Arduino while the circuit is on the breadboard. Figure 4-5 shows the H-bridge pinout, Figure 4-6 shows the regulator pinout, and Figure 4-7 shows the transistor pinout.

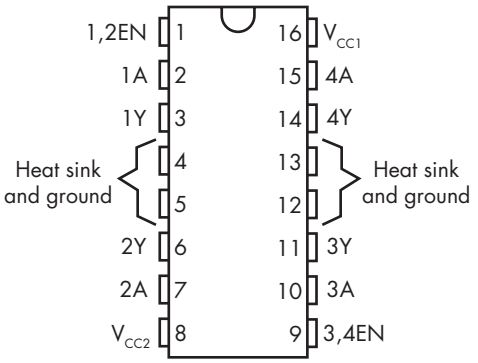


Figure 4-5: SN754410 H-bridge pinout in a DIP form factor

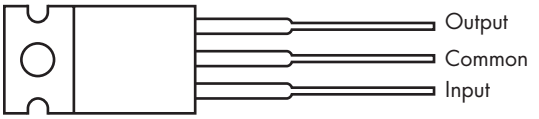


Figure 4-6: LM7805 regulator pinout in a TO-220 package

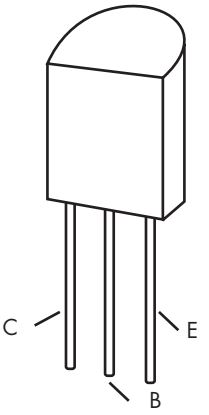


Figure 4-7: ZTX 649 transistor pinout in a TO 92 package

The magnet was mounted to a plug on the motor shaft using double-sided adhesive foam tape. For the plug, you can use almost anything—a cork, a rubber stopper, and so on—as long as it puts the magnet in a position so it will be about 3/8 inches from the sensor as the magnet rotates.

After you complete the breadboard, upload the *WatchWinder.ino* sketch to the Arduino. Just follow the instructions in "Uploading a Sketch to an Arduino" on page XX.

The Sketch

The Watch Winder employs functions and arrays to show different flashing sequences on the LEDs without rewriting the sequence each time. There are also some excruciatingly long delays: about 829 rotations in 24 hours translates to a motor at 20 RPM being on for 32.5 minutes out of 1,400 minutes in the day. This means that if the sketch were to handle an entire day of turning, it would be idle for 1,367 minutes a day.

But you can divvy up the rotations so that the sketch can be repeated and need only some fraction of the 24 hours to complete. For example, if an hour is selected as the length of time it takes for the sketch loop to complete, the motor has to do some 24 turns. It could do 12 each way or some other combination.

In the following sketch, I also made an effort to make the lights and motor movements as visually interesting as possible, leaving very little time when nothing is happening—but that's an artistic choice.

```
/*This gives about 829 revs/day*/

const int HallPin = A0; //Identify those things that will not change
const int CWpin = 12;
const int CCWpin = 13;

const int LED11 = 11;
const int LED10 = 10;
const int LED9 = 9;
const int LED8 = 8;
const int LED7 = 7;
const int LED6 = 6;
const int LED5 = 5;
const int LED4 = 4;

int autoDelay = 1000;
int timer = 500;
int timer2 = 3000;
int repeats = 10;

int previous;
int HallValue = 1; //Response from the Hall effect sensor
int time = 0;
int state;
int count = 0;
```

```
int q = 0;
int i;
int j;

int ledPins[] = {
  11, 4, 7, 6, 8, 10, 5, 9,
};
int pinCount = 8;

void blinkIt()
{
  //Initiate rapid blink sequence
  for (int thisPin = 0; thisPin < pinCount; thisPin++) {
    //Turn the pin on:
    digitalWrite(ledPins[thisPin], HIGH);
    delay(timer2);
    //Turn the pin off:
    digitalWrite(ledPins[thisPin], LOW);
    delay (timer2);
  }

  //Loop from the highest pin to the lowest:
  for (int thisPin = pinCount - 1; thisPin >= 0; thisPin--) {
    //Turn the pin on:
    digitalWrite(ledPins[thisPin], HIGH);
    delay(timer2);
    //Turn the pin off:
    digitalWrite(ledPins[thisPin], LOW);
    delay (timer2);
  }
}

void flashIt()
{ //Initiate rapid blink sequence
  for (int thisPin = 0; thisPin < pinCount; thisPin++) {
    //Turn the pin on:
    digitalWrite(ledPins[thisPin], HIGH);
    delay(timer2);
    //Turn the pin off:
    digitalWrite(ledPins[thisPin], LOW);

  }

  //Loop from the highest pin to the lowest:
  for (int thisPin = pinCount - 1; thisPin >= 0; thisPin--) {
    //Turn the pin on:
    digitalWrite(ledPins[thisPin], HIGH);
    delay(timer2);
    //Turn the pin off:
    digitalWrite(ledPins[thisPin], LOW);

  }
}
```

```
void allatOncefast() {
  {
    digitalWrite (LED4, HIGH);
    digitalWrite (LED5, HIGH);
    digitalWrite (LED6, HIGH);
    digitalWrite (LED7, HIGH);
    digitalWrite (LED8, HIGH);
    digitalWrite (LED9, HIGH);
    digitalWrite (LED10, HIGH);
    digitalWrite (LED11, HIGH);

    delay (500);

    digitalWrite (LED4, LOW);
    digitalWrite (LED5, LOW);
    digitalWrite (LED6, LOW);
    digitalWrite (LED7, LOW);
    digitalWrite (LED8, LOW);
    digitalWrite (LED9, LOW);
    digitalWrite (LED10, LOW);
    digitalWrite (LED11, LOW);

    delay (500);

  }
}

void allatOnce() {
  {
    digitalWrite (LED4, HIGH);
    digitalWrite (LED5, HIGH);
    digitalWrite (LED6, HIGH);
    digitalWrite (LED7, HIGH);
    digitalWrite (LED8, HIGH);
    digitalWrite (LED9, HIGH);
    digitalWrite (LED10, HIGH);
    digitalWrite (LED11, HIGH);

    delay (4000);

    digitalWrite (LED4, LOW);
    digitalWrite (LED5, LOW);
    digitalWrite (LED6, LOW);
    digitalWrite (LED7, LOW);
    digitalWrite (LED8, LOW);
    digitalWrite (LED9, LOW);
    digitalWrite (LED10, LOW);
    digitalWrite (LED11, LOW);

    delay (2000);

  }
}
```

```
void setup()
{

  pinMode (HallPin, INPUT);  //Identifies inputs and outputs
  pinMode (CWpin, OUTPUT);
  pinMode (CCWpin, OUTPUT);

  Serial.begin (9600);


  for (int thisPin = 0; thisPin < pinCount; thisPin++)  {
    pinMode(ledPins [thisPin], OUTPUT);
  }
}

void loop()
{
  int HallValue = (digitalRead (HallPin)); //Sets the value of the

  if (HallValue == HIGH && previous == LOW) {
    if (state == HIGH)
      state = LOW;
    else
      state = HIGH;

    //Increments the counter each time the Hall effect sensor passes the
magnet
❶   count++;
  }

  /* The "Serial.print" line was used in development. I left it in so that
     you can experiment and look at some of the values on a serial
     monitor. You might even want to change the parameters of what you
     are looking at in the monitor.
  */
  Serial.print ("HallValue      ");
  Serial.println (HallValue);
  Serial.print ("count                ");
  Serial.println (count);
  Serial.print ("CCW                     ");
  Serial.println (" ");


  if (count == 1) {
    digitalWrite (CCWpin, HIGH);
    digitalWrite (CWpin, LOW);
  }

  if (count == 3) {
    digitalWrite (CWpin, HIGH);
    digitalWrite (CCWpin, HIGH);
  }
  if (count == 3) {
    for (i = 0; i < repeats; i++) {
      allatOncefast ();
```

```
  }
  count = count + 1;
}
if (count == 3) {
  digitalWrite (CWpin, LOW);
  digitalWrite (CCWpin, HIGH);
}
if (count == 4) {
  digitalWrite (CWpin, HIGH);
  digitalWrite (CCWpin, HIGH);
}
if (count == 4) {
  for (q = 0; q < repeats; q++) {
    blinkIt();
  }
  count = count + 1;
}
if (count == 5) {
  for (j = 0; j < repeats; j++) {
    allatOnce();
  }
  delay (50);
  count = count + 1;
}

if (count == 6) {
  digitalWrite (CCWpin, LOW);
  digitalWrite (CWpin, HIGH);
}

if (count == 7) {
  digitalWrite (CWpin, LOW);
  digitalWrite (CCWpin, LOW);
}

if (count == 7) {
  for (i = 0; i < repeats; i++) {
    flashIt();
  }
  count = count + 1;
}

if (count == 8) {
  digitalWrite (CCWpin, HIGH);
  digitalWrite (CWpin, LOW);
}
if (count == 10) {
  for (i = 0; i < repeats; i++) {
    allatOncefast();
  }
  count = count + 1;
}
```

```
if (count == 11) {

  digitalWrite (CCWpin, LOW);
  digitalWrite (CWpin, LOW);
}

if (count == 11) {
  for (i = 0; i < repeats; i++) {
    blinkIt();
  }

  delay (2000);
  count = count + 1;
}

if (count == 12) {
  digitalWrite (CCWpin, LOW);
  digitalWrite (CWpin, HIGH);
}

if (count == 13) {
  digitalWrite (CCWpin, HIGH);
  digitalWrite (CWpin, HIGH);
}
if (count == 13) {
  for (i = 0; i < repeats; i++) {
    flashIt();
  }
  count = count + 1;
}

if (count == 14) {

  for (i = 0; i < repeats; i++) {
    allatOnce();
  }
}

if (count == 14) {
  digitalWrite (CWpin, LOW);
  digitalWrite (CCWpin, HIGH);
  delay (autoDelay);
}
if (count == 17) {
  digitalWrite (CWpin, HIGH);
  digitalWrite (CCWpin, HIGH);
}
if (count == 17) {
  for (i = 0; i < 20; i++) {
    blinkIt();
  }
}
{
```

```
    for (i = 0; i < repeats; i++) {
      allatOncefast();
    }
    count = count + 1;
  }
  if (count == 18) {
    digitalWrite (CCWpin, HIGH);
    digitalWrite (CWpin, LOW);
    delay (2000);
    digitalWrite (CCWpin, LOW);
    digitalWrite (CWpin, HIGH);
    delay (2000);
  }
  if (count > 20) {
❷   count = 0;
  }
  previous = HallValue;
}
```

First, the sketch creates several constants, integers, and arrays, which assist with timing turns by reading from the Hall effect sensor and counting the turns. Next, come a few function definitions: `blinkIt()` and `flashIt()` blink the LEDs in different patterns, while `allatOnceFast()` and `allatOnce()` blink the LEDs all at the same time with different delays.

As usual, the `setup()` function tells the Arduino which pins are inputs and outputs. At the start of the `loop()` function, the Hall effect sensor is read, and the sketch increments the counter at ❶ as needed, printing a few useful debugging values to the serial monitor along the way. This sketch uses the `count` value to turn different sequences on or off and limit the repetitions. However, because `count` is reset at the end of the sequence at ❷, it cannot be used as a totalizer.

Finally, for various counts, the sketch uses `if` statements to hard-code different patterns for turning the watches and flashing the LEDs; I show a few here, but I encourage you to set up your own. The sketch is written with many functions you can use as-is or repeat in a `for` loop to give multiple iterations.

The Shield

As in some of the other Arduino projects, the shield is not terribly complex, but it looks a little busy. For simplicity, this shield is a single-sided board. The circuit uses an LM 7805 voltage regulator to handle excess current that could result from using a different motor. The on-board regulator built into the Nano is intended only for currents less than 300 mA.

NOTE *I have used the regulator in this project at up to 500 mA, but the regulator tends to get pretty warm, and I don't feel comfortable using it at that level.*

You may be able to leave the regulator out; the collectors of the transistors feeding the high-output LEDs are configured as emitter-followers and wired directly to the positive 9V supply, so they are not contributing to the load on the regulated 5V.

Figure 4-8 shows the foil pattern for the shield (left), and the silk-screen layer (right).

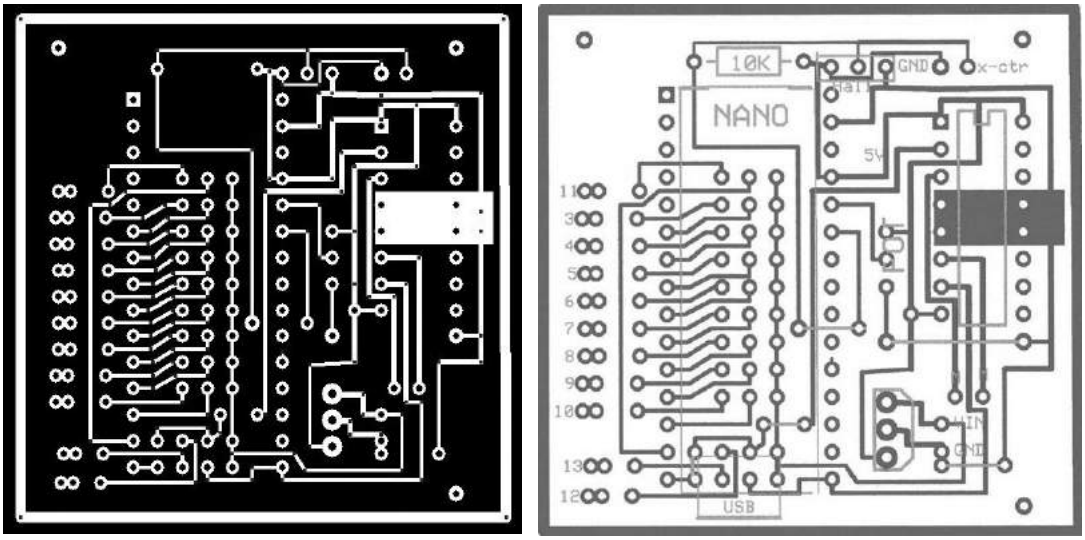


Figure 4-8: The foil pattern of the shield (left) and the silk-screen layer (right). The Watch Winder shield's silk-screen layer shows the approximate placement of the Nano, H-bridge, contacts for the external counter, Hall effect sensor, potentiometer, LED connections, jumpers, input voltage (VIN), and ground (GND). The PCB Express file is available to download from http://www.nostarch.com/arduinoplayground/.

Notice the contacts for the Hall effect switch at the top of the board, labeled *Hall*, I soldered wires that connected the Hall effect device directly to the PCB, though you could use a connector if you prefer.

In the center of the board, I left connections for a potentiometer (POT) for external adjustment of the period, an option I describe in "Total Rotation Adjustment" on page 100. The numbers of the digital outputs are labeled at the left-hand side.

If you choose to assemble the Watch Winder shield, note that the Nano is meant to be plugged into female headers soldered onto the shield and that the transistors are underneath the Nano board. Push the transistors down far enough before you solder them so they will not be in the way of the Nano when it's plugged in. I had to place the transistors fairly close to fit all the connections into the PCB layout. The ZPT 649 transistors I selected fit well enough within the 0.100-inch spacing allowed by the footprint of the Nano.

You will also have to add a few jumper wires to complete the connections on the shield. They are marked on the silk-screen pattern. In Figure 4-8, those appear as five black lines. Don't forget to include them

when wiring up the board. I also left out the capacitors from the LM7805 regulator's input and output to ground; in the finished board, they are soldered on externally. If you want those capacitors, simply solder a 10.0 mF tantalum and a 0.1 mF ceramic capacitor directly to the pins of the regulator, as shown in the schematic in Figure 4-2.

Overview of the Motor Assembly

When you've had enough fun watching the LEDs blink on the breadboard, watching the motor start and stop, and playing with the sketch, it's time to address the mechanical side of this project, which offers a few special challenges. This winder won't be functional until the motor has something to turn; see Figure 4-9 for a detailed diagram of the motor, motor mount, transmission, bearing box, and driveshaft, which comprise the turning assembly.

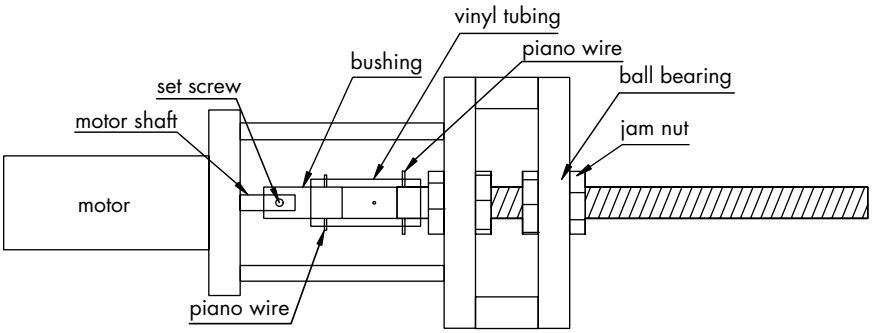


Figure 4-9: The construction entails making a small box that retains the bearings through which the driveshaft is mounted and held in place by jam nuts. The motor, mounted on standoffs, is connected to the driveshaft, and the watch basket will be attached to the other end of the driveshaft.

The driveshaft will need to be mounted through the bearings, and the two can be held together with jam nuts. I chose a fairly standard R4A-2RS bearing, which is a relatively common part and has a 3/4-inch outer diameter, a 1/4-inch inner diameter, and a 9/16-inch thickness. I suggest ball bearings because the prebuilt winder I bought used the brass bushing of the motor as the only bearing, and that's what failed. Because the inside diameter of the bearing was 1/4 inches, I decided to use a standard threaded rod at a 1/4-inch × 20 tpi (turns per inch) or 1/4-inch × 28 tpi rather than attempt to press-fit a 1/4-inch bar into the bearing.

NOTE *I used jam nuts to fasten the rod to the bearings because they were thinner and less obtrusive looking than regular nuts, but you could also use standard nuts.*

Construction

Construction of the winder provided a number of challenges, particularly working with the acrylic material—which was unfamiliar to me before this project. Though there were some rough spots to get over, I learned by trial and error some ways to get the job done. The toughest part was cutting the acrylic and drilling the holes for the ball bearings.

There are several ways to cut acrylic, and none of them is particularly easy. If you use a supplier that will laser cut the acrylic parts for you, this will be a lot easier. Several companies offer that service for a little more than the price of the raw materials. I mentioned one of them, ZLazr, earlier.

If you have access to a circular saw, that's about one of the easiest DIY approaches. Otherwise, just about any saw will do. I've used a hacksaw, which works better than most if you take your time. (If you go too fast, the acrylic will heat up and start to melt, causing the saw to bind.) I even know some people who have had success scoring and snapping the acrylic sheet. Just use whichever approach works best for you.

See "Acrylic" on page 69 for a list of acrylic shapes needed for the bearing box, the watch basket, and the stand. Cut these pieces now, if you've not done so already, and take the following steps to build the pieces.

Preparing the Motor Plate and Bearing Box Acrylic

First, print the motor mount template from this chapter's folder (see Figure 4-10), cut it out, and align it on the acrylic for the motor plate using the centerline.

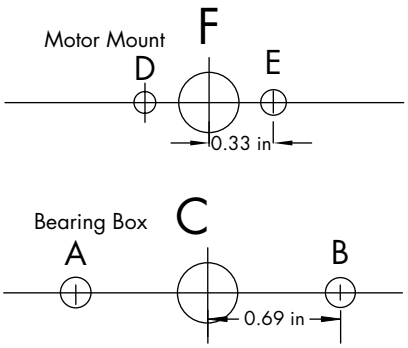


Figure 4-10: Templates for motor mount and bearing box. You can download the templates at http://www.nostarch.com/arduinoplayground/.

Tape the template to the acrylic, lining up the centerline on the center of the acrylic piece, and mark the drill centers for holes A, B, D, E, and F. To mark the holes, just punch the centers with a center punch or nail. Now, drill them; use a 1/8-inch bit for A, B, D and E and a 3/8-inch bit for F.

Then, set this piece aside, and gather the acrylic for the bearing box. The final bearing box will look like Figure 4-11 once we put it all together.

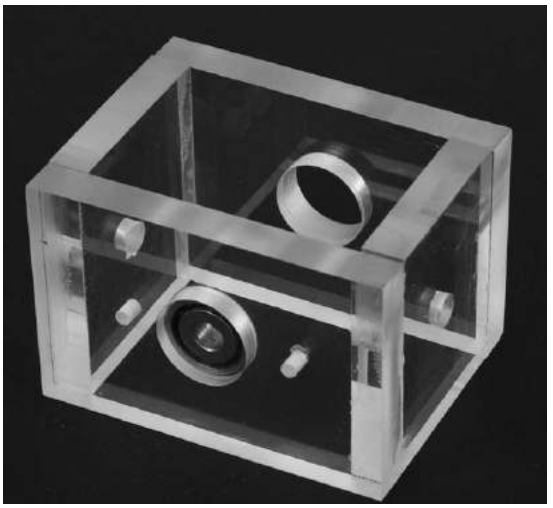


Figure 4-11: Completed bearing box, before final trim and polish. One bearing is temporarily in place.

Use the bearing box template to mark the drill centers for holes A, B, and C on one of the bearing mount pieces. C will be the bearing hole, and A and B will be for the standoffs that mount to the motor plate. Center punch and drill one of the bearing plate's two smaller holes with a 2.5 mm (or # 39) drill and tap the hole with a M3-05 tap. These holes will accept the standoffs for mounting the motor. Use the same template to mark only the bearing hole (C) on the acrylic for the opposite side of the bearing box (see Figure 4-12). Then, take the two pieces of bearing box acrylic that won't hold bearings, and mark the center by drawing lines from corner to corner. Center punch them, and drill 1/4-inch holes for mounting to the stand.

Figure 4-12: Drilling a large hole in the acrylic. Note that the acrylic piece is securely clamped down.

WARNING *When drilling any size hole in the acrylic, securely clamp the piece, as shown in Figure 4-12. Do not hold it manually. If the drill or hole saw binds, the acrylic will want to spin. In any case, keep the drill at a slow speed and advance it very gradually into the work.*

The best solution I found to make the holes for the bearing and other large holes in the acrylic is to drill a relatively small hole—perhaps 1/4 or 3/8 inches—and ream them out with a tapered reamer to the finished dimension (see Figure 4-13). This is, by far, the safest and easiest approach and the one I strongly recommend.

When you ream out the bearing hole, make sure to ream from both sides. This will result in the center of the hole being a slightly smaller diameter than the outsides. Ream until the bearing is a tight fit and then, if necessary, you can use an anaerobic bonding agent to fill in around the edges.

Figure 4-13: Using a tapered reamer to enlarge the bearing hole to the finished dimension

The final prep stage for the acrylic is to sand and finish it. How you cut the acrylic originally will determine how much finishing it will take to get the edges ready. If you had the pieces laser cut, little finishing will be required. For all finishing, I used ascending grades of sandpaper, starting with 220 grit and going up to 1500—that is, 220, 320, 400, 600, and then 1500. Automatic sanders—orbital, belt, vibratory, and so forth—often are too rough, and without special care, they will melt the acrylic. If you use one, try it on a scrap piece first. The sanding process worked well even though some extra sanding was required on roughly cut sections. Additional sanding was required on the opposing piece to make everything fit together as a rectangle—or as a cube in the case of the bearing box.

Use a liquid polish or jewelers rouge to achieve the final polish. Make sure to remove all the wax from the polish from the surface before bonding. Try not to round the edges of the sections so the thinner bonding agent (Weld-On 4) will work well. You want to assure that you have sufficient bonding surface in contact to make a secure bond.

Bonding the Acrylic for the Bearing Box

Now, it's time to use a bonding agent to connect the pieces of your bearing box. Fortunately, bonding the acrylic actually turned out to be somewhat easier than I anticipated.

Where the edges are smooth but not too badly rounded, Weld-On 4 thin bonding fluid should work quite satisfactorily. It partially dissolves the acrylic and forms an actual weld. The most difficult part is keeping the fluid from running where it shouldn't go. If you have larger gaps, or have rounded the edges, try Weld-On 16, which has a higher viscosity and a clear acrylic filler, to fill gaps and voids where necessary.

In both cases, you should follow the instructions on the product, but here's how the acrylic weld works in general: just clamp the dry pieces of acrylic together, and then, using a needle applicator included with the bonding agent, apply a thin layer of acrylic cement to each joint. Capillary action will draw the cement into the joint. For joints where the surfaces are a little more uneven, you can apply the thicker Weld-On 16 to one surface and then attach it to the other surface. Clamping is required for only a few minutes, but allow several hours for final curing.

The system doesn't need a lot of strength, but it should not fall apart when touched. For the parsimonious, a paint stripper like Klean Strip also works well to bond the acrylic. (The chemical behind the bonding agent in Weld-On is methyl chloride, which is the key ingredient in the paint stripper.) Klean Strip is less than one-fourth the price of Weld-On.

NOTE *There are several tutorials on bonding acrylic on the web, too. If in doubt, look one up, and experiment with a few scrap pieces of acrylic first before trying it out on the pieces you worked so hard on.*

After bonding the bearing box, as shown in Figure 4-9, and bonding the side pieces to the bottom, check the alignment. Run the threaded rod through the bearings, and put the jam nuts in place without over tightening them. Make sure the bearings don't bind. If they are not well centered, this can happen, but usually, they will align themselves as you tighten the jam nuts a little. If your alignment is off, you may need to adjust the holes a little with the reamer and touch up with some acrylic cement, but I never ran into that problem. For now, remove the driveshaft from the bearing box.

The Stand

The stand is the least complex part of the project. It comprises the two vertical members, 1 1/2 × 5 × 1/4 inches, and the base, 5 1/2 × 3 × 3/8 inches. I included the lightbar, 3 × 1 1/2 × 3/8 inches, and the piece of acrylic that I mounted the shield to, 2 1/2 × 2 1/2 × 3/8 inches, with the stand (see Figure 4-14).

First, drill 1/4-inch holes in each vertical member of the stand 1/2 an inch from the top, centered left to right. Then, bond the two vertical members to the base 1 1/2 inches in from the edge of the base that will be the back. Next, drill the holes for the LEDs in the lightbar. I used five LEDs (red, blue, white, yellow, and green) that were 10 mm in size. You can use whatever color combination you choose.

Finally, drill and mount the shield. To find the center of the piece, mark from corner to corner. Then, in the center, drill a #43 hole and tap for a 4-40 screw. Drill a corresponding hole, centered and 2 inches from the rear edge, in the base. Next, use the shield itself, or the drawing from the ExpressPCB print, as a template to drill a hole for the three mounting screws. In designing the board, I failed to leave room for a fourth screw. However, three are more than sufficient, as there is no mechanical force on the board. I used a 2.5 mm (#39) drill and tap for a M3-05 screw that the standoffs will fit into. Figure 4-14 shows the dimensions of the parts and the partially assembled base, including supports, the lightbar, and the shield mount. Figure 4-15 shows the completed Watch Winder on its side.

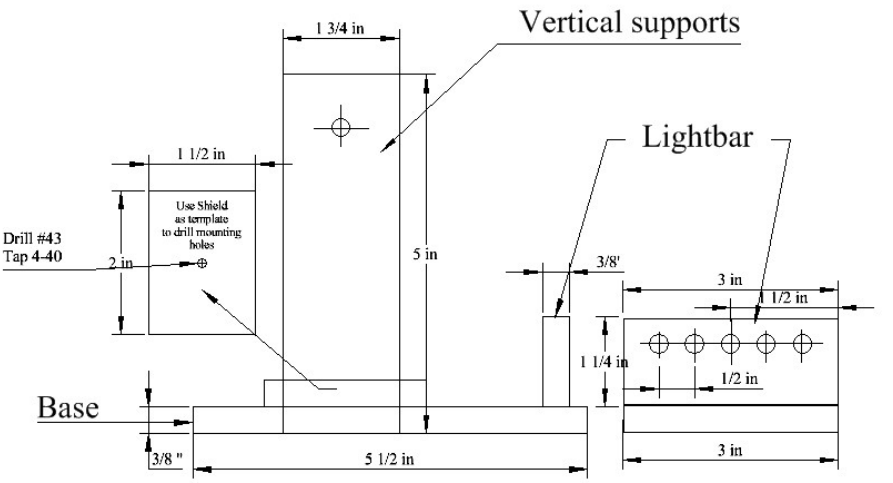


Figure 4-14: The components and configuration of the base and lightbar for the Watch Winder

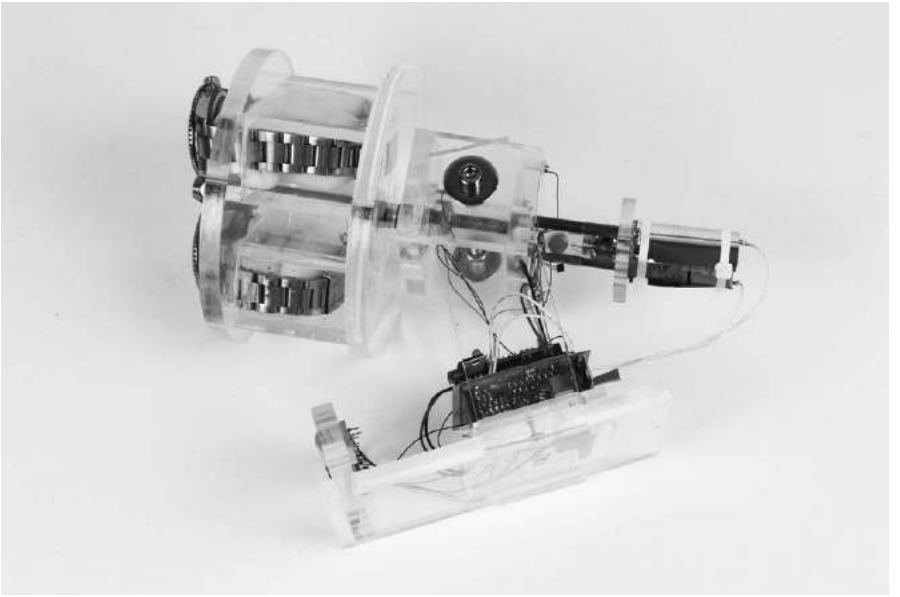


Figure 4-15: The completed Watch Winder on its side, showing the base fabrication, bearing box, motor mount, and watch basket

Preparing the Motor and the Driveshaft

Despite a concerted effort to mark and drill the holes accurately, you may still end up with misalignment between the motor and driveshaft, so this build aims to keep the coupling flexible. My solution might not be on the hit parade of industrial engineers, but I used a length of vinyl tubing to couple the motor to the drive shaft. This coupling has been working for over a year with no sign of deterioration or problems.

A 1-inch length of heavy-wall vinyl tubing with an outside diameter of 7/16 inches and an inside diameter of 3/16 inches should do the job.

It won't fit the motor shaft or the 1/4-inch threaded shaft without a bit of work, though. (I drove the sales clerk at Lowe's batty buying six of each size they had in stock.) We simply need to reduce the diameter of the threaded shaft and craft a small bushing for the motor shaft.

Trimming the Threaded Shaft

First, trim the diameter of the threaded shaft. Clamp a hand-held electric drill in a vice or parallel clamp, using a folded towel to keep from damaging the drill, and place the shaft where the drill bit would normally go (see Figure 4-16). Then, turn on the drill, and use a sharp file to trim the shaft.

Figure 4-16: Reducing the diameter of the threaded shaft

It should take only a minute or two to reduce the shaft diameter to a little over 3/16 inches for a tight fit on the vinyl tubing.

Creating the Motor Bushing

Next, take a small piece of round stock approximately 3/4 inches long and 3/8 to 1/2 inches in diameter. (I used a 3/8-inch diameter, as it required less work.) Drill a 11/64-inch (#21) hole in the end approximately 3/8 inches deep. The bushing hole needs to be as close to the center as possible, so you might want to mark it with a center punch first (see Figure 4-17).

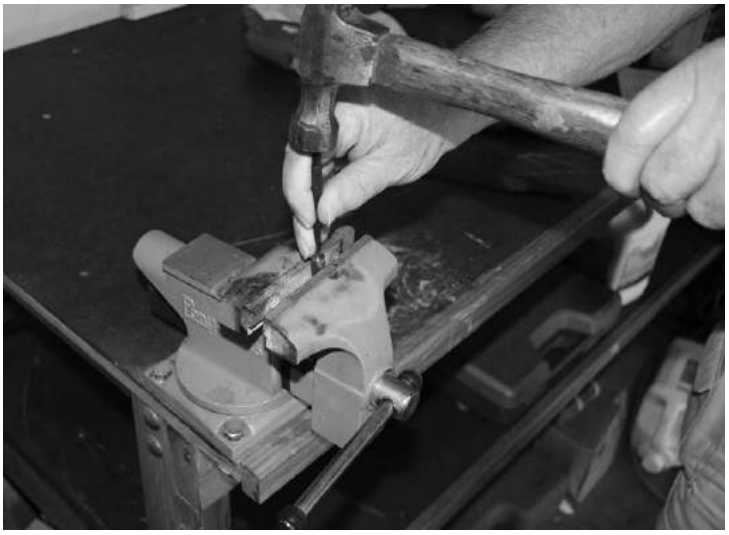


Figure 4-17: Clamp the shaft of the piece used as the bushing in a vice or pair of vise-grip pliers, and center punch a mark before drilling the hole. Get as close to the center as possible.

Then, place the short piece of stock you cut in the drill as you did the threaded shaft with the center hole toward the drill. File the end without the hole in it. Reduce the diameter by about 1/4 inches to approximately equal the filed end of the drive shaft.

Next, drill a 0.041-inch hole through the bushing, about 3/8 inches from the edge of the bushing.

While you're at it, drill a corresponding hole in the drive shaft. To center punch and drill the drive shaft and the bushing holes, file the end flat so the center punch can find a purchase (see Figure 4-18). These are the holes that will accept the piano wire through the vinyl tubing.

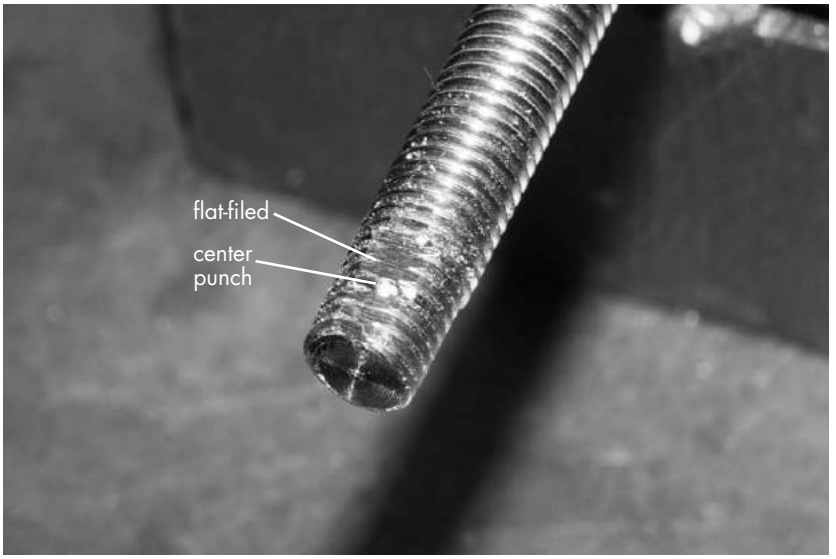


Figure 4-18: The easiest way to drill the holes in the threaded rod and bushing is to file a small flat on the shaft and center punch.

Finally, drill a 2.5 mm (#39) hole and tap an M3-.05 hole in the bushing, perpendicular to the hole drilled for the motor shaft. This will accept a set screw for the motor shaft. If you prefer, you can drill a #43 hole and tap for a 4-40 set screw. This set screw holds the bushing in place on the motor shaft, and the two 0.041-inch holes drilled in the drive shaft and motor bushing, respectively, will pin the vinyl tubing in place with piano wire.

Cutting Piano Wire Pins and Completing the Motor Assembly

Cut two pieces of 0.039-inch wide piano wire, 1/2 to 5/8 inches long. This can be a bit of a job. I used an abrasive cutoff wheel on a Dremel. Using the corner of a small grinding wheel attached to the drill should work to put a groove in the wire. Once you have a groove in the wire, you can snap it by hand. You can also use any sharpening stone to score the wire, and it should easily snap.

Once that's done, the rest of the assembly should go easily. First, mount the bushing to the motor; put a drop of thread-lock liquid on the set screw before tightening. Then, fit one end of the vinyl tubing over the bushing, and run the piano wire through the 0.041-inch hole in the bushing. Hold one end of the wire tightly in a pair of pliers (small vise-grip pliers work well), and force it through the vinyl, into the hole in the motor bushing, and into the vinyl on the other side. If this proves difficult, try heating the piano wire with a small flame, and then it should go through easily. Figure 4-19 shows the tubing over the motor bushing.

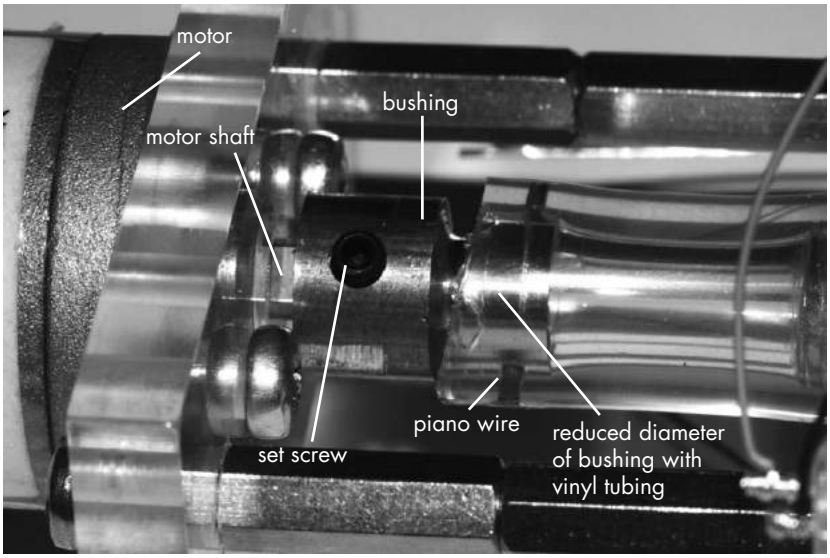


Figure 4-19: The motor shaft with the bushing comprises half of the transmission to the winder. The other half is the reduced driveshaft that goes through the bearing and holds the watch basket.

Mount the motor plate to the motor using M3 × 3/8-inch screws. Next, screw the motor standoffs into the bearing cage; see Figure 4-20 for how to place them. Take the motor assembly—that is, the motor, mounting plate, bushing, and vinyl—and fit it to the bearing cage standoffs using M3-05 × 1/2-inch screws.

To make the holes in the vinyl for the drive shaft, just install the driveshaft into the bearing box without the jam nuts, push the vinyl tubing onto it, and install the piano-wire pin as I described in the previous section. If you are concerned about the piano-wire pins coming out, you can wrap a wire tie over them or cover them with a piece of tape. (I never had a problem.) For now, remove the piano-wire pin from the driveshaft and remove the driveshaft from the bearing box until you're ready for the final assembly.

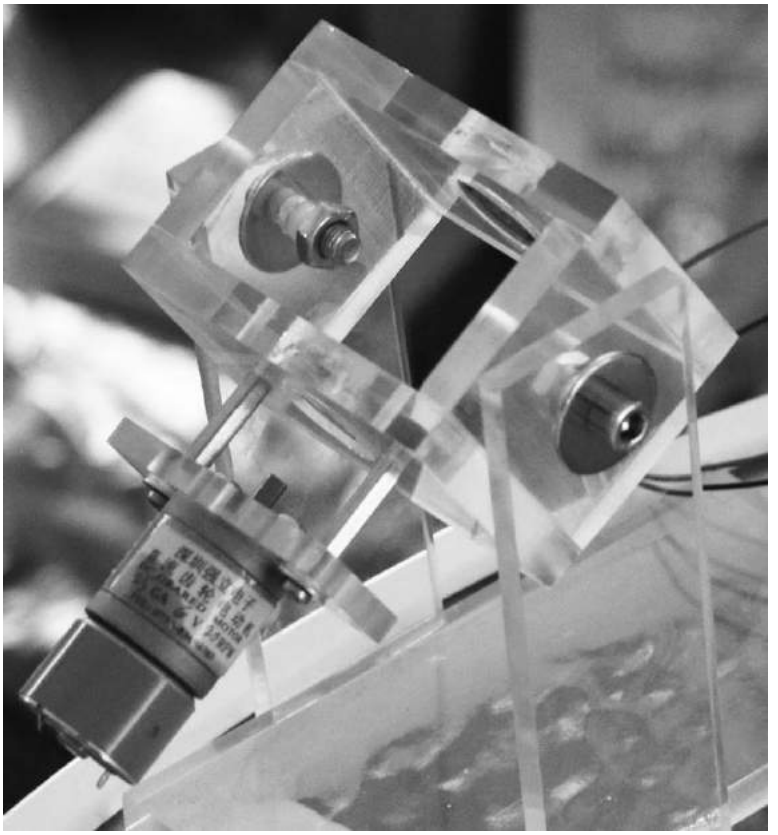


Figure 4-20: Motor and motor plate connected to bearing box with standoffs. The standoffs are threaded into the bearing box while the motor plate is fastened with 3 mm × 1/2-inch screws and washers. The bearing box is mounted to stand with 1 inch long, 1/4 inch × 28 screws and nuts (see below).

Making the Watch Basket

There are many ways to construct the part of the project that holds the watches. I chose to make my watch basket from acrylic. The construction is relatively straight forward, though it does require some patience. First, take two 5 × 3/8-inch acrylic disks and carefully mark the center on each. Drill 1/4-inch holes in the center of each. Then, on what will be the top disk, mark the rectangles shown in Figure 4-21 (left).

If you've not done so already, cut the rectangular acrylic pieces I described in the "Parts List" on page 69 for the watch boxes and assemble them following the same instructions given under "Bonding the Acrylic for the Bearing Box" on page 90. (In most of the samples I've made, I used 1/4-inch acrylic; however, 3/8-inch acrylic works fine.) Then, carefully mark out the openings on the disk (see Figure 4-21). Now, bond the watch baskets to one disk, as shown in Figure 4-22. Then, cut the openings to match up with the inside of the watch baskets.

The simplest way to cut the rectangles out is to drill a hole at the corners, being careful not to drill into the watch basket, and use a keyhole or coping saw (or if you're careful, a Saber saw) to cut the openings. They have to be cut only on one disk, which will be the top. You can clean up the edges of the cuts a little with a file or sandpaper, but don't spend too much time as it will not be noticeable with the watches and cushions in place.

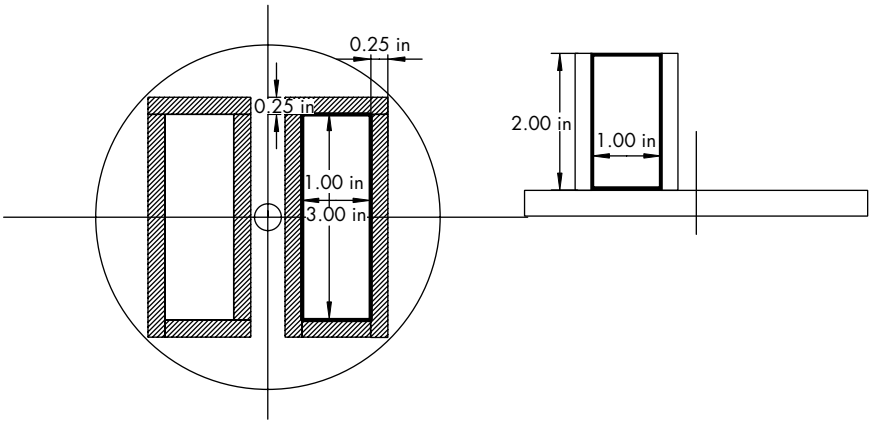


Figure 4-21: Cut out dimensions for the watch holder basket. This pattern can be downloaded as a PDF file on http://www.nostarch.com/arduinoplayground/.

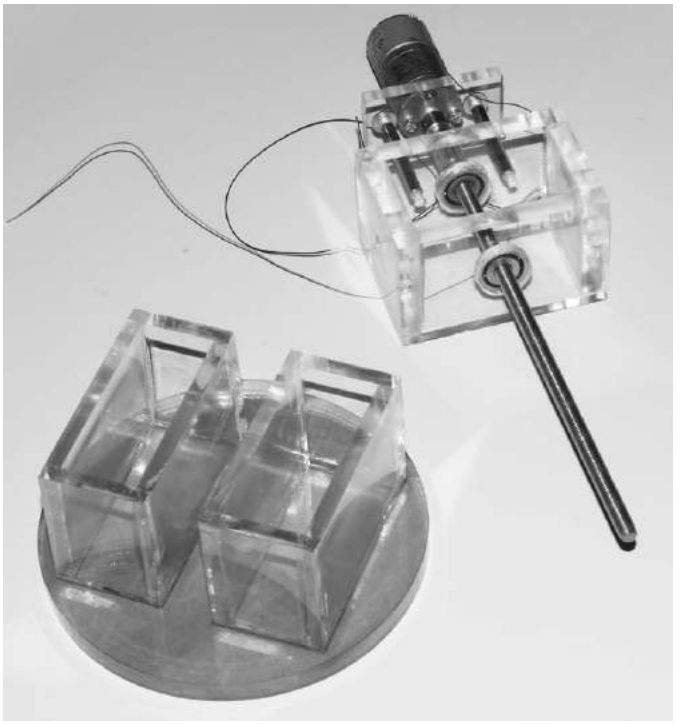


Figure 4-22: The watch holder baskets mounted to the acrylic disk. Above them is the fully assembled motor, bearing box, and drive shaft.

Finally, mount the basket on the drive shaft. First, thread two nuts onto the driveshaft and lock them against each other—that is, tighten one nut against the other. Then, add a washer, followed by the lower disk of the basket, the top assembly with the watch baskets, a washer, and a nut at the top. If you desire, go to the hardware store to search for a decorative nut to cap off the project.

NOTE *Bonding the bottom disk isn't necessary if you use locking nuts as I described and tighten the basket securely. I didn't bond the boxes to the lower disk, and it has not been a problem.*

At this point, you can mount the finished assembly to the stand using the two decorative 1/4 inch × 28 bolts and nuts. You can also bond the lightbar to the base using acrylic cement and drill the hole for mounting the shield mounting plate.

Adding the LEDs

You're just about done. Locate and mount the LEDs on the acrylic anywhere you like, and wire them up to the shield. You can see where I placed mine in Figure 4-1 on page 67. (The way the acrylic conducts the light produces some neat effects.) You may want to drill some blind holes to mount the LEDs in. Simply drill a hole the diameter of the LED but not completely through the acrylic. If you're careful , you can probably do a neat job in running the wires so they can barely be seen.

Because the LEDs are critical to the ultimate appearance of the winder, their placement and mounting is an important component of the finished product. Drilling the holes for mounting the LEDs can be a little tricky because if you bought a variety, they may have slightly different diameters. As a starting point, try 3/16-inch holes for 5 mm LEDs and 25/64-inch holes for 10 mm LEDs. I found the best way to get the right size was to drill a sample hole in a scrap piece of stock and try it before venturing to drill into a finished piece. If the hole is a bit small, a simple touch up with the tapered reamer should fix that. If a hole ends up a little large, try filling it in with some acrylic cement, such as Weld-On 16.

If you have some wire-wrap wire and a wire-wrap tool, you can wire-wrap the LEDs to the shield instead of soldering. This makes a neat connection to the back of the LEDs and is relatively inconspicuous because of the small diameter of the wire. It also lets you connect the wire close to the LED, as in shown Figure 4-23. If you soldered it, you might risk overheating the junction of the LED.

Wire-wrap or solder the leads from the LEDs to the shield. I soldered a header to the shield so that it was easy to wire-wrap or solder the leads from the LEDs to the shield directly. You do have to solder leads from the motor to the other end of the shield, but it should not be a problem.

The wires from the Hall effect sensor can be soldered to the shield and to the leads of the sensor. Measure the wires so they fit neatly where you

plan to mount the Hall effect sensor. You can use a connector, but let's keep it simple. The sensor itself and the magnet are mechanically mounted using double-sided foam tape.

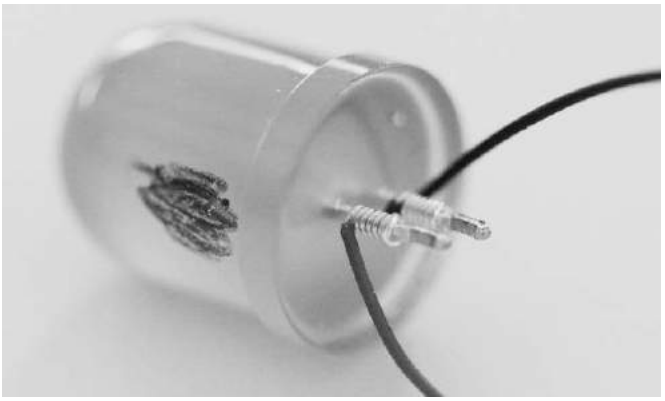


Figure 4-23: Wire-wrap wire on an LED. These wires are very fine (#30) with thin insulation, so they are unobtrusive. Small wire ties can neaten up the wiring. I suggest marking the LED's positive terminal ahead of time, as I've done here, and wiping it off later.

Leaving the Components on Display

Now, what to do with the shield and Nano? The theme of this project has been transparency, so I suggest letting everything hang out: mount the bare board on standoffs right out in the open with the switch and power jack at the back (see Figure 4-24).

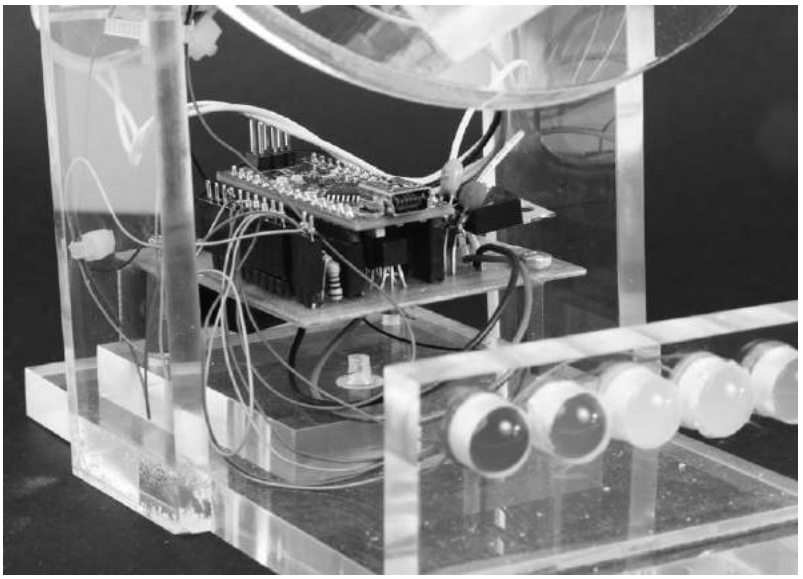


Figure 4-24: The Nano and shield mounted on the completed Watch Winder. Only three standoffs were used on the shield.

I placed the electronics directly under the bearing box, between the uprights holding the entire assembly to the base. Then, I mounted the board on a separate piece of acrylic screwed to the base with a flat-head 4-40 screw.

Keeping the Watches in the Basket

To hold the watches in the watch basket, I simply cut a block of fine foam sponge, and it worked well. If you want something a little dressier than a sponge, you can sew small pillows that you put the watch band around. I don't have any sewing ability, so I stuck with the sponge.

NOTE *The open frame works well overall, but it could collect dust. If you have a fastidious streak, you could build an acrylic box to cover the entire winder from 3/16-inch thick acrylic sheet. Or you could just buy a can of dust spray, as I did. Some have also suggested to me that the entire winder could be mounted on a piece of hardwood, such as walnut or some other decorative hardwood, to add a finishing touch.*

Design Notes

Now that you've seen how I built the Watch Winder, I'll walk you through some key design decisions I made that you might want to do differently.

Total Rotation Adjustment

It's possible to vary the total number of turns the Watch Winder makes without changing the sketch, though I chose not to do this

You can use a potentiometer to create a variable voltage and input it to one of the analog inputs of the Arduino. Then, you can substitute that value for one of the delays in the sketch to vary the number of revolutions per day. Here's how to install the potentiometer and the tweaks you'd need to make to the sketch.

Hardware Changes

Connect the upper and lower terminals of a potentiometer to the positive and negative rails of the system. Solder pads have been included on the shield for this purpose. You needn't use a full-size potentiometer; a small trimmer (10 turn is best) will do nicely, and it saves a lot of space. Connect the potentiometer's center pin (in the shield) to an analog pin of the Nano. Because the Hall effect switch uses pin A0, I suggest using analog input A1. Solder points have been provided in the shield.

Software Changes

On the sketch, there are several things you must do. First, tell the Nano that input A1 is in play. Go to the top of the sketch to add the following:

```
const int revSet = A1
```

Then, a little further, identify the value the potentiometer is set at as follows:

```
int revNumber = 0
```

`revSet` is the arbitrary name I gave to the input A1, and `revNumber` is the arbitrary name I gave to the number you will substitute in the sketch for one of the delays.

The potentiometer will give values from 0 to 5 volts. Because the analog input, A1, is connected to a 10-bit ADC, it will generate 1,024 digital values between 0 and 1,023. In other applications, it's been necessary to map these 1,024 values to some other set of values. However, in this particular situation, it's easiest to use the values as is.

In the sketch file, move to the line after `void loop() {` and assign the value of A1 to `revNumber` as follows:

```
revNumber = analogRead (revSet);
```

Go back to where we define some of the delays in the sketch. Change

```
int timer = 500
```

to

```
int timer = 0
```

Finally, go back to where you entered `revNumber = analogRead (revSet);` and after that, enter the following:

```
timer == revNumber
```

Now, each time the sketch calls for the timer value, the `revNumber` value should be used automatically, which will give you a wide variation in delay. The resulting variation runs from 200 to 1,200 revolutions per day.

How Many LEDs to Use and Where to Put Them

I originally imagined the Watch Winder having only two LEDs to indicate the direction of rotation. The first version used LEDs attached to the motor-direction pins, D12 and D13, of the Nano (see Figure 4-2 on page 68). One pin was on for the duration of rotation in one direction, and vice versa. Red and green LEDs were used to indicate which direction the winder was going in, like running lights on a boat.

But that's still pretty boring, and there were all those pins sitting there not doing anything. Furthermore, if you invited a friend over to see your winder, it would sit there, doing nothing most of the time—and so would your visitor. So I decided to spice up the project with several more decorative LEDs. I also decided to add more variability by having the sketch

provide some animation, calling for the watches to turn a varying number of times—sometimes shorter but more often. I even added a "ping-pong effect."

Because the half of my brain dealing with artistic matters apparently never developed, I'll leave the placement of the LEDs to you. On the unit in Figure 4-1 on page 67, there are four in the bearing box and five in a lightbar, but you could place them anywhere.

I arbitrarily chose nine LED channels, and in some cases, I use two LEDs per channel. I used two LEDs for each direction of the motor—the two channels, D12 and D13, that serve double duty driving the LEDs as well as driving the motor. D2 and D13 power two LEDs each. D4 through D10 power the other LEDs, the two behind the watch basket—D9 and D10, each with two LEDs—and the five out in front in the lightbar—D4–D8. D11 is reserved for future developments you may want to include.

Motor Voltage

One beta tester of the Watch Winder experienced difficulty running a 6V motor from the 5V supply. The complaint was insufficient torque. The solution, should you run into this problem, is to run the second supply (V_{CC2}) of the H-bridge directly from the 9V supply. To do this, you are either going to have to cut the traces to pin 8 or remove that pin and solder it separately to the 9V supply. Because the motor runs so intermittently, there is little risk of burning it out. It may not be an elegant solution, but it works. Incidentally, out of about 20 different motors sampled, that was the only one that experienced that problem. And as addressed earlier, the higher speed of the motor at the higher voltage will have no, or little, effect on the number of rotations.

How Many Rotations Does the Watch Winder Make?

If you really have to know how many rotations the Watch Winder makes, here's a solution. The internal counter serves to sequence the sketch but does not accurately reflect the total number of revolutions the motor makes. While we could have counted the rotations internally, it would have required a separate readout or being hooked up to the serial monitor. But if you need to keep count, you can add a small external counter. Because you will need it only on rare occasions when changing the revolution count, you can plug it in—a provision is made in the shield—when you need it and save it for other projects when you don't. The external counter in Figure 4-25 is self-powered and costs under $8. See the "Parts List" on page 69 for details.

The external counter is not required; however, it could be a nice accessory to include for this and other projects. It is not included in the design because it is used only on occasion and can be plugged in. I used a two-pin female header on the shield—the connections are labeled *GND* and *X-ctr* on the screen layer of the shield—and included a two-pin plug on wires from the counter. The count connection goes from ground to pin 4 of the counter and from the Hall effect sensor to pin 1. You can add a reset button

from ground to pin 3 of the counter. The counter comes from the manufacturer with little information, so Figure 4-22 shows a view from the back with the push-button reset on the counter on your right.

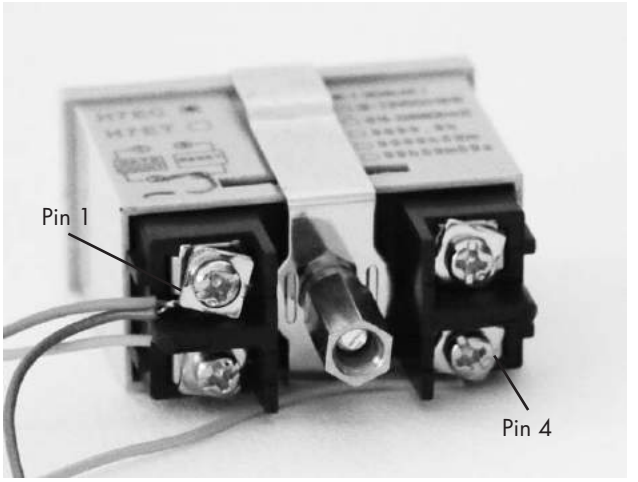


Figure 4-25: An external hardware counter with a reset button added

Closing Thoughts

Even if this Watch Winder didn't keep my watches wound, I think it would still be a great sculpture. And when you are done with this project, perhaps your next Arduino build will be just that: a kinetic, blinking, moving piece of art.

5

THE GARAGE SENTRY PARKING ASSISTANT

This project is a reliable electronic device to gauge the distance you need to pull your car into your garage. If you park in a garage, you're probably familiar with the problem: how far do you pull your car into the garage to make sure there's room in front for whatever is there and enough space behind so the garage door will close? Some people suspend a tennis ball on a string from the ceiling and stop at the point when the ball meets the windshield. That works fine, but the ball is a pain to set up and adjust, and it often gets in the way if you want to use the garage for something other than parking the car.

Arduino offers a better solution. This Garage Sentry project is the electronic version of the classic tennis-ball-on-a-string device, only better. The Garage Sentry accurately detects when your car reaches exactly the right position in the garage and sets off an alarm that blinks so you know when to hit the brakes.

In addition, at the end of the chapter, I'll show you how to modify the basic Garage Sentry into a deluxe version that alerts you when you're getting close to the perfect stopping point.

INSPIRATION BEHIND THE GARAGE SENTRY

This project evolved out of playing with an *ultrasonic transceiver module*, a device that emits sound waves and then detects them after they travel to an object, reflect off that object, and travel back to the module. The output of the module allows a microcontroller to measure the time it takes to travel to and from the object and, knowing the speed of sound, determine the distance. To test the ultrasonic transceiver's sensitivity and limits, I used the battery-operated breadboard version in my garage, which had enough space to move objects around for different distances. It turns out cars are great reflectors for ultrasonic energy. From this experimentation, I was inspired to turn my test apparatus into a Garage Sentry.

Required Tools

This project doesn't require many tools or materials, but you will need the following tools for both the standard and deluxe versions:

- A drill with a 3/8-inch or 1/2-inch chuck (powered by battery or with 110/220V from the wall)
- Drill bits for potentiometer (9/32 inches), power input (1/4 inches), and LED (3/8 inches)
- Soldering iron and solder
- Tapered reamer
- Screwdrivers (See the Introduction for screwdrivers you should have on hand.)
- Pliers (I recommend needle nose.)
- 28- or 30-gauge hookup wire
- (Optional) Wire-wrap tool and wire
- (Optional) 1/4-inch tap

Parts List

You'll need the following parts to build the basic Garage Sentry:

- One Arduino Nano
- One HC-SR04 ultrasonic sensor (Try eBay, Adafruit, Sparkfun, and so on.)

- Two high-intensity LEDs (>12,000 MCD; available from eBay and other online stores)
- Two 1/4 W (or more), 270-ohm resistors (to limit current to the LEDs)
- One 1/8 W, 20-ohm potentiometer
- Two NPN-signal transistors rated for a collector current of at least 1.5 A (I used ZTX-649 transistors, which you can find at Mouser, Digikey, and Newark.)
- One enclosure (I recommend a blue Hammond 1591 ATBU, clear 1591 ATCL, or something similar.)
- (Optional) One 0.80-inch aluminum strip for mounting bracket
- (Optional) Two 1/4-inch × 20-inch × 3/4-inch bolts with nuts
- One section (approximately 1 inch × 1 inch) perforated board (can include copper-foil rings on one side)
- One 3.5 mm jack for power
- Two 2-56 × 3/8-inch screws and nuts
- Two additional 2-56 nuts to use as spacer
- One 9V, 100 mA plug-in wall adapter power supply (Anything from 7.5V to 12V DC at 100 mA or upward should work well.)
- One length double-sided foam tape
- One LM78L05 (TO-92 package) regulator (for the breadboard build only)

Because the basic version doesn't require a lot of additional components, I suggest building the circuit on a standard perforated circuit board instead of a shield. To power your circuit, you can use a 9V, 100 mA wall adapter plugged into a 3.5 mm jack (see Figure 5-1). You shouldn't need an on/off switch.

Figure 5-1: I used a Magnavox AC adapter, but any similar power supply with a DC output from 7.5V to 12V should work. These are readily available online and cost from under $1.00 to about $3.00.

Be sure to use two bright LEDs that are clearly visible, even when a car's headlights are on. Bright LEDs range from 10,000 MCD (millicandela) to more than 200,000 MCD. The brighter, the better; just remember that brighter LEDs require more power, so the current-limiting resistor will need a higher power rating for the brighter lamps. The 270 W current-limiting resistors result in a current drain of about 30–40 mA each with the 12,000 MCD LEDs I used at 5V. (Remember that power equals volts times amps, or $P = VI$, so at 40 mA and 5V, you'd have 0.20 W.) It's best to use a 1/2 W or greater resistor even though you can easily get by with a smaller value—as I did with 1/4 W—because the LEDs are on only intermittently.

Optional Parts

In addition to the components for the basic Garage Sentry, you'll need the following extra components if you want to build the deluxe version:

- Two high-intensity green LEDs
- Two high-intensity amber LEDs
- Two additional 270-ohm resistors (1/4 W)
- Two additional transistors (ZTX-649)
- One enclosure Hammond 1591 BTCL (to replace the 1591 ATCL)
- One PCB (shield)

Downloads

- Sketches: *GarageSentry.ino* and *GarageSentryDel.ino*
- Drilling template: *Transducer.pdf*
- Drawing: *Handle.pdf*
- Shield file for Deluxe Garage Sentry: *GarageSentreDel.pcb*

Basics of Calculating Distance

This project measures the time it takes for a sound to originate, bounce off an object, and be received back at the point of origin, and it uses that time to calculate the distance between the object and the sensor.

The basic distance calculation is not much different from determining the distance of a storm by counting the seconds between a lightning flash and a thunderclap. Each second represents a distance of 1,125 feet, or about 0.2 miles. Given that sound travels at 1,125 feet per second in air at sea level, if there's a five-second delay between a lightning flash and the thunderclap, you can determine that the storm is roughly a mile away. In the case of the Garage Sentry, once you know how long it takes for the sound to make a

round trip and know the speed of sound, you can calculate the distance according to the time-speed-distance formula:

$$Distance = Speed \times Time$$

How the Garage Sentry Works

This project takes advantage of *ultrasonic sound*, which, unlike thunder, is above the hearing range of most individuals. If your hearing is good, you can detect sound ranging from about 30 Hz to close to 20 kHz, although hearing attenuates quickly above 10 kHz or 15 kHz.

NOTE *For reference, middle C on the piano is 261.6 Hz. Young children (and most dogs) can often hear high frequencies, but hearing, especially in the upper registers, deteriorates quickly with age.*

The ultrasonic transceiver module used in this project sends out pulses at a frequency of about 25 kHz and listens for an echo with a microphone. If there is something for the signal to bounce off, the system receives the return echo and tells the microcontroller a signal has been received and to calculate the distance. For the Garage Sentry, the unit is placed in the front of the garage, and the signal is sent out to bounce off the front—or rear if you are backing in—of your vehicle. To calculate your car's distance from the ultrasonic transceiver, the Arduino measures the time it takes for the signal's round trip from the transceiver to the target and back. For example, if the Arduino measures a time of 10 milliseconds (0.010 seconds), you might calculate the distance as:

$$\text{Distance} = 1{,}125 \text{ ft/s} \times 0.010 \text{ s} = 11.25 \text{ ft}$$

Ah, but not so fast. Remember the signal is traveling to the car and then back to the microphone. To get the correct distance to the vehicle, we will have to divide by two. If the controller measures 10 ms, then the distance to your car would be:

$$\text{Distance} = \frac{1{,}125 \text{ ft/s} \times 0.010 \text{ s}}{2} = 5.625 \text{ ft}$$

The HC-SR04 ultrasonic module sends out a signal at the instruction of the Arduino (see Figure 5-2). Then, the sketch instructs the transmitter to shut down, and the microphone listens for an echo.

If there is an object for the signal to bounce off, the microphone picks up the reflected signal. The Arduino marks the exact time the signal is sent out and the time it is received and then calculates the delay.

The HC-SR04 module is more than a speaker and microphone, though. The module includes transducers—a loudspeaker and mic—and a lot of

electronics, including at least three integrated circuits, a crystal, and several passive components. These components simplify its interface to the Arduino: the 25 kHz tone is actually generated by the module and turned on and off with the microcontroller. Some of the components also enhance the receiver's, or the microphone's, sensitivity, which gives it a better range.

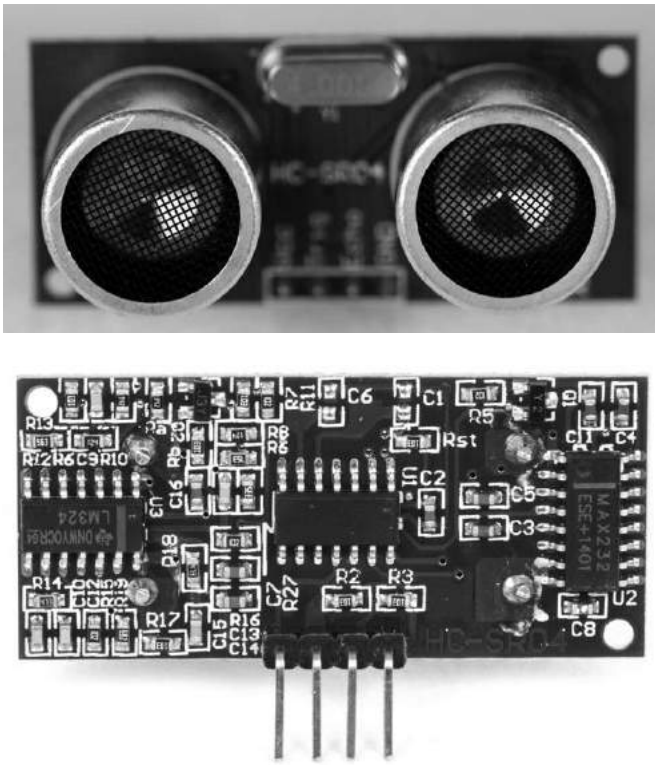


Figure 5-2: The ultrasonic sensor module. The back of the module (bottom) has connection terminals at the bottom.

The range of the HC-SR04 ultrasonic transducer is approximately 10 to 12 feet. The returning signal is always a lot weaker than the transmitted signal because some of the sound wave's energy dissipates in the air (see the dotted lines in Figure 5-3).

The arithmetic to calculate the distance between the sender and the object is not difficult. You take the number of microseconds it takes for the signal to return, divide by the 73.746 microseconds it takes sound to travel an inch, and then divide by two because the signal is going out and coming back. The full arithmetic for this appears later in "Determining Distance" on page 117.

The sketch provides a response in inches or centimeters depending on your preference. We'll use inches for setting up the distance for the alarm, but converting to centimeters simply requires a remapping of the analog input and setting the numbers a bit differently. The sketch also does the basic arithmetic for determining the centimeter measurement for you.

With the high-level overview out of the way, let's dig in to how you'll wire the Garage Sentry.

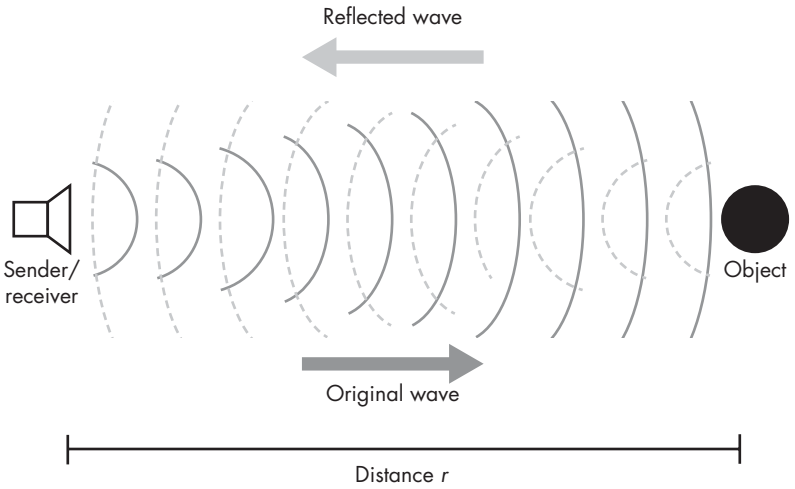


Figure 5-3: In this project, sound is transmitted from a sender, bounces off an object, and is received.

The Schematic

Figure 5-4 shows the schematic for the Garage Sentry.

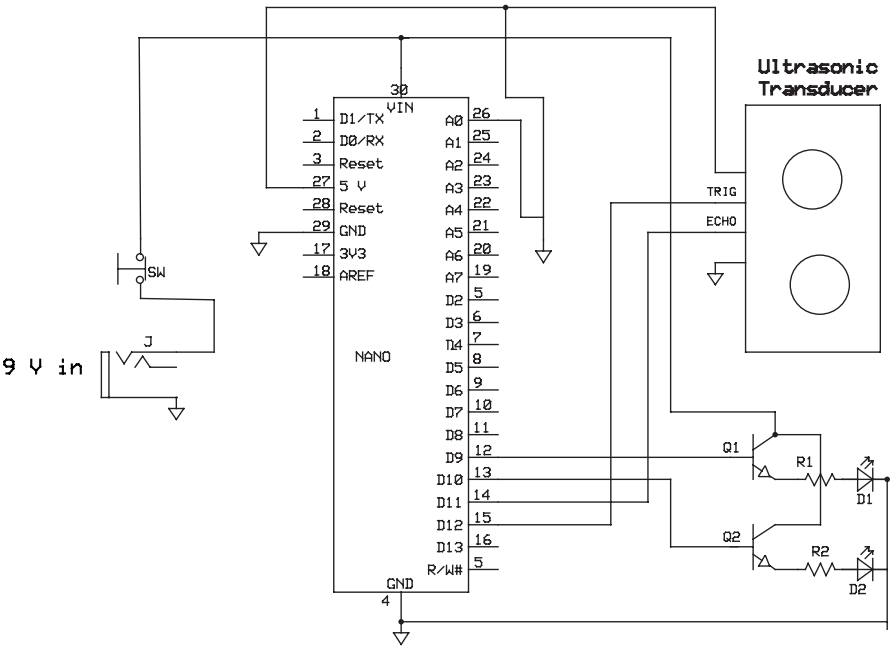


Figure 5-4: Schematic diagram of the Garage Sentry

R1 and R2 are the 270 W resistors for the LEDs and should be 1/4 W or larger. If a higher wattage resistor is not available, you could place several resistors in parallel to gain the required wattage. First, find the right resistor value with the formula:

$$\frac{1}{R_{total}} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \ldots + \frac{1}{R_n}$$

You can also use an automatic calculator, such as the one at *http://www.1728.org/resistrs.htm*, which is a lot easier than doing the math yourself.

To avoid extra calculations, select resistors of the same value. This way, the same amount of current flows through each one. For example, two 1/8 W resistors in parallel will give you a 1/4 W value.

If you do use resistors of different values, you will have to calculate the current flowing through each and the total dissipation.

This schematic also leaves you with room to customize your alarm. While this version of the project uses LEDs to create a visual alarm, with a slight modification, you can easily create an audible alarm as well. Simply replace either the red or blue LED with an audible device, such as a Sonotone Sonalert, and the alarm will sound. To replace an LED, you would need to connect the Sonalert across that LED's connections; just make sure to get the polarity correct. Alternatively, you could keep both LEDs and add an audible device for a third warning.

NOTE *In this project, the Nano takes advantage of its on-board voltage regulator, which is why there's no external regulator in the schematic.*

The Breadboard

The entire Garage Sentry fits on a small breadboard, so you can set it up, program it, power it with a battery, and walk around to test it out. As you play with it, I'm sure other applications of ultrasonic technology will come to mind. The breadboard I assembled appears in Figure 5-5.

In Figure 5-5, the breadboard is powered by a 9V battery. Usually, you could wire the battery directly to the VIN of the Nano and use the Nano's built-in voltage regulator. But you'll power the Nano with a USB cable when you program and test it for the first time, so on the breadboard, you'll set up the positive and negative rails for 5V for both the Nano and the ultrasonic module. To avoid risking damage to the Nano or the module and avoid overcomplicating the build, I included a single-chip external voltage regulator (LM78L05) so the entire breadboard runs on 5V. Take a look at Figure 5-6 to see how it's wired up.

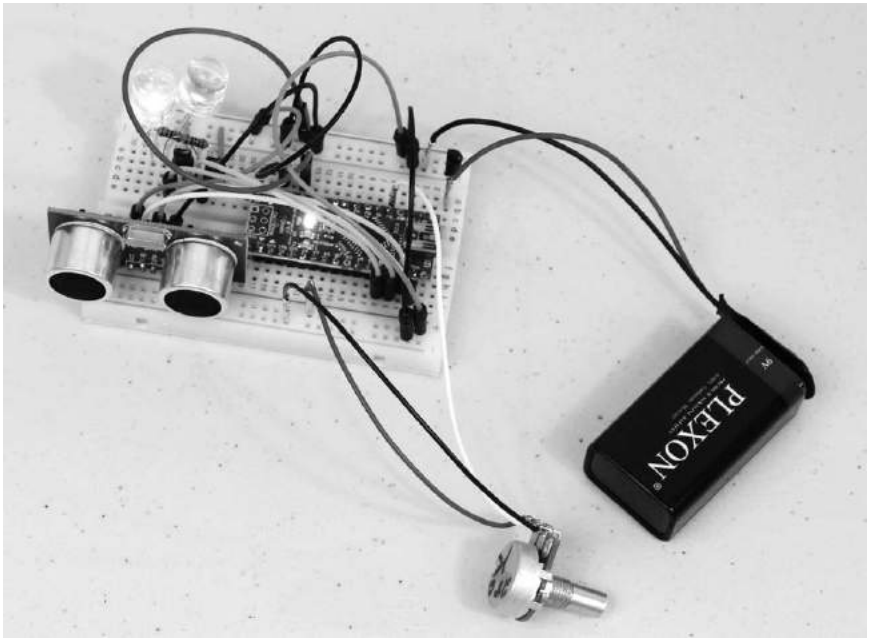


Figure 5-5: Here's the breadboard wired up. I used a 9V battery so I could experiment in different environments. Both LEDs look illuminated because of the length of the exposure of the camera.

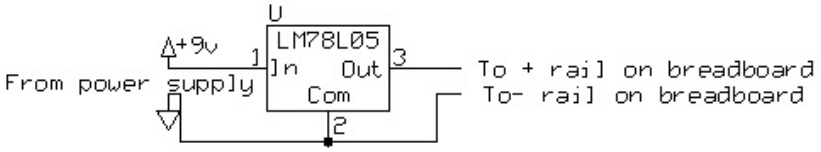


Figure 5-6: This is how the LM78L05 TO-92 regulator is wired up on the breadboard. Bypass/filter capacitors are not required.

Here's a blow-by-blow list of the steps to wire the breadboard:

1. First, put the ultrasonic module at the lower end of the breadboard facing out, and plug the Nano in to the breadboard, leaving four rows of connections above it.
2. Make sure the positive and negative (red and blue) strips on the left and right are connected properly—red to red, blue to blue. If you connect red to blue, it will cause a major problem.
3. Connect the red strip to the 5V power supply (pin 27 of the Nano, labeled *5V*). This is necessary if you are operating from the USB connector.

4. Connect pin 4 of the Nano (labeled *GND*) to the breadboard's negative rail (blue strip).
5. Connect VCC of the HC-SR04 transducer to the positive rail.
6. Connect GND of the HC-SR04 transducer to the negative rail.
7. Connect TRIG of the HC-SR04 transducer to pin 15 (D12) of the Nano.
8. Connect ECHO of the HC-SR04 transducer to pin 14 (D11) of the Nano.
9. Insert two ZTX-649 transistors into the breadboard. Select an area where all three pins of each transistor can have their own row.
10. Connect pin 12 (D9) of the Nano to the base of transistor Q1.
11. Connect pin 13 (D10) of the Nano to the base of transistor Q2.
12. Connect the collectors of both transistors to the positive rail (red strip).
13. Connect the emitter of transistor Q1 to one end of a 270-ohm resistor.
14. Connect the other end of the 270-ohm resistor connected to the emitter of transistor Q1 to a blank row on the breadboard.
15. Connect the emitter of transistor Q2 to one end of another 270-ohm resistor. Connect the other end of the 270-ohm resistor connected to the emitter of transistor Q2 to another blank row on the breadboard.
16. Connect the + (long end) of LED (D1) to the 270-ohm resistor and the other end to ground (blue strip).
17. Connect the + (long end) of LED (D2) to the second 270-ohm resistor and the other end to ground (blue strip).
18. Connect one end of the 20 kW potentiometer to the positive rail (red stripe).
19. Connect the opposite end of the potentiometer to the negative rail (blue stripe).
20. Connect the wiper (center) of the potentiometer to analog pin A0 (26) of the Nano.

You should be good to go! If you use the AC connection, simply connect it to the VCC connection of the Nano.

To add a battery connection, include the 78L05 with its center pin to ground (negative rail), the input to the positive side of the battery, and the output to the positive rail (see Figure 5-7). Connect the negative terminal of the battery to the negative rail.

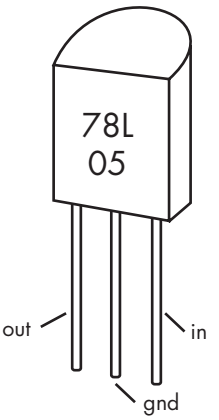


Figure 5-7: Pinout of the 78L05 voltage regulator

The Sketch

Once the breadboard is complete, the sketch can be loaded onto the Nano. Download the *GarageSentry.ino* file from *http://www.nostarch.com/arduinoplayground/*. To load the file onto the Nano, follow the instructions

outlined in Chapter 0 on page XX. Remember to select the correct board type. Once it's loaded, the unit is ready for experimentation.

The sketch for the Garage Sentry serves several functions. It tells the ultrasonic sensor to generate a wave and detects how long it takes the echo to return. It then calculates the distance based on that time and, if necessary, alerts you to stop by turning on the LEDs. Here's the sketch in full; I'll walk you through it next.

```
/* Garage Sentry 3b
*/

int ledPin = 10;
int ledPin1 = 9;
int count;
int analogPin = A0;
int val;
int y;

void setup() {

  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(analogPin, INPUT);
}
void loop() {
  val = analogRead (analogPin);
  long duration, inches, cm;
  //Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
❶ pinMode(12, OUTPUT);  //Attach pin 12 to Trig
  digitalWrite(12, LOW);
  delayMicroseconds(2);
  digitalWrite(12, HIGH);
  delayMicroseconds(5);
  digitalWrite(12, LOW);

  pinMode(11, INPUT);  //Pin 11 to receive Echo
  duration = pulseIn(11, HIGH);

  //Convert the time into a distance
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);
  val = map (val, 0, 1023, 0, 100);
  if(inches == 0)
    digitalWrite(ledPin, LOW);

  if(count == 0 && inches > 0 && inches < val) {
❷   for(y = 0; y < 200; y++)
    {
      digitalWrite(ledPin, HIGH);
      digitalWrite(ledPin1, LOW);
      delay(100);
      digitalWrite(ledPin, LOW);
```

```
      digitalWrite(ledPin1, HIGH);
      delay(100);
    }
    count = count + 1;
  }

  digitalWrite(ledPin1, LOW);
  if(inches > 10) {
    //delay(1000);
    count = 0;
  }
  Serial.print(inches);
  Serial.print("   inches ");
  Serial.print(count);
  Serial.print(" count   ");
  Serial.println();
  Serial.print(" Val       ");
  Serial.println (val);
  delay(100);
}
long microsecondsToInches(long microseconds)
{
  return microseconds / 74 / 2;
}
long microsecondsToCentimeters(long microseconds)
{
  return microseconds / 29 / 2;
```

First, we define several variables, establish parameters, and load libraries (if any). In this case, define `LED1pin` and `LED2pin`, which will serve as the alarm. Other definitions (`int`) include `cm` and `count` (a variable that will be used internally), `analogPin` (as A0), `val` (to hold the limit information), and `y` (used in the loop).

Inside the setup() Function

Next is the `setup()` function. Here, you set up Arduino features that you might want to use; this sketch includes the serial monitor, which you probably will not need in the final product but is often useful in debugging code, particularly if you want to change the code. This sketch sets the rate of the monitor at 9600 baud, which is standard in many applications. It also defines the mode of the pins you'll use as either input or output. You could set the `pinMode` values at almost any point in the code, including before or inside the setup; they're also often defined within the main loop, particularly if the definitions are expected to change.

Inside the loop() Function

The `loop()` function is where everything really happens. The loop continually executes unless it's delayed or halted by a command. So even when it appears that nothing is happening, the controller is continually cycling

through the code. In this application, one of the first tasks the controller performs in the loop is to set the variable `val` to store the input from the potentiometer connected to the analog pin (`analogPin`).

In order to initiate the ultrasonic module's transmit/receive function, the sketch first calls for a low signal to be sent to the transmitter (`Trig`) to purge the module to assure that the following high signal will be clean. You can see this in the lines starting at ❶.

Next, there's a delay to let things settle before the sketch writes a high to the transmit pin, which orders the transmitter to transmit an ultrasonic signal. This is followed by another delay, and then the sketch drives digital pin 12 low to turn off the transmitter and activates the receiver by calling the `pulseIn()` method.

Determining Distance

If there's no echo—that is, if `inches == 0` or inches approaches infinity—the controller continues to run the code until it reaches the end and then starts again at the beginning. If it detects an echo, the number of microseconds between turning the transmitter on and receiving signal (`duration`) is then converted to both inches and centimeters. This gives us a measurement of how far the transceiver is from the object. Note that throughout this explanation, I will refer to inches, but you could follow along in centimeters, too.

The `microsecondsToInches` and `microsecondsToCentimeters` commands convert the time measurement to inches and centimeters, respectively, according to the arithmetic discussed in "How the Garage Sentry Works" on page 109. The data type `long` is used, as opposed to `int`, because it provides 4 bytes of data storage instead of just 2, and the number of microseconds could exceed the 2-byte limit of 32,767 bits. So far, so good.

In a regular formula, the distance arithmetic looks like this:

$$\text{inches} = \frac{\text{time}}{2} \div \frac{74\,\mu\text{s}}{\text{inch}}$$

$$\text{centimeters} = \frac{\text{time}}{2} \div \frac{29\,\mu\text{s}}{\text{centimeters}}$$

In either case, we first divide by 2 because the signal travels from the transducer to the target and back, as previously discussed. In the inches function, we then divide the halved number of microseconds by 74, and in the centimeters function, we divide by 29. (It takes 74 µs for the signal to travel 1 inch, and 29 µs for it to travel 1 cm; I arrived at those numbers by following the arithmetic in "Time-to-Distance Conversion Factors" on page 118.)

TIME-TO-DISTANCE CONVERSION FACTORS

You could simply trust my math and copy the time-to-distance conversion code, but you can apply this arithmetic to any project using a similar ultrasonic module or other sensor, so I encourage you to work through the math yourself.

As I describe in "How the Garage Sentry Works" on page 109, the speed of sound is roughly 1,125 feet per second. Multiply that by 12 inches per foot to get 13,500 inches per second.

To get the number of seconds per inch, you simply divide this value by 13,500 inches:

$$\frac{1\text{ s}}{13{,}500\text{ in}} = 0.000074\text{ s/in}$$

It takes about 74 microseconds, or 0.000074 seconds, for sound to travel an inch. To determine the distance in centimeters, go through the same exercise, but use 343 meters per second for the speed of sound, multiply it by 100 centimeters per meter, and take the reciprocal.

Triggering the Alarm

The sketch is not done yet. Now we have to look at the number of inches (or centimeters) measured and compare it to the predetermined value—`val`, in this case—to see whether the alarm should be activated. To establish the variable `val` as a numeric value, take a potentiometer (R2) straddling the power supply on either end and tie the wiper to pin A0 (see Figure 5-4). Because A0 is the input to a 10-bit analog-to-digital converter, it converts that voltage (between 0V and 5V) to a numeric digital value between 0 and 1,023. Reading that value with an `analogRead` command results in a value between 0 and 1,023 depending on the position of the potentiometer.

That value is then used to establish the trigger point for the alarm. But allowing all 1,024 values would essentially allow the distance to be set from 0 to 1,023 inches. Because the control rotates only 270 degrees, to adjust between, say, 40 and 42 inches would represent a very minuscule rotation—beyond the granularity of most potentiometers.

To scale this for the potentiometer, the sketch maps the value so the entire rotation of the potentiometer represents a distance of only about 100 inches with the following line of code:

```
val = map (val, 0, 1023, 0, 100);
```

Mapping the potentiometer value changes the maximum distance from 1,023 inches down to 100 inches while leaving the minimum distance of 0 inches unchanged. You can map any set of values so the Garage Sentry's target distance can be from *X* to *Y*, with full rotation of the potentiometer, so when you set up your Garage Sentry, you may want to test it and this range until it's right for your garage.

A conditional control structure sets the limit for the alarm. This structure makes sure that the LED is turned off when the measured distance is 0, regardless of whether the sketch is using inches or centimeters. First, the value `inches` is compared to `val` in the following expression:

```
count == 0 && inches > 0 && inches < val
```

If this statement is true, the alarm is set off and the `for` loop at ❷ is activated (see page 115), which alternately blinks the LEDs 200 times before timing out and turning the LEDs off.

The `for` loop just counts from 0 to 200, but that can be easily changed. After each count, it turns on an LED, delays briefly and turns off the same LED, delays slightly and turns on a second LED, delays slightly and turns off the LED, and then goes to the next count. At the end of the 200 count, the system turns off the LEDs and the program continues to the next line where it is reset. That is, the program starts again at the beginning.

Construction

The trickiest part of the Garage Sentry is mounting the ultrasonic module on the box. Because the module can send out sound waves only in a straight line, you need to be able to adjust its direction so that the ultrasonic sensor can hit its target and receive the echo. But the module includes only two mounting holes, diagonally opposed from each other, so there's no easy way to fasten it to a flexible mounting. We'll tackle that first.

Drilling Holes for the Electronics

To solve this problem, I mounted the transceiver directly to the box and just aimed the box as required. To mount the module, drill 5/8-inch holes in the mounting box and use standoffs to hold the board securely. See the template in Figure 5-8 for drilling measurements. A PDF of the template is available in this book's online resources at *http://www.nostarch.com/arduinoplayground/*, in case you want to print it and lay it over your box as a guide. The box I recommend is made of polycarbonate plastic and is less likely to crack than styrene or acrylic; however, it tends to catch the drill, so be careful.

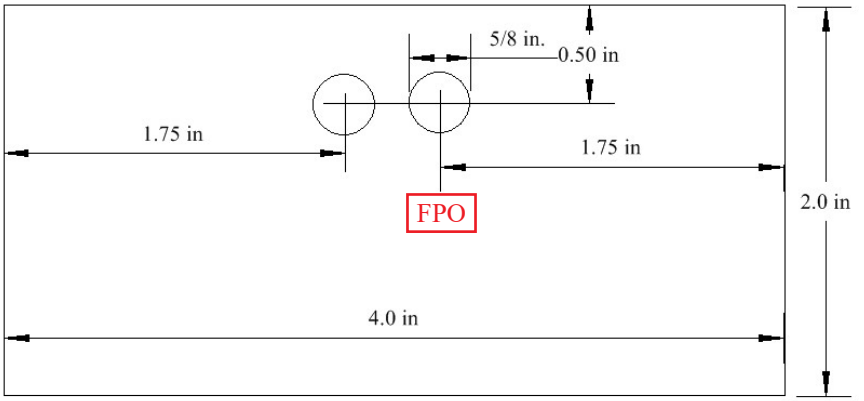


Figure 5-8: Template for drilling transducer holes

There are several ways to drill the 5/8-inch holes. If you are good at drilling, you could simply use a 5/8-inch drill bit and bore the holes directly. But I discovered that the holes can be bored safely and easily by first drilling a hole about 1/4 to 3/8 inches in diameter and then enlarging it with a tapered reamer, available from Amazon for under $15. The larger reamer in the Amazon set will ream a hole up to 7/8 inches in diameter, and it is handy to have around for other projects. Use a 1/8-inch drill to drill the holes for the standoffs, as shown in the drawing, which you can use as a template.

If you ream out the hole, make sure to ream from both sides. Enlarge the hole to a size that holds the transducer elements tightly—but not too tightly. While this is not the most precise way to bore a hole and would probably be frowned upon by professional machinists, it works well enough here.

WARNING *Regardless of the size of the hole, do not hold your work piece with your hand when drilling. Always clamp it securely. If the drill binds, the work will want to spin or climb up the drill. Drill at a slow speed and go gently.*

Next, drill the holes for the potentiometer, power jack, and two LEDs. Select a drill size based on the particular power jack and potentiometer you have. I used a 9/32-inch drill for the potentiometer, a 1/4-inch drill for the 3.5 mm jack, and a bit of approximately 25/64 inches for the LEDs. The size of the 10 mm LEDs tends to vary a bit from manufacturer to manufacturer, so I would recommend that you select a smaller drill bit, say 3/8 inches, and ream until the LED fits tightly. Because the LED is tapered, ream from the rear of the box so that the LED will fit better.

The location of both the potentiometer and power jack is not important, but make sure that neither crowds the transducer or Nano. You want them to be on the bottom of the enclosure so that they are accessible after the box is mounted (see Figure 5-12).

Mounting Options

Before you stuff the Arduino, ultrasonic sensor, and perforated board circuit into your enclosure, figure out how you want to mount the Garage Sentry. There are several ways to mount the box onto whatever surface you need.

Velcro Strips

If you have a good flat surface to mount the assembly to, you could simply affix the box with adhesive Velcro (see Figure 5-9). Two sentries have been in place in my garage that way for several months, with no sign of slippage or deterioration.

Figure 5-9: Adhesive Velcro mounting strips used to mount the Garage Sentry enclosure

A U-Bracket That Can Be Aimed

If you don't have a good surface and need to aim the module at an angle, mounting it on a U bracket that lets the sensors swing up and down or left and right will work. In this section, I'll describe how to build the U bracket mount shown in Figure 5-10.

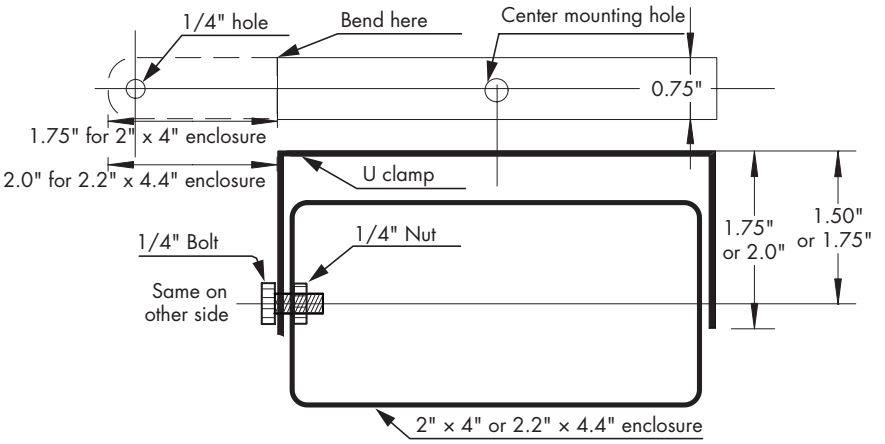


Figure 5-10: This drawing illustrates the size and shape of the optional U bracket handle and how it connects to the enclosure. Where you see two measurements for a single dimension, the smaller applies to the Standard Garage Sentry, while the larger applies to the deluxe version.

To make the U bracket for the 1591 ATCL 2 × 4-inch box, take a strip of 3/4-inch × 0.080-inch ×5 1/2-inch long aluminum (available at Ace Hardware, Home Depot, or Lowe's), and drill 1/4-inch holes 5/16 inches from the ends of the aluminum strip. Drill corresponding holes with a No. 7 or 15/64 drill in the side of the enclosure centered on the ends, and thread the holes with a 1/4-inch-20 tap. Bend the aluminum strip 1.5 inches from each end for the standard version and 2 inches for the deluxe version (see Figure 5-10). Using a vise is the easiest way to bend the metal, but if that's not convenient, you can sandwich it between a bench and piece of metal, clamp it down, and bend it by hand (see Figure 5-10).

For the U bracket for the 1591 BTCL 2.2 × 4.4-inch box, use a 6 3/8-inch long strip of the same material, and drill the 1/4-inch holes 1/2 inches from the ends. Then, bend the aluminum at right angles at 3/4 inches from either end for the standard version and 1 inch from each end for the deluxe version.

To fasten the U bracket to either enclosure, you can start by drilling a hole in the center of each end of the enclosure. It's simplest to drill a No. 7—15/64 is close enough—hole at either end of the box. The easiest way to center the holes is to draw a line along each diagonal on both ends. Where the lines intersect is the center. Thread the holes with a 1/4-inch-20 tap, and you'll be able to fasten the box to the U bracket directly. The threads in the thin ABS plastic will not be very strong, so be careful not to overtighten the bolts.

When you're finished attaching the bracket to your enclosure, it should look like Figure 5-11.

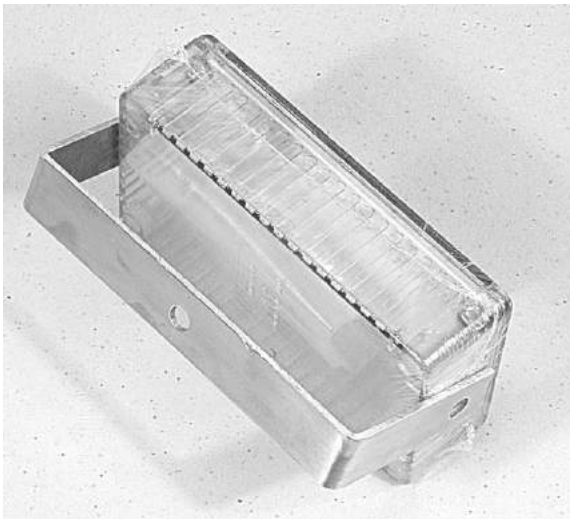


Figure 5-11: The enclosure for the Garage Sentry can be mounted with the bracket so it can be tilted or rotated to point the transducers in the correct direction.

Soldering the Transistors and Current-Limiting Resistors

After testing your circuit on a breadboard and deciding how to mount the Garage Sentry, solder the driver transistors and current-limiting resistors to a small section of perforated phenolic or FR-4 predrilled board. Use the schematic in Figure 5-4 or the instructions in "The Breadboard" on page 112 as a guide to wiring and soldering the components in the perforated board.

Make the connections in the schematic, but otherwise, there is no right or wrong way to assemble the perf board. I do recommend using perforated board with copper pads for each hole to simplify soldering. Solder all the hookup wires for the power, potentiometer, Nano, ultrasonic module, and LEDs before attempting to mount the board on the inside of the box.

When you're done soldering, mount the perforated board anywhere in the box where you can find room. I used double-sided foam adhesive, and it worked well. Mount the Nano, LEDs, and ultrasonic module next.

Wiring the Pieces Together

Finally, use 30-gauge hookup wire to connect the Nano, ultrasonic sensor, perforated board circuit, and LEDs according to the schematic in Figure 5-4. Optionally, you can use wire-wrap wire and a wire-wrap tool to wire up the sections, but it is not necessary and can be expensive if you don't already have the tool and wire.

Wiring the components and fitting them in the box may be a little messy, but it saves building a shield. Figure 5-12 shows the box as it was being assembled.

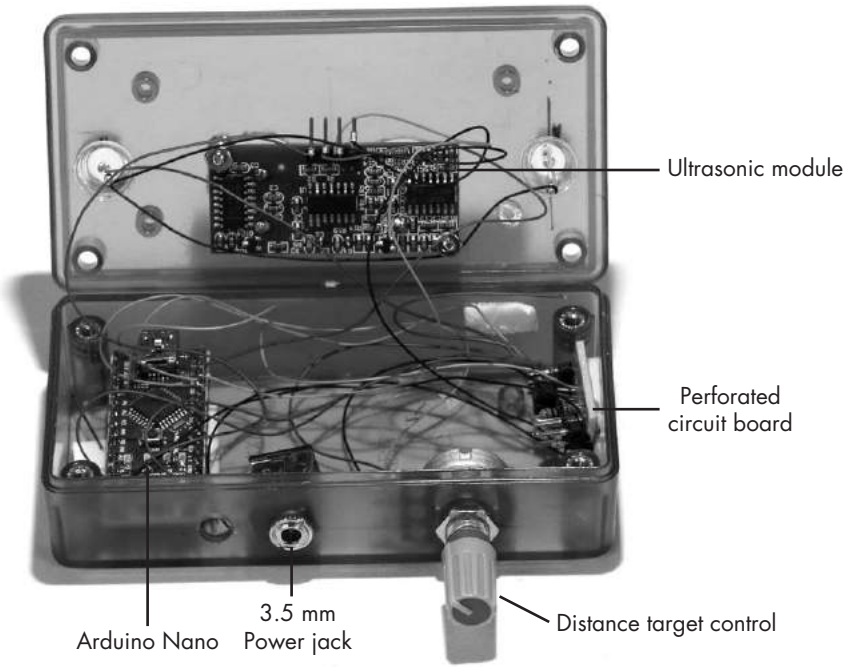


Figure 5-12: The Standard Garage Sentry uses wire wrap for the final connections.

The Deluxe Garage Sentry

That's it! Or is it?

I have been using the standard model in my garage for several months; it does what it's supposed to do and does it well. But it seems like something's missing. The alarm goes off when you reach the desired spot in the garage, but why not have it give you a little warning before you get there so you can slow down as you approach the stopping point?

The idea is to have the system warn you at some pre-established distance from the stopping point so you don't have to stop suddenly. It isn't much extra effort to add two more LEDs to go off at different distances. Figure 5-13 shows the Deluxe Garage Sentry.

Now, let's discuss how to assemble the Deluxe Garage Sentry.

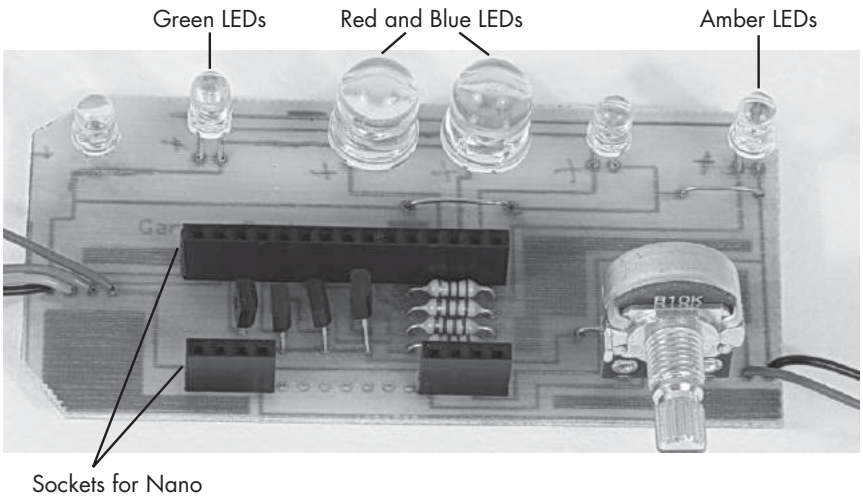


Figure 5-13: The Deluxe Garage Sentry sets off three stages of alarms.

The Deluxe Schematic

Hand-wiring everything in the standard version is tedious. So for the deluxe version, I developed a shield (PCB) that holds the LEDs, potentiometer, Nano, transistors, and current-limiting resistors (see Figure 5-13). Adding the LEDs and extra transistors required some changes in the circuitry. Figure 5-14 shows the revised schematic.

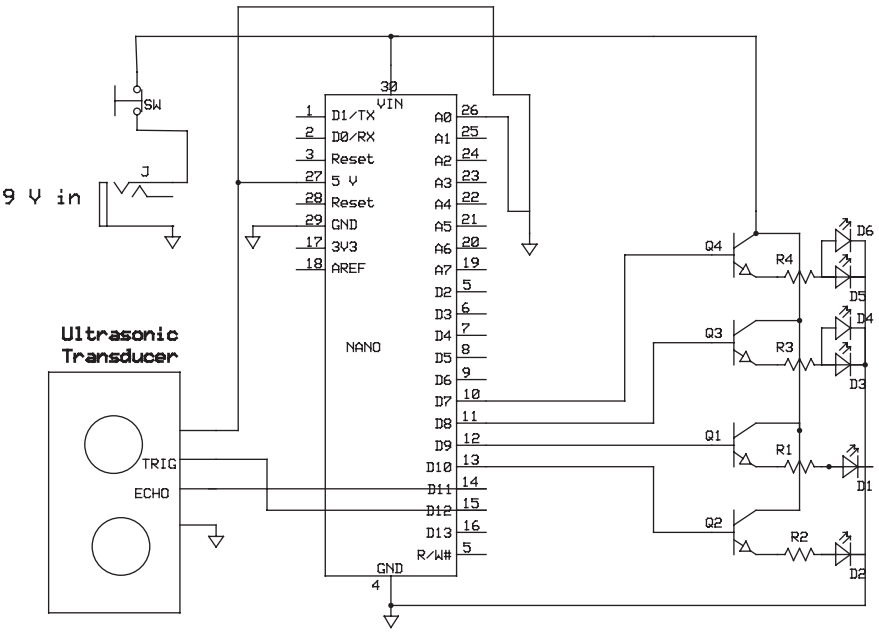


Figure 5-14: The deluxe schematic has additional LEDs, driver transistors, and current-limiting resistors on the right-hand side.

Note that this circuit drives the transistors as emitter followers. As such, the base shows a high resistance, and therefore no resistor is required between the Arduino and the resistors Q1 through Q4. If you were driving using a common emitter, however, you would need a resistor, as current would flow through the base-emitter junction, short out the driver, and burn out the transistor.

NOTE *To improve the Garage Sentry further, you could also conceivably double or otherwise increase the range using special transducers and electronics, but in this application, the 10-foot operating range is more than enough.*

A Bigger Box

Both the green and amber LEDs operate as pairs, so only a single driver transistor is required for each pair. But with all this new circuitry, the deluxe board does not easily fit in the same enclosure as the standard version.

You'll need to find a larger box for the deluxe version, which provides you with some other benefits. With a larger, clear polycarbonate enclosure, like Hammond 1591 BTCL, the LEDs can stay inside the box and still be visible so you won't have to drill holes to mate with the LEDs in the PC board. You'll have to drill only four holes: the two large holes for the ultrasonic sensor, a hole for the power jack, and one for the potentiometer. These holes make it possible to mount the ultrasonic sensor on top of the Nano board with double-sided foam tape, which is in turn mounted to the shield (see Figure 5-15). In other words, you create a sandwich with the Nano in the middle, the shield on the bottom, and the ultrasonic module on the top. This design eliminates the need for the mounting screws used in the first version.

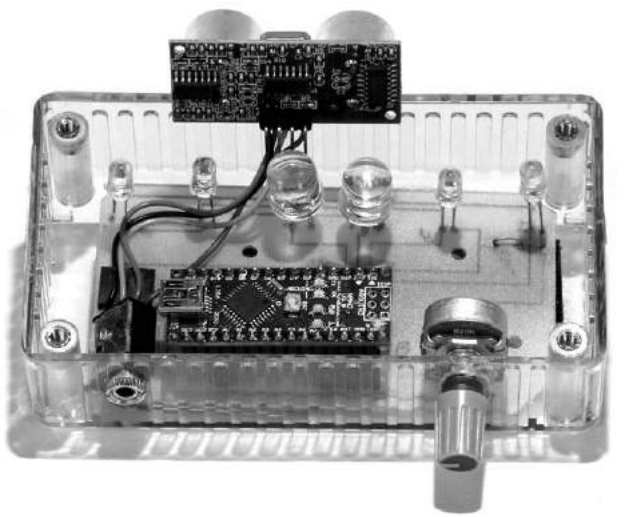


Figure 5-15: Compared to the Standard Garage Sentry, there is virtually no hand wiring in the deluxe version. The driver transistors and current-limiting resistors for the LEDs are located under the Nano board.

Simply use the same template you used in the standard version, and drill (and/or ream) the two 5/8-inch holes for the two ultrasonic elements. The shield itself can be fastened to the bottom of the box with small flat-head screws and nuts or with more double-sided foam tape. When you affix the potentiometer to the box, it should hold the board in place, as its leads are soldered to the shield. Figure 5-15 shows the deluxe version; note how much neater it is than the standard version from Figure 5-12.

Before drilling the hole for the potentiometer, carefully measure the height of the potentiometer hole and drill the hole slightly oversized so that the shaft and screw can be inserted at an angle into the box. You'll also note in Figure 5-15 that the corners of the printed circuit board have been clipped off so as not to get in the way of the studs used for the top screws of the enclosure.

I suggest mounting the power connection on the same surface of the box as the potentiometer—that is, the bottom. Here, the power connection and potentiometer will be accessible after mounting, leaving the top free to fit snugly against a shelf. The unit can just as easily be mounted upside down with the adjustment and power jack on the top.

The Shield

Figure 5-16 shows the shield for the Deluxe Garage Sentry. If you want to build this shield, download this book's resource files, look for the file *GarageSentryDel.pcb*, follow the etching instructions in "Making Your Own PCBs" on page XX, and solder your components to the board. You can also take the file and send it out to one of the service bureaus to have the board made for you.

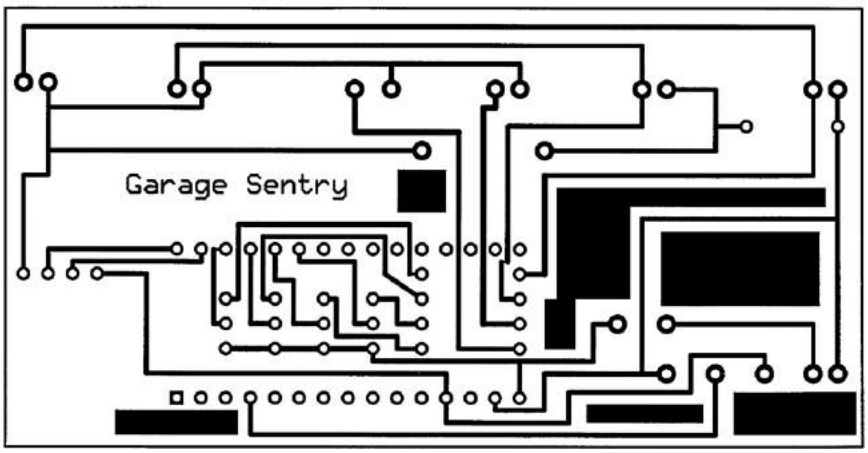


Figure 5-16: This is the shield for the Deluxe Garage Sentry, which simplifies the individual wires that had to be soldered to complete the earlier version.

The potentiometer is soldered directly to the shield, though you'll still have to solder wires to the power jack and to the ultrasonic module. As you can see in Figure 5-16, the connections for the ultrasonic sensor are located on the left-hand side.

The green LEDs are on the outermost edges, the amber LEDs are next, and the flashing red and blue LEDs are in the middle. The connections for the potentiometer are on the lower right-hand side, next to the first two connections, which are ground and VIN (from left to right). The potentiometer helps hold the shield in place in the enclosure.

This version uses high-power LEDs that draw a fair amount of current. Because the sentry uses the 5V voltage regulator on the Nano, the transistors driving the LEDs are wired directly to the 9V input voltage, allowing the unit to function without a separate voltage regulator. The LEDs are configured with the driver transistors as emitter followers, so the voltage to the LEDs will "follow" the voltage on the base of the transistor—that is, 5V—and not present LEDs with 9V.

There are several jumpers required on this shield, including the jumper for the power, which connects to the collectors of the transistors and connects the raw input to the (VIN) Nano. There are also jumpers to connect the ground to the LEDs.

I mounted the transistors and current-limiting resistors under the Nano to save some space. Also, note that the connections for the ultrasonic module in the standard version call for a right-angle female header, but doing that in the deluxe version means the length of the male part gets in the way of the LEDs, so I simply soldered the connections to the PC board and to the ultrasonic module to keep the wires out of the way.

The Sketch for the Deluxe Garage Sentry

Before you build the Deluxe Garage Sentry, download *GarageSentryDel.ino* from this book's resource files at *http://www.nostarch.com/arduinoplayground/*, and upload it onto your Arduino Nano according to the instructions provided in Chapter 0 on page XX. The sketch is basically the same as the Standard Garage Sentry sketch, but it's updated to include the new LEDs.

```
/* Deluxe Garage Sentry: goes with the shield PCB
*/

int ledPin = 8;
int ledPin1 = 7;
int ledPin2 = 10;
int ledPin3 = 9;
int count;
int analogPin = A0;
int val;
int y;
```

```
void setup() {
  // initialize serial communication:
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2,OUTPUT);
  pinMode(ledPin3,OUTPUT);
  pinMode(analogPin, INPUT);
}

void loop() {
  val = analogRead(analogPin);

  long duration, inches, cm;

  pinMode(12, OUTPUT);
  digitalWrite(12, LOW);
  delayMicroseconds(2);
  digitalWrite(12, HIGH);
  delayMicroseconds(5);
  digitalWrite(12, LOW);

  pinMode(11, INPUT);   //attached to Echo
  duration = pulseIn(11, HIGH);

  // convert the time into a distance
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);

  val = map(val, 0, 1023, 0, 100);
  //map the value of the potentiometer to 0 to 100

  if(inches == 0)
    digitalWrite(ledPin, LOW);

❶ if(count == 0 && inches > 0 && inches < val + 15)
    digitalWrite(ledPin2, HIGH);
  else digitalWrite(ledPin2, LOW);

❷ if(count == 0 && inches > 0 && inches < val + 7.5)
    digitalWrite(ledPin3, HIGH);
  else digitalWrite(ledPin3, LOW);

  if(count == 0 && inches > 0 && inches < val) {
❸   for(y = 0; y < 200; y++) //repeating blink sequence {
      digitalWrite(ledPin, HIGH);
      digitalWrite(ledPin1, LOW);
      delay(100);
      digitalWrite(ledPin, LOW);
      digitalWrite(ledPin1, HIGH);
      delay(100);
    }
```

```
    count = count + 1; //turn off instruction
  }
  digitalWrite(ledPin1, LOW);

  if(inches > 10) { //reset if inches > 10
    delay(1000);
    count = 0;
  }

  Serial.print(inches);
  Serial.print("   inches ");
  Serial.print(count);
  Serial.print(" count   ");
  Serial.println();
  Serial.print(" val       ");
  Serial.println(val);
  delay(100);
}

long microsecondsToInches(long microseconds) {
  return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds) {
  return microseconds / 29 / 2;
}
```

The most notable difference between the deluxe sketch and the standard sketch is that in the two `if-else` statements at ❶ and ❷, the green and amber LEDs are activated at different distances, based on the stopping point. For example, if the stopping point is set to 36 inches, or `Val` in the sketch, then the green LED turns on at `VAL + 15`, or 52 inches, and the amber LED turns on at `VAL + 7.5`, or 43.5 inches. This way the green LEDs will turn on when the car is 15 inches from the final stopping point, and the amber LEDs will turn on when the car is 7.5 inches from the stopping point. These numbers were selected arbitrarily, and you can change them.

The red and blue LEDs start flashing when the car has reached the stopping point. You can see how the LEDs are flashed at ❸.

Figure 5-17 shows the completed Deluxe Garage Sentry mounted on my garage workbench with the car in place.

Figure 5-17: Completed Deluxe Garage Sentry mounted on my garage workbench with the car in place

The completed sentry unit works flawlessly. Depending on your particular garage and where you place the unit, you might want to adjust the sketch to make the green and amber lights turn on at different distances. The unit pictured has been working perfectly for almost six months now, and I don't know how I'd be able to pull my car in the garage without it.

6

THE BATTERY SAVER

This project was actually born back in the 1970s, when I built a very similar device for the first time. Its purpose is to disconnect a vehicle's battery when an inadvertent drain would discharge the battery to the point where the vehicle would not start. Before the advent of computerized automobiles, it was common for drivers to park and leave the lights turned on only to come back to find a dead battery. Then, they had to find a way to call a service station (cell phones weren't available) and get a boost. Worse, if the battery died and the car was left sitting long enough, the battery would become useless and have to be replaced. After remaining in a discharged state between 12 and 18 hours, most lead-acid batteries would go totally dead and could not be rejuvenated.

But that was then. Now, many vehicles—particularly those equipped with automatic lighting systems—protect against such inconveniences with automatic (often delayed) shutoff for electrical systems. However, several

types of vehicles still don't have automatic shutdown systems or alarms to warn you that the lights are on. This project, shown in Figure 6-1, is particularly useful for those vehicles.

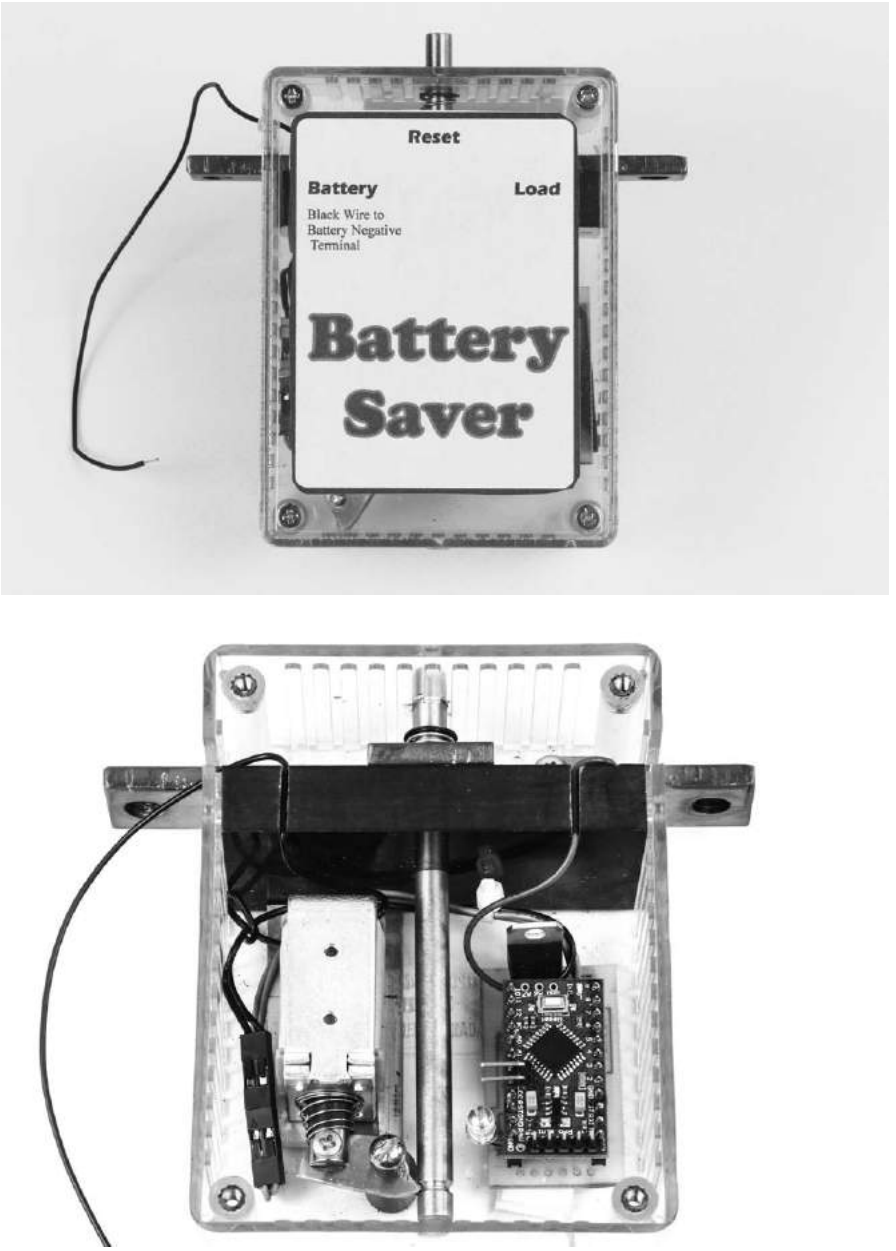


Figure 6-1: The finished Battery Saver, boxed and ready to go, as well as a look inside

Boats, Tractors, and Other Vehicles

Many road-worthy vehicles could benefit from the Battery Saver, but this project is targeted at other systems where an accidental discharge of a battery could be annoying—and expensive. Boats are particularly vulnerable. Even my small runabout has experienced problems. On more than one occasion, the running lights were left on during the day, and I did not notice until a couple of days later. The battery was totally dead, and it had to be replaced.

Leaving the running lights on isn't the only way to accidentally drain a boat battery. Most inboard boats have a blower system designed to safely expel potentially explosive gases and fuel vapors from the bilge. Conventional wisdom (and the Coast Guard) says to keep the bilge blowers on before starting the engines, the entire time the boat is in service, and for at least 10 minutes after shutdown. It's very easy to forget the blowers are on and then find you need to replace the battery or batteries the next time you use the boat.

NOTE *Before building the Battery Saver for a safety system like the blowers on a boat, read "Notes of Caution" on page 136. There are some considerations to keep in mind that may require some minor wiring changes to the electrical system.*

Boats aren't the only vulnerable vehicles, though. Riding mowers and tractors are also at risk for three reasons:

- They are usually used only intermittently.
- Parking and/or headlight switches are often placed where it's easy to accidentally bump them when getting on or off the vehicle. The switches can also be damaged by flying debris or rough terrain.
- They are usually used in daylight hours, so it can be hard to tell when the lights are on.

The Battery Saver can also protect powered tools, like tow-behind sprayers. It's easy to leave these tools turned on when finished, and while most sprayer pumps don't take too much current, leaving one on for, say, a day or two, will likely drain the battery.

NOTES OF CAUTION

If you create the Battery Saver as described in this chapter, it should work well. However, it is always possible for Murphy's Law to cause a problem, so here are a few points to consider before you build.

Should the Battery Saver fail, it can be reset, but in the meantime, it will remove all power to the vehicle's electrical system. *Do not* use the Battery Saver on any system where an electrical failure could cause catastrophic problems (such as on an aircraft) or result in bodily harm or property damage. That said, I have used the following unit in a car, on three agricultural tractors, and on two boats for well over a year (and earlier versions for several years) with no failures.

The Battery Saver includes a very high-current switch. It is possible that, under certain conditions, a resistance could develop such that when current is applied, the switch becomes extremely hot—perhaps hot enough to be a fire hazard. (For what it's worth, this has never happened even on prototype versions.) All efforts have been made to eliminate problems like this—for example, the block that holds the Battery Saver's copper contacts is made of a fire- and melt-resistant phenolic material—but overheating remains a hazard. Always check the heat of the Battery Saver before you touch it to make certain that it is not hot enough to burn you. Even the reset plunger could become warm enough to burn if the vehicle or Battery Saver malfunctions.

This last warning is specific to mariners. If you keep your boat in the water, chances are it has a built-in bilge pump and automatic float switch to keep it afloat when you're not aboard. If you have multiple batteries, make sure the bilge pump is wired to a battery that isn't in the circuit with the Battery Saver, as in Figure 6-2. Otherwise, find the wire that provides current to the bilge pump/switch, and simply bypass the battery saver.

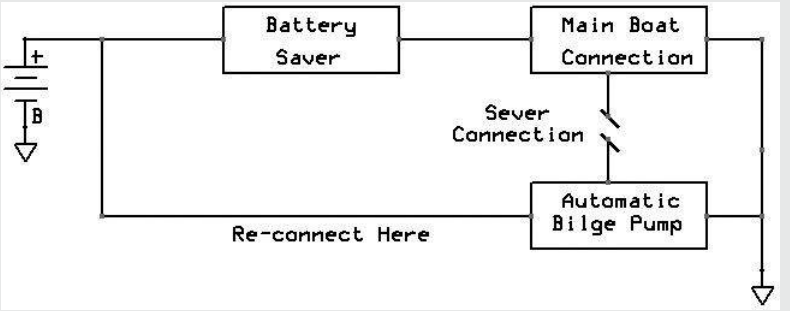


Figure 6-2: Wiring the Battery Saver on a boat

Parts List

This parts list might look somewhat like a scavenger hunt, but everything can be found easily from a variety of sources. Getting the particular size or quantity, however, may be challenging, so read the list carefully to make sure you have everything before you start. And before shopping, look ahead to Figure 6-13 on page 149 to see some of the more unusual parts, like the copper contacts.

- One Deek-Robot Pro Mini Arduino clone microcontroller board (There are several available, and some have different pinouts—particularly for pins A4 and A5. Figure 6-3 shows the pinout for the one I used; it's available from eBay and other retailers. Other units with different pinouts should work, but the connections on the shield must be changed.)
- 12V solenoid (This project uses an Electronic Goldmine G19852, but I have seen several others on eBay that were cheaper.)
- One 1/2-inch phenolic sheet (Many vendors package this in a greater amount than you need for this project. I found a 1/2 × 6 × 12–inch piece on eBay from a company called Nova Plastics.)

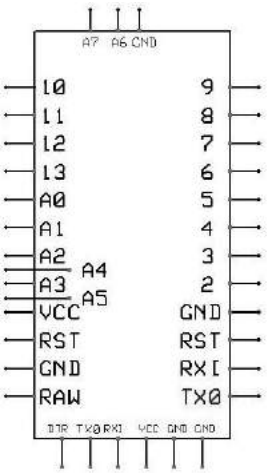


Figure 6-3: Pinout of the Deek-Robot Pro Mini Arduino clone

- One 6 × 3/4 × 3/16–inch copper bar, which is part of a high-current switch (This is actually a piece of copper bus bar that is generally used in large electrical installations. See Figure 6-4 for an example. To find this, you may need to do a little digging with an online search for "copper bus bar." I suggest starting with eBay. At a modest price, I bought a piece that was 3 feet long.)

Figure 6-4: A copper bus bar, drilled out for the Battery Saver

- One ABS plastic enclosure, Hammond 1591 STCL (Available at electronic suppliers like Mouser, Digi-Key, Newark, and so on.)
- One 1/4-inch brass round (I've been able to pick up small pieces like this at a surplus scrap metal dealer, but eBay usually has all lengths available, starting at $1.50.)
- The pylon (Another 1/4-inch brass round.)
- The release (I purchased a small piece of sheet metal from Ace Hardware, but the same is available from many other hardware stores.)
- One e-clip (I purchased an assortment of 300 e-clips on eBay. This will probably be a lifetime supply with enough left over for several future generations. If you don't want that many, you can probably find one at your local hardware store. Home Depot and Ace Hardware sell them individually.)
- One 1/2-inch, 4-40 flathead screw (Available at almost any hardware store. You can also purchase a box of 100 on Amazon cheaply.)
- One 3/8-inch, 4-40 roundhead screw (Available at your local hardware store.)
- 3 oz of Permatex Silicone RTV sealant
- One matching pair of inline connectors (You can buy simple, cheap, inline connectors online, or you can make your own. I used Pololu's 1×2 connector housing and male and female crimp pins, shown in Figure 6-5. See the appendix for crimping techniques. You can also use telephone-style chicklets if you can find them.)

Figure 6-5: A pair of mating connectors used to connect the solenoid

- Assorted #28 hookup wire (Check your local hardware store or online electronics hobby supply shops like Jameco, Pololu, and so on.)
- One solder lug (Check your local hardware store; if you can't find one, you can work around it by following the instructions in the project.)
- One battery cable to fit your storage battery on one side and a lug to attach to the Battery Saver on the other

Special Tool Requirements

- Countersink (Almost any 82° countersink will do. The material being formed is relatively soft, so no special materials are required.)
- Needle files (While only one is required, they usually come in sets and can be purchased inexpensively at Harbor Freight and other discount tool outlets. See Figure 6-18 on page 154.)
- Triangle file (This can be found in any hardware store.)

Downloads

- Sketch: *Battery_Saver_Rev_3.ino*
- Templates: *release lever.pdf*, *Enc template.pdf*
- Shield: *Battery Saver.PCB*

The Schematic

While the circuit is relatively simple, as shown in Figure 6-6, there are a couple of key design elements to be aware of.

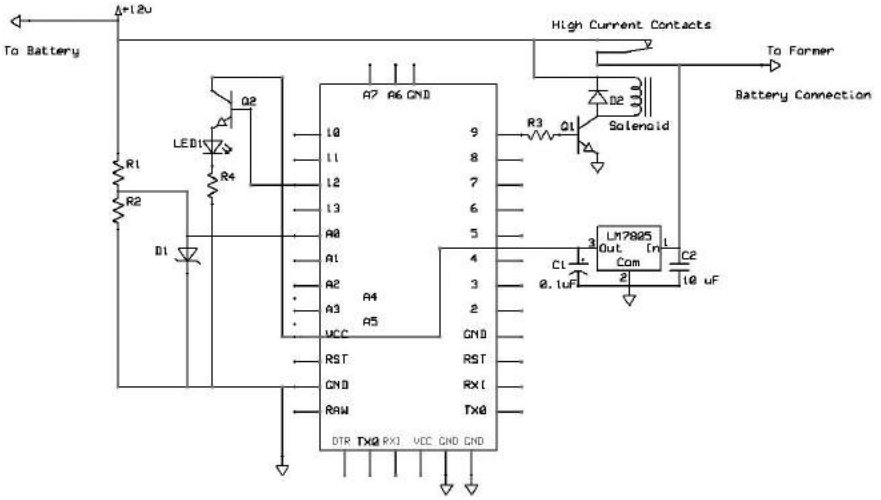


Figure 6-6: Schematic for the Battery Saver

Notice that the circuit uses an LM7805 voltage regulator. In theory, the small regulator included in the Pro Mini board would be more than satisfactory because the load is relatively light. But under the hood of a car, and around high-current systems and high-voltage electronics, there is a lot of

stray electromagnetic energy bouncing around. While it's improbable that this energy would cause a problem for the Battery Saver, it is well within the realm of possibility. The more robust 1.5A LM7805 regulator offers the Pro Mini another level of protection. In addition, capacitors C1 and C2 have been included to bypass any AC sneaking into the circuit and to prevent unwanted oscillation. Similarly, the 4.7V Zener diode (D1) protects the input of the Arduino from a voltage spike—which is very likely—on the 12V supply. It limits the voltage to A0 to only 4.7V.

The configuration of resistors R1 and R2 may look familiar from other projects. They comprise a voltage divider to lower the 12V supply to a level below the 5V maximum the Arduino input can tolerate. To be on the safe side, I selected a value of 10 kilohm for R1 and 5.6 kilohm for R2. Both are standard values. This design should allow the battery voltage to jump to 14V before reaching the point where the Zener diode kicks in.

Diode D2 (1N4002 or equivalent) provides yet another level of defense: it protects against an inverse current that could be created when the magnetic field in the solenoid collapses. This is a standard protection device in inductors with an iron core, which can store magnetic energy and release it rapidly into the coil. The reverse voltage can reach relatively high levels and create significant currents which could, in this case, damage the driver transistor and other components in the system.

Transistors Q1 and Q2 are both ZXT 649 silicon NPN transistors. I chose these transistors because they have sufficient drive capability and are inexpensive and readily available. I used the same model as in other projects. The high side of Q2 is brought to the 5V rail (VCC), as the LM7805 regulator supplies plenty of current. (You could just as easily connect the high side, or collector, of Q2 to the 12V supply, because Q2 is wired as an emitter follower and the voltage at the emitter will follow only the voltage at the base.) R4, a 470 ohm resistor, limits the current to LED 1.

LED1 indicates when the unit is operating. The circuit containing Q2 and LED1 provides a blinking LED with a very short on cycle, which is consistent with this volume's goal of blinking LEDs as often as possible.

NOTE *The Deek-Robot Pro Mini also has a red LED that indicates when it is turned on. This additional, although very small, constant current drain could contribute to the discharge of the battery. In the units that I installed, I unsoldered one end of the Pro Mini's LED, which is kind of a delicate operation.*

Transistor Q1, wired as a common emitter circuit, provides the drive for the solenoid. Resistor R3 (4.7 kilohm) drives the transistor and also protects it—a direct connection would allow too much current to flow in the base-emitter junction, resulting in potential damage. The high side of the solenoid is connected to positive 12V to allow maximum voltage and current to flow through the solenoid coil without taxing the voltage regulator.

How the Battery Saver Prevents Draining

The Battery Saver comprises a very high-current switch that can be electrically turned off and a sensor circuit to detect when the battery is in jeopardy of dying. The high-current switch is required because it interrupts the main power from the battery, which includes the feed to a starter motor that can draw up to several hundred amps.

The high-current switch—which could also be considered a relay—is made of the three pieces of copper bus bar: a release bar, a solenoid, and a release lever. When the pieces of copper bus bar are connected, the battery is connected to its circuit as normal. When the solenoid is pulled in, the power is disconnected.

The sensor circuit uses the power of the Arduino microcontroller. The Arduino continually monitors the battery, and if it senses that the power is diminishing, it calls for the high-current switch (relay) to be thrown. There are many ways to determine when a battery is reaching exhaustion, and this project simply looks at the voltage left in the battery. Table 6-1 shows the voltage versus the remaining charge in a standard 12V, lead-acid storage battery.

Table 6-1: Battery Charge State Versus Voltage

Battery Charge Level	Battery Voltage
100%	12.7V
90%	12.5V
80%	12.4V
70%	12.3V
60%	12.2V
50%	12.1V
40%	11.9V
30%	11.8V
20%	11.6V
<10%	11.3V

The battery charge level quickly deteriorates with only a minimal reduction in voltage. In order to have at least 40 to 50 percent of the charge remaining, the drain on the battery must be stopped somewhere between 11.9V and 12.2V, a voltage I will refer to from now on as the *trigger point*. In practical applications, empirical evidence shows that there is still plenty of juice left, even when battery voltage drops to 12.0V, 11.90V, or even below. A battery in good condition under ideal circumstances could perform with perhaps as little as 30 percent of its capacity, but that would depend on the load, ambient temperature, and other factors, so I'm taking a very conservative view of about 12+V. Newer design batteries tend to do better.

While the state of charge is a good indication of remaining capacity in a battery, other factors—like internal resistance, battery discharge rate, and so on—could impact the usable state of charge remaining in a battery. A more accurate measurement of the capacity remaining in a fully charged battery might be current used, which can be calculated if the total energy in the battery is known. For example, if a battery has a capacity of 1100 ampere-hours (A×h), you could calculate the point where 550A×h remain and disconnect the battery then. However, because this project targets systems with batteries that range widely in capacity, I decided that measuring the battery voltage would be more than adequate.

Arduino to the Rescue

The Arduino microcontroller steps up to the task of measuring the battery voltage and throwing the disconnect switch at the appropriate voltage, but if that's all it did, you could be in trouble. When turning on a vehicle, the starting motor uses a lot of current. Depending on the state of the battery (internal resistance and so on) and ambient conditions (temperature, for example) during the cranking process, the battery voltage can conceivably fall well below the critical shut-off voltage. The Battery Saver depends on the Arduino to give the vehicle enough time to start the motor.

Further, should the system shut down due to an inadvertent drain, you'll need to ensure that when the Battery Saver is reset, it doesn't immediately sense a critical battery voltage and shut down again. All of these functions are handled by the microcontroller, under orders from the sketch. While there are probably several different ways of handling it, the sketch implements *timing sequences*—rules to follow for when to take certain actions—to allow for the voltage drop during engine cranking. Similar timing rules give the user time to restart the engine after the Battery Saver is reset.

The Breadboard

Even though this project doesn't require a lot of extra components and peripheral equipment, I still believe it's useful to go through the exercise of wiring a breadboard. A working prototype gives a definitive proof of concept, and it allows you to play with the hardware and software prior to committing to the finished version. When working with the breadboard, I tested the circuit with a solenoid similar to the one used in the finished product (see Figure 6-7).

Figure 6-8 shows the breadboard operating the high-current switch; "Construction" on page 148 describes how to build that switch. To test the circuit without making the high-current switch first, or to use a solenoid by itself (see Figure 6-7), you can connect a lamp, LED, relay, or some other component in place of the switch. Remember: if you use an LED or other polarized device, make certain to get the polarity correct.

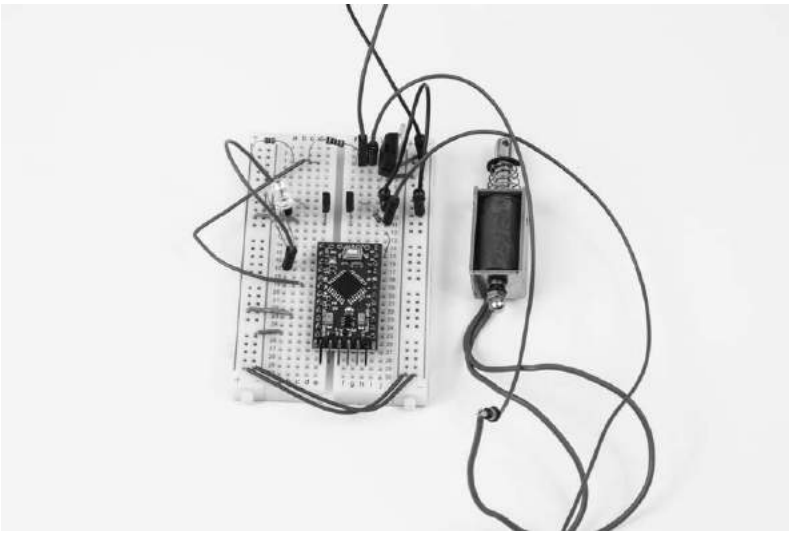


Figure 6-7: The breadboard I used to test the Battery Saver concept and put the software together. Testing your circuit with a solenoid is not required.

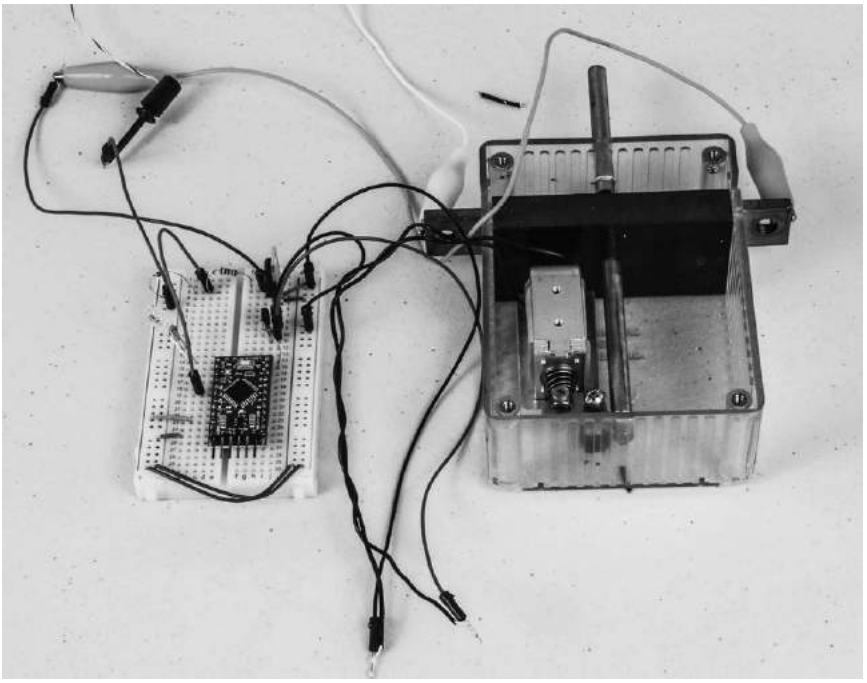


Figure 6-8: The breadboard circuit operating the solenoid after the high-current switch has been assembled. Note the use of clip leads to hook up the Battery Saver to the breadboard. Diodes D1 and D2 and capacitors C1 and C2 were not included in the breadboard primarily because they are needed only when the project is in use.

The basic breadboard is fairly straightforward. Just follow these instructions to assemble it:

1. Connect the breadboard's positive rails (red strips) together, and connect the negative rails (blue strips) together. Do not connect the positive rails to the negative rails as that will result in a direct short circuit.
2. Insert the LM7805 voltage regulator so that the three terminals span three different rows. (See "The Schematic" on page 139 for why an external voltage regulator was used rather than the Pro Mini's onboard regulator.)
3. Connect the input of the regulator to a positive 12V source. (See Figure 6-9 for the pinout of the LM7805.)

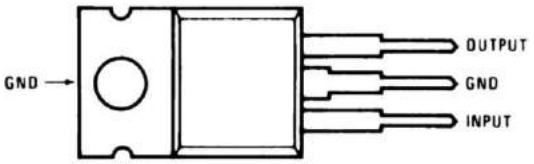


Figure 6-9: Pinout of the LM7805 voltage regulator

4. Connect the center pin of the regulator to ground (blue stripe) and the output pin to the positive rail (red stripe). When you power the circuit, the positive rail should have 5V coming from the output of the regulator.
5. Insert the Pro Mini microcontroller board in the breadboard. Its position is not critical; anywhere in the general vicinity of where it is in Figure 6-7 is fine.
6. Insert resistors R1 (10 kilohm) and R2 (5.6 kilohm) into the breadboard. It's easiest to insert them near the regulator. Then, connect one side of R1 directly to the regulator (input pin). The joining point of R1 and R2 should be located in an independent place on the board, and the other side of R2 goes directly to ground.
7. Connect a jumper from the point where R1 and R2 join to pin A0 of the Pro Mini.
8. Insert transistor Q1 (ZTX649) into the breadboard in an area with three open rows (see Figure 6-10 for the pinout).
9. Connect resistor R3 from the base of Q1 to pin D9 of the Pro Mini.
10. Connect the emitter of Q1 to ground.
11. Connect the collector of Q1 to the load (solenoid or other).
12. Connect the other side of the load to positive 12V.

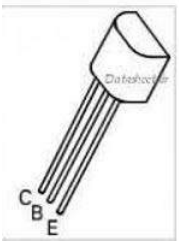


Figure 6-10: Pinout of ZTX649 transistor

13. Insert transistor Q2 into the breadboard. Connect its collector to one of the positive 5V rails (red strip).
14. Connect the base of Q2 to pin D12 of the Pro Mini.
15. Connect the emitter of Q2 to the positive side of LED1.
16. Connect the negative side of LED1 to R4 (470 ohm).
17. Connect the other end of R4 directly to ground (blue strip).
18. Connect the negative side of the 12V supply to ground (blue strip).

Now, load the sketch onto the Pro Mini (see "Uploading Sketches to Platforms Without a USB Interface" on page XX), and take it for a test run. If you have a variable power supply with a voltage readout, setting the voltage will be easy; if you don't have a variable power supply, I suggest building the Regulated Power Supply project in Chapter 3. If you use a variable supply without a readout, just use your multimeter to monitor the voltage and observe the trigger point.

Start the power supply at 13V (12.7V is normal for a charged 12V lead-acid storage battery), and the monitor LED should blink slowly. Gradually lower the voltage, and write down the voltage when the flashing goes from slow to rapid. That voltage is the trigger point, and it should be around 11.9V to 12V.

The Sketch

In developing the Battery Saver sketch, I went through several iterations to assure reliable functionality. One difficulty was avoiding false triggers when the sensed voltage jumped around because an engine or an accessory, such as a hydraulic tilt, was activated.

I used functions to adjust various sequences of operation independent of the main program. This isn't a complex sketch, but writing simple functions is a useful technique when you want to avoid repeating code. I could have avoided functions in the final sketch, but I let them remain because they work well, and writing the sketch this way provides a good lesson in the use of functions.

```
/*The Battery Saver sketch, which uses multiple functions
  to create timing sequences */

int led = 12;
int Battin = A0;
int Relay = 9;
int volts = 0;
int volts2 = 0;
int volts3 = 0;
int B = 387;  // Threshold trigger set point

void timer3 () {  // Shut off timer function
  delay (200);
  volts = analogRead (Battin);  // Reset trigger point
```

```
  volts3 = map (volts, 0, 1023, 0, 500);
  if (volts3 > B) {
    digitalWrite (Relay, LOW);
  }
  else {
    digitalWrite (Relay, HIGH);
  }
}

void timer2 () {  // Fast blink timer function -- low voltage
  if (volts2 < B)
  {
    for (int j = 1; j < 1800; j++)
    {
      digitalWrite (led, HIGH);
      delay (10);
      digitalWrite (led, LOW);
      delay (90);
    }
  }

}

void timer () {   // First timer function -- high voltage
  if (volts2 > B)
  { digitalWrite (led, HIGH);
    delay (200);
    digitalWrite (led, LOW);
    delay (1000);
  }
}
void setup ()
{
  Serial.begin (9600);
  pinMode (Relay, OUTPUT);

}
void loop () {
  delay (1000);
  volts = analogRead (Battin);
  volts2 = map (volts, 0, 1023, 0, 500);

  timer ();
  if (volts2 < B) // Set trigger point
  {
    timer2 ();
  }
  if (volts2 < B)
  {
    timer3();
  }
}
```

In this sketch, `timer`, `timer2`, and `timer3` are the three functions used. The `timer` function is for normal operation when the battery voltage is above the trigger point. The trigger point is the voltage at which the Battery Saver goes into timeout mode prior to shutting down. The `timer2` function sets off a rapid flashing sequence once the trip threshold is reached and provides the timing—the fast LED sequence—prior to shutoff. Once `timer2` has finished, `timer3` is invoked, provided the voltage remains below the threshold; this disconnects the battery by activating the solenoid. If the voltage increases above the trigger point at any time during the timeout period, the Battery Saver returns to normal operation after `timer3` times out. Many of the variables in this sketch can also be changed to vary blinking and delay sequences and threshold; they're addressed in "Operating the Battery Saver" on page 159.

The Shield

As you might guess from the breadboard, the shield is straightforward, too. Compared with handwiring, however, I believe it's a lot easier and faster.

The PCB Layout

Figure 6-11 shows my finished PCB, and Figure 6-12 shows the layout image with silkscreen indicating the placement of the components. You can download a PCB layout for the shield at *http://www.nostarch.com/arduinoplayground/*.

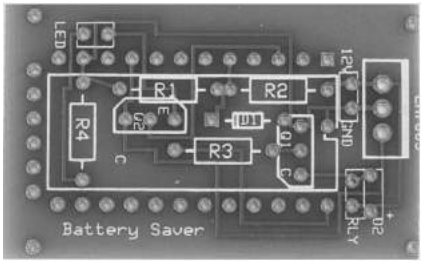


Figure 6-11: The unpopulated Battery Saver board. The RLY connection is where you would connect the solenoid.

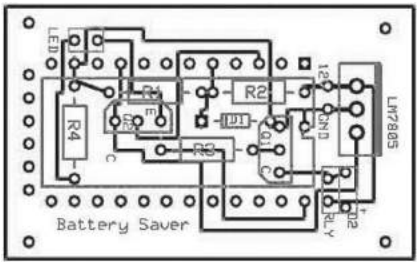


Figure 6-12: The foil pattern for the shield on the Battery Saver. The silkscreen image is in gray.

Most of the components are located under the Pro Mini to save space. Other than that, there is nothing special about populating the shield. As in other projects, it's not necessary to include headers for all the Pro Mini pins. Use all the pins that have connections with board traces and enough other pins to add mechanical stability. I always try to use at least one header for the very first pin, to aid in aligning the board while plugging it in.

The LED can be mounted directly to the board or can be mounted somewhere remotely with long wires. Just be sure to observe polarity. The Battery Saver may be located in an area where the operator can't see the LED, so placing it in a remote location is a practical solution.

Preparing the Shield and Pro Mini Controller

If you want to use a PCB, you can make the shield according to the layout provided with this book's resource files, whether you etch that yourself or send it off to be professionally manufactured. You could also design your own shield PCB if you're feeling ambitious, or just solder everything to the prototyping board, but if you are using everything else from the Parts List, just make sure your board has the same dimensions as the provided shield layout.

If you etch the PCB design I provide, drill the component holes next. I usually use a #66 drill. Solder a 2-inch wire to the plus 12V side of the shield, and solder the other side of the wire to a small solder lug. Solder a 15-inch wire to the ground terminal and then connect two wires to the solenoid connections. I used a small inline connector—made with Pololu #1950 crimp connector housings, #1931 male crimp pins, and #1215 female crimp pins—so I could remove the board easily if necessary. Almost any connector can be used.

Both transistors are located under the shield, so when you solder them, make sure to push them down enough that they clear the bottom of the Pro Mini. See Figures 6-11 and 6-12 for the transistor placement on the PCB.

I soldered the monitor LED directly to the board so it could be seen through the box. However, you could also solder wires to the board and place the monitor LED in a location where it might be more visible outside your vehicle.

Construction

Building the rest of the Battery Saver involves a few mechanical challenges and requires the wits of a scavenger (see the Parts List on page 137 for supply details). Figure 6-13 shows all the parts of the Battery Saver laid out. There is nothing complex about any of the parts.

There are only a handful of components in the Battery Saver: the enclosure, the phenolic contact support, the shield and Pro Mini controller, the copper contact assembly, the solenoid and mounting, the release lever and pylon, the release rod, and the springs and e-clip. The assembly instructions that follow are a little involved, but if you get stuck, just use Figure 6-13 to keep things in perspective.

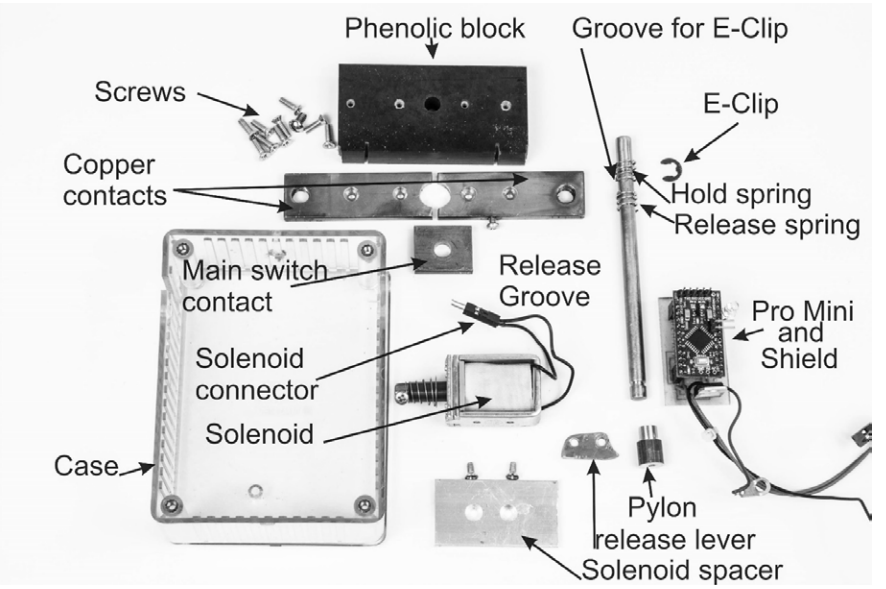


Figure 6-13: The components of the Battery Saver, completely disassembled. The cover of the enclosure is under the clear box. Note the short screws for fastening the spacer to the solenoid, so they don't damage the solenoid coil.

Preparing the Enclosure

Because the enclosure, a Hammond 1591 STCL, is an integral part of the design, I suggest starting there. Beyond a couple of holes and slots machined into the box, there is very little else to do. Figure 6-14 shows the holes I cut in detail; a template is included in this project's resource files. There are only a few holes to cut, though, so you should be fine without the template if you follow these instructions carefully.

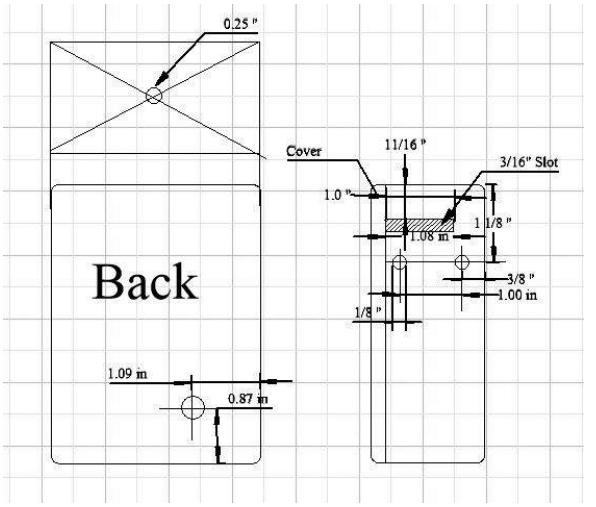


Figure 6-14: The holes and slot in the Battery Saver enclosure

On the long sides of the enclosure, measure approximately 1 1/8 inches from the top—that is the side where the reset plunger will protrude through. This should locate you roughly at the fourth mounting rib. In the center of that rib, drill two #30—approximately 1/8-inch—holes 3/8 inches from either edge, and countersink both for a 4 screw. Do this on both sides of the enclosure.

On each side, at the top of the next rib (going toward the top of the enclosure), cut a 1-inch-deep groove for the copper contact assembly. The shaded part of the side view in Figure 6-14 shows where the groove should go on each side.

The ABS plastic enclosure cuts easily with a hacksaw. Remove the cover and cut from the opening of the box toward the back. For each groove, make two cuts so that when you remove the material between them, you will have a 3/16-inch-wide channel in both sides of the enclosure. A little over 1 inch from the edge at the opening will do, but it's not necessary to cut down to the back of the enclosure. You can even cut both sides at once, but just make sure the channels are directly opposite each other. Don't worry if your cuts are a little off; that can be corrected later with a file. After you make the cuts, break off the material in the center and clean up the channel with a small triangle or flat file.

Now, drill a 1/4-inch hole in the top of the enclosure, where the reset button will go, exactly centered on the top surface. Draw lines along that side's diagonals from corner to corner, as illustrated in Figure 6-14, and drill where the lines intersect. This hole is where the brass release rod for the reset plunger will go.

The Contact Support

The contact support is perhaps the easiest part to make. First, cut the phenolic to a 3 × 1 3/8–inch block. The phenolic material cuts easily with a hacksaw. Insert the block in the enclosure so the two holes you drilled through the ribs are in the center of the block. Mark the holes on both sides, drill them with a #43 drill bit, and tap them for a 4-40 screw. While you are working on the phenolic contact support, you can cut two grooves, or channels, for the power and ground wires on either side, as shown in Figure 6-16. The location is not critical as these are used only to run the positive and negative wires for the shield. I used the small Dremel saw in Figure 6-23 to cut the grooves. Use two 4-40 × 3/8-inch flathead screws on each side to screw the phenolic piece in place.

With the phenolic contact support screwed in place, hold the drill as close to vertical as possible in the center of the largest face of the contact block. Drill a 1/4-inch hole that lines up with the 1/4-inch hole you previously drilled in the top of the enclosure. The easiest way to do this is to mount the contact support block and then use the hole in the top of the enclosure as a guide to drill the hole in the block. After drilling, put the contact support aside until you're ready for the copper contact assembly.

NOTE *The enclosure is not a perfect rectangle because some relief is included to allow it to come out of the mold easily. Factor this in as you line up to drill the center hole. When I drilled the hole, I held the enclosure in a vise to eliminate the effect of the relief.*

Preparing the Copper Contact Assembly

The copper contact assembly requires only a handful of holes. Cut a 4 3/4-inch section of the 3/16 × 3/4–inch copper bar, and drill the holes along the center of the 3/4-inch dimension, as outlined in Figure 6-15.

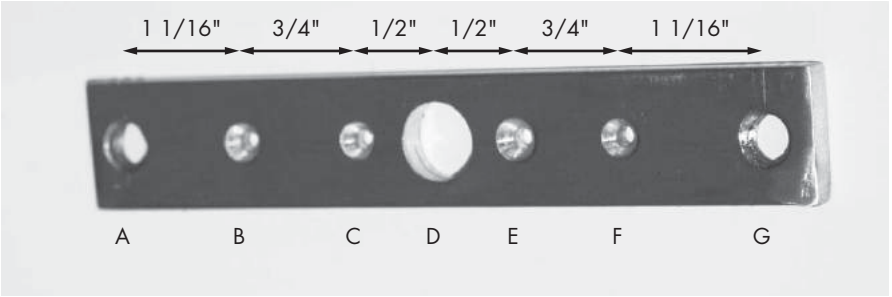


Figure 6-15: The base of the copper contact assembly, showing holes and spacing

For holes A and G, use a 9/32-inch bit. Holes B, C, E, and F should be drilled with a #30 bit; make sure to countersink them deeply enough for a 4-40 flathead screw. Finally, drill hole D with a 1/2-inch bit. Tap holes A and G for a 5/16 × 18–inch bolt, which will hold the battery cables.

CAUTION *While copper is not hard, it tends to grab a drill bit and climb up the bit. Always hold the copper piece in a vise, a clamp, or with pliers when drilling. Never attempt to hold the copper with unprotected hands.*

When all seven holes are drilled, set the base of the copper contact on the phenolic contact support so that the 1/2-inch center hole in the copper is centered as closely as possible on the 1/4-inch center hole in the phenolic support and the bar is centered along the entire length of the contact support. Make sure the bar is equidistant from both sides of the support, hold the two pieces together firmly, and mark holes B, C, E, and F. Center punch the marks you just made in the contact support, drill them with a #43 drill, tap them for a 4-40 screw, and set the contact support aside again.

Now, mark the exact center of the copper contact, which should perfectly bisect the 1/2-inch hole. Cut the piece in two at this marker; a hacksaw should work well for this. Then, cut a 3/4 × 3/4–inch square of your leftover 3/16 × 3/4–inch copper bar, and mark the center by making

diagonal lines from corner to corner. Center punch and drill a 1/4-inch hole where those lines intersect. This copper square will become the actual contact.

The final hole to make in the copper contact assembly is somewhere between holes E and F, on the outer edge of the bar; look for the little screw sticking out of the contact bar in Figure 6-16 (circled). Drill a #43 hole and tap for a 4-40 screw. This is where the switched positive voltage to the Pro Mini comes from. Figure 6-16 shows all the contact and mounting hardware drilled.

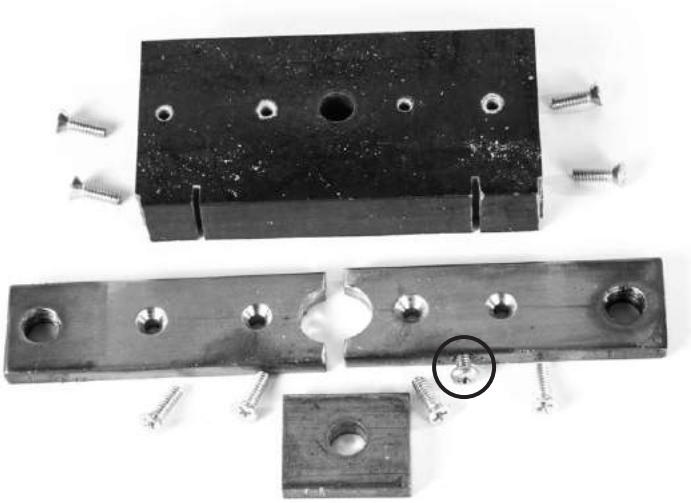


Figure 6-16: The contact and support hardware ready for assembly

Mounting Supplies for the Solenoid

Depending on your solenoid, the mounting process may vary slightly from the one described here. The frame of the solenoid I used had two holes tapped for a 4-40 screw in the bottom (see the Parts List on page 137). However, there was precious little room between the frame and the coil, so I elected to look for an alternative mounting approach—a very aggressive double-sided adhesive. If you still want to mount the solenoid with screws, judge the screw length carefully. With either mounting approach, the solenoid was not high enough to line up with the release mechanism, so I had to add a platform.

Preparing the Release Rod, Springs, and E-Clip

As with the other mechanical components of the Battery Saver, the release rod to reset the contacts just needs a little TLC. Begin with a section of 1/4-inch brass rod, and cut it to 4 1/4 inches in length. Measure 1 1/8 inches down from one end, and make a groove for the spring retaining clip to fit into, as shown in Figure 6-17.

In order to cut the groove, I mounted the bar in the chuck of an electric drill that I clamped to my workbench and used a hacksaw blade to carefully groove the piece, guiding the hacksaw blade with my fingers. The set on the hacksaw blade is a little wide, but the depth of the groove, not the width, is the important part. That said, the groove doesn't have to be very deep. I recommend you cut the groove in small increments and keep trying the retaining clip until it fits snuggly.

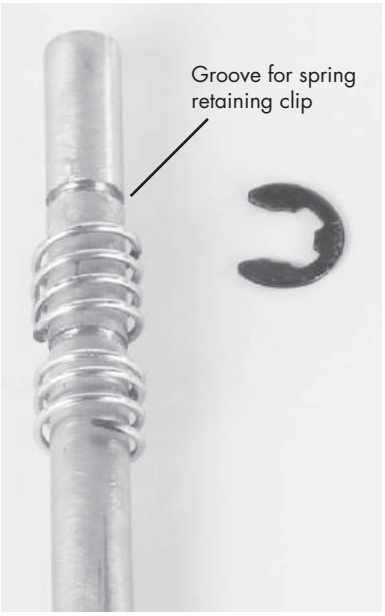


Figure 6-17: The upper section of the release bar. There are two grooves in the bar; the lower one was made in error. The springs used on the release rod were selected from a standard spring assortment from Ace Hardware and cut down to fit the project

Figure 6-17 also shows the configuration of the springs. The bottommost spring will rest on the phenolic block and keep the copper contact off the contact areas until the release rod is depressed. It should be small enough that it does not touch either side of the copper contacts. (Remember, the hole in the square contact is 1/2-inch wide, so the spring needs to be smaller than that.) When depressed, the top spring overrides the lower spring and keeps the upper part of the switch in firm contact with the lower sections. You can make fine adjustments of the spring length on a small grinder, or you can use your Dremel tool or a hand whetstone.

Now, measure 4 inches from the same end, and file the release groove in the lower half of the 1/4-inch release bar using a small needle file. Figure 6-18 shows both the needle file used and the shape of the release groove. (You might want to use a hacksaw first to make the groove, and shape the upper side with a file.)

The shape of the release groove is not overly critical, but make sure there is a slight bevel on the top side. The swirls you see on the shaft were made with 400-grit sandpaper I used to smooth the rod so it slides smoothly through the phenolic block and hole in the case and contact piece.

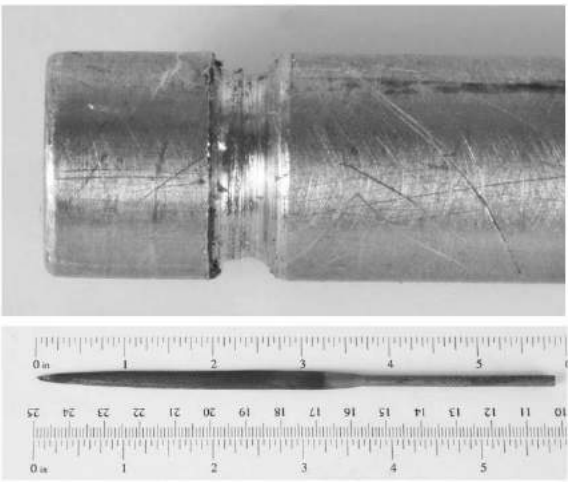


Figure 6-18: A close up of the release groove in the bottom of the release rod (top) and the small needle file used to make the groove (bottom)

Making the Release Lever and Pylon

The release lever is made of a small piece of light-gauge steel sheet metal, approximately 0.060 inches thick. Use the pattern in Figure 6-19 to cut the lever. In this book's resource files, you can find a PDF file of the lever template. Because the template is pretty small, you might try bonding the template to a blank piece of similarly sized stock, hold it in a pair of pliers (vise grips work well), and shape the piece on a grindstone. Once you've bonded the template to the stock, you can also use a file to shape it.

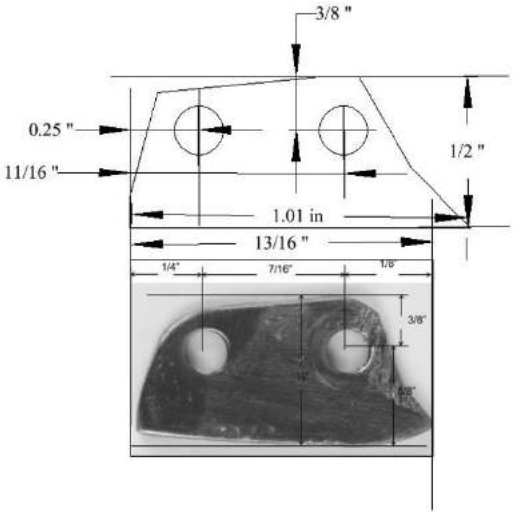


Figure 6-19: The release lever was shaped initially with a pair of tin snips and finished with a file. Some of the marks are from holding the piece tightly with the vise-grip pliers.

With a pair of tin snips, I cut a rectangle first and then cut the shape of the release lever. I firmly clamped the piece in place with a vise so I could finish shaping it with a file. Mild steel files easily with a satisfying feel. (You could also shape the lever with a small grinding wheel or a Dremel tool with an abrasive wheel.) Make the piece a little oversized at first, as indicated in the template, as it may require some adjustment in the final assembly. The lever should be under pressure from the tension spring, and should securely hold the release rod so that the square copper piece is firmly in contact with the both sides of the bottom copper contact assembly. Both pivot holes in the lever—one to attach to the solenoid and one to attach to the release pylon—are drilled with a #30 drill.

The pylon is for mounting the release lever to the enclosure, and it can be made of any scrap brass or aluminum you may have around. The pylon in Figure 6-20 was made from a 3/8-inch diameter brass rod. I reduced the top section's diameter because I didn't want the pylon to rub on the release rod, but that probably was not necessary.

Figure 6-20: The release lever, the pylon, and two screws

Cut the pylon to a length of about 0.61 inches so that when mounted inside the enclosure, the end attached to the release lever hits the release rod just below the center of its diameter. This dimension is dependent on how precisely the release rod hole is drilled into the center of the top of the enclosure. Run the release rod through the hole, and measure how far from the back of the enclosure its center is. The pylon should be a little shorter than that.

You can make the top of the pylon smaller by chucking it into a drill like a bit and spinning it on a file. When you finally assemble the Battery Saver, you may also have to adjust the height of the pylon slightly to accommodate the thickness of the release lever and altitude of the release rod if the hole was not drilled perfectly straight.

Drill a #43-sized hole near the center of the pylon all the way through, and tap for a 4-40 screw from both sides. One side will mount to the back of the enclosure, and the other will hold the release lever. To make sure the screw doesn't go down too far and tighten against the release lever, I tapped down only about 3/16 inches so the screw bottomed out and jammed. Figure 6-21 illustrates what the tapped pylon should look like.

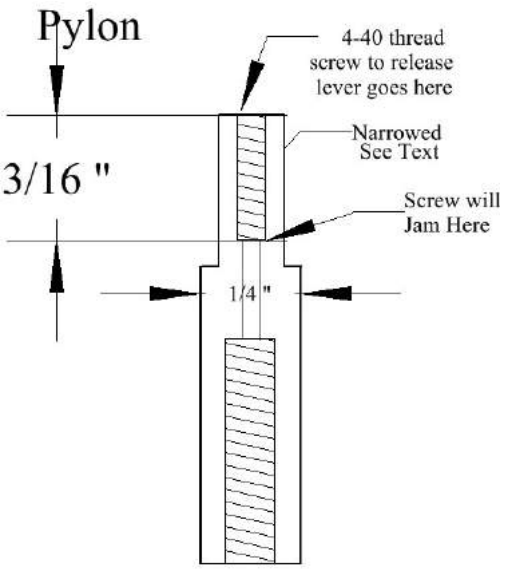


Figure 6-21: Inside the pylon

You can adjust either the length of the tapped hole or the length of the screw to make sure the release lever is free to move but secure, using the screw head as a bearing surface. For fastening both the release-lever bearing screw and the pylon mounting screw, I recommend using an anaerobic adhesive (such as Loctite Threadlocker) to secure the threads.

Assembling All the Parts

Now, we're ready to start putting all the parts from Figure 6-13 together and fastening them where they belong. When you're done, the Battery Saver should look like Figure 6-22.

The order of assembly is not overly critical, but more a matter of common sense. I'll go through it step by step:

1. Start by screwing the copper contact pieces into the phenolic support. Make sure the screw heads are below the surface of the copper. This is critical to assure good contact with the contact bar. Prior to assembly, as an added measure to improve the contact area, I sanded both the contact pieces and contact cap with a 400-grit sandpaper. To assure flatness, I put the sandpaper on a flat surface and rubbed the copper on it.
2. Slide the phenolic support into the enclosure, and fasten it with the four 4-40 × 1/2-inch screws.
3. Insert the release rod through the enclosure, then thread it through the pressure spring, then through the copper contact, and finally through the release spring and down through the phenolic base. Insert the e-clip. This may be a little tricky. You'll have to hold the clip with a pair of needle-nose pliers.

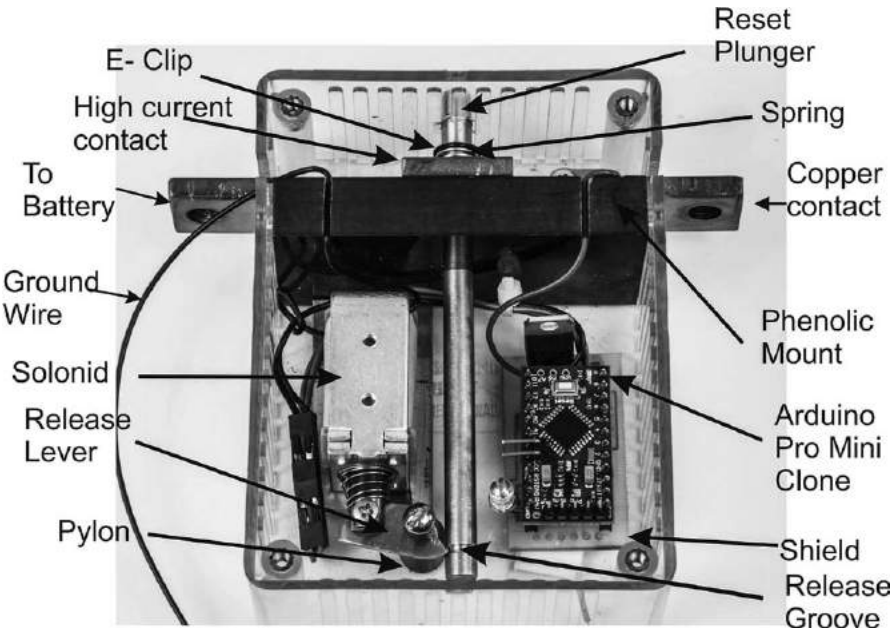


Figure 6-22: The completed Battery Saver with the cover off and the switch in the operating position. In this iteration, grooves were carved in the phenolic block for the positive and ground wires.

4. Measure for the location of the pylon, and mount it to the enclosure with a 4-40 × 1/2-inch flathead screw. The pylon should be inside the enclosure, and the screw should be threaded in from the outside.
5. Screw the release lever to the pylon, and check by hand that it engages the release groove and secures the release rod. This is a little tricky, and you may have to adjust the release lever by grinding or filing it a little to fit in the release rod's groove. Once you have the lever in place, you can operate it with your fingers and assure that it locks tightly to the release-rod groove.
6. Install the spacer on the solenoid.
7. Temporarily mount the solenoid to the enclosure, and screw the release lever to the solenoid. To do this, I first unfastened the release lever from the pylon and fastened the other end to the solenoid plunger. I then applied double-sided tape to the bottom of the spacer, juggled everything in place, and loosely placed the screw through the release lever and into the pylon.
8. Exercise the release mechanism with the solenoid held in place. Once the release rod holds the top switch contact firmly down, mark the holes for the solenoid spacer.
9. Drill and tap holes in the enclosure to securely mount the solenoid and spacer to the enclosure. Alternatively, fasten the solenoid with the double-sided adhesive (3M outdoor double-sided adhesive).

10. Run the wires that will connect the solenoid to the Pro Mini circuit under the release rod, and connect them to the solenoid using the connectors.
11. Run the ground wire from the Pro Mini circuit under the release rod and out through the slotted opening for the copper bar. (It's the dark wire sticking out of the enclosure on the left in Figure 6-22.) You may want to either drill small holes in the phenolic for the wires or carve small grooves to run the wire through, using a Dremel tool and a small circular saw blade like the one in Figure 6-23. You can also make a groove with a file or hacksaw.

Figure 6-23: Dremel tool with small circular saw blade attached

12. Attach the red, positive voltage wire with the lug to the copper switch pole at the last hole you tapped in "Preparing the Copper Contact Assembly" on page 151. Use a 4-40 × 3/8-inch screw.
13. Mount the shield using small spacers. If you can't find a solder lug, you can solder-tin the wire, form it so it fits around the screw snugly, and then tighten the screw.

Compare your Battery Saver to the finished device in Figure 6-22 to make sure everything looks right. When the battery voltage reaches the trigger point, the Arduino triggers the solenoid, which releases the lever. Freed from the release lever, the rod pops up from the spring tension of the lower spring. The lower spring holds the contact piece above the copper mounted to the phenolic block, opening the circuit. To reset the Battery Saver, just push the rod back down again. Before doing a final test, check the latching of the release lever and rod several times.

Installing the Battery Saver into a Vehicle

Connecting your Battery Saver to your vehicle takes only minutes. First, disconnect both the positive and negative terminals from your battery. Then, take a short battery cable (see the Parts List on page 137) and connect from the positive side of the battery to the input side of the Battery Saver. Take the output side of the Battery Saver and connect it to the cable that originally connected to the battery. Connect the black wire to the negative terminal that will be reconnected to the battery.

Mounting the entire enclosure will depend on where and how your vehicle's battery and/or battery box are located. In many cases, the Battery Saver can simply hang from the battery cables. In other applications, I've used a heavy-duty cable tie to wrap around the entire battery and Battery Saver to hold it in place. In some cases, double-sided Velcro works well.

Operating the Battery Saver

In operation, once the Battery Saver is installed (see Figure 6-24), restore the ground connection to the battery and hook that ground connection to the Battery Saver. Then, set the Battery Saver by depressing the reset button—that is, the top of the release bar—until the unit is armed. You should hear or feel a click as you depress the reset button and the release lever engages.

Normal Operation

As long as the battery is fully charged and above the threshold voltage, the "on" indicator will blink at the rate established in the `timer()` function, which is approximately once every 2.2 seconds. When the battery voltage drops below the threshold level, which is set at approximately 11.9V via variable `B` in the sketch, the indicator LED begins flashing rapidly. When the LED sequence finishes, which is after about 3 minutes, the voltage will be checked once again. If the voltage is still below the threshold, the battery is disconnected; otherwise, it just returns back to normal operation.

When you reset the Battery Saver after it has shut off, it will resume operation. After reset, if the battery voltage is above the threshold, the indicator will blink at the usual rate of once every 2.2 seconds. Often when the drain is removed from a battery discharged to some level, the battery recovers somewhat on its own after a relatively short time. If, however, the voltage is below the threshold level, the indicator will blink rapidly for the timeout period, as indicated earlier, to allow for the operator to start the boat, tractor, or other vehicle. You can set the timeout period by changing the value of `j` in the `timer2()` function.

The maximum value of the `j` variable is set at 1,800 in the initial sketch, and incrementing or decrementing `j` will add or subtract 100 ms from the total timeout period. Thus, to set the timeout period to 5 minutes, you would set the maximum value of `j` to 3,000.

Setting the Threshold Voltage

The threshold voltage is established by resistors R1 and R2 with values of 10,000 and 5600 ohms, respectively (see the schematic in Figure 6-6). These are set up as a voltage divider. According to the voltage divider calculation, whether you use a calculator or work it out with the formula, the voltage will be approximately 4.31V for a 12V input. Thus, you can calculate the exact threshold voltage at which you want the device to shut off the battery current by setting the threshold trigger point, which is `B` in the sketch, to whatever value you wish. While I calculated the theoretical value of the threshold, I experimented and found that a value of 387 establishes the shutoff threshold point at about 11.9V.

ON BATTERY TYPES

I experimented with several batteries and loads and found setting `B` to 387 consistently leaves at least half the energy in the battery after shutdown to restart the engine. On several different batteries, ranging from the small battery on a portable generator to large-capacity batteries for starting a truck, the same value seemed to work well.

That said, however, I have little experience with deep-cycle batteries and know that they have very different discharge parameters. If you want to attempt to use the Battery Saver for such batteries, take a look at their discharge rates and voltages, and set the threshold accordingly.

Protection from the Environment

Unfortunately, the Battery Saver is not weatherproof, and most applications call for it to be used in somewhat hostile environments. There are, however, a couple of possible solutions. On a variety of vehicles, including boats and tractors, I have wrapped the device in a plastic bag and tightly wrapped wire ties around the cables where they enter and exit the device.

But that's not terribly attractive, and I took some abuse from the distaff side of the family, so I applied Permatex silicon RTV sealant around all the openings where the copper bar comes through—but not where the reset (brass) rod comes through—and sealed all the screws.

For the reset button, I had difficulty finding a protective covering nipple (you would be surprised at what I found on the web), so I settled for affixing the top of an eye dropper to the enclosure with the silicone sealant. This also keeps the reset bar—which is hot to 12V when depressed—from shorting out.

Applying Cool Amp

While the copper-on-copper contacts work well, I burnished them with very fine sandpaper (400 grit) before assembly. Even though the untreated contacts have been used on a number of versions of the Battery Saver and have never failed, I decided to use Cool Amp, a simple-to-use silver-plating compound, for this version.

For very little cost, I was able to silver-plate the contact area of the contact bar and plate and thus reduce their resistance. Figures 6-24 and 6-25 show the difference between unplated and plated copper.

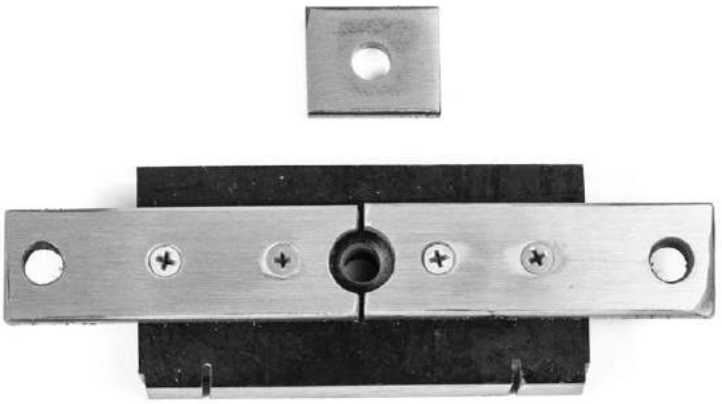


Figure 6-24: Battery Saver contacts after burnishing but before treatment with Cool Amp

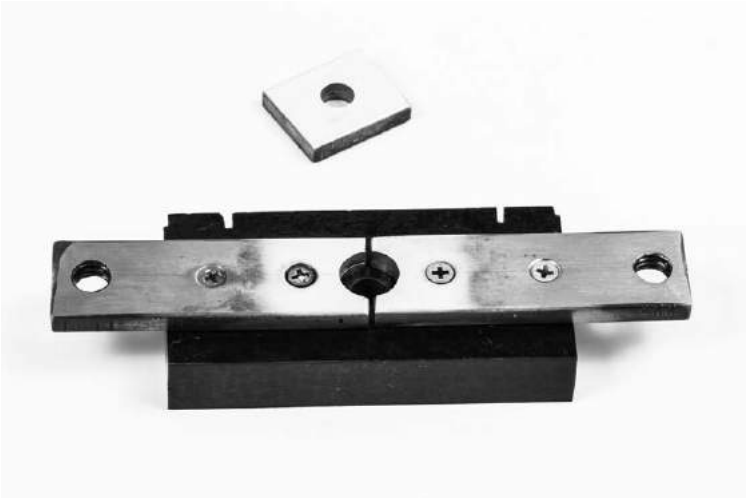


Figure 6-25: Battery Saver contacts after silver plating with Cool Amp

While this additional lowering of resistance is probably not necessary, all the current for the vehicle passes through this contact, so I figured it wouldn't hurt to be on the safe side. Further, I have used the Battery Saver in marine applications around saltwater, and the copper parts have acquired a green patina while the silver-plated areas have not.

I have used Cool Amp on a number of contacts, from motor-starting contacts to heavy-duty relay contacts, and it works well. You can learn more about Cool Amp at *http://www.coolamp.com/*.

NOTE *Copper does not oxidize too rapidly and the most common oxide of copper is highly conductive, which is one reason it is commonly used for current switching. Silver has the same properties—only better.*

DISCLAIMER

I hold a U.S. patent (4,149,093, now expired) on a device similar to the Battery Saver. The patent drawing is shown in Figure 6-26. Notice how similar it is to this project, despite being made decades earlier. It had essentially the same function, even though microcontrollers hadn't been invented yet!

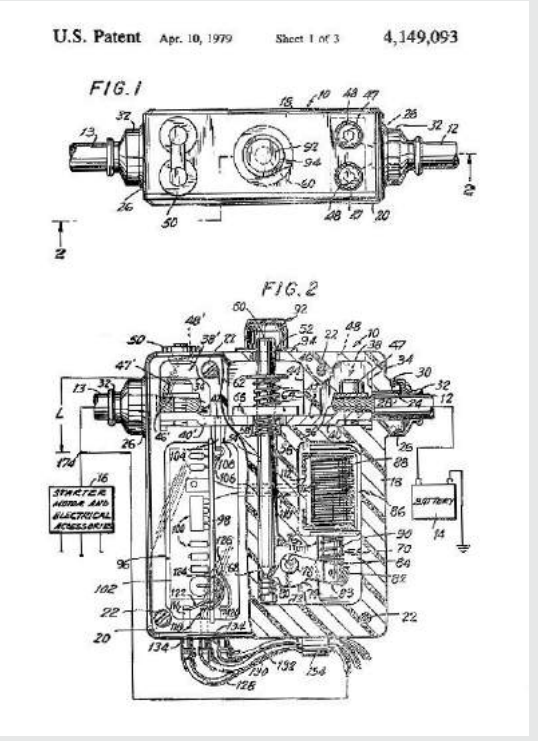


Figure 6-26: A patent drawing of the Battery Saver's predecessor