PROJECTS ■ THEORY ■ APPLICATIONS ■ CIRCUITS ■ TECHNOLOGY

NUTS AND VOLTS

NUTS AND VOLTS
www.nutsvolts.com
February 2016
EVERYTHING FOR ELECTRONICS

Build A Raspberry Pi
WIND SPEED TRACKER
That Will Blow You Away

WORKBENCH ESSENTIALS
build the digital decade box

INCREASE THE SENSITIVITY
of your arduino with an analog front end

THE CSS555 TIMER
a micropower programmable version of the 555 family

CHARGE WITH CONFIDENCE

Experience the ease and reliability of battery charging with our dual port AC/DC charge units, the X2 AC Plus and X2 AC Pro! Designed with maximum efficient power distribution capability, these power-packed chargers provide flexible battery charging for all your chemistry types whether you are in the workshop or out in the field. Grab our optional WiFi module to configure your settings and keep an eye on your charge progress with your Smartphone or PC!

GET CONFIDENT WITH HITEC!

X2 AC PRO

- AC Mode Output: Total 200W Power with Power Distribution
- DC Mode Output: 200W Port A, 100W Port B for Total 300W Power
- Includes 60W Soldering Iron, 400 to 840˚F

X2 AC PLUS

- AC Mode Output: Total 100W Power with Power Distribution
- DC Mode Maximum Output: 100W Per Port for Total 200W Power
- 10 Programmable Profile Settings

BOTH CHARGERS FEATURE:

- Power Distribution Mode for Ultimate Flexibility
- Built-in Power Supplies
- Use Either 11-18V DC or 100-240V AC Power Sources
- Adjustable Charge Current Rate of 0.1 ~ 10.0 Amps Each Port
- USB Support for PC and Wi-Fi Smart Phone Interface

LiPo	LiHV	LiFe	LiIon	NiMH	NiCd	PB
1-6 cell	1-6 cell	1-6 cell	1-6 cell	1-15 cell	1-15 cell	2-20V

HITEC RCD USA, INC. | 12115 PAINE STREET | POWAY, CA 92064 | (858) 748-6948 | WWW.HITECRCD.COM

www.saelig.com
sales@saelig.com

Saelig .com
unique electronics

585-385-1750
888-7saelig

Your Full-Service Test & Measurement Equipment Supplier*

***Full Specs on Our Website... Or Call Us for Free Tech Advice to Match the Best Product to Your Needs & Budget.**

PC Scopes

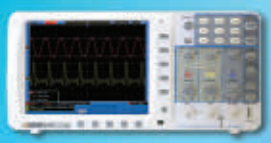
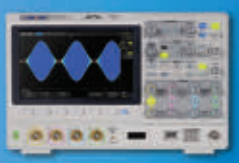
pico Technology
cleverscope



Benchtop Scopes

GW INSTEK
OWON
RIGOL Beyond Measure
SIGLENT
TELEDYNE LECROY



Power Supplies

GW INSTEK
OWON
RIGOL Beyond Measure
SIGLENT




Signal Generators

TTi
GW INSTEK
OWON
RIGOL Beyond Measure




RF Test Equipment

AARONIA AG
TTi
RIGOL Beyond Measure
TEKBOX
Triarchy





• Great Customer Service • Free Tech Support •

February '16

Subscription Information
Nuts & Volts — PO Box 15277
North Hollywood, CA 91615-9218
Call 877-525-2539 or go to www.nutsvolts.com
Subscribe · Gift · Renewal · Change of Info

Departments

Nuts & Volts (ISSN 1528-9885/CDN Pub Agree #40702530) is published monthly for $26.95 per year by T & L Publications, Inc., 430 Princeland Court, Corona, CA 92879. PERIODICALS POSTAGE PAID AT CORONA, CA AND AT ADDITIONAL MAILING OFFICES. POSTMASTER: Send address changes to **Nuts & Volts, P.O. Box 15277, North Hollywood, CA 91615** or Station A, PO. Box 54, Windsor ON N9A 6J5; cpcreturns@nutsvolts.com.

NUTS AND VOLTS

Published Monthly By
T & L Publications, Inc.
430 Princeland Ct.
Corona, CA 92879-1300
(951) 371-8497
FAX **(951) 371-3052**
Webstore orders only **1-800-783-4624**
www.nutsvolts.com

Subscription Orders
Toll Free **1-877-525-2539**
Outside US **1-818-487-4545**
P.O. Box 15277
North Hollywood, CA 91615

FOUNDER
Jack Lemieux

PUBLISHER
Larry Lemieux
publisher@nutsvolts.com

ASSOCIATE PUBLISHER/ ADVERTISING SALES
Robin Lemieux
robin@nutsvolts.com

EDITOR
Bryan Bergeron
techedit-nutsvolts@yahoo.com

VP OF OPERATIONS
Vern Graner
vern@nutsvolts.com

CONTRIBUTING EDITORS
Fred Eady, Tim Brown, Paul Verhage, Chuck Hellebuyck, Ron Hackett, Lou Frenzel, James Senft, Eric Bogatin, Roger Secura, David Goodsell

CIRCULATION DEPARTMENT
subscribe@nutsvolts.com

SHOW COORDINATOR
Audrey Lemieux

WEBSTORE MARKETING
Brian Kirkpatrick
sales@nutsvolts.com

WEBSTORE MANAGER
Sean Lemieux
sean@nutsvolts.com

ADMINISTRATIVE STAFF
Re Gandara

DEVELOPING PERSPECTIVES

by Bryan Bergeron, Editor

Leaded Components: Reports of their death have been greatly exaggerated

I've been reading and writing about the imminent demise of leaded components for decades. Even so, at least half of my work still involves leaded components. After all, what's not to like? Leaded components are easy to work with. It's easy to identify the value of a leaded resistor or capacitor with the naked eye, and leaded components are readily available.

Besides, I've already committed the band values — red for two, orange for three, yellow for four, etc. — to long-term memory. Then, there's the muscle memory of how to bend leads and how to work a soldering iron tip around the porcupine-like mass of leads when the component side is down.

Why let all that learning go to waste?

I don't know what I would do without a good supply of 1/4 watt 10K leaded resistors to use as circuit probes. When I'm working with an Arduino or other microcontroller board, it takes only a few seconds to wire-wrap a 10K pull-up or pull-down resistor to an I/O pin. Try that with a surface-mount (or SMT) resistor.

Then, there's the differential in infrastructure cost and workbench real estate. For leaded components, I have a simple Weller temperature controlled soldering iron, good old-fashioned needle-nose pliers, and desk lamp magnifier.

For surface-mount work, I have a hot air station that has the footprint of an oscilloscope, a tool drawer full of tweezers and stainless steel picks, and a half dozen containers of various solder pastes and fluxes.

And forget the magnified desk lamp — I have to don a Bosch & Lomb stereo magnifier and get my nose within inches of the board to see what's going on.

One sneeze, of course, and every SMT component not glued or soldered down will be forever lost in the dust balls behind my workbench.

I know that experimenters aren't alone in the battle between SMT and leaded components. I routinely tear down equipment for both fun and profit, and it's unusual to find an electronic device devoid of leaded components.

Many of the inexpensive devices made in China — from drone controllers to electronic measuring devices — are made with a mystery chip embedded in a black epoxy blob that is surrounded with leaded capacitors and resistors.

This is understandable, given the cost of converting an electronics assembly plant from leaded to SMT devices. Unless you're building iPhones or electronic watches, why upgrade an assembly plant unless you have to?

The bottom line is that if you're just getting into electronics, don't be dismayed — or distracted — by the world of SMT. A traditional perfboard, a good supply of leaded components, and a few schematics to work from will get you started.

When you're ready to make your circuit semi-permanent, then break out your soldering iron or wire wrap tool. SMT components, the special boards, pastes, and the rest will be there if you ever need them. **NV**

READER FEEDBACK

Testing Your Metal

Referring back to the August 2015 Developing Perspectives column about replacing old non-polarized plugs/cords. I've also had to do this a number of times. If the device is all plastic on the outside, I don't worry about it; if it is metal, I first make sure no AC line is in contact with the metal case. If that is the situation, then just go ahead with the replacement. Otherwise, find the problem. Or, if there is no "problem," then make sure the common wire (white) is connected to this connection. That should help it be a little safer.

Phil KE3FL

Just a Fraction Off

A reader spotted an error in Figure 2 of my January 2015 Ham's Wireless Workbench column. The dimension "1/2 wavelength" should be "1/4 wavelength." Sorry about that!

Ward Silver

Watch This

Regarding Bryan Bergeron's recent editorial on magnetics and watches, I have observed that the tiny spring's windings can be touching each other when magnetized. This shortens the spring somewhat, which results in the watch gaining time. Increased friction at this point could counter it a bit.

Open the watch and observe.

Duco Weytze

TOYing with Buggy Code

If you are having trouble compiling the code from my article in the November 2015 issue on the TOY project, I have isolated the problem to the new 1.6.6 version of the Arduino IDE for Windows.

It seems there is a bug in that version of Arduino which is causing the problems. The solution is to go back to version 1.6.5 and the problems all go away.

However, I identified another issue compiling the code for the ToY project.

Not only did the Arduino IDE code change from version 1.6.5 to 1.6.6 causing the first issue, the ESP8266 code changed from version 1 to version 2 which introduced another problem. The problem is the ESP8266 developers changed the *WiFi.SSID()* function from returning a pointer to character to returning a string. Luckily, the fix in my code is simple.

Continued on page 61

In this column, Tim answers questions about all aspects of electronics, including computer hardware, software, circuits, electronic theory, troubleshooting, and anything else of interest to the hobbyist. Feel free to participate with your questions, comments, or suggestions. **Send all questions and comments to: Q&A@nutsvolts.com.**

- **Cat-5 Cabling**
- **Electronic Filters Explained**
- **Mailbag**

Post comments on this article at **www.nutsvolts.com/magazine/article/february2016_QA.**

Cat-5 Cabling

Q AT&T U-verse uses a four-pair Cat-5 cable between its gateway and set top box. Are all four pairs in use for the TV service? I'd like to break out one pair to serve as a regular phone jack. I do not have AT&T's IP phone service but just a regular wire line phone.

— Art Wiegand

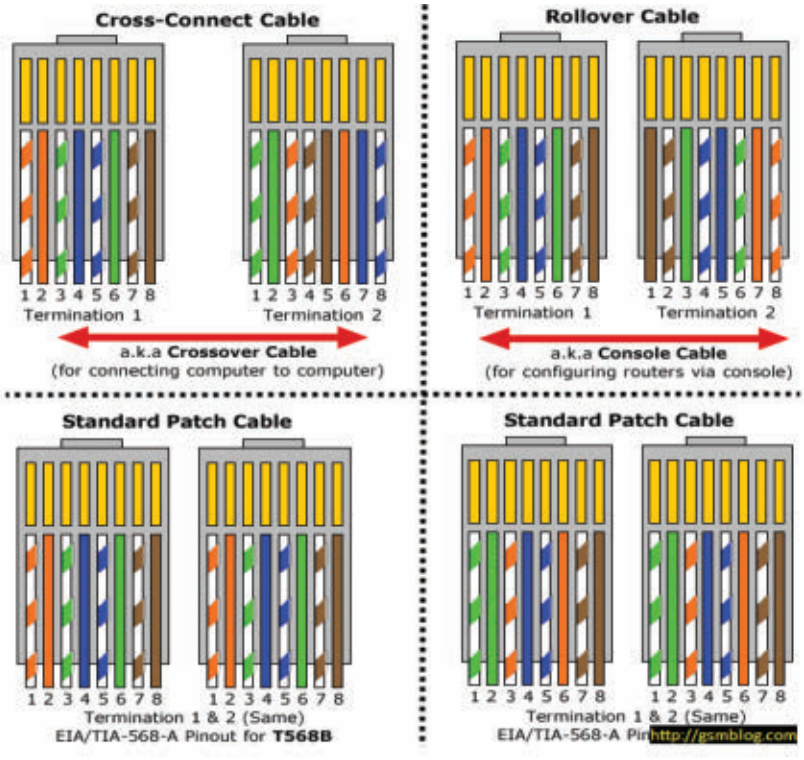
A Category 5 cabling uses unshielded twisted wire pairs to reduce the effects of crosstalk and interference without using shielding as is utilized in coaxial cables to reduce costs and simplify installation (it is a lot easier to crimp RG45 plugs onto cables than to attach connectors to coax). Category 5 cabling has four sets of unshielded twisted pairs arranged by color groupings (color and white with color stripe) using the colors green, orange, blue, and brown, which can be hooked up as TIA 568A or TIA 568B in either straight through, rollover, or cross-over configurations (see **Figure 1**).

These pairs of wires have tight twists to ensure a level of resistance to crosstalk and interference. Changing this twist such as when installing the RJ45 plug is a hot topic in the networking field, so you don't want to change this twist unless absolutely necessary. Recommendations essentially are don't untwist more wire than is required to fit in the plug, which is about 1/2 inch for RG45 plugs and receptacles. Untwisting more wire than necessary will leave your installation subject to crosstalk and interference which can greatly reduce data rates.

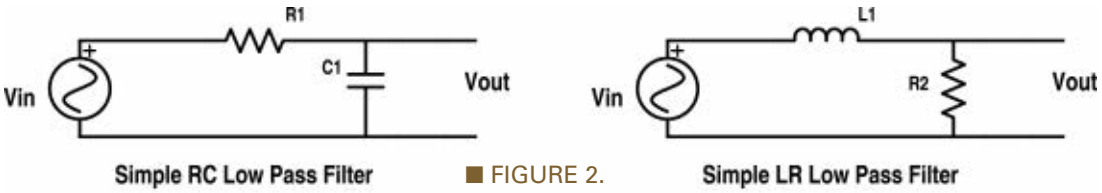
In case you want to run a phone line from the Network Interface Device (NID) instead, I found an excerpt from the AT&T U-verse Forum web page on how to hook up U-verse Cat5 cables from the NID (the box on the outside of a residence which hooks your home to the telco's network) to the Residential Gateway (RG; the "modem" box inside the residence which splits signals to user devices) at **https://forums.att.com/t5/U-verse-2014-Archive/what-2-wires-from-cat-5-go-to-the-outside-nid-box/td-p/3536669**. According to the site: "Traditional phone pairs used by U-verse techs are blue pair line 1 phones, orange pair for line 2 phones, green pair for U-verse. If this is Internet only in the house, use either blue or green pair; if you planning on using POTS or VOIP, we recommend using green pair."

Now, for your specific question about Cat5 cabling from the RG to the Set Top Box (STB); I will use my U-verse equipment as an example (I only have a standard phone line and Internet service, so I use one Ethernet port for the computer, and wireless for laptops and smartphones). My Motorola NVG510 RG has cable receptacles for phone lines 1 and 2, DSL broadband input, and five Ethernet outputs — plus wireless capability. Below is a list of the standard Ethernet Cat5 connector pin assignments. Ethernet cable pin assignments (TIA 568A/B cables) do not use blue and brown pairs:

■ FIGURE 1.



1: Data (TX +)
2: Data (TX -)
3: Data (RX +)
4: Ground (-)
5: Ground (-)
6: Data (RX -)
7: Power (+)
8: Power (+)



■ FIGURE 2.

Your STB most likely has an external power connection, so you are probably not using Power Over Ethernet (POE). Theoretically, the wires hooked to pins 7 and 8 could be used as a phone line pair and — if your RG does not use the Ethernet grounds — you could use pins 4 and 5 also. As for me, I would not disturb the Ethernet cable for just one pair and risk degrading my network speed and reliability.

I am assuming that you have a Cat5 cable with RJ45 plugs on each end (plugged into the receptacles on the RG and STB), so you would have to "cut into" the cable to access the pairs. Regular phone cable is cheaper than new Cat5 cables (about $2 for a 20 foot phone cable with RJ1 plugs installed, versus $6 for Cat5 with RJ45 plugs if you damage the Cat5 cable in the process), so I would leave the Cat5 cable alone.

Another option — if there is no phone receptacle already in the room — is to use a wireless phone which will hook to your regular phone line and transmit the phone signals throughout the house (cuts down on crawling under the house to replace old phone cables which at this stage in my life is no longer fun).

The Mysteries of Electronic Filters Explained

Q Could you explain how an electrical signal filter works?

— Jodie Pokabla
Dearborn, MI

A Signal filters for electronic devices are used to allow certain frequencies to pass (bandpass filters — BPF), block higher frequencies (low pass filters — LPF), block lower frequencies (high pass filters — HPF), block certain frequency bands (band rejection filters or notch filters — BRF), or pass all frequencies (all pass filters — APF). Filters can also be classified as passive (use only inductors, capacitors, and resistors) or active (use inductors, capacitors, and resistors along with transistors or operational amplifiers).

Before I discuss filters, let's go back and look at the basics: resistance, capacitive reactance, and inductive reactance. Resistance is the opposition to the flow of electrical current regardless of the signal's frequency from DC to the maximum signal frequency. Capacitive reactance is the opposition to the rate of change of the voltage of a signal. Capacitive reactance is a function of the capacitance and the frequency of the signal, $X_C = 1/(2\pi fC)$, where X_C is the capacitive reactance, f is the signal frequency, C is the capacitance, and π is the number 3.1417.

Inductive reactance is the opposition to the rate of change of the signal current, $X_L = 2\pi fL$, where X_L is the inductive reactance and L is the inductance. In theory, this is simple. In the real world, however, all electronic components have varying degrees of resistance, capacitance, and inductance, so all three opposing entities are present in all device components. Resistors are designed to minimize capacitance and inductance; capacitors are designed to minimize resistance and inductance; and inductors are designed to minimize resistance and capacitance.

Let's look at passive signal filters first. Looking at the equations above, we can deduce that for a pure resistance, opposition to a signal is independent of frequency so it passes all signals equally. For a pure capacitor, the capacitive reactance decreases with increasing frequency. So, a capacitor will pass higher frequencies better than it will pass lower frequencies (capacitors will block DC or 0 Hz signal frequencies). For a pure inductor, the inductive reactance increases with increasing frequencies. So, an inductor will pass lower frequencies better than it will pass higher frequencies (inductors will pass a DC signal unattenuated).

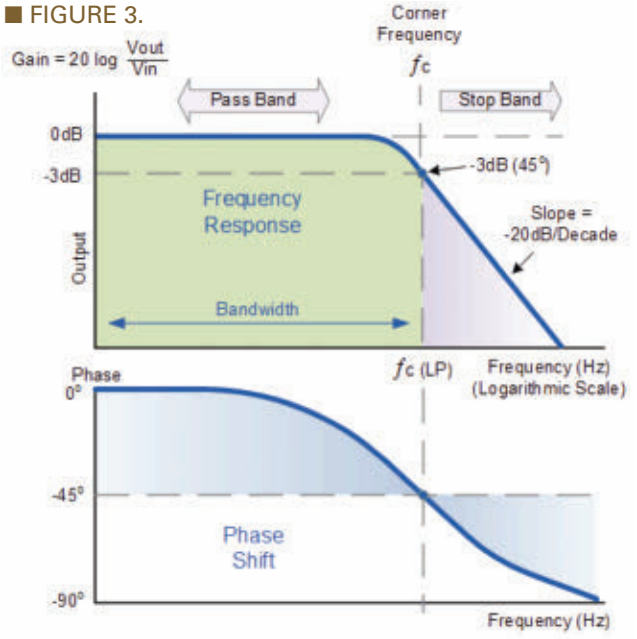
Figure 2 shows the schematic diagrams for simple low

Q&A SIDELINES

Cat-5 Cabling
https://forums.att.com/t5/U-verse-2014-Archive/what-2-wires-from-cat-5-go-to-the-outside-nid-box/td-p/3536669

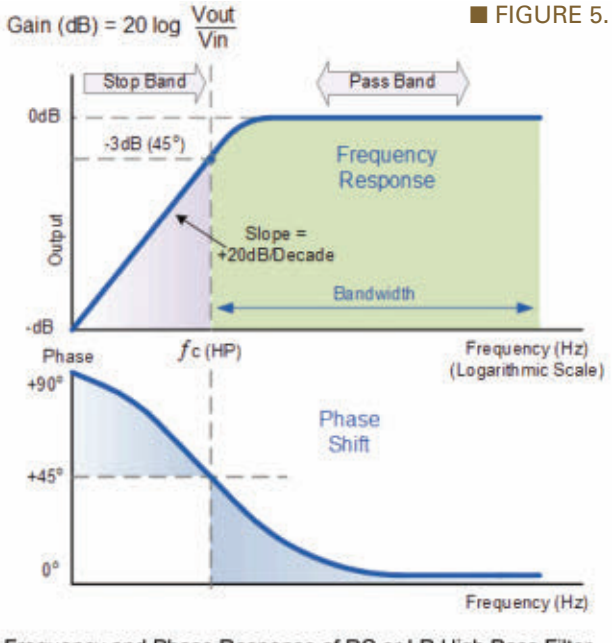
The Mysteries of Electronic Filters Explained
http://sim.okawa-denshi.jp/en/Fkeisan.htm
www.changpuak.ch/electronics/Butterworth_Bandpass_active.php
www.calculatoredge.com/electronics/ch%20pi%20low%20pass.htm
http://t-filter.engineerjs.com

■ FIGURE 3.



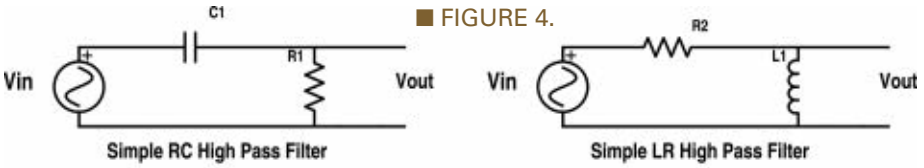
Frequency and Phase Response of RC or LR Low Pass Filter

■ FIGURE 5.



Frequency and Phase Response of RC or LR High Pass Filter

■ FIGURE 4.



pass filter circuits using a resistor and capacitor (RC filter) or a resistor and an inductor (LR filter). **Figure 3** shows the signal versus frequency response (Bode Plot) of a low pass filter.

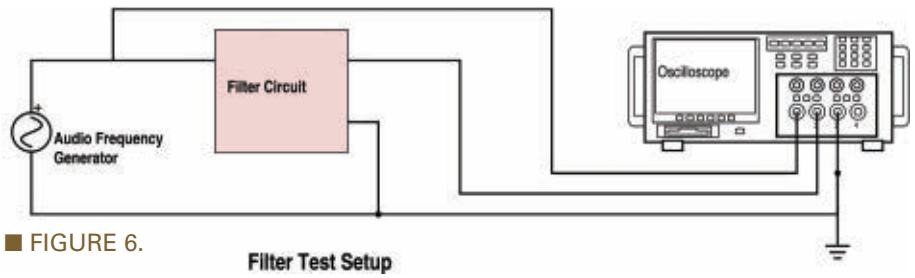
At the cutoff frequency for the RC filter, R = X_C. For the LR, R = X_L. The cutoff frequency for the RC LPF is f_C = $1/(2\pi RC)$, and the voltage gain (V_{out}/V_{in}) is A_V = $1/(2\pi fC(SQRT(R^2 + (1/(2\pi fC)^2))))$. For the LR LPF, the cutoff frequency is f_C = $R/(2\pi L)$ and the voltage gain (V_{out}/V_{in}) is A_V = $R/(SQRT(R^2 + (2\pi fL)^2))$. Where f_C is the frequency at which the mid range signal level is reduced by 70.7 percent or three decibels [Voltage Gain in dB = $20\log_{10}(V_{out}/V_{in})$, the decibel gain can be defined using the power ratio as 10 $\log_{10}(P_{out}/P_{in})$ because the number is the same since power is proportional to voltage squared.

Voltage is easier to measure (most amplifier gain is specified as voltage gain), so, R is the value of circuit resistance, C is the value of circuit capacitance, and L is the value of circuit inductance. The phase shift for the RC circuit is Θ = -arc tan($2\pi fRC$); for the LR circuit, the phase shift is Θ = -arc tan($2\pi fL/R$).

Arc tan is the arc tangent function found on spreadsheets and scientific calculators. I have shown the phase shift characteristics of the filter circuits to illustrate that the output signal's phase will be changed relative to the input signal phase around the cutoff frequency, in case maintaining signal phase relations is important to a design.

Figure 4 shows the schematic diagrams for simple high pass filter circuits using a resistor and capacitor (RC filter) or a resistor and an inductor (LR filter). **Figure 5** shows the signal versus frequency response of a low pass filter. The cutoff frequency for the RC HPF is f_C = $1/(2\pi RC)$ and the voltage gain is AV = $R/(SQRT(R^2 + (1/(2\pi fC)^2)))$. For the LR HPF, the cutoff frequency is f_C = $R/(2\pi L)$ and the voltage gain is A_V = $2\pi fL/(SQRT(R^2 + (2\pi fL)^2))$. The phase shift for the RC circuit is Θ = arc tan($1/(2\pi fRC)$) and for the LR circuit, the phase shift is Θ = arc tan($R/2\pi fL$). Sorry about all of the parentheses in these equations, but that is the way equations are presented for entry into a spreadsheet or programmable calculator.

High pass and low pass filters can be used to modify the lower and



■ FIGURE 6.

Filter Test Setup

MAILBAG

Re: Furnace Data Acquisition

In the August 2015 issue, you gave several ideas to Edward Ganshirt on how to go about monitoring a furnace. You were focused on using a current sensor to interface with a high voltage motor wire inside the furnace. I just wanted to point out that the low voltage lines of the thermostat are another option, and no current sensor would be needed.

The thermostat wires offer the additional benefit of being available both inside the furnace and at the actual thermostat, in case someone wanted to build a nice device showing live data on an LCD.

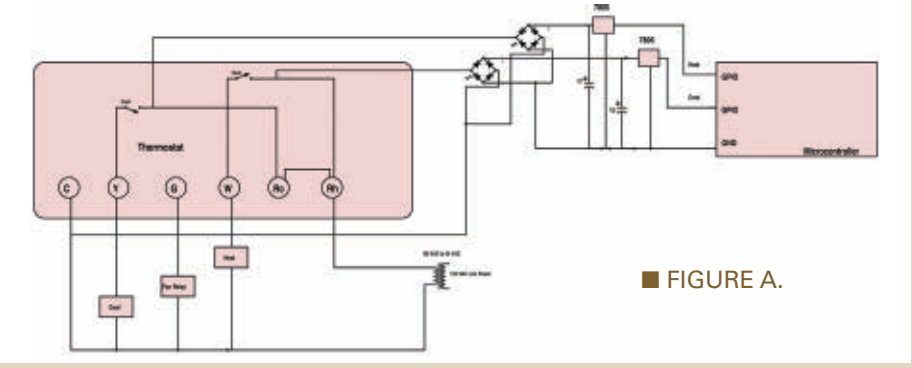
Judy May W1ORO, Union, KY

Judy, looks like you caught me in an Einstellung moment where I just looked at the method proposed in the questions and not a better solution. I am excited about receiving information from you because we need more women involved in the field of electronics (I taught at a community college for 23 years and that was the case throughout our state). The fact that you are a ham radio operator shows a real love of electronics.

Using the thermostat contacts would be a better solution since they are easily accessed inside the residence and safer since they use a lower voltage. An HVAC thermostat operates using 24 volts AC derived from the HVAC unit line power via a transformer. ***Figure A*** *shows the wiring of an HVAC thermostat with my idea for reading the heating and cooling contact closures with a microcontroller such as a PIC, Atmel, or Arduino. The voltage in the thermostat is 24 VAC, so you need a couple of bridge rectifiers to convert to DC; a couple of capacitors to smooth the ripple; and a couple of 7805 three-pin voltage regulators to convert the final output to 5 VDC for input to the microcontroller.*

Once you have the signal to the microcontroller, it can be used to log on/off times, calculate usage rate, etc. Be aware that some thermostats do not have the "C" ground, while others used in heat pumps have W2 for secondary heat, and O and B for reversing valves. The wiring colors (red, green, yellow, orange, blue, and white) match the letters except for C which has a black wire (usually).

Thanks for giving us insight from a different perspective.



■ FIGURE A.

Re: Automotive Battery Amp-Hour Capacity at Slow Discharge

I enjoyed the November 2015 Q&A question on the automotive battery amp-hour. Can you add the golf cart battery to this discussion? Many of us RVers use a golf cart battery for house batteries for a couple of reasons: (1) They are usually less expensive; and (2) The house batteries don't need to be short time/heavy discharge automotive batteries. Other than that, I don't know exactly how they compare between auto and deep cycle (more expensive) batteries.

Bob Smith, Prescott, AZ

Batteries used in Recreational Vehicles (RVs; a.k.a., house batteries) are similar to the batteries used to crank a vehicle (a.k.a., chassis batteries or starting batteries). The exception is RV batteries are designed for deep discharge at a relatively low current draw for extended periods of time while they power the appliances and other devices inside the RV. RV batteries operate without being immediately recharged as is the way starting batteries are operated.

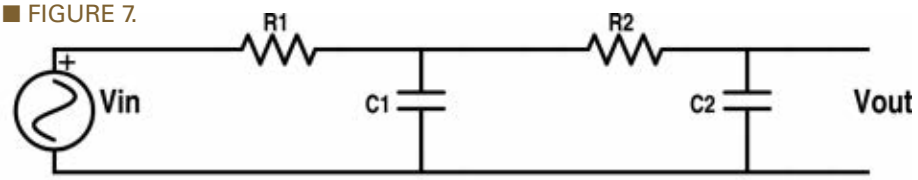
RV batteries are rated in amp-hours (Ah – roughly time of useful operation at a given current draw) and reserve capacity (RC – number of minutes at 80 degrees Fahrenheit that the battery can deliver 25 amps until it drops below 10.5 volts) instead of cold cranking amps (CCA) as is specified for starting batteries which are immediately recharged. A 12 volt group 24 RV battery provides 70 to 85 Ah. Wiring two batteries in parallel will double the Ah, but you must be careful that the batteries supply current equally to the load.

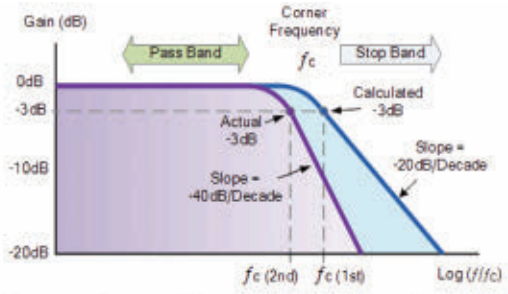
Two of the larger six volt golf cart batteries wired in series can be used in place of the RV battery (provided there is sufficient space) to provide 180 to 220 Ah. As in all rechargeable lead acid batteries – in most cases – RV batteries fail due to overcharging (poor regulation of the alternator causes water loss and plate deterioration) and undercharging (repeatedly deeply discharged without fully recharging or leaving for months without charging – solar trickle chargers are a good investment for RVs that are not used frequently enough to keep the batteries charged).

Battery life is dependent upon how deeply the battery is discharged before recharging (a battery discharged to 50 percent before recharging will last twice as long as a battery discharged to 80 percent before recharging).

upper ranges of signals, respectively, such as an audio signal. Try to envision how you could use the LPF and HPF to modify the bass and treble portions of an instrument signal. For an experiment, use an oscilloscope and an audio frequency function generator as shown in **Figure 6** to measure the frequency responses of the LPF and HPF circuits on sine signals (30 to 1,000 Hz for LPF and 10,000 to 15,000 Hz for HPF).

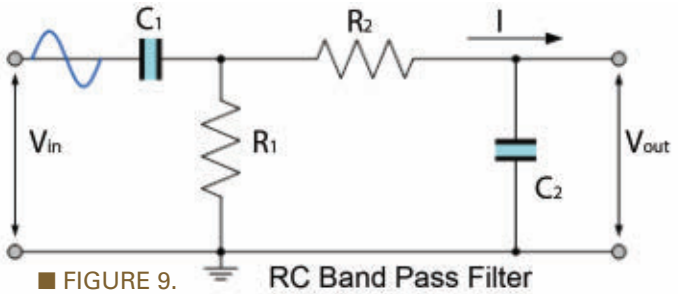
■ FIGURE 7.



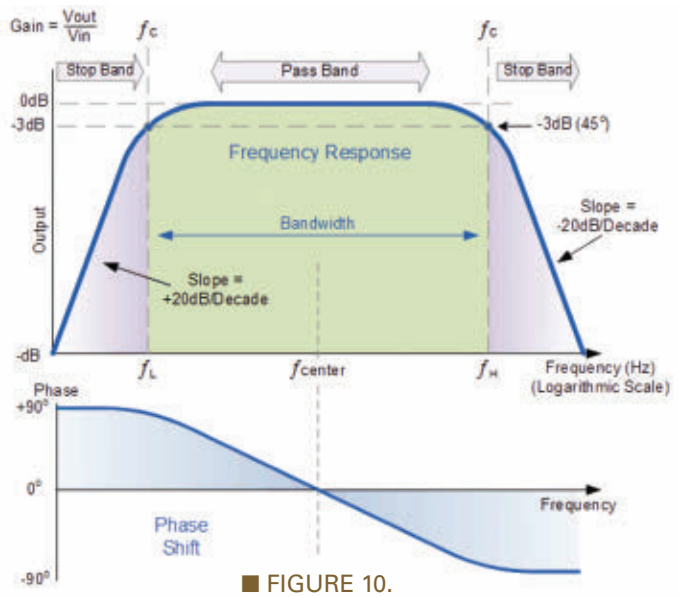


■ FIGURE 8.

Frequency Response Curves for Simple (Blue) and Cascaded (Purple) RC Low Pass Filters ($R_2 = 10R_1$ and $C_2 = C_1/10$)



■ FIGURE 9. RC Band Pass Filter



■ FIGURE 10.

RC Bandpass Filter Frequency and Phase Response Curves

CAUTION: Use isolation transformers on the generator and o-scope to avoid shock hazards and short circuits that can damage your expensive equipment. If your o-scope will do ratios, the gain calculation is simplified. Otherwise, you can calculate the gain by dividing the V_{out} on channel 2 by the V_{in} on channel 1 for several frequencies and use the results in a spreadsheet to draw the Bode Plot.

You can measure the phase shift by connecting V_{in} to the vertical channel (one of the normal inputs) and V_{out} to the horizontal channel (sometimes called X or sweep), and then determine the phase difference from the resulting Lissajous figures. By substituting potentiometers for resistors, you can vary the frequency response of these filters. Any readers out there who are interested, send me your results and I will try to publish as many as possible.

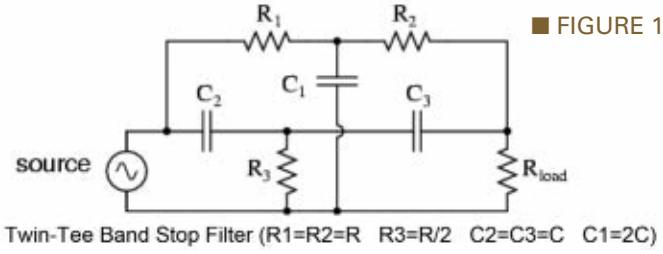
If there are simple filter circuits, then there are complex filter circuits. If one filter is good, then adding a second filter should be twice as good, right? In reality, two cascaded RC LPFs make the system better than twice as good. **Figure 7** shows the cascaded low pass filter and **Figure 8** shows frequency response curves for the simple (in blue) and two cascaded (in purple) RC filters which demonstrate the cascaded filter's cutoff frequency is lower, and the slope of the "roll off" portion of the curve is steeper (sharper filter). By adding additional cascaded RC LPF filter stages, the roll off slope can be made even steeper, which brings us closer to an ideal filter in which the roll off portion would be straight up and down.

In reality, adding filter stages can produce predictability problems since each cascaded stage affects the other stages (this is beyond the scope of Q&A to cover, but there are many engineering texts dealing with this aspect of filter design). The cutoff frequency for the two-stage cascaded RC LPF is $f_C = 1/(2\pi SQRT(R_1C_1R_2C_2))$.

If cascading two RC LPF gave us a sharper frequency response, what about cascading an HPF with an LPF? **Figures 9** and **10** show this arrangement and the associated frequency response curve. You can easily see that this filter has both high and low cutoff frequencies, but it allows frequencies between these two to pass. Thus, it is called a bandpass filter. The two cutoff frequencies are: $f_H = 1/(2\pi R_1C_1)$ and $f_L = 1/(2\pi R_2C_2)$. The center frequency is $f_{CENTER} = SQRT(f_H X f_L)$. The phase response can be calculated from $\Theta = \arctan((f/(f_H - f_L))(f/f_{CENTER} - f_{CENTER}/f))$.

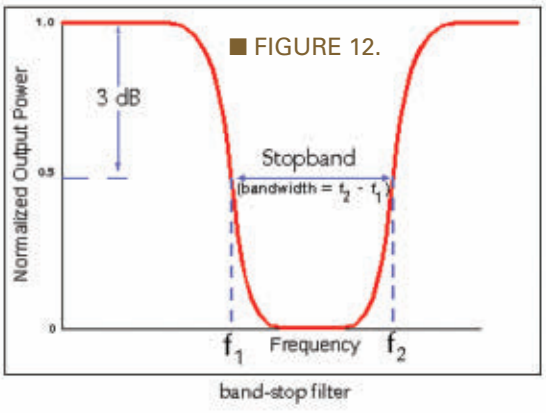
For the BPF, we define the bandwidth as $BW = f_H - f_L$ and the quality factor $Q = f_{CENTER}/BW$, or Q is the circuit reactance divided by the circuit resistance. The bandwidth is inversely proportional to Q; as Q increases, the bandwidth decreases. So, by decreasing the circuit resistance, you can decrease the bandwidth. BPFs are used to eliminate unwanted signals, while allowing wanted signals such as audio or television signals.

Band Rejection Filters (BRF) are also known as notch filters or band stop filters, and are used to eliminate unwanted signals such as a nearby FM radio station that



Twin-Tee Band Stop Filter (R1=R2=R R3=R/2 C2=C3=C C1=2C)

■ FIGURE 11.



band-stop filter

■ FIGURE 12.

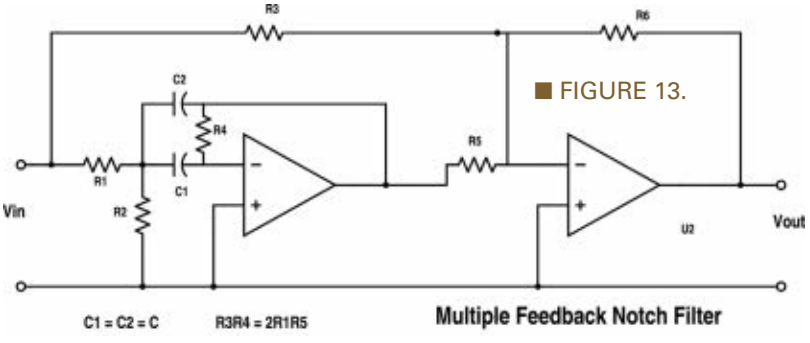
interferes with the station you wish to listen to. **Figure 11** shows a twin T band stop filter schematic and **Figure 12** shows the frequency response curve for the twin T.

The BRF frequency response is like an inverted version of the response for the band pass filter. The center frequency for the twin T is $f_{CENTER} = 1/(2\pi RC)$ and a bandwidth of BW = 2/(RC). The twin T is actually a high pass filter (C_2, C_3, and R_3) in parallel with a low pass filter (R_1, R_2, and C_1) where the notch is created by the combined parallel resonances of these two filters which produces a high impedance to the signal in the stop band. Sometimes the BRF is used to eliminate power line frequencies in audio systems.

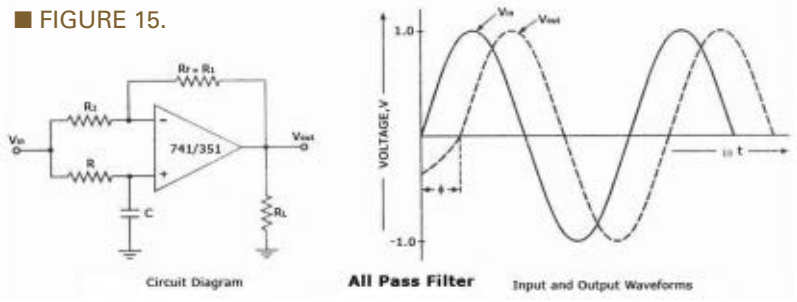
So far, we have looked at passive filters which have no active elements (transistors or amplifiers). By adding an active element such as an operational amplifier (op-amp), the filter can be built without inductors, which are large at low frequencies and are difficult to implement in an integrated circuit. The active filter bandwidth and center frequency can be controlled by varying resistors; the output of the filter is greater which helps subsequent stages, and the filter is isolated from the load which eliminates loading effects which can change the tuning of the filter. **Figure 13** shows the schematic of an active multiple feedback band stop filter and **Figure 14** shows this filter's frequency response. The center frequency of the circuit is $f_{CENTER} = 1/(2\pi SQRT((1/R_4C^2)(1/R_1+1/R_2)))$ and band width is BW = $2/(R_4C)$, where the gain of the op-amp G = $-(R_6/R_3)$. By adjusting the op-amp gain, you can easily adjust the notch bandwidth.

All Pass Filters (APF) provide equal gains for all input signals over a range of frequencies (flat frequency response) by phase shifting the various frequencies in a predictable manner. The APF can be used as a delay equalizer or phase corrector. **Figure 14** shows an active APF and the phase delay produced by this filter. APFs are used in audio system reverberation units and speaker crossovers.

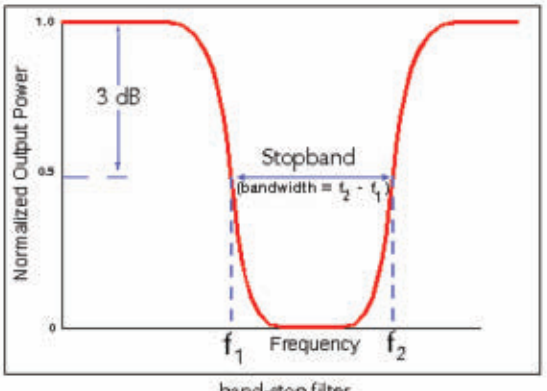
I have left out a lot of filters such as Butterworth, Chebyshev, Bessel, Pi, L, Surface Acoustic Wave (SAW), multi-pole, Sallen Key, etc. Also, I have not covered a lot of filter design in the form of Laplace transforms, transfer functions, and impulse response, which are the subjects of college engineering classes. See Q&A **SIDELINES** for online filter calculator websites. **NV**



■ FIGURE 13.



■ FIGURE 15.



band-stop filter

■ FIGURE 14.

■ BY RON HACKETT
www.jrhackett.net

Building a Simple Data Logger

As I mentioned in the previous installment of the Primer, this time we're going to continue our exploration of the FRAM breakout board that we experimented with last time (Adafruit.com product #1895), and implement a simple 08M2 based data logger that monitors and records the real time temperature of a backyard grill. However, before we get into the details of our current project, I think a little background information will be helpful.

It's probably an understatement to say that I'm a bit of a backyard grilling and smoking enthusiast. My wife would probably say "fanatic" since I do own a half dozen or so grills and smokers (she may be right!).

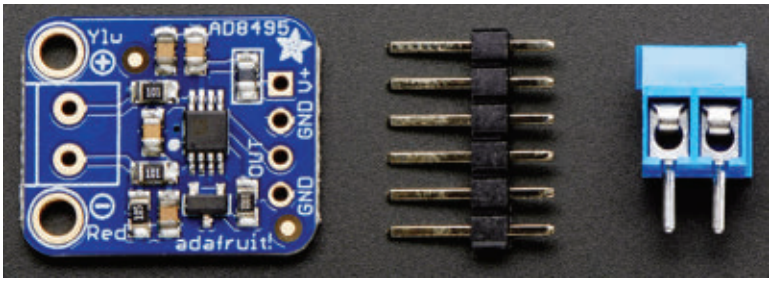
For quite a while now, I've been thinking about how I would go about developing a temperature control system for one of my kamado grills. For readers who aren't obsessed with charcoal grilling and smoking, kamados are among the most versatile backyard cookers available (IMHO), primarily because they are capable of maintaining a cooking temperature anywhere from 70°F or 80°F, to well over 800°F! In case you're wondering what anyone in their right mind would want to cook at those extreme temperatures, 70°F or 80°F is ideal for cold-smoking items like salmon or cheese, 225°F is perfect for slow-smoked baby back ribs, and commercial pizza ovens often operate at temperatures as high as 900°F!

Until now, two major problems have prevented me from actually getting started on a grill/smoker temperature control project: one is hardware related, and the other is software related. We'll get to the software problem before we're finished this month, but let's start with the hardware problem which is related to temperature sensors. The two sensors that we've used so far (DS18B20 and MCP9700A) both max out at 257°F, so they certainly aren't usable when baking a pizza. Even the usual grilling temperature of 450°F to 550°F is far beyond the operating range of both the DS18B20 and the MCP9700A. The obvious solution is to use a thermocouple sensor. Some thermocouple sensors are capable of measuring temperatures in excess of 2,000°F, and many inexpensive units can easily handle 900°F and higher.

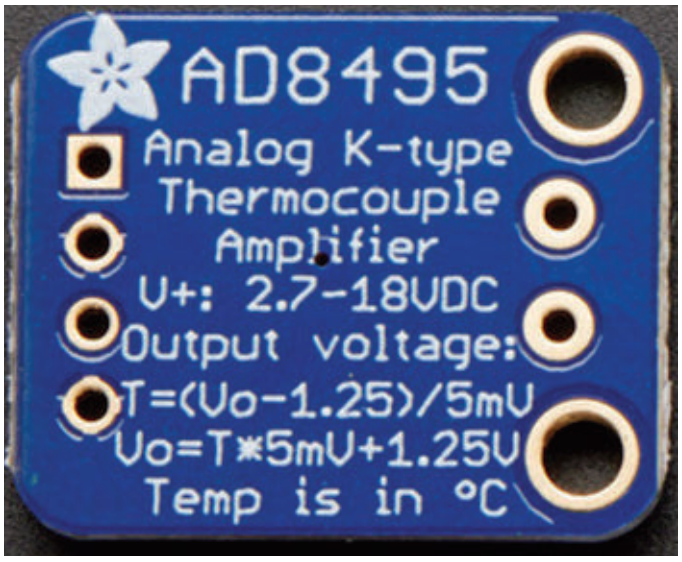
Similarly to thermoelectric coolers, thermocouple sensors are based on the "Peltier effect," but they function in a "reverse" manner. In a thermoelectric cooler, a DC current flows through the junction of two dissimilar pieces of metal, causing the temperature of one piece of metal (outside the cooling chamber) to increase, and the temperature of the other piece of metal (inside the cooling chamber) to decrease. A thermocouple sensor also contains a junction of two dissimilar metal wires, and applying heat to this junction causes a minute current flow that's proportional to the temperature of the heat source. These sensors are available in several different "types" which are distinguished by the two specific metal alloys used in the sensor.

The problem, however, is that thermocouples require an extremely sensitive analog amplifying circuit to provide accurate readings. Until recently, thermocouple amplifiers have been somewhat expensive and slightly complicated to use in a PICAXE project. However, just recently, I was once again searching for a thermocouple amplifier, and I found a new one at **Adafruit.com** that was exactly what I needed to get started (see **Figure 1**). All the surface-mount parts are pre-soldered, so it's only necessary to install the two-pin screw terminal for the thermocouple sensor wire and a four-pin male header for inserting the amplifier into a breadboard.

The Adafruit thermocouple amplifier board is based on the AD8495 K-type thermocouple amplifier from Analog Devices; it outputs an analog voltage signal, so it



■ FIGURE 1. AD8495 thermocouple amplifier board (front).

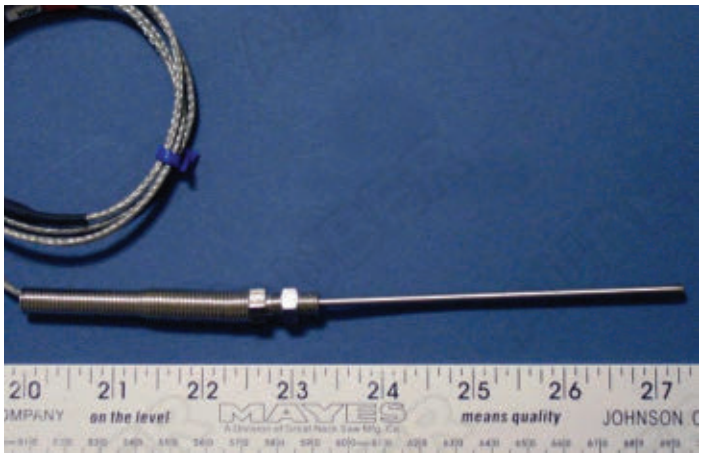


■ FIGURE 2. AD8495 thermocouple amplifier board (back).

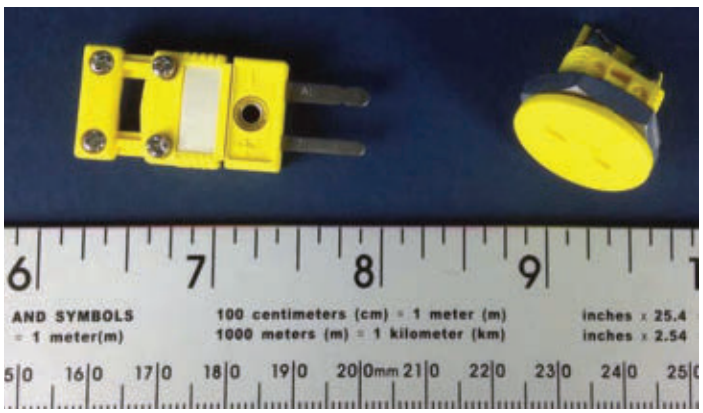
only requires one pin for interfacing. The board is so easy to use that a manual isn't even required. All the necessary information is printed on the back of the board (see **Figure 2**). We'll get into the details when we begin our experiments this month, but there's one thing I want to mention at this point.

Because the board provides an analog output, interfacing it with a PICAXE processor is almost identical to the MCP9700A based temperature measurement project we covered back in the December 2012 issue of *Nuts & Volts*. As a result, if you have no interest or need to monitor any temperatures higher than 257°F, you can easily substitute an MCP9700A sensor for the AD8495 board and thermocouple sensor in any of this month's experiments. The only software change would be to edit the portion of the software that converts raw ADC (analog-to-digital converter) input to degrees Fahrenheit, so that it matches the one we used in the original MCP9700A project. Also, the techniques we'll be using to record and display data can be readily applied to just about any data logging project you may want to implement.

Table 1.	Product #	Source
I²C Non-Volatile FRAM Breakout Board — 256 kBit (32 kByte)	1895	adafruit.com
K-Type Thermocouple Amplifier Breakout Board (AD8495)	1778	adafruit.com
K-Type Thermocouple Probe for Smoker or Oven (max 900°F)	TC-K3MM	auberins.com
Thermocouple Connectors, Male and Female	Connectors	auberins.com



■ FIGURE 3. Thermocouple probe.



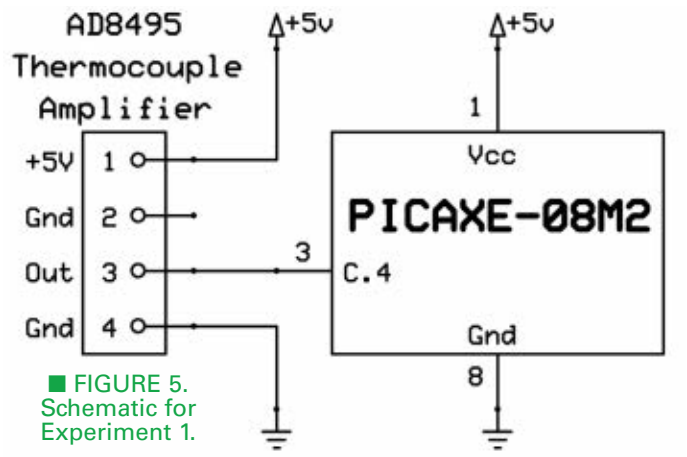
■ FIGURE 4. Thermocouple connector pair.

The AD8495 printed circuit board (PCB) can be used with any K-type thermocouple sensor. This type of sensor is readily available on the Internet, but I chose one from Auber Instruments (see **Figure 3** and **Table 1**) because its physical dimensions suited my requirements and it can measure up to 900°F, which is sufficient for my application. The Auber thermocouple comes with two standard spade connectors that can easily be removed, so the probe can be directly connected to the Adafruit thermocouple amplifier board.

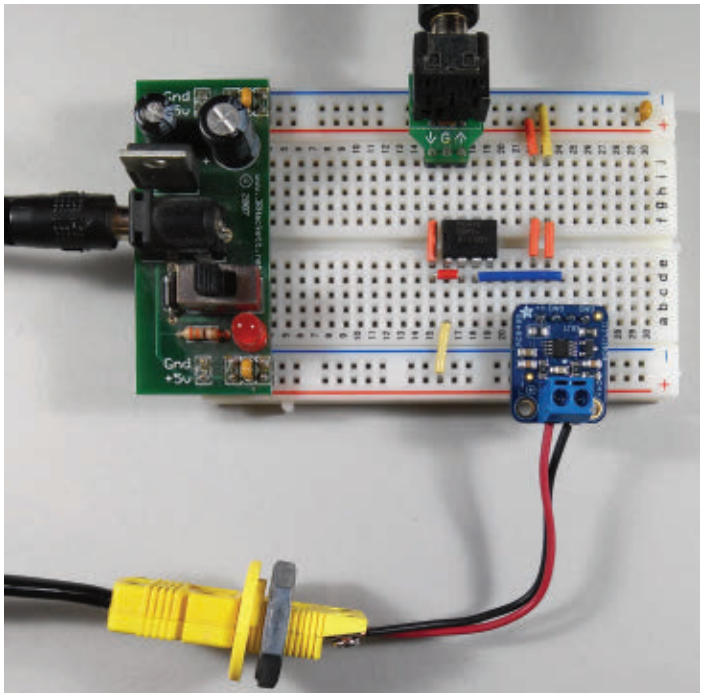
However, when I ordered the probe, I also ordered a pair of thermocouple connectors — one of which is a female panel-mount connector (see **Figure 4** and **Table 1**) because I plan to eventually mount the electronics inside a plastic project box. Also, the male connector fits into the standard female connector on one of my multimeters, so I can use the probe that way as well.

Experiment 1: Using a Thermocouple to Measure Temperature

In this experiment, we're simply going to test the thermocouple interface to make sure it's functioning correctly. The schematic for the experiment is presented in



■ FIGURE 5. Schematic for Experiment 1.



■ FIGURE 6. Breadboard setup for Experiment 1.

Figure 5. As usual, it doesn't include the standard PICAXE programming circuit, so, of course, you will need to include it to download the program. My breadboard setup is shown in **Figure 6**. As you can see, it's really simple.

There are only two points I want to mention. First, there's no need to ground both the AD8495 ground pins; just connect either one of them to ground. Second (and more important), the two thermocouple leads are polarized. Make sure you observe the correct polarity when connecting them to the AD8495. (The red lead is generally the positive lead, but check the documentation for the thermocouple you use to be sure.)

The software for this experiment (*ThermoTest.bas*) is also simple, but some of the conversion computations are not. So, let's take a look at them. (Before continuing, you may want to go to the article link, download all the files for this month's Primer, and print a copy of the *ThermoTest.bas* program for reference.)

Let's begin with the conversion of the 10-bit ADC value that's input from the AD8495 board. As you can see in the program listing, we're using the 08M2's internal fixed voltage reference (FVR) for the *readadc10* command. We used the FVR back in the December 2012 column with the MCP2700A temperature sensor in a battery-powered system, so I won't repeat all the details here. (You may want to re-read that article before proceeding.)

In the earlier article, we were using a three-cell alkaline battery pack, so we used an *fvrsetup FVR2048* command to configure the ADC reference voltage at 2.048V. This time, we're going to use a +5V supply so we can configure the reference voltage at 4.096V, because our supply voltage will never be lower than the reference voltage. As you know, a 10-bit ADC reading can have any one of 1024 values. This means that each ADC step is equivalent to an increase of 4 mV because 4 mV * 1024 steps = 4.048V, which is our reference voltage. This makes the conversion from the ADC reading to mV simple. Using the variable names defined in the ThermoTest.bas program:

$$\text{mVolt} = 4 * \text{ADCval}$$

Now, let's focus on the millivolt to Celsius conversion. As I mentioned earlier, all the documentation we need is printed on the back of the AD8495 board, so we'll start with the temperature formula that's printed on the back:

$$T = (Vo - 1.25)/5\ mV = \frac{Vo - 1.25}{5\ mV}$$

In the above equation, the two terms in the numerator are in volts, but the denominator is in millivolts. That's a little confusing and we can't work in volts anyway, so let's re-write the above equation so everything is in millivolts:

$$T = \frac{mV - 1250}{5}$$

Next, we multiply the top and bottom of the fraction by 2 (we'll soon see why!):

$$T = \frac{mV - 1250}{5} * \frac{2}{2} = \frac{2\ mV - 2500}{10}$$

At this point, we have the following equation:

$$T = \frac{2\ mV - 2500}{10}$$

If you refer back to the "documentation" on the back of the board, you will see that the variable "T" is in degrees Celsius. However, don't forget that we can't

directly compute a result like 22.6°C, so we need to "fool" the PICAXE compiler (which can only work with integers) by working in **tenths** of a degree, and then inserting a decimal point in the appropriate position. We can accomplish that by multiplying the above equation by 10, which simplifies the equation by eliminating the denominator. For clarity, the following equation uses the variable names that are defined in the *ThermoTest.bas* program:

$$tCx10 = 2 * mVolt — 2500$$

Now that we've computed the temperature in tenths of a degree Celsius, we need to convert that result to Fahrenheit. By now, you're probably familiar with the standard equation:

$$F = \frac{9 * C}{5} + 32$$

However, we need to work in tenths of a degree Fahrenheit, so we'll again use the variable names that we defined in the *ThermoTest.bas* program and change the 32 (degrees) to 320 (tenths of a degree):

$$tFx10 = \frac{9 * tCx10}{5} + 320$$

Keeping in mind the fact that the compiler performs all calculations in order (from left to right), we can re-write the above equation in the following form:

$$tFx10 = 9 * tCx10 / 5 + 320$$

Now that we've calculated the real time temperature in tenths of a degree Fahrenheit, we need to separate each of the digits in the *tFx10* variable so that we can correctly place the decimal point. The *bintoascii* command is a convenient way to do that, and the *sertxd* command inserts the decimal point in the correct position.

To experiment with the program, download it to your breadboard setup. You should see the real time temperature updated in the terminal window every two seconds.

I conducted three simple experiments with the program. For the first experiment, I used a small electric "hot pot" to bring some water to a boil, and then inserted the thermocouple probe into the boiling water. The temperature increased fairly quickly and stabilized at 213.2°F, which is accurate enough for me. (I live at sea level.) For my second experiment, I used a small butane mini-torch that I have set up as a hot-air gun for heat shrinking wire insulation. I held the tip of the hot-air emitter a fraction of an inch from the end of the probe, and watched the temperature increase to more than 400°F.

For my final experiment, I reconfigured the mini-torch to emit a pointed blue flame, and waved the flame back and forth about two inches from the end of the probe. The temperature gradually increased to more than 750°F, at which point I stopped because that's about as hot as I intend to run my kamado. You may want to run your own experiments to test your setup. Just make sure you don't exceed the maximum safe temperature for the thermocouple probe you use.

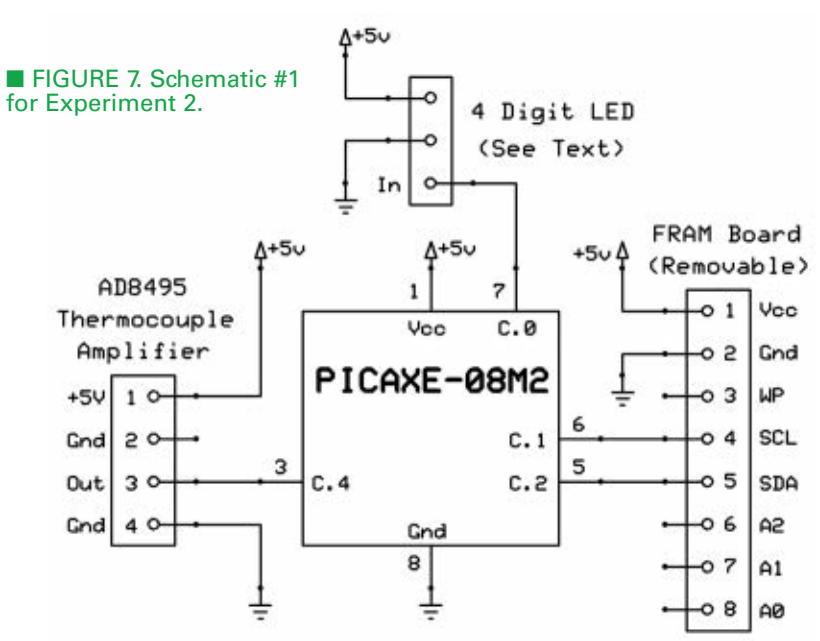
Experiment 2: Implementing a Simple Data Logging System

At this point, we're ready to put together a simple data logging system for monitoring and recording real time temperature measurements. We're going to use the Adafruit FRAM board from the previous column, the Adafruit AD8495 board thermocouple amplifier board, and a thermocouple sensor from Auber Instruments as we discussed near the beginning of this article (refer to **Table 1**). However, as I mentioned earlier, you can easily replace the last two items with an MCP9700A sensor if you don't need to do any high temperature monitoring.

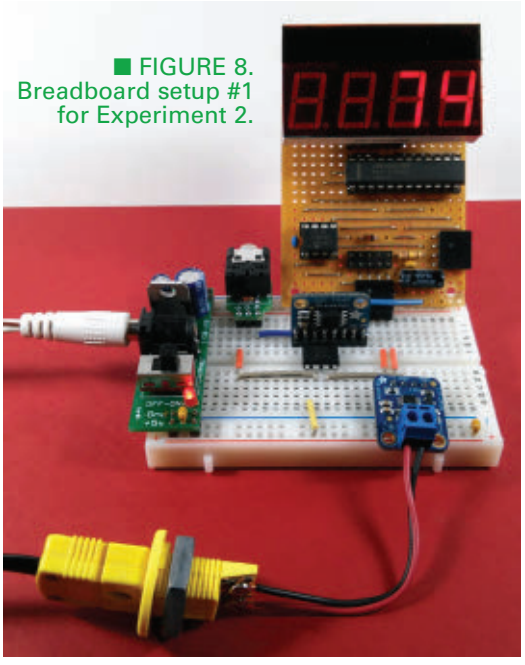
Before we proceed, I also need to explain how I intend to use the data logging system. I want to be able to monitor the kamado's real time temperature at two-second intervals, and display it on a seven-segment four-digit LED so that I can adjust the kamado vents as necessary to maintain the temperature within an acceptable range. I also want to log the temperature data to the FRAM at one-minute intervals so that I can later view the data in an Excel chart.

At some point in the future, I intend to make the entire system wireless, but for now I'm going to store the real time data in the FRAM on the monitoring circuit and use a "sneaker-net" approach to physically carry the FRAM inside the house to my Mac. There, I'll insert the FRAM into a second breadboard circuit that will read the data and serially send it to my Mac so I can view it in an Excel chart. Ultimately, I would also like to add the ability to control the temperature, so that it remains stable over long periods of time. However, I think of that goal as "phase 2" of a larger on-going project. For now, I'm just focused on the monitoring aspect. With that in mind, let's move on.

The schematic for the temperature logging circuit for this experiment is presented in **Figure 7**, and **Figure 8** is a photo of my breadboard setup. The LED display that I'm using is the stripboard project we completed way back in the April 2010 installment of the Primer, but you can use any serial LED display that you prefer. Because the LED's input pin is connected to pin C.0, which, of course, is the 089M2's *serout* programming pin, you will see a series of garbage characters rapidly displayed whenever you download a new program to the 08M2 on the



■ FIGURE 7. Schematic #1 for Experiment 2.



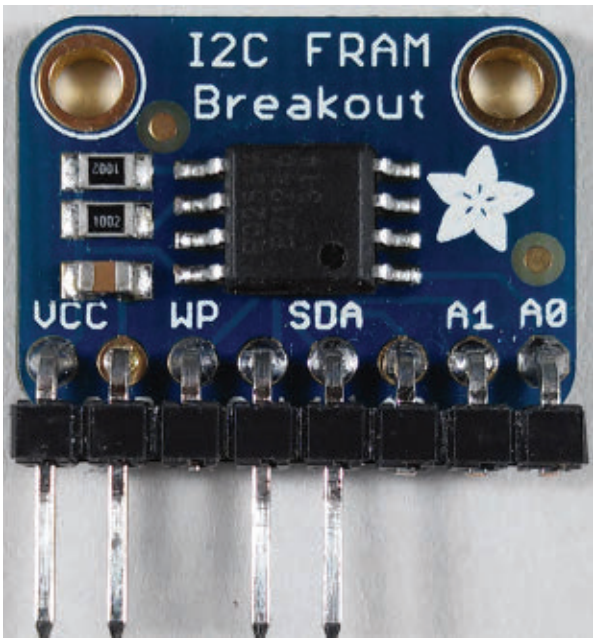
■ FIGURE 8. Breadboard setup #1 for Experiment 2.

breadboard. I expected that it might be necessary to cycle the breadboard power after each new download in order to properly initialize the LED display, but that didn't turn out to be true. However, it could be an issue for some LED displays, so you may want to keep it in mind.

The FRAM breakout board is the same one I used in the previous Primer. In other words, four of its pins have been snipped as shown in **Figure 9**. As we discussed last time, this arrangement simplifies the breadboard setup, and it also makes it easy to move the FRAM board back and forth between the two breadboard circuits we will use in this experiment.

Before we move on to the software for our data logging circuit, I want to mention one more point: Don't forget to connect pin 1 of the FRAM board to the upper +5V power rail. That connection isn't visible in **Figure 8**, but it's the reason the PICAXE programming adapter has to be offset from its usual position in line with the 08M2.

The program for the temperature logging circuit (*FRAMdataLogger.bas*) does require some explanation. As usual, the number preceding each of the following comments refers to the corresponding number along the left edge of the program listing:



■ FIGURE 9. FRAM board with four pins removed.

1. The iteration time of the main *do/loop* is two seconds. The accuracy of the loop timing is accomplished by using the 08M2's built-in time variable, which automatically increments once every second at 4 MHz or 16 MHz. (At 8 MHz, the incrementing interval is two seconds; at 32 MHz, the incrementing interval is 0.5 seconds.) The program runs at 16 MHz so that it can easily accomplish all its processing tasks within the two second time frame.

2. If you implemented Experiment 1, you may have noticed that the temperature reading tends to "wobble" up and down a bit. To minimize this, the program takes 10 temperature measurements in a one second time span, and then computes the average of the 10 readings.

3. Here, the program executes a secondary *do/loop* that allows the remainder of the two second iteration time to elapse.

4. Just before looping back, we compute the remainder that occurs when the time variable is divided by 60. The remainder will only be zero when one minute (60 seconds) has elapsed. If so, the data is stored, the *minute* variable is incremented, and the *time* variable is reset to zero

5. This is an easy method of

rounding off. At this point, the *tempF* variable is in tenths of a degree, so if we simply divided it by 10, we would truncate the "tenths" digit, not round it off. By first adding 5 to *tempF*, any value of 5 or greater in the tenth's digit of *tempF* will be increased by enough to cause a "carry" to the one's digit; if the value of the tenth's digit is less than 5, adding 5 will not cause a "carry" to occur. As a result, when we then divide *tempF* by 10 (truncating the tenth's digit), the presence or absence of the "carry" correctly rounds the result to an integer.

6. The LED display that I'm using is based on the MAX7219 display driver, which requires a value of 15 to blank any digit in the display. Therefore, if you use a different LED display, you may need to modify this portion of the code. Also, the four values are displayed from left to right, so sending 15 as the first value results in the left-most digit being blanked.

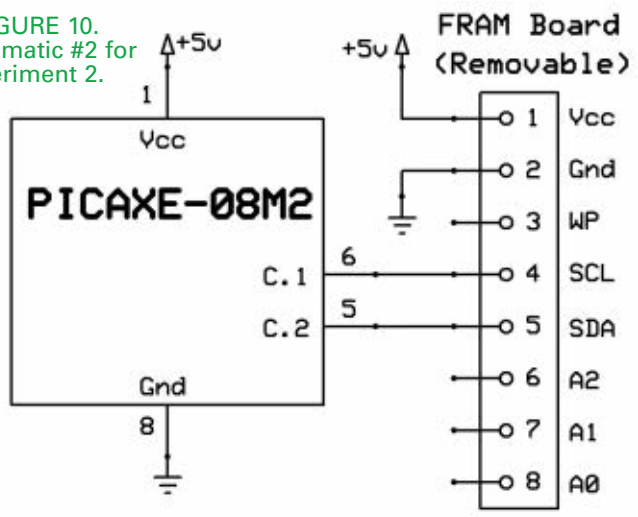
7. We want to right justify the displayed temperature value, so digits don't "bounce back and forth" whenever *tempF* increases or decreases above or below the value of 100. Therefore, if *tempF* < 100 degrees, we send another value of 15 to blank the second digit from the left, and then send the two-digit value of *tempF*. On the other hand, if *tempF* >= 100 degrees, we just send the three-digit value of *tempF*. As a result, the LED display is always right justified.

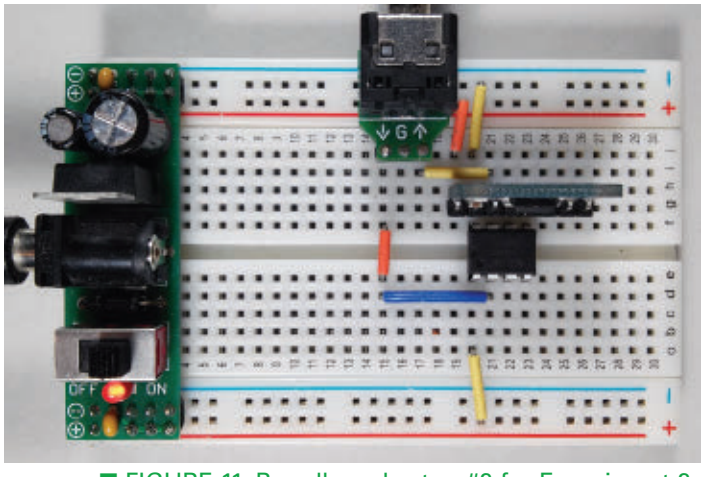
8. Each time the four bytes of data are stored in the FRAM, two additional "end of file" (*EOF*) markers (i.e., two values of 255) are also stored. When we power-down the program, carry the FRAM inside the house to the data reader breadboard, and transfer the data to the PC, the data reader software uses one of those values to determine when to stop transmitting the data to the PC. Of course, it is possible for the *minuteL* variable to have a value of 255. For example, if a "low and slow" cooking session lasts longer than four hours and 15 minutes, *minuteL* would equal 255 at that point in time. However, we would have to cook something for more than 45 days to store a valid data value of 255 in the *minuteH* variable! So, as we'll soon see, the data reader program stops reading the data as soon as *minuteH* contains the EOF marker.

At this point, we're ready to move on to the data reading portion of this experiment. Fortunately, this portion of the experiment is much simpler than the data logging portion we just discussed. The schematic for the data reading circuit is presented in **Figure 10**, and my breadboard setup — which is identical to one we used in the previous Primer — is shown in **Figure 11**.

This circuit really doesn't require an explanation. The only point I want to mention is that the +5V connection to pin 1 of the FRAM board — which wasn't visible in the temperature monitoring and logging circuit (see **Figure 8**) — can be clearly seen in **Figure 11**.

■ FIGURE 10. Schematic #2 for Experiment 2.





■ FIGURE 11. Breadboard setup #2 for Experiment 2.

The software for the data reading circuit (*FRAMdataReader.bas*) is also really simple. There's also only one point here I want to clarify about how the program uses the EOF marker to determine when to stop processing the data. When I first wrote the program, it included the following code:

```
hi2cin FRAMloc,(minL,minH,tempFL,tempFH)

do
  sertxd (#minute,",",#tempF,CR,LF)
  FRAMloc = FRAMloc + 4
  hi2cin FRAMloc,(minL,minH,tempFL,tempFH)
loop until minH = EOF
```

If you compare the above code snippet to what's written in the program listing, you can see that both versions essentially implement the same set of tasks. What's the difference? (I'm glad you asked.) See if you can figure it out before I explain why I modified the original code.

In the above code, the first four-byte data entry is read

■ FIGURE 12. Experiment 2 in action.

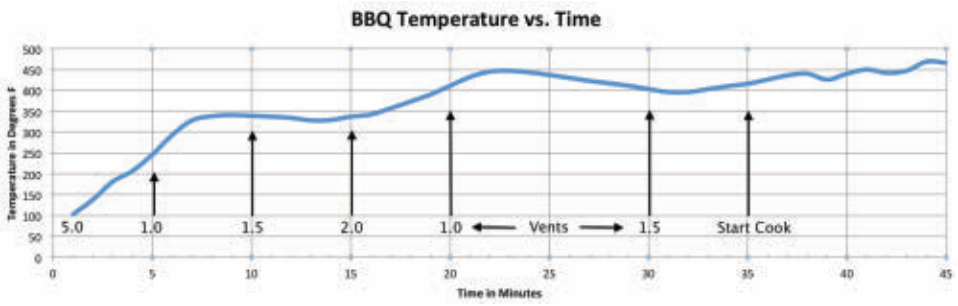
before entering the main *do/loop*, where it's processed. The last statement inside the loop reads the next line of data, and then the loop iterates. When the *EOF* marker is read into the *minH* variable, the "*until*" portion of the *do/loop* statement immediately ends the processing. In other words, the data line that includes the *EOF* marker is not sent to the terminal window. That's what I thought I wanted, but each time I tested the program I had the uncomfortable feeling that somehow the processing may have been terminated before all the data was sent to the terminal. (I guess you could say I have a trust issue.)

To make myself feel better, I re-wrote the software as shown in the program listing. If you trace through that *do/loop*, you will see that the data line that includes the EOF marker is sent to the terminal before the *until* portion of the *do/loop* tests for it and terminates the loop. That makes me feel better, and removing the invalid data line before making the Excel chart is easier (and cheaper) than going into therapy for my trust issue.

It was easy to set up the temperature data logging system with my kamado. I just drilled a small hole in the side of the kamado, inserted the thermocouple probe through the hole, and placed the breadboard data monitoring circuit on one of the kamado's side tables as you can see in **Figure 12**. The Excel chart for one of my recent grilling sessions where my "target" temperature was 450°F is presented in **Figure 13**. In the chart, you can clearly see the (somewhat delayed) effect on temperature whenever I manually adjusted the opening of the intake

air vent on the kamado.

I've been using this setup for a few weeks now, and it has provided me with a considerable amount of data for phase 2 of my larger project (i.e., using a PICAXE circuit to control the kamado's temperature for longer "low and slow" cooks). However, even if I never get to that project, the temperature monitor has provided an unexpected side benefit. I can now see the kamado's current temperature from my kitchen window. I no longer need to run outside every few minutes to do that.



■ FIGURE 13. Test results for Experiment 2.

In fact, after I finish this installment, my very next project is to design a PCB version of the monitoring circuit, install it in a project case, and include a larger LED display to make viewing the kamado's current temperature even easier.

What's Next?

Now that we can accurately monitor the temperature of an outdoor grill and/or smoker, the next logical step would be to develop a PICAXE project that can also control the temperature. Of course, there are already commercially available devices that provide this capability for charcoal grills and smokers by controlling the speed of a fan that feeds air to the fire. In fact, I already use a DigiQ DX2 digital controller from **BBQguru.com**, and it does its job very well.

However, in addition to my outdoor charcoal grills and smokers, I also have an electric smoker installed in the furnace room of my basement (you can never have too many smokers!), so I decided to experiment with using a PICAXE circuit to monitor and control its temperature.

Obviously, we can't use a fan to control the heat in an electric smoker, but it's easy to interface a solid-state relay (SSR) with a PICAXE processor, and plug the smoker into an AC outlet that's controlled by the SSR.

In fact, I've already conducted a few preliminary experiments to see if I can use an 08M2 circuit for that purpose, and so far the results are very good.

We can't get into the details this month because we're out of space again, but next time, that's what we'll do. Even if you have no need to control a smoker of any sort, being able to interface a PICAXE processor with an SSR makes a variety of electrical control projects possible. I'm sure you can think of one or two possibilities for your own projects.

See you next time. **NV**

NEW PRODUCTS

- HARDWARE
- SOFTWARE
- GADGETS
- TOOLS

TRACKED ROBOT KIT

Agent 390™ is a powerful tracked robot platform now available from Actobotics®. Incorporated is a smooth ball bearing supported drive track system to achieve high maneuverability and omnidirectional control. Although the contact patch on the tracks is smooth, they still offer fantastic traction and climbing ability.

The 18" x 16.42" aluminum chassis provides an extremely rigid base which can carry well over 50 lbs. The flat top plates offer an easy way to attach additional components directly to the platform, and the center track idler can be shifted to be closer to the front or back.

The side plates (triple channel brackets) are tied together using 6-32 tapped aluminum standoffs that also serve to hold the belt in place.

These plates incorporate the Actobotics hole pattern for mounting other components. Because builders may already have motors they want to utilize for driving the Agent 390, ServoCity offers this kit with or without the 313 RPM HD precision planetary gearmotors and mounts.

Pricing with motors is $389.99; pricing without motors is $329.99.

PERPENDICULAR/ PARALLEL DUAL GEAR RACK KITS

Also available from ServoCity is the 785 Dual Gear Rack Kit which is a simple way to create linear motion using a rotational servo. This kit is ideal for steering rack setups or other applications requiring smooth linear motion.

The kit comes with the multi-rotation Hitec HS-785HB servo which allows for up to 9.6" of travel (up to 19.2" of total travel) from each of the gear racks when sending the proper PWM signal from a servo controller. The kit is constructed of 6061 T6 aluminum components and wear-resistant Delrin plastic to create a durable yet lightweight assembly.

The framework has several mounting options and integrates seamlessly with the rest of the Actobotics line of products. The 785 Dual Gear Rack Kit is great for projects that require symmetric linear movement from a common center point using a PWM signal.

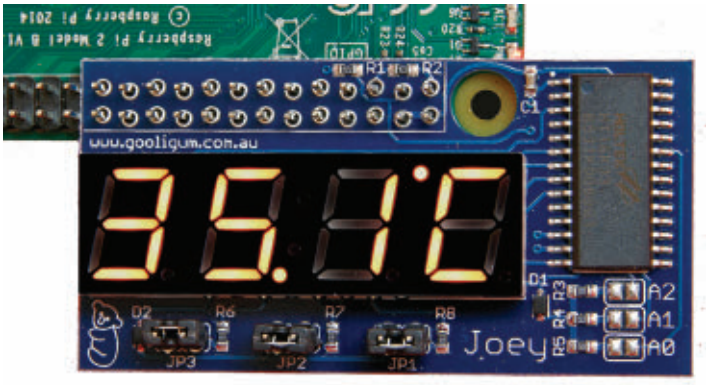
The kit includes the servo and all of the required hardware. Both the perpendicular and parallel versions of the kit retail for $99.99.

For more information, contact:
ServoCity
www.servocity.com

SEVEN-SEGMENT LED DISPLAY FOR THE RaspPi

The Joey is a compact four-digit seven-segment LED display board for the Raspberry Pi — recently released by Gooligum Electronics — which makes it easy to display numeric data or simple text which remains visible even when another expansion board is mounted on the Pi. It is physically thin and slides over the Pi's GPIO pins, allowing other add-ons to be attached in the usual

way. It connects via the Raspberry Pi's I²C bus and coexists nicely with other I²C devices. In the unlikely event of a clash, the Joey's I²C address can be changed. It doesn't use any dedicated GPIO pins, so it can coexist with almost any other add-on.



The Joey provides three digital inputs in addition to those available on the Raspberry Pi. These inputs can be accessed by your programs to enhance the functionality of your project.

The Joey comes with comprehensive documentation and a Python support library which makes it easy to display formatted numeric values, and even short text messages.

To get started, a set of example applications is provided, including a CPU temperature monitor (as shown in the photo), a clock, and an IP address display (which can be set up to run whenever the Raspberry Pi boots).

The Joey LED display board retails for $8. Bulk purchase and educational discounts are available.

For more information, contact:
Gooligum Electronics
www.gooligum.com

DC/DC MULTI CHARGER

Hitec's powerful X2-700 Multicharger is a sophisticated charging choice for all high amperage batteries. This DC powerhouse features two 700 watt output ports for increased capacity and supports all battery chemistries, including the latest LiHV cells.

The microprocessor control and an internal resistance meter make this a key component of the workshop. Hitec's free "Charge Master" software allows full PC control of the X2-700 through its integrated USB port, from basic setup to full software analysis of a battery's health and performance.

Users can streamline operations with the new Synchronous Mode as the second output port automatically mirrors channel 1 settings to charge multiple similar batteries with minimal setup time. The Multicharger is reliable and durable, and a must-have charger for all seasoned hobbyists.

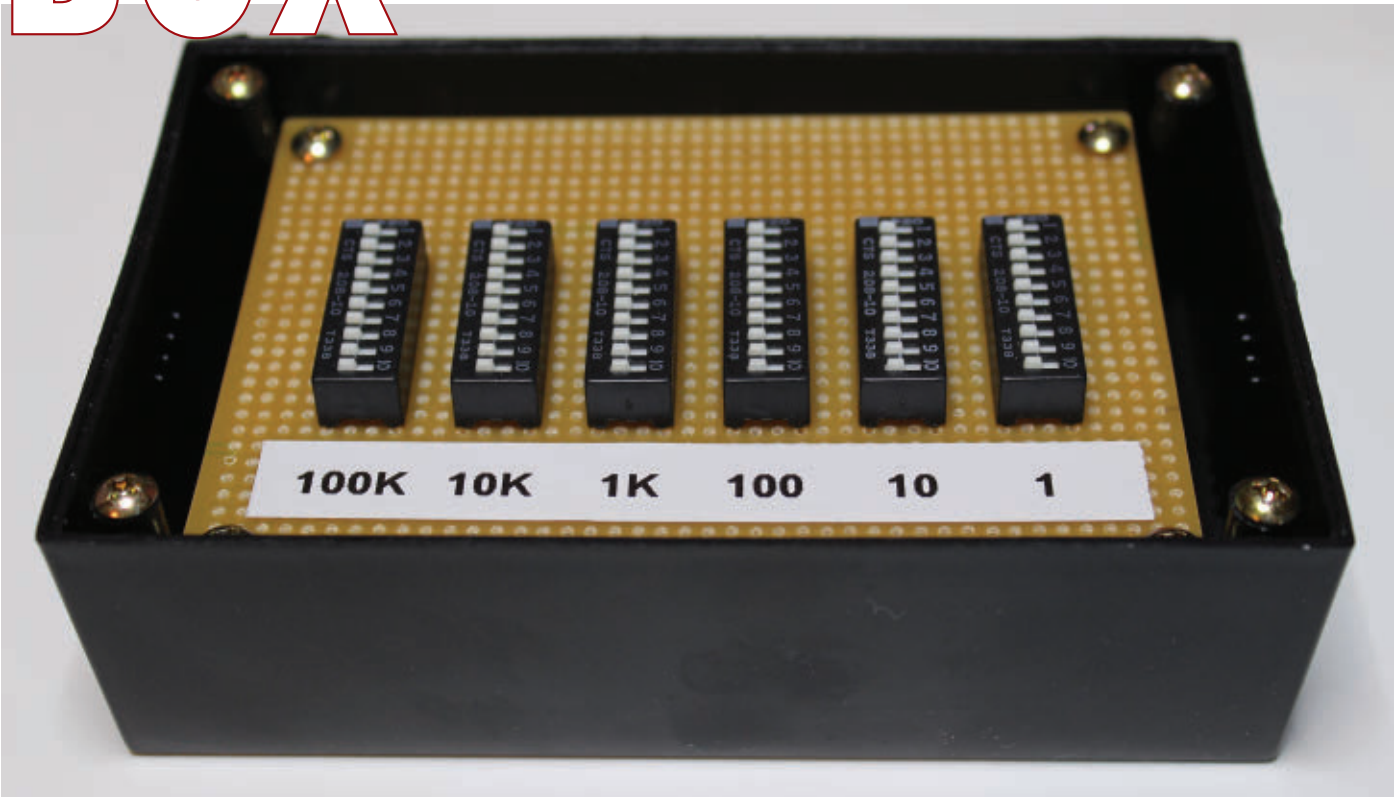
The estimated retail price is $214.99.

For more information, contact:
Hitec RCD USA, Inc.
www.hitecrcd.com

BUILD A DIGITAL DECADE BOX

By Roger Secura

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/magazine/article/february2016_DigitalDecadeBox.**

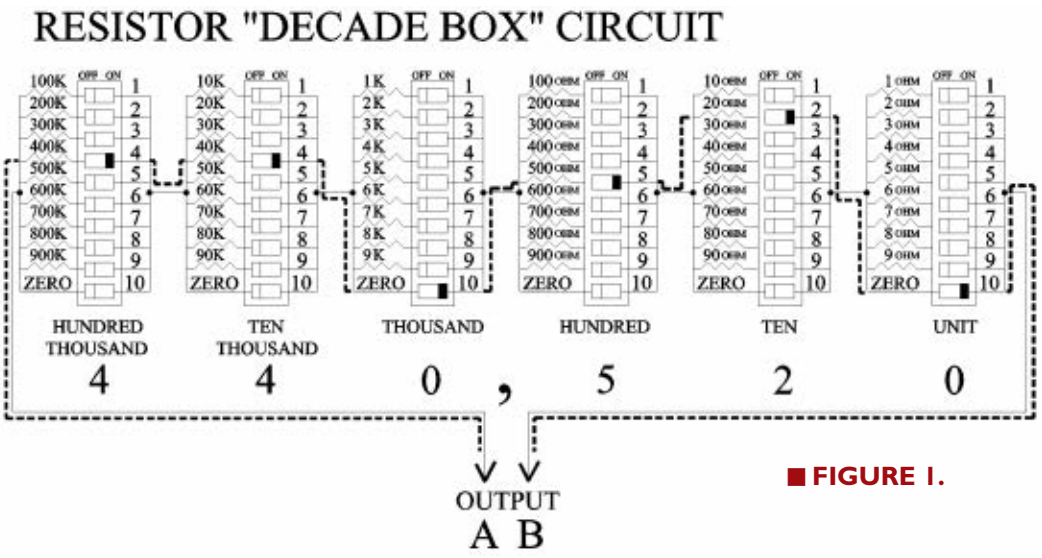


Having the right tool makes any job easier. Just try and remove a sparkplug without a good deep-well socket and wrench. The same holds true in the field of electronics. The multimeter, signal generator, oscilloscope, and frequency counter are just some of the "tools" that not only speed up the design process, but reduce the urge to pull one's hair out in frustration.

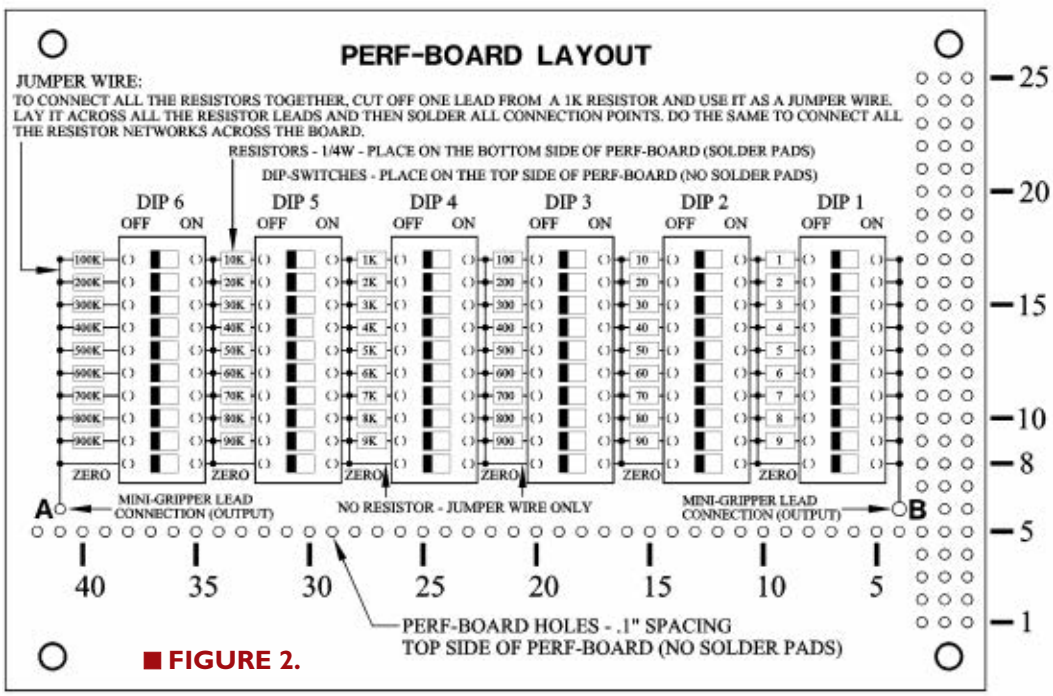
In addition to the main players, there are a few hacker tools (black boxes) that some of the financially challenged (like me) build to help in the process of circuit design and testing. One of these electronic contraptions is the resistor "decade box." A resistor decade box is a chain of resistor networks which allows you to "dial in" any resistor value you need simply by adjusting a few knobs. If you can't afford a commercial decade box, I would suggest checking out Frank Muratore's article in the April 2014 issue of *Nuts & Volts* and build your own. His simple black box design is low in cost and real easy to build — I use it all the time.

As nice as Muratore's design works, however, the potentiometers he employed in his design are notorious for their sensitivity. That means you have to hook up your ohmmeter and adjust the knobs back and forth several times in order to get the exact resistance you need. What I really wanted was a decade box where I could select the required resistance with a couple of switches — without having to hook up an ohmmeter to see the actual resistance value. To solve that problem, I designed the digital decade box presented here.

This is a great project for the beginner. You'll get to sharpen your soldering skills for your next project. What follows is the construction and operating instructions you'll need to build and operate the decade box.



■ FIGURE 1.



■ FIGURE 2.

Circuit Operation

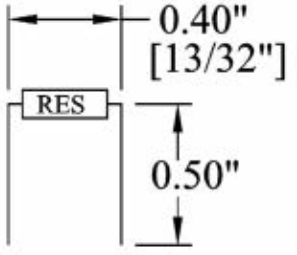
The circuit for the decade box is shown in **Figure 1**. You'll notice that in order to select a resistance value (440.520K, for example) we select the appropriate DIP switches in each column according to the value we need.

This completes the circuit, and the total resistance of all the switches will be available at the output (A, B). It's important that you set all DIP switches that are not selected for a resistance value to zero ohms (switch # 10). In other words, if you select 10K for a resistance value, all the other DIP switches must be set to zero ohms (switch #10 - ON). Remember, you have to complete the electrical circuit in order to get a 10K resistance at the output (A, B).

On the other hand, don't forget to move the zero switch to the off position for any DIP switch selected for a resistance value.

Construction

■ FIGURE 3.



The layout for the circuit is shown in **Figure 2**. Place the DIP switches on the top side of the perfboard (no solder pads) and all the resistors on the bottom side of the board (solder pads). The board is small, so count the holes carefully. You should see two holes between each DIP switch. Insert DIP switches #1and #2. Place them on the top side of the board (no solder pads).

Before you start soldering anything,

HARDWARE

ITEM	QTY	DESCRIPTION	PART#	SOURCE
Black Box	1	Plastic Box — 5.3" x 3.3" x 1.6"	TB-4	www.allelectronics.com
Perfboard	1	3-1/8" x 4-5/16"	PC-4	www.allelectronics.com
Test Leads	1	Banana Plug to Mini-Grabbers	TL-5	www.allelectronics.com

PARTS LIST

ITEM	QTY	DESCRIPTION	PART#	SOURCE
R1	1	1 ohm,1/4W, 5%	1.0QBK-ND	Digi-Key
R2	1	2 ohm,1/4W, 5%	2.0QBK-ND	Digi-Key
R3	1	3 ohm,1/4W, 5%	3.0QBK-ND	Digi-Key
R4	1	3.9 ohm,1/4W, 5%	3.9QBK-ND	Digi-Key
R5	1	5.1 ohm,1/4W, 5%	5.1QBK-ND	Digi-Key
R6	1	6.2 ohm,1/4W, 5%	6.2QBK-ND	Digi-Key
R7	1	7.5 ohm,1/4W, 5%	7.5QBK-ND	Digi-Key
R8	1	8.2 ohm,1/4W, 5%	8.2QBK-ND	Digi-Key
R9	1	9.1 ohm,1/4W, 5%	9.1QBK-ND	Digi-Key
R10	1	10 ohm,1/4W, 5%	10QBK-ND	Digi-Key
R11	1	20 ohm,1/4W, 5%	20QBK-ND	Digi-Key
R12	1	30 ohm,1/4W, 5%	30QBK-ND	Digi-Key
R13	1	40.2 ohm,1/4W, 5%	40.2XBK-ND	Digi-Key
R14	1	49.9 ohm,1/4W, 5%	49.9XBK-ND	Digi-Key
R15	1	60.4 ohm,1/4W, 5%	60.4XBK-ND	Digi-Key
R16	1	69.8 ohm,1/4W, 5%	69.8XBK-ND	Digi-Key
R17	1	80.6 ohm,1/4W, 5%	80.6XBK-ND	Digi-Key
R18	1	90.9 ohm,1/4W, 5%	90.9XBK-ND	Digi-Key
R19	1	100 ohm,1/4W, 5%	100QBK-ND	Digi-Key
R20	1	200 ohm,1/4W, 5%	200QBK-ND	Digi-Key
R21	1	300 ohm,1/4W, 5%	300QBK-ND	Digi-Key
R22	1	402 ohm,1/4W, 5%	402XBK-ND	Digi-Key
R23	1	499 ohm,1/4W, 5%	499XBK-ND	Digi-Key
R24	1	604 ohm,1/4W, 5%	604XBK-ND	Digi-Key
R25	1	698 ohm,1/4W, 5%	698XBK-ND	Digi-Key
R26	1	806 ohm,1/4W, 5%	806XBK-ND	Digi-Key
R27	1	909 ohm,1/4W, 5%	909XBK-ND	Digi-Key
R28	1	1K ohm,1/4W, 5%	1.0KQBK-ND	Digi-Key
R29	1	2K ohm,1/4W, 5%	2.0KQBK-ND	Digi-Key
R30	1	3K ohm,1/4W, 5%	3.0KQBK-ND	Digi-Key
R31	1	4.02K ohm,1/4W, 5%	4.02KXBK-ND	Digi-Key
R32	1	4.99K ohm,1/4W, 5%	4.99KXBK-ND	Digi-Key
R33	1	6.04K ohm,1/4W, 5%	6.04KXBK-ND	Digi-Key
R34	1	6.98K ohm,1/4W, 5%	6.98KXBK-ND	Digi-Key
R35	1	8.06K ohm,1/4W, 5%	8.06KXBK-ND	Digi-Key
R36	1	9.09K ohm,1/4W, 5%	9.09KXBK-ND	Digi-Key
R37	1	10K ohm,1/4W, 5%	10KQBK-ND	Digi-Key
R38	1	20K ohm,1/4W, 5%	20KQBK-ND	Digi-Key
R39	1	30K ohm,1/4W, 5%	30KQBK-ND	Digi-Key
R40	1	40.2K ohm,1/4W, 5%	40.2KXBK-ND	Digi-Key
R41	1	49.9K ohm,1/4W, 5%	49.9KXBK-ND	Digi-Key
R42	1	60.4K ohm,1/4W, 5%	60.4KXBK-ND	Digi-Key
R43	1	69.8K ohm,1/4W, 5%	69.8KXBK-ND	Digi-Key
R44	1	80.6K ohm,1/4W, 5%	80.6KXBK-ND	Digi-Key
R45	1	90.9K ohm,1/4W, 5%	90.9KXBK-ND	Digi-Key
R46	1	100K ohm,1/4W, 5%	100KQBK-ND	Digi-Key
R47	1	200K ohm,1/4W, 5%	200KQBK-ND	Digi-Key
R48	1	300K ohm,1/4W, 5%	300KQBK-ND	Digi-Key
R49	1	402K ohm,1/4W, 5%	402KXBK-ND	Digi-Key
R50	1	499K ohm,1/4W, 5%	499KXBK-ND	Digi-Key
R51	1	604K ohm,1/4W, 5%	604KXBK-ND	Digi-Key
R52	1	698K ohm,1/4W, 5%	698KXBK-ND	Digi-Key
R53	1	806K ohm,1/4W, 5%	806KXBK-ND	Digi-Key
R54	1	909K ohm,1/4W, 5%	909KXBK-ND	Digi-Key
SW1-SW6	6	DIP Switch SPST, 10SEC	CT20810-ND	Digi-Key

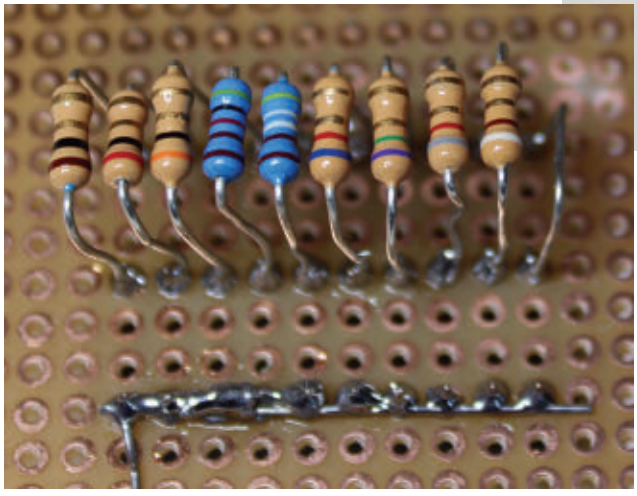
scotch tape both DIP switches to the surface of the board so they don't pop out during the soldering process (trust me, it works). Take the one ohm resistor from the **Parts List** and bend it according to **Figure 3.**

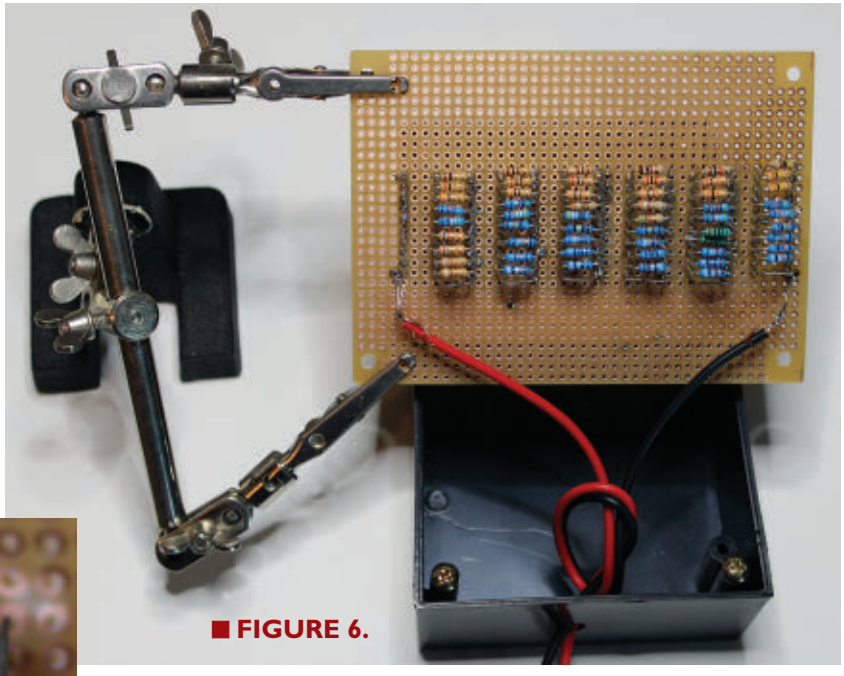
Now, turn the board over and insert the one ohm resistor into the DIP switch terminal holes as shown in **Figure 2**. Yes, it's a little tight, but the resistor leads and the DIP switch terminals will fit into the same solder pad holes. Solder one side of the resistor (one ohm) lead to DIP switch #1 and the other lead to DIP switch #2. Don't insert the next resistor (two ohm) until you are confident it's a good solder joint. Repeat this procedure (one resistor at a time — one ohm to nine ohms) until you have all the resistors mounted between DIP switch #1 and DIP switch #2 (see **Figure 4.**) Notice that switch 10 on each DIP switch is just a jumper wire.

Okay, here's the tricky part. Once all the resistors are mounted and soldered across DIP switch #1 and #2, solder all the other terminals of DIP switch #1 together as shown at the bottom of the photo in **Figure 4** and at the top of the photo in **Figure 5.** The trick is to cut off the lead of a resistor you don't need and lay it across the DIP switch terminals, then solder all the terminals together (again, scotch tape helps). You'll need to use six resistor leads as jumpers for all six DIP switches, and one lead for the resistors located on DIP switch #6.

Author's note: VERIFY ALL RESISTANCE VALUES BEFORE SOLDERING. Unfortunately, the company where I purchased my resistors sent the wrong resistance value for one resistor. I was forced to string three resistors together to make 60K. This made it difficult to fit in between the other resistors. Also, don't bend and cut all the resistors in this project at the same time. You'll get confused as to what resistor value you have laying on your work bench. This will force you to get out the ohmmeter and verify the resistance value – annoying!

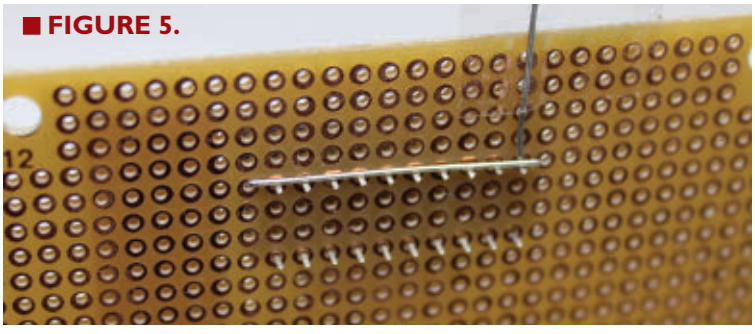
■ **FIGURE 4.**





■ **FIGURE 6.**

■ **FIGURE 5.**



When you complete the soldering, your decade box should look something like **Figure 6.**

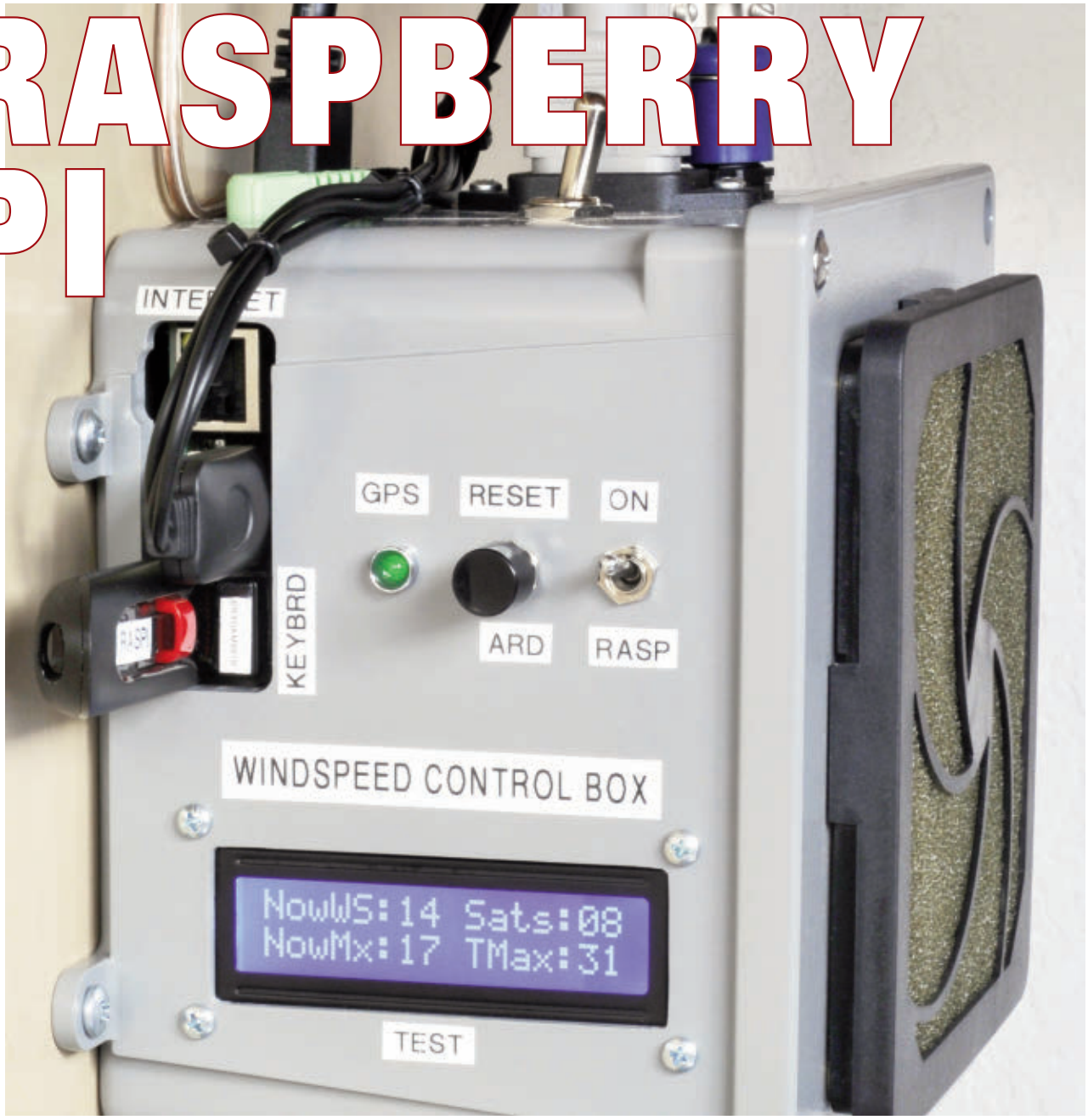
Final Note

Some precision resistors can only be purchased in minimum quantities of 100 or 1,000 pieces. When I designed this decade box, I made the decision to buy a few resistors that were as close to the required value as possible (60.4K for a 60K), rather than purchasing 100 resistors just to get one that is exactly 60K. If you want a more accurate response from your resistor decade box, then you may want to search the Internet for lower tolerances.

I hope you enjoy your new tool for decades! **NV**

BUILD A FUN WIND SPEED TRACKER WITH A RASPBERRY PI

By David Goodsell



For years, I've had a Davis weather station and enjoyed tracking the wind — especially the sudden gusts that rattle the windows. However, the tiny plot on the station's LCD screen is not very resolved and I wanted it to be better ... a whole lot better. The solution: a Raspberry Pi!

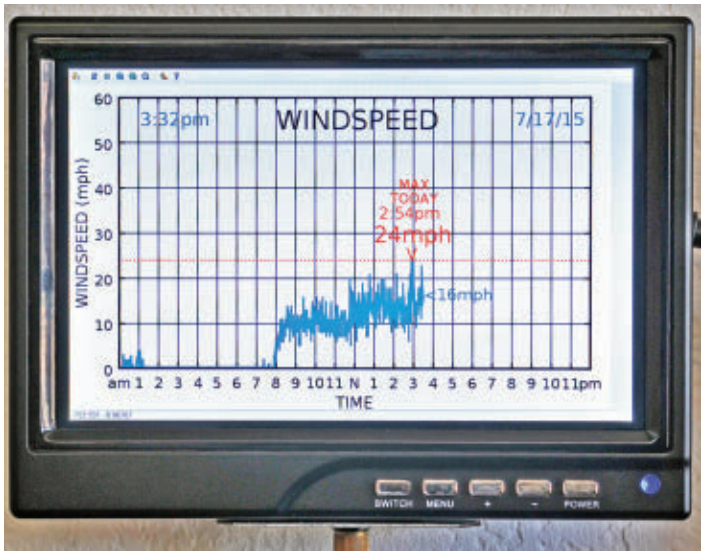
Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/magazine/article/february2016_RaspPiWindSpeedTracker.**

The Big Picture

I was gassed when I realized that a Raspberry Pi (RPi) could run a plotting program like "Gnuplot" and generate spiffy real time plots of wind speed on one of the small LCD displays from Adafruit (refer to **Figure 1**). The best part was customizing the display to look just the way I wanted it to.

The seven inch 1280 x 800 display looks great on my wall. The colors are vibrant and the resolution is more than I could hope for. During operation, the RPi updates the peak wind speed once per minute, 1,440 times a day, 24/7, and resets at midnight. It's fun to watch the peaks climb higher and higher, as I wonder if the house is going to blow away.

The schematic diagram of the project is shown in **Figure 2**. I realize that some of the functions could be combined to reduce the parts count, but I already had most of them in my stock so I shoved them all into a box and started programming. **Figures 3** and **4** show the box mounted under my desk.



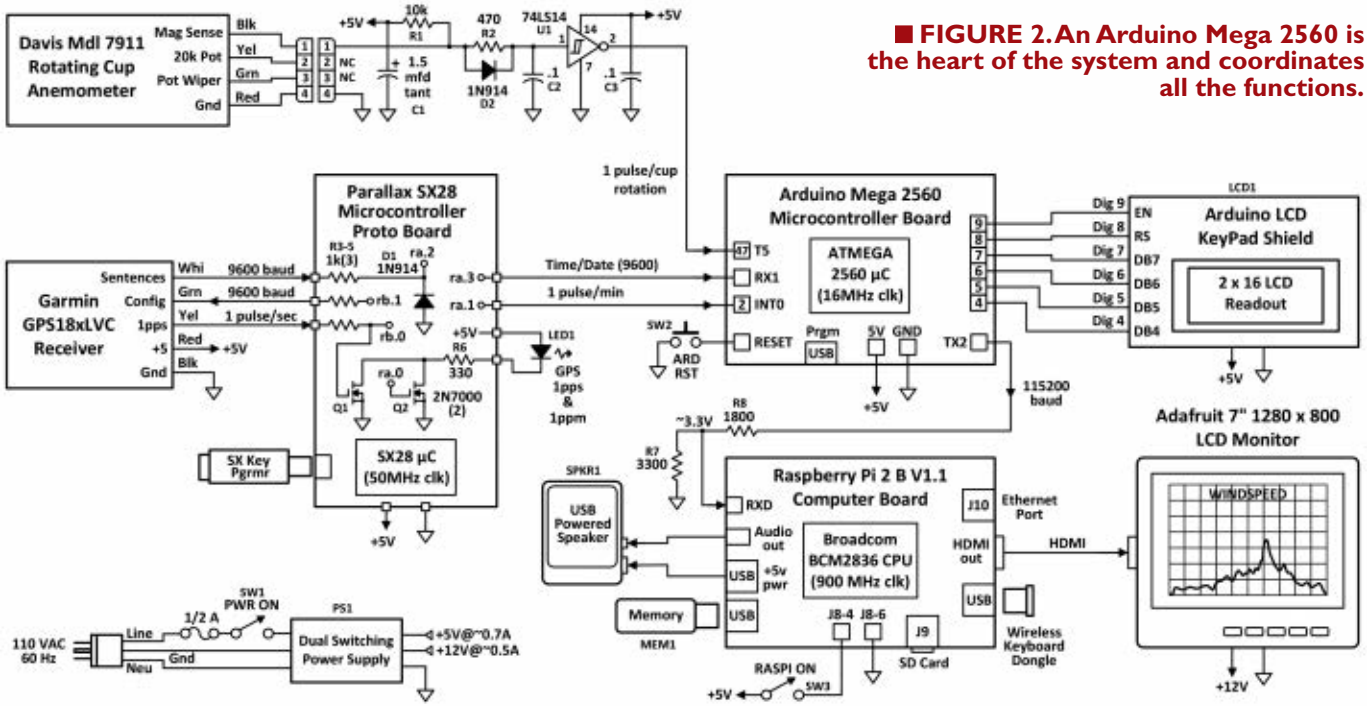
■ **FIGURE 1. The Adafruit seven inch 1280 x 800 LCD plots the max wind speed at one minute intervals, 1,440 times a day.**

Davis Anemometer

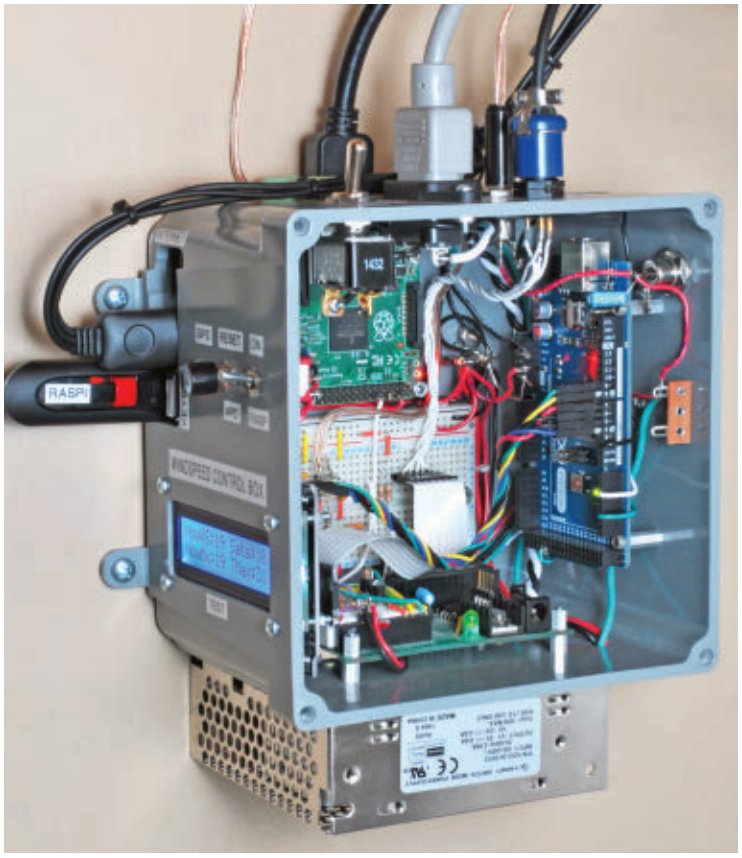
As seen in the schematic, the project starts with the anemometer on the left. Many years ago, I purchased a Davis Model 7911 anemometer and it's still working today (see **Figure 5**). I checked eBay and there are some less expensive units from overseas, but their outputs may not

RESOURCES

www.adafruit.com (displays and Raspberry Pis)
www.ebay.com (anemometers)
www.davisnet.com (anemometers)
www.dfrobot.com (cheap 2x16 LCD shield)
www.digikey.com (general components)
www.parallax.com (SX28 IDE)
Arduino Cookbook, by Michael Margolis, 2013



■ **FIGURE 2. An Arduino Mega 2560 is the heart of the system and coordinates all the functions.**

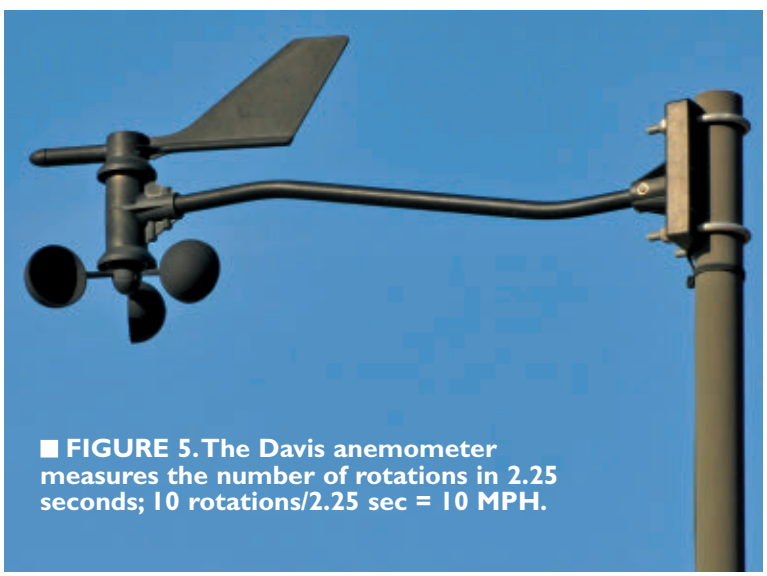


■ **FIGURE 3. The control box contains three processors: a Parallax SX28, Arduino Mega 2560, and Raspberry Pi 2.**

■ **FIGURE 4. Here, 8-10 GPS satellites are typically in use (as shown on the LCD), even with the GPS receiver mounted indoors.**

be compatible. Both eBay and Amazon sell the older model 7911.

The cups in the Davis unit rotate a magnet past a magnetic sensor once every revolution. The calibration is extremely simple. You count the number of pulses occurring in 2.250 seconds: 10 pulses = 10 MPH; 20 pulses = 20 MPH; and so on. The unit also contains a 20K ohm continuous-rotation potentiometer which can be used to track the direction of the wind. However, I didn't use the pot in this project.



■ **FIGURE 5. The Davis anemometer measures the number of rotations in 2.25 seconds; 10 rotations/2.25 sec = 10 MPH.**

The output of the anemometer is filtered to eliminate any noise from the sensor or the 75 feet of cable running through the attic. I didn't employ any lightning arrestors, although whenever a thunderstorm comes along I can't help but think about whether the next lightning strike is going to electrify my desk and me with it!

The pulses from the cups are counted by an Arduino (also used in this project) for exactly 2.250 seconds and stored in a small array. This counting process repeats every 2.250 seconds until a full minute has elapsed, so there are approximately 25 readings/minute. Next, the peak wind speed from the 25 readings is determined and stored as the next data point to be plotted, i.e., each minute the peak during the past minute is plotted. Of course, you could plot the average value for the past minute instead of the peak.

GPS Receiver and Decoder

For the time base, I chose the Garmin GPS receiver shown in **Figure 6**, mainly because I already had it left over from a previous project. I could have used an RTC (Real Time Clock) but then I would have had to add more buttons to set the time. Besides, in my junk box I also had a Parallax SX28 microcontroller programmed to parse the sentences and output the time and date over a 9600 baud five volt line. I bought a bunch of SX28 protoboards when Parallax had an End-Of-Life sale several years ago; they are handy for small projects like this one.

The time and date are parsed from the GPS RMC sentence and the number of satellites in use is taken from the GGA sentence. As you'll notice in **Figure 4**, I display the number of satellites on the 2x16 LCD, only because I am awed by the whole GPS concept and love to see it working. The Garmin unit works fine inside my single-story house, but your reception may vary.

For some obscure reason, I decided to write my own Arduino routine to convert UTC time to local time/date, instead of adapting a Library routine. Big mistake! It looked simple enough, but then I realized that I would have to include time zones, leap years, and daylight savings. Garf! After many hours of debugging, I got it working and confirmed that it converts the UTC data to the correct time and date for all 50 states. Just enter your Time Zone (TZ) in the Setup Section of the code before you load it. If you are not in the US, simply adapt the code to fit your location.

Program Flow

The basic program flow is as follows:

1. In the background, the Arduino measures wind speed at 2.250 intervals, and the SX28 reads the GPS time/date and number of satellites.

2. At the end of each minute, the SX28 sends a pulse to the Arduino.

3. The Arduino then:

a. Determines the peak velocity for the past minute and adds the value to the daily array (1440 points).

b. If the daily maximum velocity has increased, it moves the red fid mark and value to a new point, and sounds an audio signal.

c. It serially transmits updated config and data array files to Gnuplot in the RPi.

4. The RPi sends a new HDMI plot to the LCD monitor, but in a small window.

5. The RPi expands the window to fill the LCD screen.

6. Steps 3, 4, and 5 are repeated at the end of each minute, 1,440 times per day.

7. Everything resets at midnight.

■ **FIGURE 6. Using a GPS receiver to extract the time and date doesn't require setting, but an RTC could be substituted.**

8. The GPS GMT time/date from the SX28 is converted to local time as it comes in.

The biggest problem I had with the programming was juggling the code for the three different processors. The SX28 used assembly language, the Arduino used C, and the RPi required a combination of command-line programming and Python. I had a heck of a time keeping track of which language I was using at any one time. Plus, it was my first experience with Python, so there was a learning curve. Fortunately, the Internet has a cornucopia of helpful hints and I was able to work my way through the problems. I also bought several books and tapped my knowledgeable friends.

This project was also my first experience with the fantastic free Gnuplot plotting program. It was fun to customize the display, but it was almost impossible to stop tweaking it. (You know how it is.) I chose to locate the main Gnuplot configuration file in the Arduino because I needed to update many of the displayed numbers 1,440 times a day. Normally, the config file would be permanently located in the RPi main directory. Instead, I send an updated config file to the RPi at the end of each minute, at 115200 baud.

The source code for the Parallax SX28, Arduino Mega, and RPi is available at the article link. Also available is a document explaining how to install the necessary support applications/tools for the RPi.

Booting Up and Other Details

Getting all three processors to boot up concurrently

PARTS LIST

DESIG	COMPONENT	Digi-Key (unless noted)
R1	Resistor, 10K	10KQBK
R2	Resistor, 470 ohms	470QBK
R3-R5	Resistor, 1K	1.0KQBK
R6	Resistor, 330 ohms	330QBK
R7	Resistor, 3.3K	3.3KQBK
R8	Resistor, 1.8K	1.8KQBK
D1, D2	Diode, 1N914	1N914BCT
C1	Capacitor, 1.5 mfd 35V	478-1853
C2, C3	Capacitor, 0.1 mfd 50V	399-8994-1
SW1	Toggle Switch, SPDT	480-3077
SW2	Pushbutton, N.O.	RadioShack
SW3	Toggle Switch, SPDT	CKN1023
LED1	LED, Green	RadioShack
LCD1	LCD Shield, 2x16	DF Robot, DFR0009
PS1	Power Supply, 5V, 12V	102-2527
Q1, Q2	FET, 2N7000	2N7000TACT
U1	IC, 74LS14 Schmitt Trig	296-3638-5
MEM1	Thumb Drive, 8 GB	Best Buy
SPKR1	Insignia Portable Spkr	Best Buy
BOX	Carlon/Cantex, 6x6x4	Lowe's/Home Depot

Note: The major components/boards are labeled on the schematic.

was almost my downfall. I could not figure out how to autostart the .py program on the RPi. I scoured the Internet for several days and tried numerous suggestions. No joy.

Finally, I found a reference to an autostart file located deep within the /etc/xdg/lxsession/LXDR-pi subdirectory. The reference suggested adding @lxterminal and the name of the wind speed program to the file, and bingo! It took off. Now, if the power fails, the three processors boot themselves up, collect data for several minutes, and start plotting a full screen. It's amazing that the system runs all day and night, day after day with no glitches. That's a total of 1,440 plots/day x 365 days = 525,600 plots/year.

Speaking of problems, the Arduino Mega has a subtle shortcoming. Not all of the pins of the 2560 controller chip are connected to the header on the board, and some are fairly important. For example, most of the external inputs to the six timer/counters are missing. I found this out the hard way when I thought I could use any one of the timers to count the pulses from the anemometer. I checked the Mega schematic and found that there were no external (Tn) connections to Timers 1, 2, 3, or 4. Only Timers 0 and 5 were hooked up. However, Timer 0 was tied up by the Mega to support some of the timing functions, which left just one Timer: T5.

In fact, the most important code for counting the anemometer pulses uses Timer 5 and is only five lines long:

```
TCNT5 = 0x00          //reset timer/counter 5
TCCR5B = 0x06         //start timer/counter 5, on
                      //falling edge of external pulse
                      //on T5
delay(2250)           //continue counting for 2.250
                      //seconds
TCCR5B = 0x00         //stop timer/counter 5
windspeed = TCNT5     //transfer count to global
                      //variable
```

BTW, I spent considerable time trying to downsize the "over-qualified" Arduino Mega to an Uno. The largest array in the program required 1,440 bytes and the Uno had 2K bytes. It seemed like there was plenty of extra RAM, but I just couldn't get it to play nice. Plus, with the Uno, I had to invoke a software serial port because I needed two ports with different baud rates. The software

port worked fine by itself, but when I folded it into the full program it caused a number of mysterious interaction problems that I got tired of trying to sort out. So, I stayed with the Mega. There is only so much time. (Any ideas?)

I also added a 16 x 2 LCD display to the system for debugging and to display other parameters. I really love all the USB connectors on the RPi. They allowed me to connect a wireless keyboard/trackball in order to make changes to the program and a Flash drive to save the changes — not to mention the convenient five volt power for the speaker.

Construction

Punching the round and rectangular holes in the thick plastic box and its cover was a snap this time, mainly because I had purchased a Grizzly benchtop milling machine a while back. In the past, making such holes involved a lot of elbow grease, i.e., get out the big rattail file and have at it. With the machine, I milled a big hole in the cover as shown in **Figure 7**, and mounted a fan filter over it to let air circulate around the processors. The power supply is mounted externally to allow plenty of airflow through it.

Sound

After the whole system was successfully running, I had yet another brainstorm: Why not add sound? For example, there could be a "ding" every time an extra-large gust of wind came along. So, I did it. The RPi required yet another app called Pygame to make it happen. I recorded the sound I wanted using Audacity and put the .wav file in the pi directory. Now, it dings four or five times a day when the wind picks up. I tried using the HDMI audio output jack on the little monitor, but the correct command line instructions eluded me. So, I went with the audio output jack on the RPi board and it worked great.

The Future

The future for this project is wide open. Gnuplot can easily handle multiple plots in different colors and line styles. You can measure temperature, humidity, barometric pressure, whatever. Although I think my next plot to add is going to be another wind speed measurement, but this one will be generated by a no-moving-parts ultrasonic anemometer. Ultrasonic wind sensors typically have a time constant of 0.25 seconds or less, which will catch the fast rise time gusts that the Davis unit completely misses. I can't wait to see how the two plots compare.

The RPi and Adafruit LCD have opened up a whole new world of dedicated displays for me — no more tying up my laptop or desktop monitor. There is bad news and good news, though. My house may still blow away, but at least I'll know how fast it went down the street. NV

■ **FIGURE 7. Fabricating the box and its cover was a snap with my benchtop milling machine. No more filing for hours!**

The Remarkable CSS555

The CSS555 is a micropower programmable version of the 555 family of timer ICs. It operates at a current under 5 µA and a supply voltage from 5.5V down to 1.2V. These qualities make it particularly well-suited for long lasting battery and small solar powered projects. It can be used in standard 555 configurations as supplied, but it is also user programmable to produce extended timing periods.

By James Senft (a.k.a., TinkerJim)

Standard 555 Micropower Operation

The CSS555 is pin-for-pin compatible with the customary 555 series timers as **Figure 1** indicates. Therefore, application circuits have the familiar 555 topology as, for example, in the astable circuit illustrated in **Figure 2**.

However, the micropower nature of this IC allows using external components with values that demand much less power. For example, the LED blinker in the electronic paperweight in **Figure 3** uses the above astable circuit with timing resistors R_A = 10M, R_B = 47K, and a 0.22 µF timing capacitor. It blinks perpetually on the energy stored in a 1F capacitor charged from a small photovoltaic cell taking in typical desktop lighting.

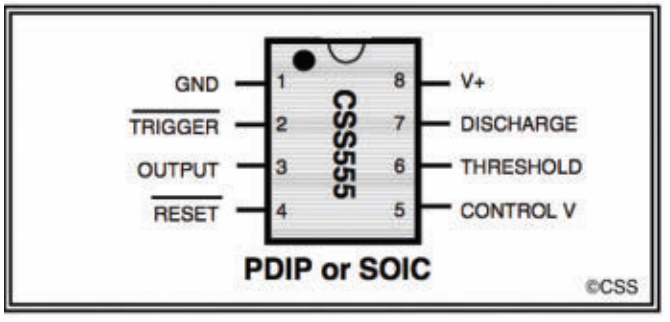


FIGURE 1. The CSS555 has the same pinout.

Another application of the CSS555 in the standard (unprogrammed) mode is the solar engine shown in **Figure 4**. Here, the circuit is a sort of monostable arrangement with a more or less constant voltage (about 1.4V usually) on the trigger and threshold pins, which is supplied by a photodiode (a suitable LED, actually). The supply voltage to the circuit varies as the storage capacitor Cs charge from a solar cell, or discharge through the load when the output pin goes high.

It goes high when the supply reaches three times the trigger voltage, and goes low when the supply drops to 1.5 times the photodiode voltage.

The original concept for this solar engine circuit was devised by Manfred Schaffran and Wilf Rigter in 2003 when only the 7555 timer was available. The CSS555 greatly improves the efficiency since the circuit now takes under 5 µA during periods when the storage capacitor is charging up. The various available 7555s required anywhere from 50 µA to 180 µA. A CSS555 equipped solar engine has operated perfectly from a tiny solar cell supplying a mere 15 µA.

The little robot "Walker" in **Figure 5** is equipped with this circuit, a good size solar cell atop his head, and, presently, a 0.22F capacitor. He periodically waddles around in a little circle (one of his arms/legs is longer than the other) whenever he has gleaned enough energy from

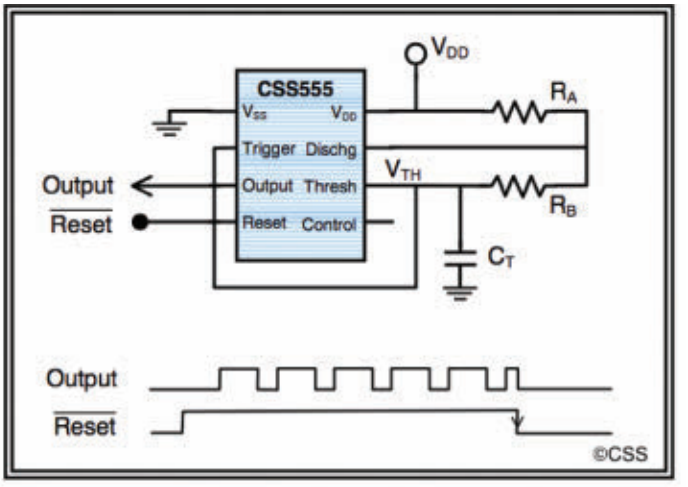


FIGURE 2. Astable circuit diagram. Note that the reset pin must be held high for the chip to be active.

the surrounding light. More information on all these projects can be found at **Instructables.com**. The ultra low power feature of the CSS555 truly excels in projects such as these.

Extended Mode Operation

The second remarkable feature of the CSS555 is that it is user programmable for extremely long timing periods — from hours to days — with small value capacitors. The chip includes an internal counter that tracks the capacitor charge/discharge cycles and changes the output pin state when the selected number of cycles has occurred. **Figure 6** illustrates this with a counter value of 10.

Selectable counter values go in multiples of 10: from $10^0 = 1$ (the standard 555 mode) up to $10^6 = 1M$. **Figure 7** illustrates a monostable circuit set up for a delay of about 16 minutes using a capacitor of only 100 pF (which is built into the CSS555C variety of the chip).

For the extended functions, access to a programming board is needed, of which two are available from Custom Silicon Solutions (CSS; **www.customsiliconsolutions.com**). The "EZ Programmer" shown in **Figure 8** works from two AAA cells, reads the current settings on the chip, shows them via LEDs, and will reprogram the chip according to the positions selected for the switches. It is extremely easy to work with this programmer.

The other programmer (a deluxe version) is supplied in the CSS555C Demonstration Kit. It works through a computer via USB and the configuration settings can be selected from drop boxes. The screenshot of this programming package in use shown in **Figure 9** shows many of the options that are selectable for this timer.

A Practical Application

The following project illustrates a very practical application of the CSS555 operating in the extended mode. The 5" battery powered fan in **Figure 10** proved itself well worth the few dollars it cost. Whenever conditions got a bit warm, I used it on the desk or next to the easy chair when just a little quiet breeze was all that was needed to maintain comfort. It was indeed quiet — especially on the low setting — which seemed to be the most used.

This fan was so quiet, in fact, that more often than not, I would altogether forget the thing was on and fail to turn it off. When I'd return to the scene and reach to turn it on, I would find to my dismay that it was already on, but with two dead D cells! Reminder notes placed in various locations proved futile.

After a couple of summers of this, it finally dawned on me that the CSS555 in extended monostable mode offered a nice solution to this problem — and a significant savings in D cell batteries! It could be configured to run for around 15 minutes after just a touch of a button.

FIGURE 3. This electronic paperweight holds down papers by automatically utilizing the ambient gravity field. It also blinks.

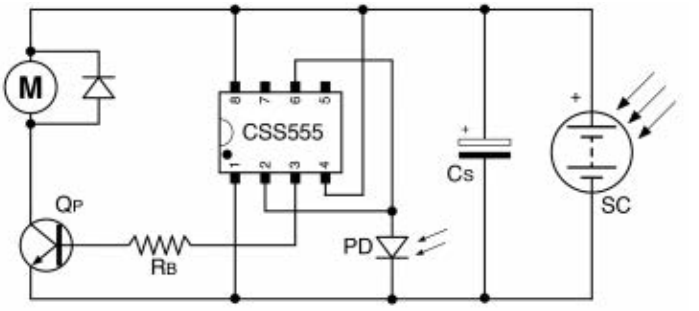


FIGURE 4. The CSS555 as the heart of a very efficient solar engine circuit.

During idle periods, it would consume a negligible level of power, taking less than a half dozen microamps between button pushes. Of course, a microcontroller first came to mind but experience with this fan showed that it could provide adequate comfort when its series-connected pair

FIGURE 5. "Walker the Robot." He is energized by the solar cell on his head.

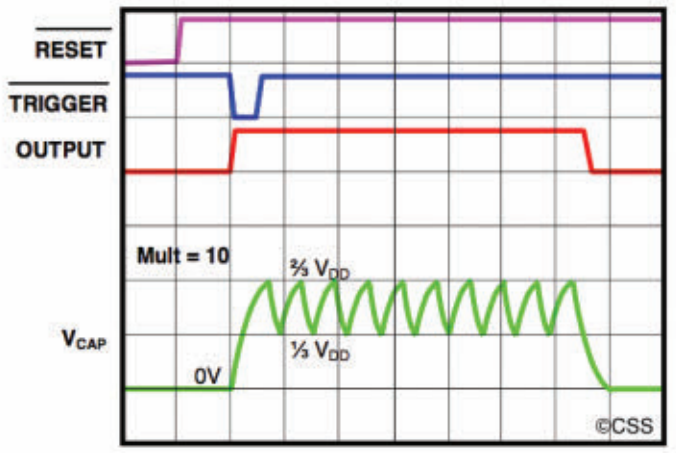


FIGURE 6. Waveforms of the CSS555 running as a monostable in extended mode with its counter set to 10.

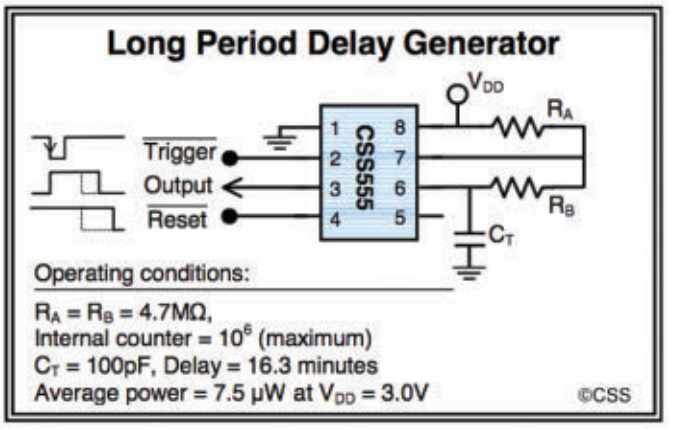


FIGURE 7. One-shot circuit using extended mode, giving an on time of 16.3 minutes.

FIGURE 8. "EZ Programmer" from Custom Silicon Solutions.

of batteries wore down considerably below 1.5V. This was well below the brownout voltage for the micros on hand. With the EZ Programmer, the CSS555 could be set for "low voltage" operation, which allows it to function from 1.2V.

Also at this setting, the timer trip levels of the IC are set farther apart: 10% and 90% of the supply voltage, instead of the usual 1/3 and 2/3. This lengthens the delay time (i.e., output pin high) by about 3X to:

$$(\text{Counter}) \cdot (2.197) \cdot (R_A + 2 \cdot R_B) \cdot C_T$$

Figure 11 shows the schematic used for the fan circuit. With a counter setting of 10^4, $R_A = 3.3M = R_B$, and $C_T = 4.7$ nF, this circuit gives a measured on time of some 17 minutes, which works out just fine. When the fan turns off (which in itself is very satisfying to one who has gone through so many D cells due to forgetfulness!), if it starts feeling a bit warmish, just another touch of the button restores comfort. When leaving the scene when the fan is still going, it could be turned off via the built-in Off-Low-High slide switch. I usually just let it go, knowing that it will turn itself off in a few minutes and then be standing by ready to go with a tap on the button tomorrow or next week — or whenever it is needed — with its batteries in essentially the same condition.

A simple MOSFET to control the motor was ruled out since (as mentioned above) it was desired to have operation when the supply gets below 1.5V. This would not have met the gate threshold voltage for the MOSFET on hand. However, I had several LCA717 OptMos chips in my parts box. The LCA717 is an excellent device for motor switching applications. It requires only 1.2V forward to turn on the load, and has very low on resistance. The LED

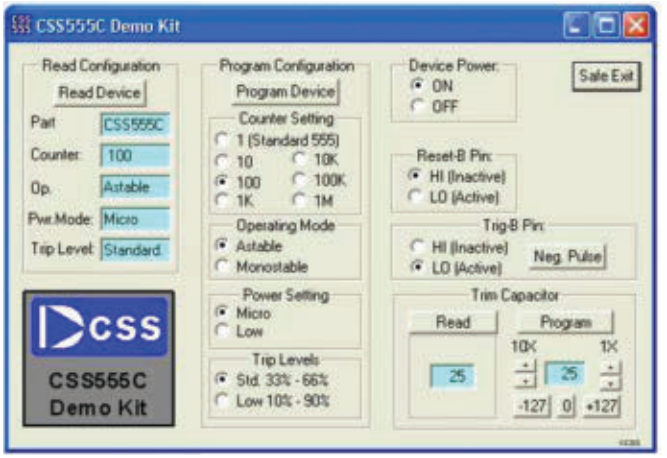


FIGURE 9. Screenshot of the computer-based CSS555 programmer.

within needs 2 mA which is negligible relative to the current draw of the motor, which in the fan ranges from about 180 mA on low to 350 mA on high (with new batteries).

The LCA717 has the advantage (for some applications) of using a separate power source for the load, which is the way the circuit schematic here is shown. In this fan application, both the timer and the load share the same two series-connected D cell batteries. **Figure 12** shows the completed circuit board, which was easy to fit in the cavity of the fan base.

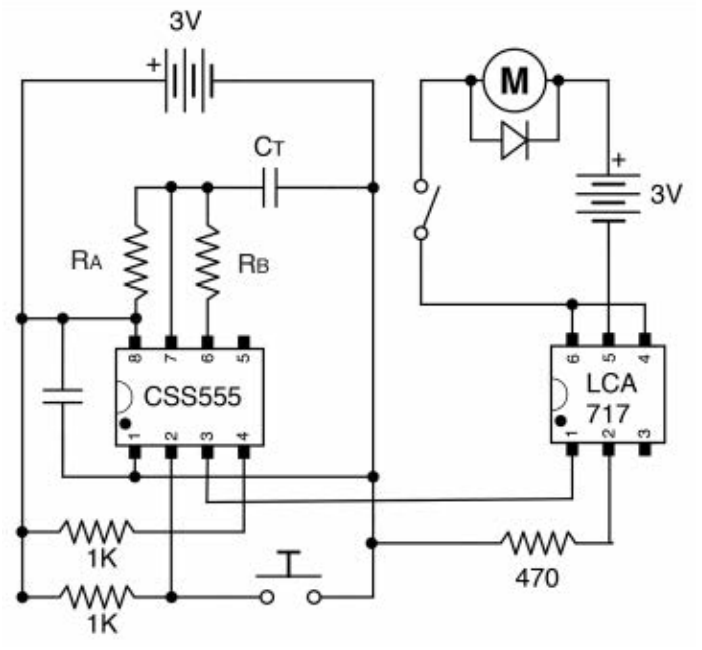


FIGURE 11. Schematic of the one-shot circuit used for controlling the fan motor. Note the bypass capacitor across the supply and ground; 0.1 µF suffices in such applications.

FIGURE 10. The "O2Cool" personal fan powered by two D cell batteries.

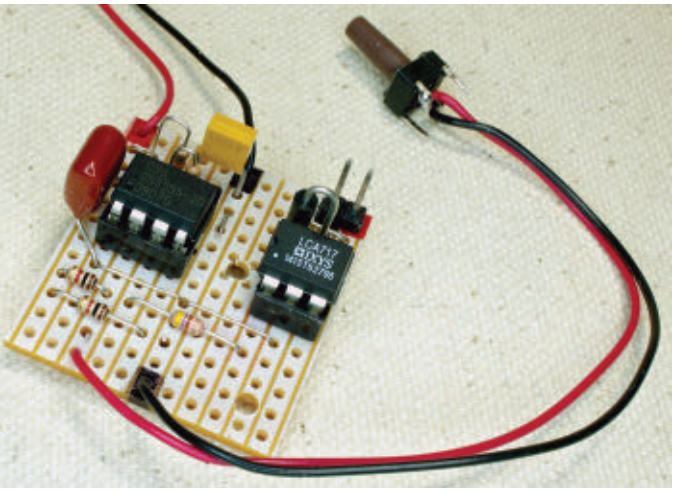


FIGURE 12. Circuit board of the now battery-saving fan.

Conclusion

I hope this short article will serve as an indication of the capabilities of this relatively new timer. The documents that CSS has prepared for these chips give much more information. They are very thorough, easy to follow, and have plenty of illustrations — some of which have been presented here (with permission of course!). The information sheets available for downloading from the CSS website include SPICE models, timer delay and adjustable duty cycle calculators, and a collection of application circuits showing the many ways this remarkable timer can be put to work. **NV**

Why You Need an Analog Front End and How to Set It Up

How to make your sensors talk nice to your microcontroller

By Eric Bogatin

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/magazine/article/february2016_AnalogFrontEnds.**

The analog input to an Arduino Uno has a resolution of only 10 bits. On a 5V scale, this is only about 1 mV of sensitivity. If you need more sensitivity, don't look at another microcontroller. Look at adding an analog front end to your Arduino.

Singly or together, a general-purpose op-amp and an instrumentation amplifier — each operating on 0V to 5V power supplies — are the building blocks to an analog front end to any microcontroller. Here's a practical way you can turn your Arduino into a high performance sensor measuring instrument.

Physical Computing

An Arduino is about as smart as a fruit fly. I base this on the gate count of the Atmel AVR 328 microcontroller — the brains of the Arduino board — which is a little less than 100,000 gates, including the 32K of memory and all the registers. A fruit fly has 100,000 neurons.

Just as the brain of a fruit fly processes sensory input information about the world around it and translates this into a fruit fly's action, an Arduino (and all microcontrollers) processes inputs from the outside world and turns them into outputs. This is really what distinguishes "physical computing" from generalized computing as with microprocessors.

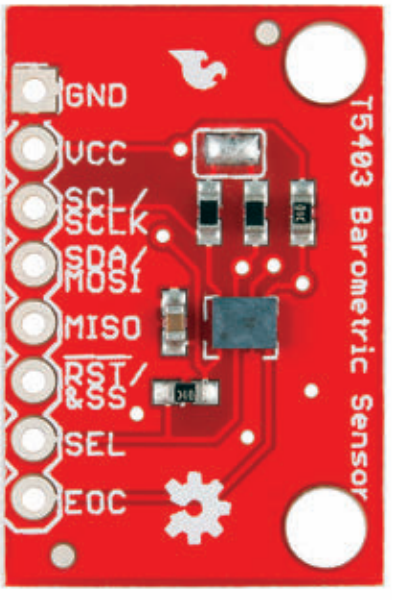


FIGURE 1. T5403 barometric pressure sensor with an I²C interface.

It's the sensors that translate some quality in the physical world and turn it into an electrical quantity — a voltage, a current, a resistance — which may vary with time or frequency. It's up to us as gadget designers to set up the microcontroller to read this electrical quantity and turn it into information we can act on.

We have three options for inputs from sensors to an Arduino: a high level encoded digital signal; a low level digital voltage; or an analog voltage.

Some sensors — like the T5403 sensor from SparkFun shown in **Figure 1** — have a lot of electronics already integrated into them. This sensor measures barometric pressure, turns it into an electrical signal, and then encodes this in an I²C digital interface.

All the electronics on the sensor board — the sensor itself, the analog front end, the ASIC, and the memory —

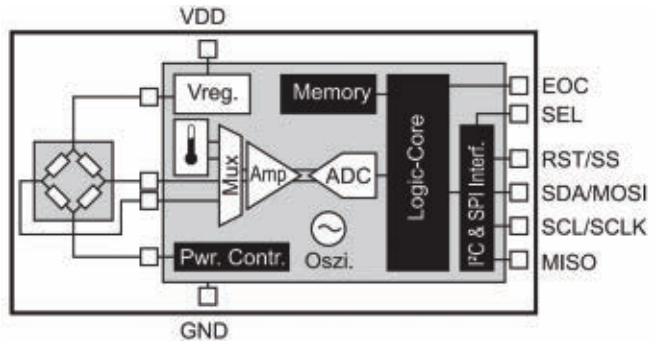


FIGURE 2. The electronics that turn the resistance measurement into an I²C signal that is then read by the microcontroller (source: EPCOS datasheet).

dramatically simplify what we must do to integrate this sensor with the microcontroller. In this case, just plug it into a few digital pins and install the library. All the low level functions of reading the resistance, converting it to a voltage, filtering it, conditioning it, even compensating the electronics with the measured onboard temperature, and ending up with a calibrated barometric pressure is all done "under the hood" for us by the electronics shown in **Figure 2**. That's a lot of processing power in a tiny space — all for less than $15.

Some sensors output a low level digital signal, like a rain gauge. This sensor just sends a click every time a bucket is filled and emptied. It's up to us to count the clicks with a digital I/O pin and keep track of the total rain fall.

Then, there are all the sensors which just output an analog signal. This is where the analog front end conditioning we add to the system can dramatically improve the quality of the measured information.

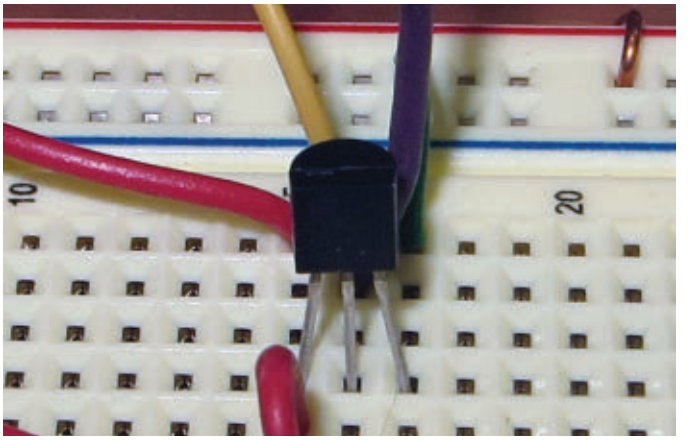
Limitations to Sensor Measurements in an Arduino

An example of a sensor which outputs a voltage is the TMP36 temperature sensor also from SparkFun. It comes in a small three-terminal plastic package shown in **Figure 3**. One pin is connected to +5V; another pin is connected to gnd; and the middle pin is the voltage output that is directly proportional to temperature, with a sensitivity of 10 mV/°C and 0V at -50°C, or 100°C/V.

The output voltage from this temperature sensor can be read directly by one of the analog input pins on the Arduino. Here's where we encounter the fundamental limitations with the Arduino analog input pins.

Each analog input pin of the Arduino is a 10-bit ADC (analog-to-digital converter). This means there are only 2^10 = 1024 discrete voltage levels the ADC can report. When we "read" an analog pin, the integer that comes back is a discrete level — a number between 0 and 1023. We sometimes refer to the units of these levels as analog-to-digital units (ADUs). When analog accuracy is important, or in general as a good habit, I use the 3.3V onboard reference as the reference to the ADC channels (see the **sidebar**). I also add a large capacitor to the reference channel just to reduce any AC noise on this rail.

FIGURE 3. Close-up of the TMP36 temperature sensor.



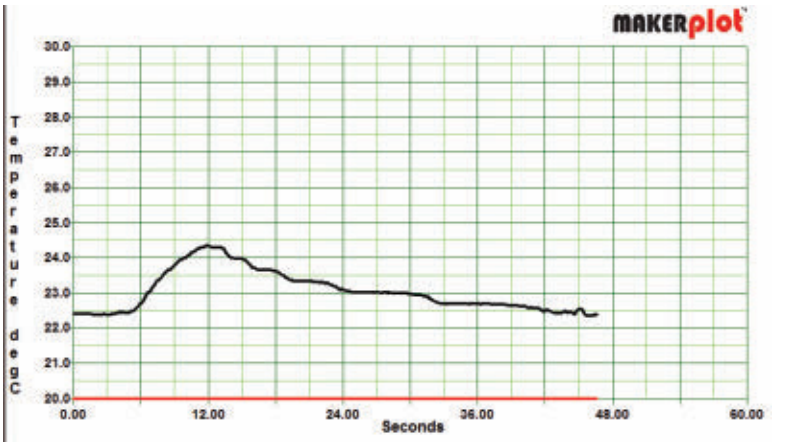


FIGURE 4. Measured temperature of the TMP36 when I touched it and let go.

With a voltage scale of 0V to 3.3V, this means the voltage resolution is 3.3/1023 = 3.2 mV resolution. In general, we should measure the reference voltage level for the ADC, Vref, with a three-digit DMM. Then, the voltage on the pin in volts is V[volts] = ADUs/1023*Vref.

The TMP36 sensitivity is 100°C/V. With a discrete level resolution of about 3.2 mV, the Arduino has about a 0.3°C per ADU resolution. This may be fine for reporting ambient temperature, but may not be sensitive enough when we care about very small temperature changes.

I wrote a sketch to read one of the analog channels of an Arduino Redboard from SparkFun. I averaged about 100 readings in about 100 msec, and scaled the voltage into a temperature value. The analog channel reading in

ADUs was converted into a temperature in my sketch using temp [degC] = (ADU/1023 x Vref) * 100 – 50. The 50 is there because the output voltage is 0V at -50°C.

I printed the temperature values every 100 msec to the serial port and used MakerPlot to read the serial port and plot the temperature. I did no calibration at all of the

Improve the Quality of Your Analog Measurement Using the 3.3V Pin as the Vref

The ADC (analog-to-digital converter) pins of the Arduino compare the voltage at their input to a reference voltage to generate the 1024 different output levels. The default condition is to use the +5V rail on the die of the microcontroller as the reference voltage.

You might think that the + 5V rail is very low noise; after all, it comes from either the regulated USB power supply or the onboard regulated +5V supply if you are using an external power supply. If the microcontroller is not driving any outputs and its current draw on the power rail is low, then the noise on the power rail can, in fact, be low.

I wrote a sketch for my Arduino to toggle five digital outputs off and on really fast. The rail noise I measured on the board was about 5 mV peak-to-peak when the outputs were switching, but not driving anything. This is about the resolution of the ADC and would contribute negligible ADC measurement noise.

However, when the digital I/O pins draw some current, this has to come through the impedance of the power rail supply and its output voltage — which everything on-die uses — will drop.

I connected the five toggling outputs into 330 ohm resistors which then drove LEDs. We can estimate the current draw of each I/O. There is about 25 ohms of output impedance on the output transistor of each Arduino I/O pin. There is about a 2V drop across an LED. If the output voltage from one I/O pin when not driving a load is 5V, then the voltage drop across the (25 + 330) ohm resistors is 5V – 2V = 3V, and the current draw from the pin and the power rail per pin is:

$$I = \frac{(5V - 2V)}{355\Omega} = 8.5\ mA$$

I had five I/O pins toggling. This was a total current draw of 8.5 mA x 5 = 42 mA of current from the +5V rail. I measured the voltage drop on the +5V rail on the board as about 35 mV peak-to-peak. This corresponds to an output impedance of the power supply of:

$$R_{output} = \frac{\Delta V}{\Delta I} = \frac{(35\ mV)}{42\ mA} = 0.83\Omega$$

This is very close to what I measured independently for a USB power supply. I measured an onboard regulator output impedance of the Redboard (when powered by an external 9V supply) to be 1.4 ohms, in comparison.

This 35 mV of noise on the board level power rail is almost identical to what I measured as the rail noise on the die itself. Of course, we can't probe the on-die power rail directly, but we can see the on-die power rail voltage by setting a digital I/O pin as HIGH, and measuring its output voltage. After all, when its output is HIGH, the output pin is effectively connected to the +5V on-die voltage rail. This is the same voltage that would be used by the ADC reference channel in its default condition. **Figure A** shows all the measured voltages.

The consequence of this is that when there is little current drawn by the microcontroller, there is probably no impact on the measurement accuracy of the ADC channels. However, if the microcontroller is also switching current around, the reference voltage will drop and this will contribute to measurement errors in the ADC values.

More importantly, this voltage noise on the Vref line will introduce errors; the magnitude of which will depend on what else the microcontroller is doing. This sort of problem is incredibly hard to debug. Better to avoid the chance of it ever happening.

The way around this problem is to use either another external reference voltage source or use the 3.3V reference supply brought out on many Arduino boards. On the SparkFun Redboard, this is a separately regulated supply which has nothing connected to it. Its voltage noise is very low. I measured it to be less than 2 mV peak-to-peak — even with 42 mA of switching current from the I/O pins.

Since it is available for free, it's always a good habit to use this as the ADC Vref. The downside is that this will limit the maximum voltage range the ADC can sense. With an analog front end, this is not an important limitation.

To use an external reference voltage, in the *void setup()* function, you must add the line:

```
analogReference(EXTERNAL);
```

I also measure the analog reference voltage with a three-digit DMM and add a line near the beginning of each sketch where I can add the specific reference voltage level:

```
float Vref = 3.290;
```

Then, when I read an analog pin, I can calculate the actual voltage on that pin using:

```
Value_volts = analogRead(A0) /1023 * Vref
```

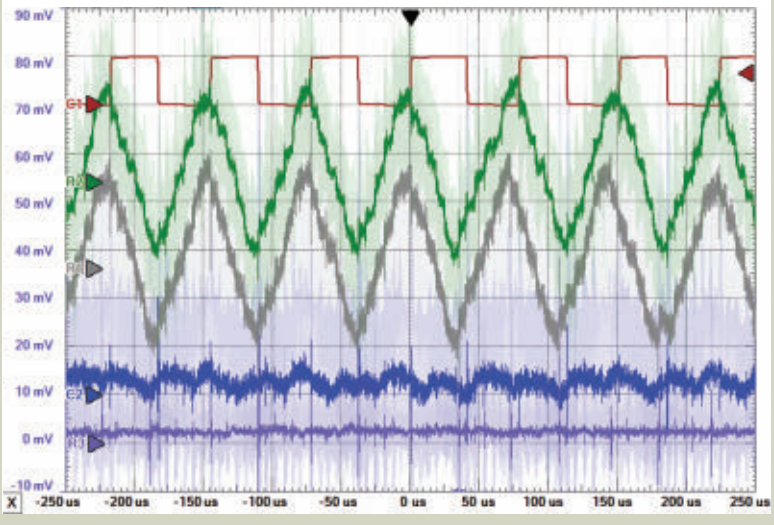


FIGURE A. Voltage noise on the various power rails while five digital I/Os toggled off and on. Top trace: Voltage on one of the digital pins showing its voltage switching. Next two traces down: Measured rail voltage noise on the board and on the output of a digital pin set to HIGH. This shows the 35 mV noise on the power rail when 42 mA of current switches. Fourth trace down: The +5 V rail voltage noise on the board with no current drawn by the I/O. The noise level is down to 5 mV. Bottom trace: Voltage noise on the +3.3 V reference with 42 mA of switching current. The scale for all except the digital I/O signal is 10 mV/div. The triangle shape is because the five I/O are sequentially turned on one at a time and then turned off one at a time.

sensor. **Figure 4** shows the measured temperature when I touched the sensor and let go.

In this example, you can see the ambient temperature of about 22°C before I touched the sensor. It rose about 2°C quickly in about three seconds. I let go and it cooled, taking a longer time.

On the cooling leg, you can clearly see the 0.3°C temperature steps corresponding to the resolution limit of the Arduino's ADC. This is not a sensor limitation; it is an Arduino limitation.

When the signal we care about is a small signal, we just don't have much sensitivity available from the 10-bit Arduino analog inputs.

Problem #1 is the small resolution available with the 10-bit ADC. We just can't see small changes with only 3 mV resolution even using the smaller Vref of 3.3V.

Problem #2 is there is a DC offset on the signal (about 0.75V in this example). We can improve this by adding some gain, but we have to be careful not to exceed the 3.3V maximum voltage input to the ADC. This would be a maximum gain of about four.

Problem #3 — just marginally an issue with this sensor — is the output impedance of the sensor and what the ADC pin needs to see. Section 23.6.1 of the Atmel manual states: "The ADC is optimized for analog signals with an output impedance of approximately 10 kΩ or less." Where this spec comes from is explained in the **sidebar**.

The output impedance of a sensor is a measure of how much current it can source or sink. According to its specs, the TMP36 has a maximum output current of about 50 uA of current draw possible. With roughly a 1V output voltage, this is an output impedance of 1 V/50 uA = 20K ohms. This is in the gray area of possibly a problem — especially when looking at fast data acquisition.

One way of getting around these three problems and enabling accurate and high resolution measurements from a wide variety of sensors is using an analog front end between the sensor output and the microcontroller input to condition the analog signal.

An Analog Front End Essential Element: the Op-Amp

We refer to all the electronics from the sensor element to the input pin of the ADC as the analog front end. While there is a huge variety of off-the-shelf building block circuits we can use in the analog front end, the two most important building blocks to solve all sensor interface problems are operational amplifiers and instrumentation amplifiers. An operational amplifier — affectionately shortened to op-amp — is a super high gain differential amplifier. Its output voltage is proportional to the difference in voltage between its two input pins, referred to as the V+ and V- inputs:

$$V_{output} = G \times (V_+ - V_-)$$

The value of G is often as high as 1,000,000. The secret to using op-amps effectively is using combinations of R and C elements in a feedback circuit to enable useful features. Whole books are written about op-amp circuits with specialized functions. A really great handbook by the legendary Walter Jung called *Op-Amp Applications Handbook* can be downloaded for free. The circuit we'll look at in this article is the non-inverting amplifier.

In the non-inverting amplifier, the input signal goes into the V+ pin, and a simple resistor divider circuit connects the output pin to the V- input as shown in **Figure 5**. The gain in this circuit is:

$$G = \left(1 + \frac{R2}{R1}\right)$$

Since this amplifier is often the first circuit to interface with a sensor, it is sometimes referred to as a pre-amp. An important feature of the amplifier — in addition to amplifying the signal — is to change the output impedance of the sensor. It can take a sensor with really high output impedance and provide a comparable signal level (or even higher) with an output impedance of a few ohms. This solves problem #3.

Among the three top suppliers of op-amps — Analog Devices, Linear Technology, and Texas Instruments (TI) — there are almost 500 different versions to choose from. I

The Input Impedance of an Arduino ADC Pin

The Arduino input impedance of an ADC (analog-to-digital converter) pin is specified as 100 megohms. It sounds high and would be wonderful, but this is not the complete story. The first circuit element the pin sees on-die is a multiplexer which switches each of the six Arduino ADC pins into the actual sample and hold circuitry to be read by the ADC circuitry. This input has the equivalent of 14 pF of capacitance.

When the Arduino pin is read, the pin is switched into this capacitance. The sensor driving this pin must charge up this 14 pF of capacitance before it is read to get an accurate value. If it is still charging when the pin is read, the value will be inaccurate.

If the source impedance of the sensor is 10K ohms, then the RC time constant is 10^4 ohms x 14 x 10^-12 F = 0.14 usec. If we wait for six time constants, the signal will be within 0.2% of its final value. This is 1 µsec — about the fastest possible acquisition time for an analog channel.

If the output impedance of the sensor looks like a resistor of no more than 10K ohms, then the voltage read by the ADC will have settled to its final value before the voltage is actually read. If the output impedance is larger than 10K ohms, there is a chance the voltage will not have stabilized before it is read and the first measurement may not be accurate.

This sort of problem is incredibly hard to debug. As risk reduction in your design, just avoid the problem by always using a low enough output impedance for the sensor.

This is the origin of the Atmel spec recommending the sensor output impedance be less than 10K ohms to drive one of the ADC pins and not lose accuracy even in the worst case.

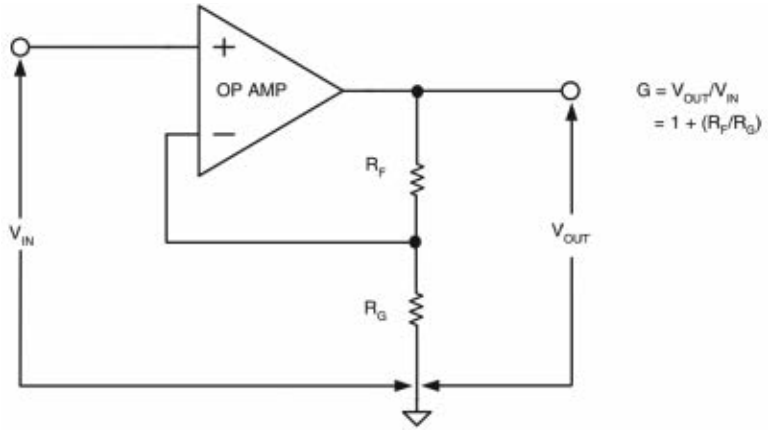


FIGURE 5. Circuit diagram of the non-inverting amplifiers using an op-amp, taken from Walter Jung's *Op-Amp Applications Handbook*.

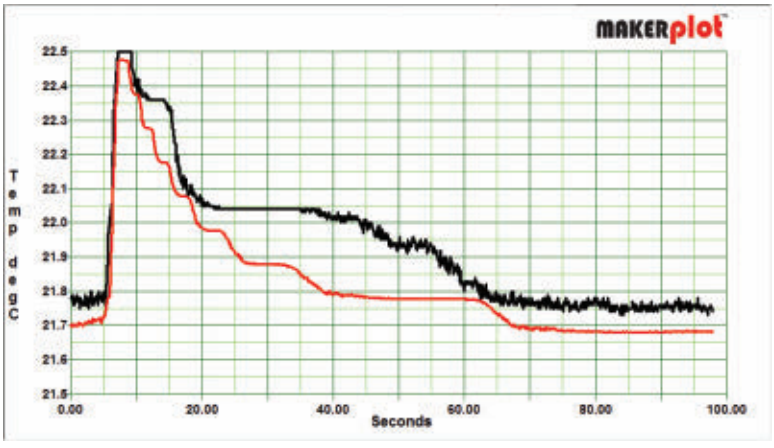


FIGURE 6. Measured temperatures of the same TMP36 using the scaled value directly from the sensor and after a gain of 3.245 in a pre-amp. Note the resolution improvement from 0.3° to 0.1°.

found one that I use all the time because it has pretty good specs, runs off the +5V single supply from an Arduino, and comes in an eight-pin DIP. Best of all, it only costs $0.29. As a bonus, it comes with two independent op-amps in one package for this low price.

I use the LM358 op-amp — manufactured by TI and available from Jameco and other distributors. It's not the highest performance op-amp, but at this price, is a great general-purpose version and is incredibly easy to use.

In the application of the TMP36 temperature sensor, the nominal output signal level is about 0.75V at room temperature. I wanted to increase this signal level so it is higher, but still below the 3.3V of the Vref. I chose to use a gain of about three, resulting in a nominal voltage level out of the op-amp of about 2.25V. I used resistors of 22K ohms and 9.8K ohms, resulting in a gain factor of (1 + 22K/9.8K) = 3.245.

The original sensitivity of the TMP36 sensor was 100°C/V. With a resolution of 0.0032V, this is a temperature resolution of 0.32°C.

With a gain of 3.245, the output sensitivity of the pre-amp is 100/3.245 = 30.8°C/V. The voltage resolution of the 10-bit ADC is still 0.00323V, but this is now equivalent to 30.8 x 0.00323 = 0.1°C. Much better.

This helps solve problem #1. **Figure 6** shows the recording of the raw output from the TMP36 sensor into A0 of the Arduino's ADC, and the output of the LM358 with the gain of 3.245 into A1. I converted the ADU values from A1 into temperature using:

Temp[degC] = (ADU/1023*Vref/Gain_opAmp) x 100 – 50

An Analog Front End Essential Element: the Instrumentation Amplifier

We still have problem #2. The voltage out of the pre-amp is about 2.5V. We want to see small changes on top of this large DC value. We really would like to subtract off a DC value and amplify what is left. This is the perfect job for an instrumentation amplifier.

An instrumentation amplifier is very similar to an op-amp. Its output voltage is related to the difference voltage between its two inputs by:

$$V_{output} = G \times (V_+ - V_-)$$

However, the gain in an instrumentation amplifier is typically adjustable from only 1 to 1,000. It does not use feedback from the output voltage to the input; instead it's just a straight up differential amplifier.

Instrumentation amplifiers are most commonly used to amplify small differential signals such as in electrocardiogram monitoring or in resistance based sensors. They are at the heart of a Wheatstone bridge.

For our temperature sensor application, we can use the instrumentation amplifier to subtract off the DC voltage we provide at the V- pin and amplify what is left.

There are hundreds of different instrumentation amplifier options to choose from — each with slightly different specs and price points. The instrumentation amplifier I use the most with an Arduino is the AD623 manufactured by Analog Devices and available from Jameco.

It runs off the single +5V from the Arduino, comes in an eight-pin DIP package, is really easy to set up, and costs less than $5. The gain is selectable with a single resistor, Rg, from 1 to 1,000 using:

Resources

T5403 barometric pressure sensor from SparkFun:
www.sparkfun.com/products/12039

SparkFun Redboard:
www.sparkfun.com/products/12757

TMP36 temperature sensor:
www.sparkfun.com/products/10988

Makerplot for reading the serial port data and plotting:
www.makerplot.com

Atmel Manual:
www.atmel.com/images/doc8161.pdf

A great op-amp applications handbook
you can download for free:
www.stu.edu.vn/uploads/documents/080509-212459.pdf

L358 from Jameco:
www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_23966_-1

AD623 from Jameco:
www.jameco.com/webapp/wcs/stores/servlet/ProductDisplay?langId=-1&storeId=10001&productId=1780985&catalogId=10001&CID=CAT151PDF

$$Rg = 100\ k\Omega/(G - 1)$$

In this application, I wanted to use a gain of 10. The DC voltage was provided by a simple 10K ohm 10-turn pot connected between the 3.3V reference and ground. The center tap signal — which I could adjust — was connected to the V- input of the AD623. The final circuit with the TMP36, pre-amp, and instrumentation amplifier is shown in **Figure 7**.

This analog front end circuit was implemented in a breadboard adjacent to the Arduino Redboard. The amplifiers were powered off of the 5V rail of the Redboard. The two sensitive voltages used the 3.3V reference pin on the Redboard.

I added a few capacitors to the voltage rails to keep the noise down. **Figure 8** shows this configuration.

In this circuit, the original 100°C/V sensitivity was changed to 100°C/V /3.245/10 = 3.1°C/V. With a resolution of the least significant bit still 0.00323V, the temperature resolution is 3.1 x 0.00323 = 0.01°C. Now we're talking.

We have three different sensitivity levels of temperature measured in this circuit: the direct readings; the direct readings increased in sensitivity by 3x; and the direct reading increased in sensitivity by 30x. All the conversion from ADUs into voltage and then into temperature is done in the sketch.

It's the final temperature values which were printed to the serial port and available for plotting by my favorite plotting tool, MakerPlot.

With these three different temperature readings available, I did the same experiment: touching the sensor very briefly and watching the temperature rise quickly and fall slowly. In the slow decline in temperature, the resolution limits of 0.3°C and 0.1°C are clearly seen.

The smoothly varying temperature on this plotted scale of 0.1°C per division is an indication of the 0.01°C resolution of the signal after the instrumentation amplifier. **Figure 9** shows these comparisons.

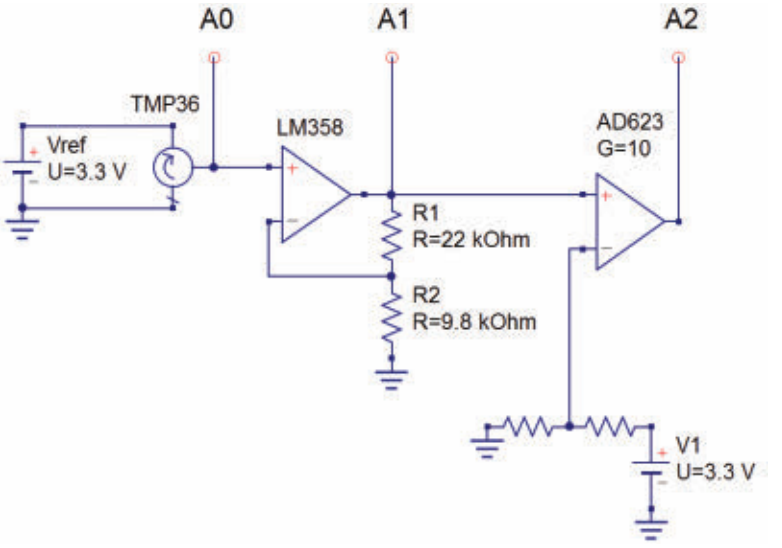


FIGURE 7. Complete circuit with sensor, pre-amp, and instrumentation amplifier.

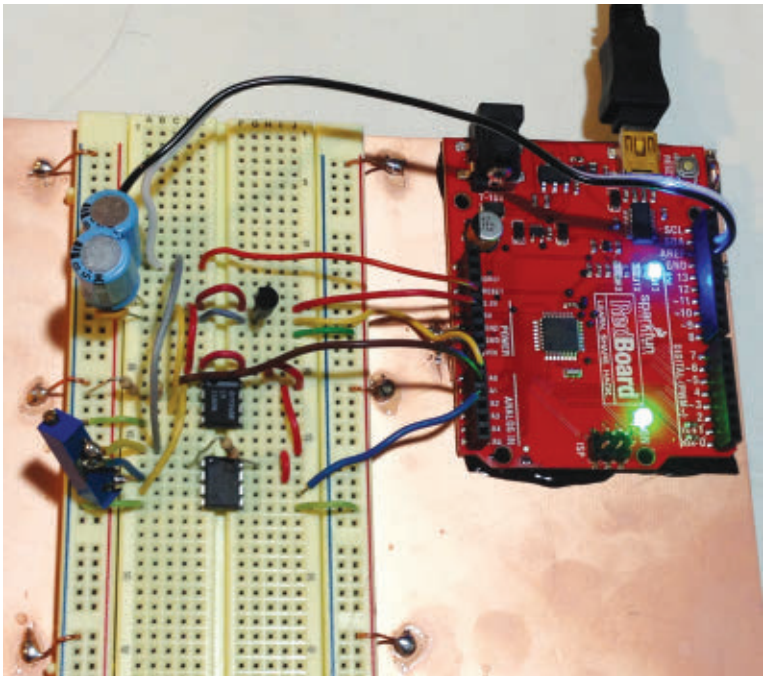


FIGURE 8. The complete system configured on a breadboard connected to an Arduino.

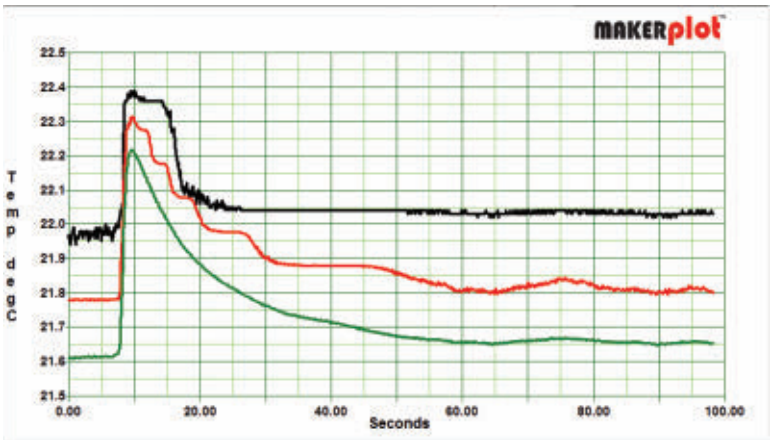


FIGURE 9. Shows the same experiment as before. I touched the sensor briefly and recorded the unconditioned voltage from the sensor, the voltage from the pre-amp, and the voltage from the instrumentation amplifier. You can see the impact of the higher sensitivity on the smoothness of the plots. Top trace: Resolution of 0.3°C from the TMP36. Middle trace: Resolution of 0.1°C from the pre-amp. Bottom trace: Resolution of 0.01°C from the instrumentation amplifier.

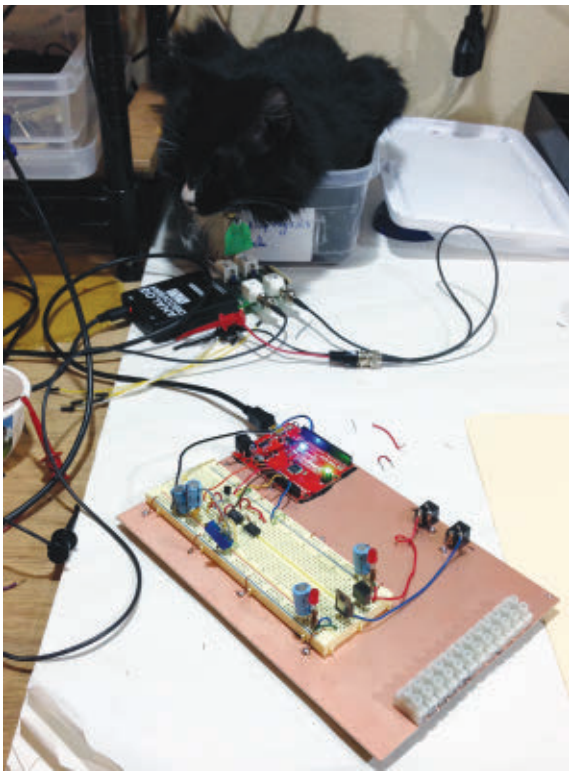


FIGURE 10. Maxwell — the cat scanner — monitoring every aspect of this experiment.

Conclusion

When we have the good fortune of using a sensor with a high level voltage and a low output impedance, we can just connect it to one of the ADC pins of an Arduino and get 10 bits of resolution with little effort.

However, if we want to push the limits of sensitivity or take advantage of sensors which have high output impedance, small scale signals, or small signals riding on a large DC offset, we can leverage an analog front end to condition the signal and get the most value from the 10-bit ADC in the Arduino.

Two essential devices dramatically simplify the design and implementation of an analog front end: the op-amp and the instrumentation amplifier. The LM358 op-amp is the perfect general-purpose version and the AD623 is the perfect general-purpose instrumentation amplifier which play well with an Arduino.

Of course, there is often more than one right answer to any design challenge and there are multiple ways of interfacing sensors to a microcontroller. The op-amp and instrumentation amplifier are sharp arrows to have in your quiver of design solutions.

To ensure the integrity of the measurements in this article, I incorporated a cat scanner to monitor each operation.

Figure 10 shows the bench set up with the cat scanner in operation. **NV**

ELECTRONET

■ BY FRED EADY

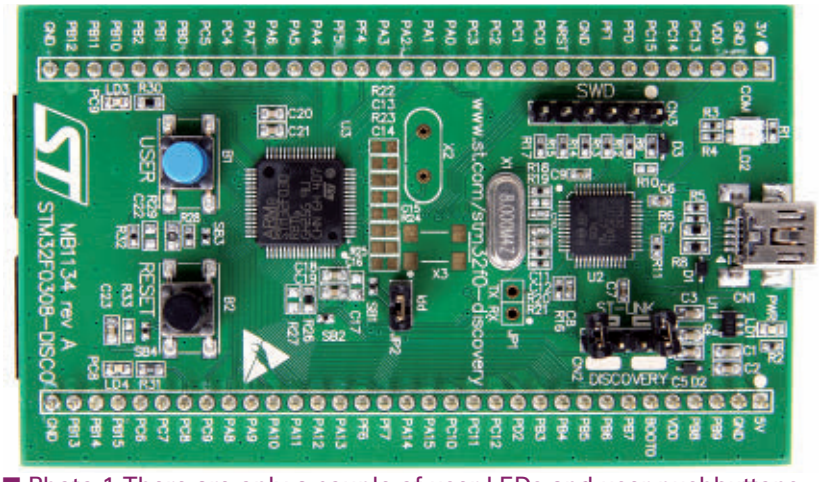
ARMed and Dangerous

I've decided to ARM myself. No, I didn't go out and buy a fully automatic black market M4A1, and no, I didn't plop down big bucks for a brand new chrome-plated Colt .45 semi-automatic pistol. I am ARMed with a Segger J-Link PRO that is fully supported by the Segger Embedded Studio. By the time you finish reading this month's offering, you will also be ARMed and dangerous.

Target Practice

The bull's eye is in the form of an STM32F030R8T6, which is mounted on the STM32F0308-DISCO (DISCO for DISCOVERY) evaluation board you see under the lights in **Photo 1**. The evaluation board is modest in terms of onboard goodies. However, every STM32F030R8T6 I/O pin is pulled out to a male header post. So, you can exercise the STM32F030R8T6 I/O subsystem to your heart's content.

The intent of the STM32F0308-DISCO evaluation board is to allow low cost ARM development using an entry-level Cortex M0 device. To that end, an onboard ST-LINK/V2 programmer/debugger is incorporated into the evaluation board's design. Power is derived from the STM32F0308-DISCO's integral USB portal. The STM32F0308-DISCO can also be powered externally with +5 or +3.3 VDC.



■ Photo 1. There are only a couple of user LEDs and user pushbuttons on this little evaluation board. If you want to light up the world or drive some external relays, the STM32F0308-DISCO comes with an extra blank printed circuit board that can be fitted to the evaluation board's pair of headers.

The STM32F030R8T6

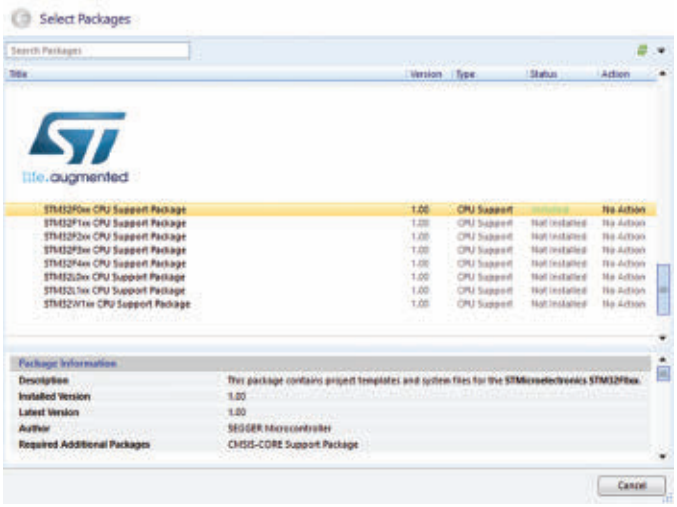
The STM32F030R8T6 is the Number 2 device in its family when measured by its internal resources. If you are new to ARM, there is no reason to fear this "new-to-you" technology. ARM devices are made up of Flash, SRAM, UARTs, SPI portals, I²C subsystems, clocks, timers, and I/O pins. You can say the same for any other general-purpose microcontroller. In addition, most ARM microcontrollers are dirt cheap and deliver a mighty punch for the penny.

The STM32F030R8T6 comes standard with 64K of Flash and 8K of SRAM. There are seven 16-bit timers, two SPI portals, two I²C portals, two USARTs, and 18 channels of ADC (analog-to-digital converters). Depending on how you use its resources, the STM32F030R8T6 can man up to 55 GPIO pins. The maximum operating speed is 48 MHz with a GPIO toggle rate of 12 MHz. If you need a watchdog timer, the STM32F030R8T6 has one of those pinned up too. If you count them all up, there are 11 timers available.

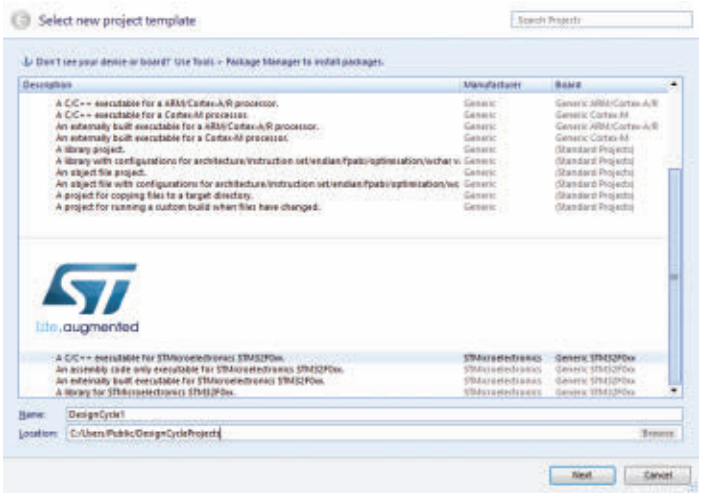
Operational power supply voltages for the STM32F030R8T6 range from 2.4 VDC to 3.6 VDC. The STM32F030R8T6 also includes a real time clock that can be backed by an external battery. A useful feature of the STM32F030R8T6 that is not found on the run-of-the-mill microcontrollers is its SysTick timer. The SysTick timer is a free running 24-bit countdown timer that can be used in various ways. The ability to assign the 55 GPIO pins to external interrupt duty is also a useful STM32F030R8T6 quality. Interrupt handling is supported by the STM32F030R8T6's NVIC (Nested Vectored Interrupt Controller).

The STM32F030R8T6 is fully supported by datasheet documentation, reference manuals, code libraries, and sample code. All of the aforementioned information can be had for a free download.

If you're an old hat at microcontroller design and programming but have no ARM



■ Screenshot 1. Since we will only be working with the STM32F030R8T6, we will only install the STM32F0xx CPU support package.



■ Screenshot 2. We are starting from scratch and will be writing code for an STM32F030R8T6 Cortex-M microcontroller. This entry window informs the Segger Embedded Studio as to the type of project template we wish to use. Note that we have also defined the location of the new project.

experience, I recommend reading its datasheet through. You will be amazed at how much the STM32F030R8T6 has in common with your favorite microcontroller.

Segger Embedded Studio

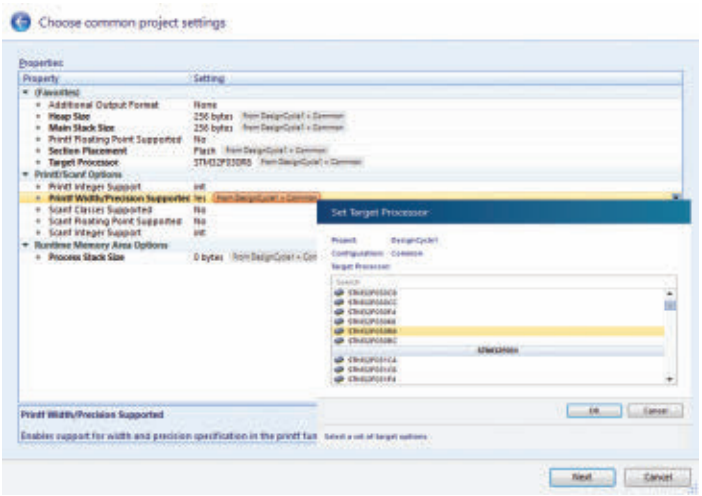
The Segger Embedded Studio (referred to as SES for the remainder of this article) consists of a code editor, a C/C++ compiler, and an integrated debugger that can take advantage of the Segger J-Link PRO hardware debugger/programmer. SES is a free download as long as you don't use it commercially. In free mode, the Studio is unrestricted and has no time limit. It's very easy to use and comes well documented.

Setting Up for the STM32F0308-DISCO

The SES is a powerful ARM development environment. However, out of the box it doesn't have a clue as to what we want it to do. So, the first thing we have to do is provide the SES with STM32F030R8T6 information that it can use to help us achieve a successful compilation of our code.

Since we will only be working with the STM32F030R8T6 this time around, we'll only load the STM32F0xx CPU support package. The support package contains project templates and system files we will need to work with the STM32F030R8T6. The STM32F0xx CPU support package is installed using the SES's Package Manager. As you can see in **Screenshot 1**, the STM32F0xx CPU support package is installed and ready for use.

In **Screenshot 2**, we inform the SES that we would like to write some C or C++ code targeting an STM32F0xx Cortex-M microcontroller. The executable code template has been selected as shown in the screen capture. The



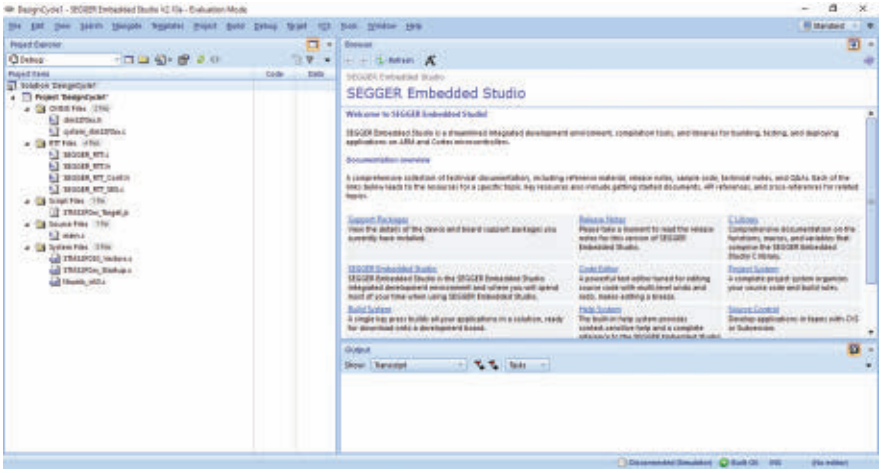
■ Screenshot 3. It would be a good idea to let the Segger Embedded Studio know which Cortex-M microcontroller we want to work with. So, we tell it here.

STM32F0xx device we wish to work with is selected in the data entry screen captured in **Screenshot 3**. At this point, we can also specify stack sizes, *printf* options, and *scanf* options. I have opted to allow the *printf* statements to specify the width and precision of the *printf* arguments.

RESOURCES

JumpStart API
JumpStartMicroBox
ImageCraft
www.imagecraft.com

J-Link PRO
Segger
www.segger.com



■ Screenshot 4. Our new project is ready to be fleshed out. This is also a good time to explore the features offered by the Segger Embedded Studio.

■ Photo 2. In addition to its standard programming/debugging duties, the J-Link PRO's Ethernet interface allows multiple unit debugging over an Ethernet network. Our immediate need is the J-Link PRO's ability to interface to our STM32F030R8T6 using an SWD programming interface. The target socket is loaded with a 1.27 mm SWD adapter.

We have given enough information to the SES to build a skeleton of our project. SES has also assembled the necessary *include* and *startup* files according to the entries we made during the new project setup process. The leftmost area of **Screenshot 4** displays our *DesignCycle1* project layout, while the right hand area of **Screenshot 4** welcomes us to the SES. This is a good place to stop and explore the features of the SES.

Attaching the J-Link PRO

Now that we have the SES pointed in the STM32F030R8T6 direction, we will need to perform some hardware work that will allow us to utilize the combined power of the J-Link PRO and the SES.

The STM32F0308-DISCO is packaged with a bare through-hole printed circuit board (PCB) whose hole pitch (0.10 inches) matches that of the STM32F0308-DISCO's pair of male headers. We will use this bare PCB to craft a programming/debugging adapter portal for the J-Link PRO.

The J-Link PRO you see in **Photo 2** sports a standard five-pin JTAG interface positioned on a 20-pin male header. The STM32F030R8T6 uses a two-pin SWD (Serial Wire Debug) programming interface. SWD replaces the five-pin JTAG interface with a clock pin (SWCLK) and bidirectional data pin (SWDIO). The SWCLK signal is attached to the JTAG TCK pin, while the SWDIO signal overlays the JTAG TMS signal. The SWD conversion is very easy to perform. The construction plan is laid out in **Schematic 1** and the physical result is shown in **Photo 3**.

We must remove the pair of CN2 jumpers. Removing the jumpers severs the logical link between the STM32F0308-DISCO's built-in ST-LINK/V2 programmer/debugger and the STM32F030R8T6. We can still get power for the STM32F030R8T6 via the STM32F0308-DISCO's USB portal.

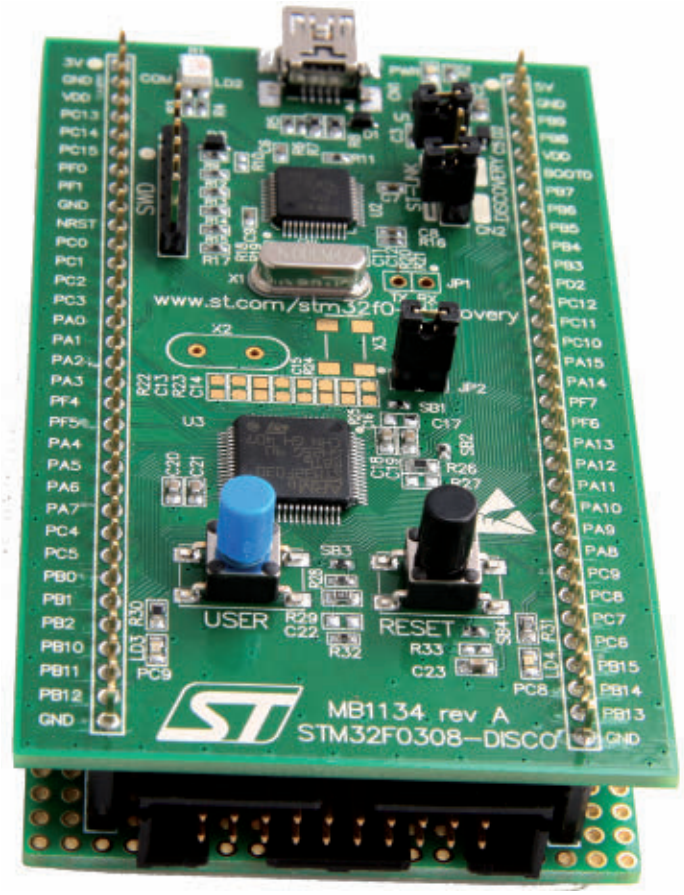
The original idea behind the CN2 jumpers is to allow the STM32F0308-DISCO's built-in ST-LINK/V2 to be used as a standalone programmer/debugger via the CN3 header pins.

If we've assembled our SWD interface correctly, the SES should see the J-Link PRO and allow a connection. **Screenshot 5** confirms our success. The STM32F0308-DISCO is powered up and attached to the J-Link PRO, which is recognized by the SES. It's time to write some code.

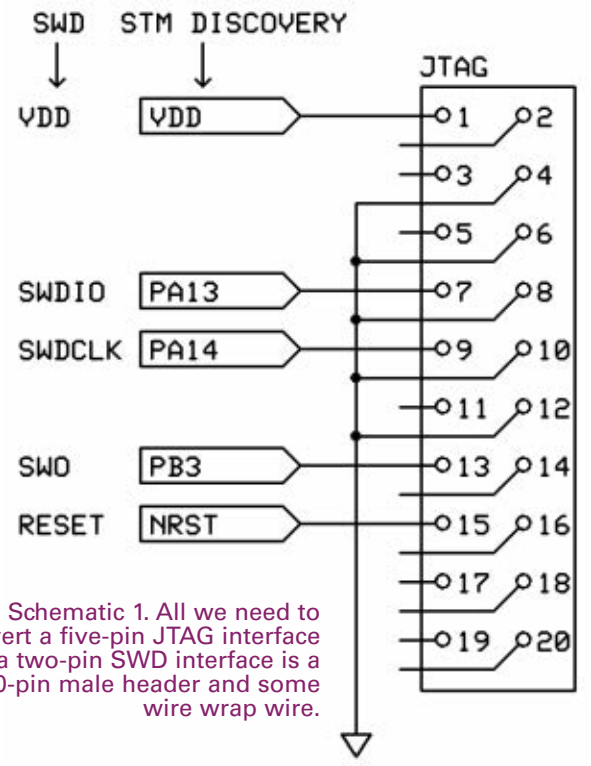
First Shots

If we are to be successful ARM programmers, we must understand the ARM architecture. Although reading the datasheets and reference manuals is a great way to get familiar with ARM, there is another way you can increase your ARM knowledge. Once you've done your reading, you can apply what you're learned to help with your understanding of the contents of the stm32f0xx.h file. In this case, the stm32f0xx.h file is formally known as the CMSIS (Cortex Microcontroller Software Interface Standard) Cortex-M0 device peripheral access layer header file. The stm32f0xx.h header file consists of a configuration section that allows us to specify the target device and crystal frequency. The stm32f0xx.h header file also contains data structures and address mapping for all of the device's peripherals.

Peripheral register declarations, bit definitions, and macros to access the peripheral hardware registers are



■ Photo 3. This is a shot of the STM32F0308-DISCO mounted on the bare printed circuit board. The 20-pin JTAG header is wired into the female headers according to Schematic 1.



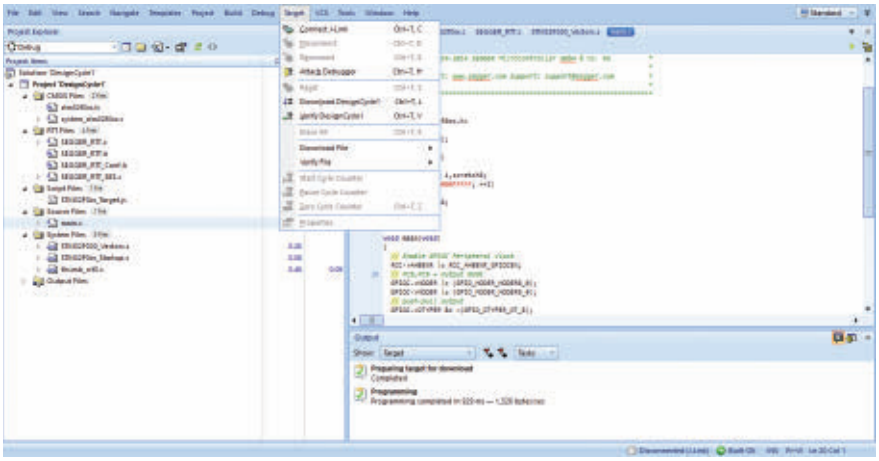
■ Schematic 1. All we need to convert a five-pin JTAG interface into a two-pin SWD interface is a 20-pin male header and some wire wrap wire.

also part of the stm32f0xx.h header file. Being able to associate the contents of the stm32f0xx.h header file with the datasheet and reference manual information is key to understanding how the ARM microcontrollers work and how to write code for them. With that, let's begin by placing a reference to the stm32f0xx.h header file at the beginning of our code:

```
#include <stm32f0xx.h>
```

The STM32F0308-DISCO schematic can be found in the user manual (UM1658). Once you access the schematic, you will see that the STM32F0308-DISCO's pair of user-accessible LEDs are attached to GPIO pins PC8 and PC9. The SES project creation process placed the system_stm32f0xx.c file in our project tree. The contents of the system_stm32f0xx.c file define our clock source (internal 8 MHz HSE with 6x PLL) and the resultant SYSCLK frequency (48 MHz). The system_stm32f0xx.c file also dictates the AHB (Advanced High-Performance Bus) and APB (Advanced Peripheral Bus) prescaler settings. The AHB and APB busses are all set to run full speed without prescaling. This is important information in that our LEDs are attached to GPIOC, which operates on the AHB2 bus. Here is the code to enable the GPIOC peripheral clock:

```
void main(void)
{
  // Enable GPIOC Peripheral clock
  RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
```



■ Screenshot 5. The STM32F0308-DISCO is powered up and attached to the J-Link. The Segger Embedded Studio is signaling that it's ready to go to work.

■ Screenshot 6. This capture of the RCC reset and clock control structure was pulled from the stm32f0xx.h header file. We are interested in the AHB entry.

Let's break down the clock enable code. Take a look at **Screenshot 6**, which is a capture of the RCC (Reset and Clock Control) structure contained within the stm32f0xx.h header file. The RCC peripheral operates on the AHB1 bus with a boundary address space of 0x4002100 – 0x400213FF. This boundary address is defined as the *RCC_BASE* within the stm32f0xx.h header file:

```
#define PERIPH_BASE          ((uint32_t)0x40000000)
#define APBPERIPH_BASE        PERIPH_BASE
#define AHBPERIPH_BASE        (PERIPH_BASE +
                               0x00020000)
#define RCC_BASE             (AHBPERIPH_BASE +
                              0x00001000)
```

Now that we have the boundary address, we can define RCC as a pointer to a structure. The structure is located at address *RCC_BASE*:

```
#define RCC                ((RCC_TypeDef *) RCC_BASE)
```

We can use the structure pointer to access elements within the RCC address space. These elements are defined in **Screenshot 6** and consist of a set of registers.

Registers are normally made up of a number of bits. We want to control bits within the registers. In this case, we want to set a bit that enables the GPIOC bus clock:

```
#define  RCC_AHBENR_GPIOCEN
  ((uint32_t)0x00080000)
```

■ Screenshot 7. The GPIO structure elements necessary to configure all of the STM32F030R8T6's GPIO subsystems are defined here.

So, our code points to the RCC boundary address (*RCC*), accesses a member of the structure within the boundary address space (*AHBENR*), and changes its value by setting a bit (*RCC_AHBENR_GPIOCEN*).

Now that the GPIOC bus clock is running, let's move on and set up the GPIO pins PC8 and PC9. We must write some code to define each pin's mode (input or output), output type (open drain or push-pull), speed, and pull-up status. Here is the code to define the mode:

```
// PC8,PC9 = output mode
GPIOC->MODER |= (GPIO_MODER_MODER8_0);
GPIOC->MODER |= (GPIO_MODER_MODER9_0);
```

GPIOC runs on the AHB2 bus in the boundary address space 0x48000800–0x48000BFF. This is verified by address entries in the stm32f0xx.h header file:

```
#define PERIPH_BASE          ((uint32_t)0x40000000)
#define AHB2PERIPH_BASE       (PERIPH_BASE +
                               0x08000000)
#define GPIOC_BASE           (AHB2PERIPH_BASE +
                              0x00000800)
```

Now, we can once again define the pointer to the structure:

```
#define GPIOC       ((GPIO_TypeDef *) GPIOC_BASE)
```

Screenshot 7's first entry is our structure element/register of interest (*MODER*). You can also see the other GPIO pin properties whose knobs we will need to twist.

Here are the GPIO mode bits that will configure PC8 and PC9 as outputs:

```
#define GPIO_MODER_MODER8_0
  ((uint32_t)0x00010000)
#define GPIO_MODER_MODER9_0
  ((uint32_t)0x00040000)
```

The GPIO pins are configured as push-pull by default on power-up or reset. So, the code we are writing is academic. We'll write it anyway:

```
// push-pull output
GPIOC->OTYPER &= ~(GPIO_OTYPER_OT_8);
GPIOC->OTYPER &= ~(GPIO_OTYPER_OT_9);
```

Note that we cleared the output type bits (&= ~) versus setting them (|=). If you are an AVR programmer, this type of bit set/clear coding is something you do quite often. I think you've got the idea. So, here's all of the GPIO configuration code:

```
// Enable GPIOC Peripheral clock
RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
// PC8,PC9 = output mode
GPIOC->MODER |= (GPIO_MODER_MODER8_0);
GPIOC->MODER |= (GPIO_MODER_MODER9_0);
// push-pull output
GPIOC->OTYPER &= ~(GPIO_OTYPER_OT_8);
```

```
  GPIOC->OTYPER &=
~(GPIO_OTYPER_OT_9);
  // max speed
  GPIOC->OSPEEDR |=
  (GPIO_OSPEEDER_OSPEEDR8);
  GPIOC->OSPEEDR |=
  (GPIO_OSPEEDER_OSPEEDR9);
  // no pullup/down resistors
  GPIOC->PUPDR &=
  ~(GPIO_PUPDR_PUPDR8);
  GPIOC->PUPDR &=
  ~(GPIO_PUPDR_PUPDR9);
```

LEDs are for Blinking

The structure pointer scheme worked well for us in the GPIO configuration code. Guess what? It works equally well for the manipulation of the GPIO pin states. This code should look familiar:

```
  do{
    GPIOC->BSRR = GPIO_BSRR_BS_9;
    //set PC9
    GPIOC->BSRR = GPIO_BSRR_BR_8;
    //reset PC8
    delay();
    GPIOC->BSRR = GPIO_BSRR_BR_9;
    //reset PC9
    GPIOC->BSRR = GPIO_BSRR_BS_8;
    //set PC8
    delay();
  }while(1);
```

You can find a reference to the BSRR register in **Screenshot 7**. We already know how GPIOC fits into the picture. As one would expect, the *GPIO_BSRR_BS_x* bit patterns are contained within the st32f0xx.h header file:

```
#define GPIO_BSRR_BS_8     ((uint32_t)0x00000100)
#define GPIO_BSRR_BS_9     ((uint32_t)0x00000200)
#define GPIO_BSRR_BR_8     ((uint32_t)0x01000000)
#define GPIO_BSRR_BR_9     ((uint32_t)0x02000000)
```

Our LED blinky program won't work without some code representing the *delay()* function:

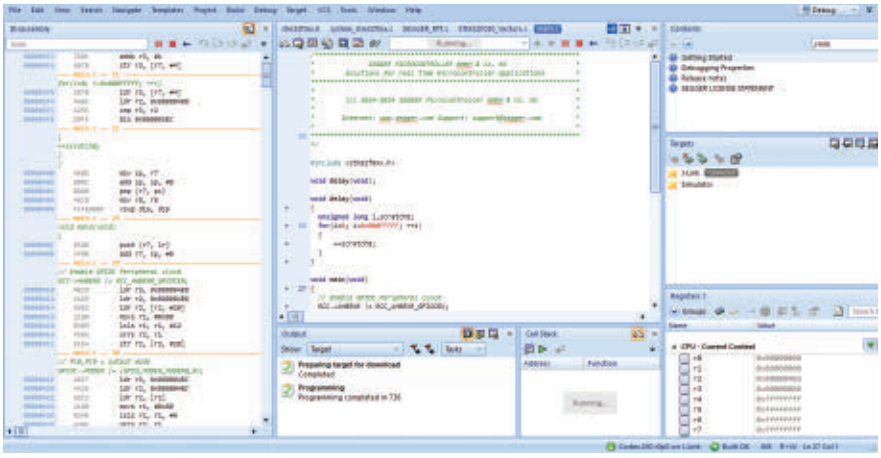
```
void delay(void)
{
  unsigned long i,scratch8;
  for(i=0; i<0x000FFFFF; ++i)
  {
       ++scratch8;
  }
}
```

The delay code is nothing more than standard C. There's no science to this delay routine. I just stuffed in enough "Fs" to make a half-second blink rate.

More to Come

I'll leave you with **Screenshot 8**, which is a screen capture of the SES running our little blinky program in debug mode. Now that we can flip bits on the STM32F030R8T6's GPIO pins, the door is open to coding for the remaining STM32F030R8T6 peripherals.

In the meantime, you can add the STM32F030R8T6, J-Link PRO, and SES to your Design Cycle. **NV**



■ Screenshot 8. I can't show you the blinking LEDs, but I can give you an idea of how the Segger Embedded Studio looks when running in full debug mode.

■ BY L. PAUL VERHAGE

North American — Guatemala Near Space Alliance

Near space encompasses the entire planet; therefore, it's not surprising to hear that amateur radio operators in other nations are running their own programs. When an amateur radio operator from Guatemala asked for help in kicking off his first launch, the near space community couldn't resist but help out.

Figure 2: The pallet is a 10 inch square of 1/4" Coroplast and the spacer is made from a sheet of the thinner 3/16" thick Coroplast that was cut, folded, and taped together. Most glue will not adhere to polypropylene, so I resorted to taping the ends of the Coroplast together. The spacer stands four inches tall.

Figure 1: A gift to Guatemala. This near spacecraft and a weather balloon were mailed to Guatemala so that amateur radio operators in this Central American country could start their own program of exploring near space.

Figure 3: Three pallets and two spacers stacked together created this three level avionics deck. I drilled holes through the center of each pallet after finding out that they were difficult to remove from the airframe without them.

Last summer, Juan Munoz TG9AJR emailed the GPSL Yahoo group asking if the near space community could help him start a program in his home country of Guatemala. Juan stated that he already had an APRS tracker and GPS for the near spacecraft, thanks to Zack Clobes W0ZC of Kansas. What he needed next was a balloon, parachute, and airframe. Michael Willett K5NOT of northeast Texas and I eagerly provided additional assistance by donating the balloon, parachute, and airframe to Juan's endeavor. The balloon and parachute were ready to send to Juan; it was the airframe that needed assembly. It's Juan's airframe and later modifications to the design that I want to talk about this time.

Promoting Amateur Near Space Exploration with Easy-to-Make Airframes

It's one of my desires that all interested amateur radio operators be able to reach near space as easily as possible. Once they start exploring, it's easy for them to keep exploring. One way to make this possible is to design airframes that are easy to assemble and flexible in function.

Reusable lunch bags are one simple way to make airframes for near space, but they can be difficult to modify for a wide range of experiments. For this reason, I think this makes them better for backup trackers and not primary near space exploration vehicles. Therefore, for Juan, I built an airframe from Styrofoam like those I use for mine. However, I wanted to avoid using Styrofoam packing peanuts to keep the avionics in place inside the module. So, I modified the internals of the airframe to secure the experiments without packing the open internal space with packing peanuts. This resulted in Juan's airframe being built around a stack consisting of corrugated plastic

pallets and separators.

The Pallets and Spacers

The name for the corrugated plastic used for the pallets and spacers is Coroplast, and it's available in some hobby shops and most plastic shops. The plastic used in Coroplast is Polypropylene — a plastic that has a slick surface that's difficult to glue anything to. However, Coroplast is very stiff and lightweight. The avionics and experiments are bolted to the pallets which are made from 1/4" thick Coroplast. Normally, I just use 3/16" thick Coroplast in my airframes, but the 1/4" thick variety is significantly stiffer and will stand up better when heavier items are bolted to it. I used 3/16" thick Coroplast for the spacers since they're turned edge up and very resistant to flexing and bending in this orientation. **Figures 2** and **3** show a spacer and the stacked pallets and spacers.

The Airframe

After completing the avionics pallets and their spacers, I could turn my attention to building the airframe since I knew how much space was required to hold the stacked pallets. The airframe — an opened-top box — is constructed from 3/4" thick Styrofoam sheets glued together with hot glue and then wrapped in package tape. The tape compresses the glued seams and adds color to the exterior of the Styrofoam box. If the color of the package tape is a dark one, then it will also increase the airframe's internal temperature by absorbing solar radiation during its mission into near space.

After gluing the airframe together, I cut openings into three of the airframe's sides. The first opening is for the control panel which is located in front of the airframe. The control panel has switches for powering the flight computer on and off, and a programming header and LED indicators. The control panel allows me to communicate and program the flight computer without opening the airframe. Plus, it also indicates the status of the flight computer through a series of four LEDs.

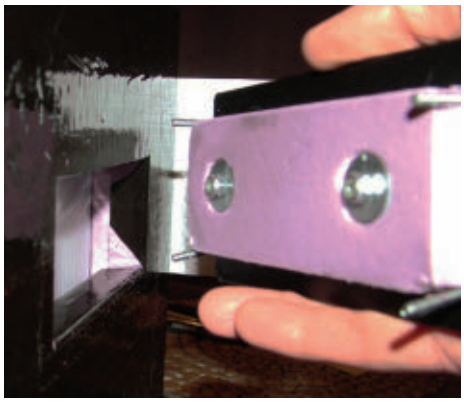


Figure 4: This block of Styrofoam fits snuggly into one of the airframe's experiment ports. The Styrofoam block is sandwiched into place by a slightly larger sheet of foamed PVC plastic on the outside of the airframe and a thin sheet of polystyrene on the inside.

Figure 5: These fabric straps are attached to a second airframe I made. The fabric straps are grosgrain tape and I get them by the yard from a crafts store. The red color of the straps ought to make the airframe stand out.

To attach the panel, I cut a square out of the bottom front of the airframe and bolted the circuit board of the control panel to the airframe. The panel's location on the airframe makes it easy to see and access.

Next, I needed a simple way to add experiments — like cameras — to the airframe without having to make new openings in the airframe and patching up old ones every time I changed experiments. So, I cut two additional openings (called experiment ports) into the airframe, but on the sides this time. These experiment ports provide a way to connect experiments to the outside of the airframe or to let sensors peer outside the airframe while their electronics remain inside the warmer airframe. The experiment ports measure three inches tall and six inches wide. The openings are sealed with Styrofoam rectangles that bolt to the openings with four bolts (one in each corner). The outside of the Styrofoam rectangle is bolted to a slightly larger rectangle of foamed PVC (Sintra is its commercial name) so the Styrofoam can't fall inside the airframe. A thin sheet of polystyrene plastic is located inside the airframe to keep the rectangle of Styrofoam from falling outside the airframe.

Nuts, bolts, and washers in each corner attach the sandwich of

Figure 6: A view from the back. The antenna plane and antenna itself are lightweight; therefore, only three bolts are required to attach the Coroplast plane to the airframe. All bolts in the airframe need to use nylon locking nuts, or nylocks. Their nylon inserts insure they will not rattle loose during a mission.

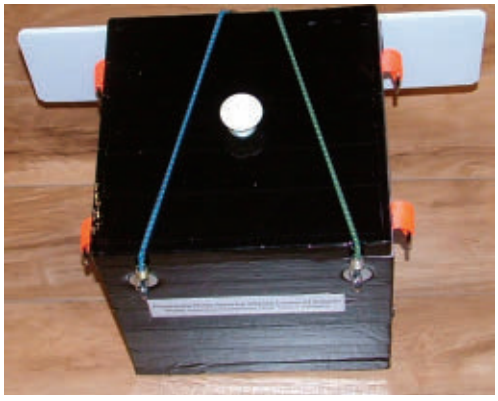


Figure 7: This is Guatemala's module. It shows the two bungee cords wrapped around the hatch. Eye bolts and fender washers provide places for the bungee cords to hook onto the airframe.

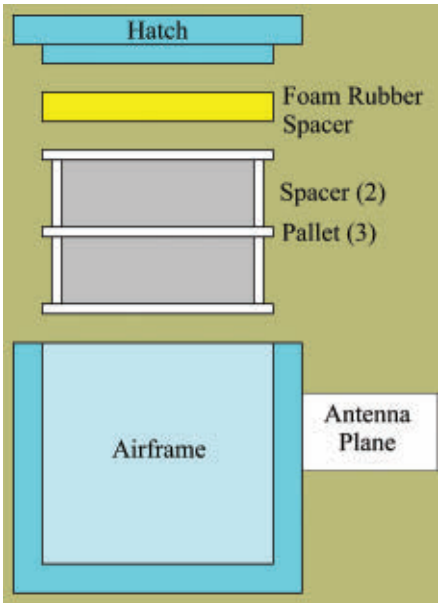


Figure 8: An X-ray view of the airframe. The sheet of foam rubber keeps the battery (which is on top of the top pallet) in place. When the hatch is bungeed down, the foam rubber squeezes the battery and keeps it from moving during the mission.

Polystyrene, Styrofoam, and Sintra securely to the airframe. When changing the experiments for a mission, I only need to make new Styrofoam rectangles that are specific for the experiment and then cover the experiment port. (This sure beats constructing an entirely new airframe every time an experiment changes!)

The airframe connects to the parachute and other modules with a pair of fabric straps. The straps — which are 7/8″ wide grosgrain — pass below and up the sides of the airframe. The straps are bolted to the airframe and form a harness for attaching the parachute and other modules. The top ends of the straps fold over so split metal rings can be used to connect the parachute. There are loops of Dacron kite line tied to holes in the bottom corners of the grosgrain where BalloonSats and other payload modules are connected to the airframe.

The Antenna

There are many types of radio antennas, but the design I recommend for near spacecraft is the quarter-wave dipole. This antenna design produces a signal that's uniformly strong in the horizontal plane and grows gradually weaker at points directly above and below the antenna (the point directly above and below the antenna is called a null, since the signal is theoretically zero here).

At first glance, this sounds like a bad design since ground assets are tracking from below the near spacecraft (the null at the top is unimportant since no one is tracking above the near spacecraft). However, the null is a small region on the ground and it's infrequent that a tracking crew is directly below the near spacecraft. Most often, tracking crews are some distance away from the near spacecraft and not directly beneath it. Therefore, chase crews receive sufficiently strong signals for the majority of the mission.

Also, because of the popularity of amateur radio, there's always a ground station some distance away from the near spacecraft and in reception of the near spacecraft. Many of these stations are called Internet Gates, or IGates and they put the airframe's position reports on the Internet where chase crews can access them over a smartphone.

Since the antenna is permanently a part of the airframe, it's held in place on a plane of Coroplast which is bolted to the back of the airframe. This places the antenna and its plane where it can't interfere with the airframe's three openings.

The antenna's RG-174 coax cable passes into the interior of the airframe through a small hole drilled through a Styrofoam wall. The opening is then sealed with a drop of hot glue to prevent it from becoming a source of cold air leaking into the airframe. A small notch cut into one of the vertical spacers creates a small gap between the pallet and spacer so the RG-174 coax can reach the transmitter on the flight computer.

Airframe Hatch

The airframe's hatch is made from two sheets of Styrofoam that I glued together. The hatch's top square of Styrofoam has the same dimensions as the outside of the airframe, and its inner sheet is 1-1/2″ smaller so it snuggly fits inside the airframe's top opening. By sliding the hatch's smaller Styrofoam square into the airframe, the hatch is prevented from sliding around the top of the airframe. I added a plastic knob to the center of the hatch just to give it a handle. A 2″ long sheet metal screw and washer attaches the knob to the hatch without crushing the Styrofoam when it's tightened.

Two bungee cords hold the hatch to the airframe. The bungee cords are 10″ long and thin, since they don't have a lot of weight to hold in place. Metal eyebolts and hooks bolted to the airframe are used to hook the bungee cords in place.

Feel free to use this near space airframe design, or modify it for your particular needs. I would be interested in hearing about any variations done to the design. As soon as Juan completes his first mission, I'll be sure to share the results here.

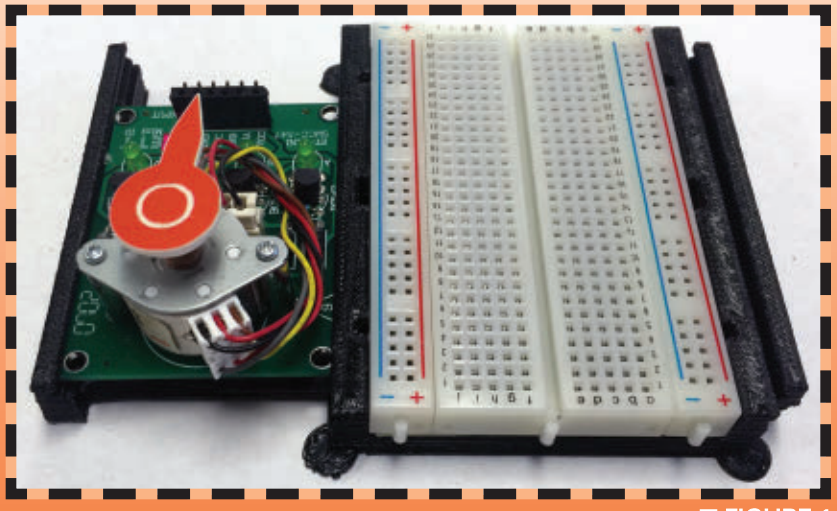
Onwards and Upwards,
Your Near Space Guide **NV**

BY CHUCK HELLEBUYCK

REAL WORLD USES FOR THE ELECTRONICS EXPERIMENTER

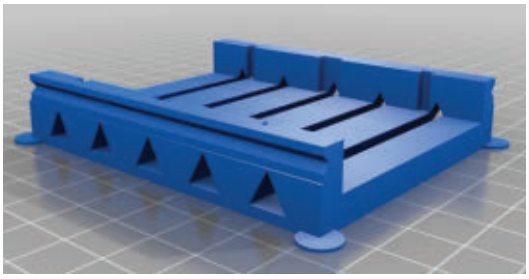
3D Printed Breadboard Base with Sidecar Supports

I almost always test out my circuit designs on a breadboard. So, when I find a unique breadboard base, it catches my eye. I was cruising Thingiverse.com and came across a rather interesting but simple breadboard base with slide-in sidecar style board supports (Figure 1). There are so many times I've connected to a board that doesn't plug into a breadboard, and the only thing holding it together was the connection wires. The connection wires as the only support makes the project difficult to move around. This 3D printed breadboard base with sidecar supports that are spring loaded with a rubber band make the whole design portable. I just had to print this on my 3D printer and try it out.

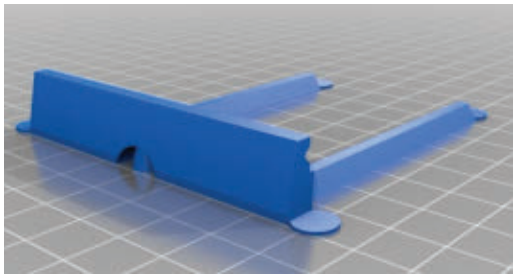


■ FIGURE 1. Assembled unit with sidecar board.

The original design is from Thingiverse user Patshead. It's called the Breadboard Spring Vise (though I like my sidecar name better). The side sections have triangulated shaped arms that slide under the breadboard base through guide holes on the sides. Two sidecars are connected to each other with a rubber band as mentioned. The design has two pieces: a base to hold the breadboard and the sidecar which can slide in from either side. This requires three objects to print: one base (**Figure 2**) and two sidecars (**Figure 3**). Each 3D design has pads on the corners to help hold them down to the platform of your 3D printer. You can get the same effect with adding a brim or a raft. The idea is

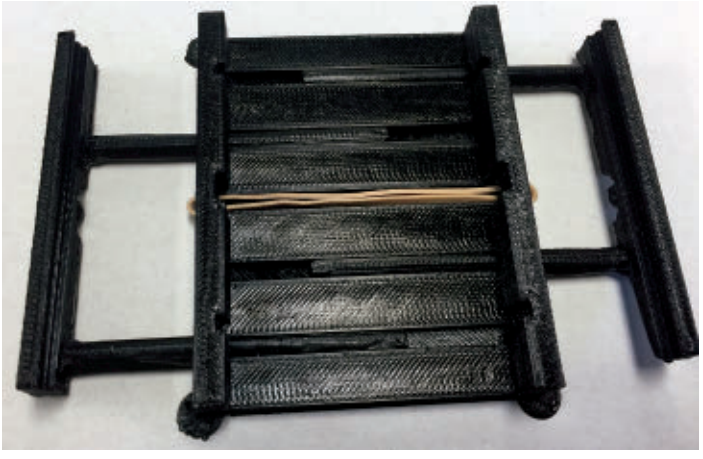


■ FIGURE 2. Breadboard base.

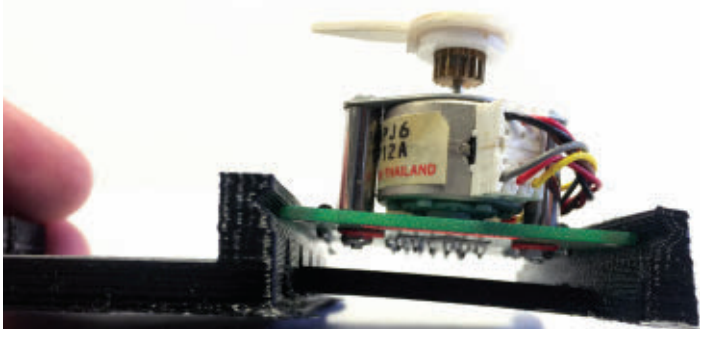


■ FIGURE 3. Sidecar.

■ FIGURE 4. Pieces to assemble.



■ FIGURE 5. Rubber band inserted.



■ FIGURE 6. Sidecar grooves hold the board.

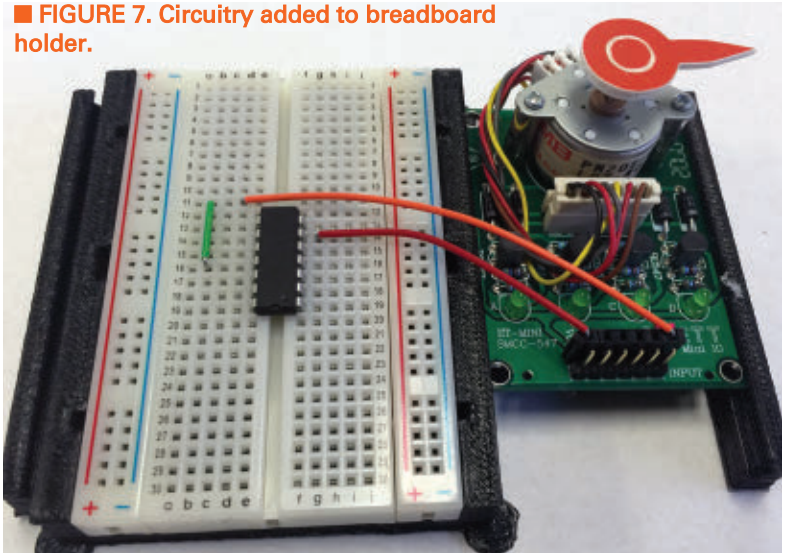
to create more surface area so the plastic stays flat and doesn't warp while cooling. These are then broken off after printing. I decided to leave them on the base to allow it to be clamped or screwed down to a board if I wanted in the future.

When I printed this on my da Vinci 1.0, one of the sidecar's surface areas seemed large enough to hold it down, but the edge lifted a little during the print. The sidecar still works but I'll probably reprint it at some point. All the design pieces fit on my larger da Vinci 1.0 3D printer (200 mm x 200 mm), but just about any printer can produce these because you can print them individually on even the smallest machine.

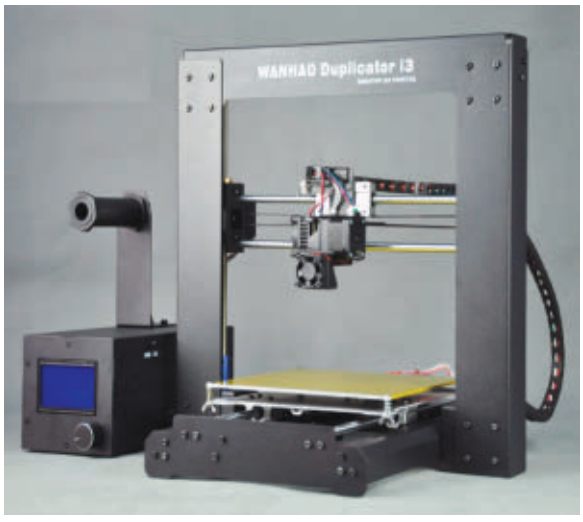
A 4" x 4" bed size (100 mm x 100 mm) can accept any of the parts individually. The base has five triangular holes in the sides and slots that run horizontally. This design was for a medium sized breadboard, however, there is also a design available for a larger size breadboard. The sidecars are offset so the one on the right uses two of the five triangular holes, while the one on the left uses two more. This leaves a center groove for the rubber band to run through. I gathered up all the pieces (**Figure 4**) I needed to assemble it, including the medium sized breadboard, rubber band, base, and two sidecars. Then, with a little help from a tiny screwdriver, I fed the rubber band into the center slot (**Figure 5**). The sidecars have cone shaped posts on the bottom to accept the rubber band loop.

The rubber band pulls the sidecars into the base at the same time from the same rubber band. The base has a V-groove on the side to accept a circuit board edge; the sidecar also has the same groove (**Figure 6**). This allows just about any circuit board two inches wide or less to be held in place. Then, you can use jumper wires to the board from the main breadboard. I installed a stepper motor board that I will control from a microcontroller. The stepper motor board is an ETT-Mini board I found at **www.futurlec.com/Mini_SMCC.shtml**. They have lots of

■ FIGURE 7. Circuitry added to breadboard holder.



■ FIGURE 8. Prusa I3 style printer from WanHao.



different mini boards to choose from, but some don't offer pins that fit in the breadboard. They do have header pins that can be pushed through with a solder iron but this sidecar option makes it easier to just leave them as they are and connect to the breadboard with a few jumper wires. The circuit shown in **Figure 7** is not a complete circuit as I took the picture as a demo. The focus of this article is to show how a 3D printer can actually make your electronics bench even easier to work at as you can easily print custom tools like this. There are a lot of different options for breadboard holders on **Thingiverse.com** and elsewhere. You can also easily design your own with programs such as Tinkercad, which is like building with blocks. Once you have a 3D printer, it's amazing how many little helpful gadgets like this breadboard holder with a sidecar can be found and printed. Another option is to create a custom board holder with grooves to fit one particular board. If you need to build up 10-20 boards, then having a holder that you can slide the board into the grooves to hold it in place while you solder can save you a lot of time. Or, just use this same rubber band adjustable method for a variable side holder. The possibilities are endless.

Fabrikator Update

In my last article, I introduced the Fabrikator Mini 3D printer since it is so small it can easily fit on any workbench. Unfortunately, **Hobbyking.com** has since discontinued that printer.

A Prusa I3 style printer (**Figure 8**) is another great option. Many are offered as kits on eBay, but you can find one assembled for around $400. Another option for those that want less complexity but are willing to settle for less printer temperature and settings control is the da Vinci Jr (**Figure 9**). It prints PLA plastic only but does a good job at just $349 fully assembled.

I did several review videos on this printer at my YouTube Channel (**youtube.com/elproducts**).

When I got my first 3D printer, I wasn't sure if I would use it enough. Now, I don't know what I would do without it. Check out my YouTube channel for more tips on how to get started with 3D printing. I host a show called Filament Friday every week. **NV**

Resources

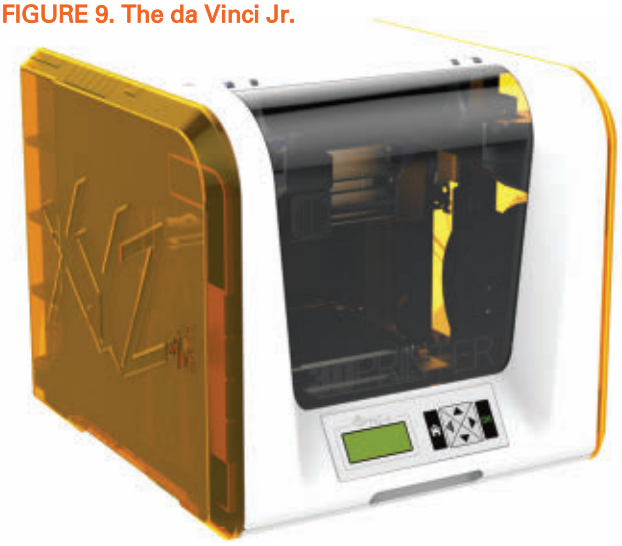
Check out my website and blog:
www.elproducts.com

My 3D designs:
www.thingiverse.com/elproducts/designs

My YouTube Channel:
www.youtube.com/elproducts

Tinkercad:
www.tinkercad.com

■ FIGURE 9. The da Vinci Jr.



BY LOU FRENZEL W5LEF

The Internet of Things: Who Needs It?

By now, you have probably heard of the Internet of Things (IoT). Also called the Internet of Everything, this is the concept of connecting every sort of device or product to one another or to a human via the Internet. Now that most of the affluent world is connected to the Internet, why not connect everything else? We have the power and the technology, so let's do this. But is this a good idea?

With everyone writing or talking about the IoT, you would have to say this is the next big technological thing. After all, it allows for a host of "interconnecting." Do you want your refrigerator to communicate with your HVAC thermostat? Or, do you care if you can turn your sprinkler system off and on while you are on vacation? How about sharing your fitness data from your smartwatch with your trainer over the unsecured Internet? That all sounds pretty amazing.

The fact is IoT is happening right now. Potentially, there could be as many as 50 billion or so devices connected by 2020, according to recent research. That will affect you, me, and just about everyone else. So, be prepared.

IoT Defined

IoT is the concept that a device can be monitored or controlled remotely from anywhere by way of an Internet connection. The basic idea is illustrated in **Figure 1**. A device such as a sensor talks to a remote server that, in turn, is connected to a tablet or smartphone. The computer could just collect the sensor data over time for later analysis. Or, someone with a smartphone could be monitoring the sensor output in real time. There are all sorts of other variations and scenarios.

One of the largest application areas is the smart home. This is a home where many devices can be connected. The HVAC thermostat is one key target (**Figure 2** is a good example). So are major appliances like the refrigerator, the washing machine, or the coffee maker. The reason for the interconnection has yet to be determined in some cases, but as an example, the refrigerator may report its physical status back to the manufacturer or you may want to be able to turn the coffee maker off or on remotely.

Other applications in the smart home are security systems, door locks, video cameras, or baby monitors. Pool pumps, sprinkler systems, garage doors, lighting, and other stuff are additional examples.

I will admit to buying a video camera that I can monitor on my iPhone. The video cam talks to my Wi-Fi router which has an Internet connection via my cable company. My iPhone has an app that links

Figure 2. The Nest thermostat is a popular IoT product using Wi-Fi that lets you control and monitor your HVAC remotely.

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/magazine/article/february2016_OpenCommunication_Iot**.

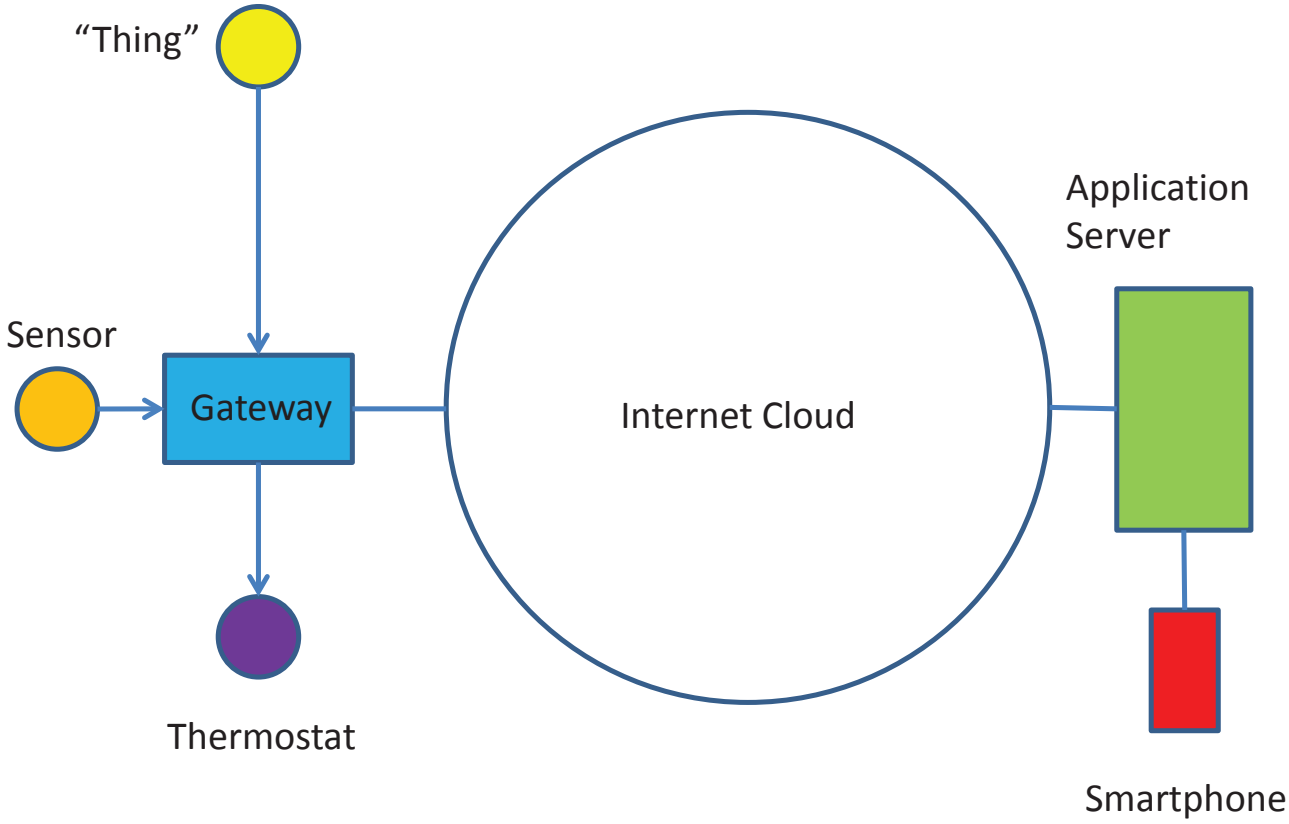


Figure 1. A simplified view of how the Internet of Things works. The "thing" connects to a gateway wirelessly. The gateway has the Internet connection to the cloud. At the other end of the link is a server dedicated to the application. This, in turn, links via a cellular connection to a tablet or smartphone.

by way of the cellular network to a remote server that collects the video for the camera and forwards it to my phone. Totally neat.

While the smart home will probably be the largest sector of the IoT, there are other application sectors. A big one is the industrial Internet of Things, or IIoT. Here is where factories, plants, oil, and gas fields can monitor and control things from afar. Sensors of all sorts are the main interest where temperature, flow, mechanical position, pressure, and other critical physical characteristics can be observed in real time. Other things to be controlled are motors, pumps, relays, solenoids, robots, programmable logic controllers (PLCs), and machine tools.

Such monitoring and control has been going on for years, however, now with new IoT technology such as wireless and Internet connectivity, it can be done easier, more affordably, and on a larger scale.

Another term you will hear related to the IoT movement is machine-to-machine (M2M). M2M has been around for years also, and has been used for remote monitoring of vending machines, unmanned facilities, and truck fleets. It primarily uses the cellular network rather than the Internet for communications. M2M is basically a blurry part of IoT (in my opinion).

There are all sorts of other applications emerging such as in medical and fitness monitoring, wearables like smartwatches, agriculture, the smart grid, and smart cities and drones. It will be interesting to see what new uses develop.

Adoption Issues

IoT is happening right now, but there are a number of critical issues that developers must consider when creating an IoT product or system. Here are just a few examples:

- Privacy and security.
- Interoperability of devices and systems.
- Ease of setup and installation.
- Managing large data collections.
- Multiple standards.

As for security, we all want our data to be secure. We do not want our systems hacked to capture and use whatever data we collect. Nor do we want some outside entity to take control of our system. This means that our devices must use some form of encryption and authentication. Luckily, most new IoT chips and products do have built-in security measures.

To have all devices talk to the gateway or to one another, they must all use the same wireless standard and

protocol. With multiple standards and protocols in use, this is a problem. In addition, even if the same standards and protocols are used, minor variations can keep devices from working together. This will require rigid testing and certification to ensure interoperability.

Ease of installation is a huge issue — at least for the consumer. If installation and setup processes are complex, arcane, and lengthy, consumers will just be frustrated and home IoT could be a flop — all the more reason to take the extra design time to make sure installation is a snap.

IoT systems are designed to capture data. This they do well. They can collect a massive amount of data is a short time. This means having enough memory to store the data. Then, what do you do will all that data? Do you display it, analyze it, or what? Some early studies of IoT indicate that 70 to 90 percent of all the data collected is never used. (That kind of statistic kind of makes you wonder why the application was implemented in the first place.) This is a major issue to think through before investing in IoT.

Multiple standards is a key issue as indicated earlier. Wireless standards are a good example. There are many short range wireless technologies that will work with IoT. All are vying for a piece of the IoT pie. Wi-Fi is probably one of the early leaders in this race. It is highly developed with multiple variations, plus an in-place certification process.

Most homes have a Wi-Fi router already in use. These days, virtually all laptops, tablets, and smartphones include Wi-Fi. It is a good reliable choice and an early leader in the standards battle. Wi-Fi is fast and convenient, but can be expensive overkill for many simple sensor applications.

Bluetooth is another good choice. It is widely available in smartphones, and the new low energy version will ensure that sensors can run for years on a single battery. Certification is available to make sure all devices work together.

ZigBee is another standard that is a major contender — especially where lots of sensors are involved. Based on the IEEE 802.15.4 standard (see **sidebar**), ZigBee's mesh topology lets you build systems with hundreds or even thousands of sensors. It's a great choice for business and industry.

Yet another standard showing up in smart home products is Z-Wave. This is a simple wireless technology that is ideal for low data rate home monitoring and control. It is low cost and interoperable.

Of course, there are many others to look at. There is Weightless that uses the white spaces on vacated VHF and UHF TV bands. LoRa is another that uses the lower frequencies below 1 GHz. Both Weightless and LoRa promise longer range connectivity up to several miles in applications where the devices are some distance away.

Cellular connectivity is also an option. There are cell phone chips and modules, and most wireless carriers offer connection services. This is a good reliable choice for more critical applications.

Let's not forget the software side of IoT. There are multiple different software protocols for handling the data. These operate with the wireless chips and modules. Some examples are MQTT, JSON, IPSO, IoTivity, AllJoyn, and Apple's HomeKit. A popular standard is 6LoWPAN that lets other protocols use the IPv6 Internet protocol.

A more recent example is Thread. Thread uses the popular IEEE 802.15.4 wireless standard, but adds a mesh topology protocol to handle many sensors or other devices.

What is 802.15.4?

802.15.4 is one of the Institute of Electrical and Electronic Engineers' (IEEE) short range wireless standards. It works in several of the unlicensed spectrum, but primarily in the 2.4 to 24.835 GHz spectrum where Wi-Fi and Bluetooth reside. It uses direct sequence spread spectrum (DSSS) with differential BPSK or offset QPSK modulation.

Data rates can be up to 250 kb/s. The typical power output is 1 mW or 0 dBm. The range is up to 10 meters or so, depending on the environment. It uses very low power, so can operate from a battery for years. This standard implements the physical (PHY) and media access control (MAC) layers of the OSI networking model. Other networking layers are built on this, based on the application.

This is a popular model as other standards are based on it, including ZigBee, ISA 100-11a, WirelessHART, and Thread.

Been There, Done That

The idea of IoT is not new. It has been around for decades. We used to call it remote monitoring and control telemetry. To me, IoT is just telemetry on a grander scale, thanks to the Internet and lots of low cost/low power wireless products.

Now that it's easier than ever to create products we can operate remotely or collect data from, I suspect IoT will become even more successful and prolific.

Although, personally, I'd be okay with us bringing back X10 power line controllers and the Clapper. **NV**

READER FEEDBACK Continued from page 7

Change line #124 in the file ThinkingOfYou.ino to:

```
const char *ssid = WiFi.SSID().c_str();
```

and the code will compile cleanly.

Craig Lindley

Errata on the Coop Boss

In the January 2016 issue, John Rucker's article on "SmartThings and the Device Maker" had some updates/changes that missed getting in. First off, SmartThings® is a registered trademark name/brand. Secondly, there are four figures that needed to be replaced because of a software update. So, the new graphics for Figures 11, 12, 14, and 15 are included here.

Finally, when referring to Figure 1, please note that the following letters represent the items in the article text that were shown in bold:

A = Motor Connection
B = Main I/O Header
C = DC In
D = Aux Port
E = Antenna

We apologize to Mr. Rucker and for any inconvenience(s) our oversights caused readers.

NV staff

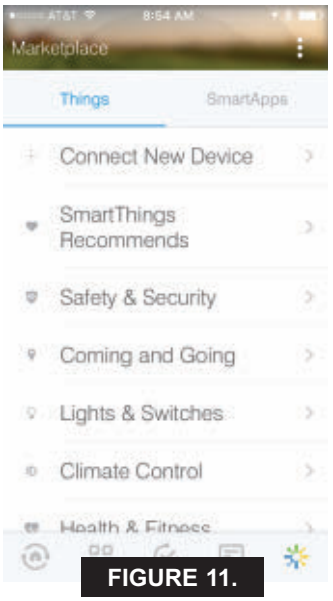


FIGURE 11.

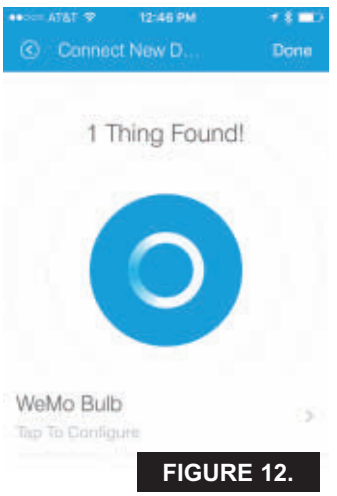


FIGURE 12.

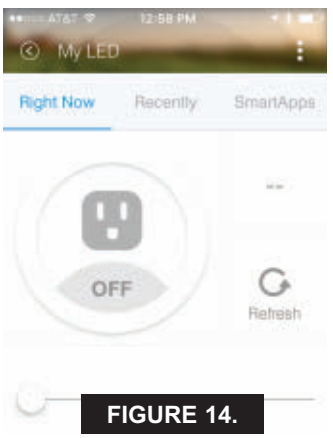


FIGURE 14.

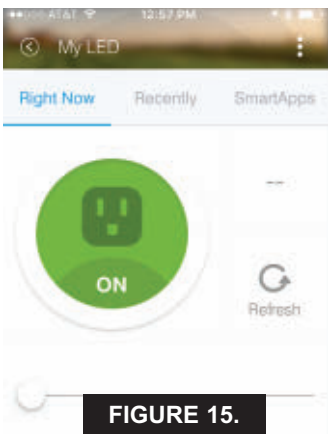
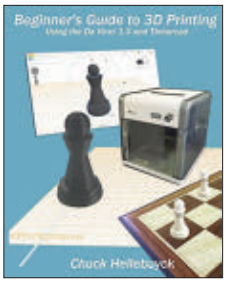


FIGURE 15.

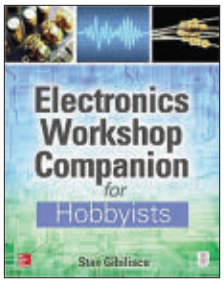
GREAT FOR DIYers!

Beginner's Guide to 3D Printing
by Chuck Hellebuyck



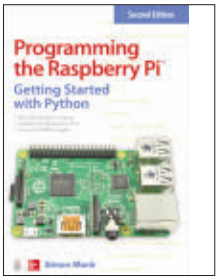
This book was written to answer the questions most beginners need answered. It covers many of the popular 3D printer choices and then uses the under $500 Da Vinci 1.0 from XYZprinting to show you how easy it is to get started. I take you through using Tinkercad software for creating your own custom designs. I go further and show you how to take a simple design and send it off to a professional 3D printer. This book was designed for anyone just getting started.
$24.95

Electronics Workshop Companion for Hobbyists
by Stan Gibilisco



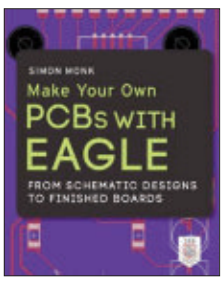
In this practical guide, electronics expert Stan Gibilisco shows you, step by step, how to set up a home workshop so you can invent, design, build, test, and repair electronic circuits and gadgets. Electronics Workshop Companion for Hobbyists provides tips for constructing your workbench and stocking it with the tools, components, and test equipment you'll need. Clear illustrations and interesting do-it-yourself experiments are included throughout this hands-on resource.
$25.00

Programming the Raspberry Pi Second Edition: Getting Started with Python
by Jack Simon



This practical book has been revised to fully cover the new Raspberry Pi 2, including upgrades to the Raspbian operating system. Discover how to configure hardware and software, write Python scripts, create user-friendly GUIs, and control external electronics. DIY projects include a hangman game, RGB LED controller, digital clock, and RasPiRobot complete with an ultrasonic rangefinder.
$15.00

Make Your Own PCBs with EAGLE
by Eric Kleinert



Featuring detailed illustrations and step-by-step instructions, *Make Your Own PCBs with EAGLE* leads you through the process of designing a schematic and transforming it into a PCB layout. You'll then move on to fabrication via the generation of standard Gerber files for submission to a PCB manufacturing service. This practical guide offers an accessible, logical way to learn EAGLE and start producing PCBs as quickly as possible.

$30.00

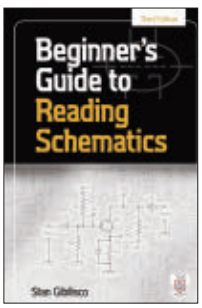
Programming the Intel Edison: Getting Started with Processing and Python
by Donald Norris

Learn To Easily Create Robotic, IoT, and Wearable Electronic Gadgets!

Discover how to set up components, connect your PC or Mac, build Python applications, and use USB, Wi-Fi, and Bluetooth connections. Start-to-finish example projects include a motor controller, home temperature system, robotic car, and wearable hospital alert sensor.

$20.00

Beginner's Guide to Reading Schematics, 3E
by Stan Gibilisco



Navigate the roadmaps of simple electronic circuits and complex systems with help from an experienced engineer. With all-new art and demo circuits you can build, this hands-on, illustrated guide explains how to understand and create high-precision electronics diagrams. Find out how to identify parts and connections, decipher element ratings, and apply diagram-based information in your own projects.

$25.00

How to Diagnose and Fix Everything Electronic
by Michael Jay Geier

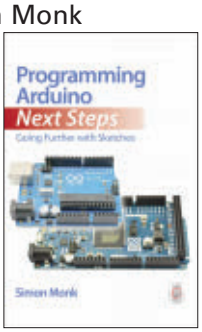
Master the Art of Electronics Repair

In this hands-on guide, a lifelong electronics repair guru shares his tested techniques and invaluable insights. *How to Diagnose and Fix Everything Electronic* shows you how to repair and extend the life of all kinds of solid-state devices, from modern digital gadgetry to cherished analog products of yesteryear.
$24.95

Programming PICs in Basic
by Chuck Hellebuyck

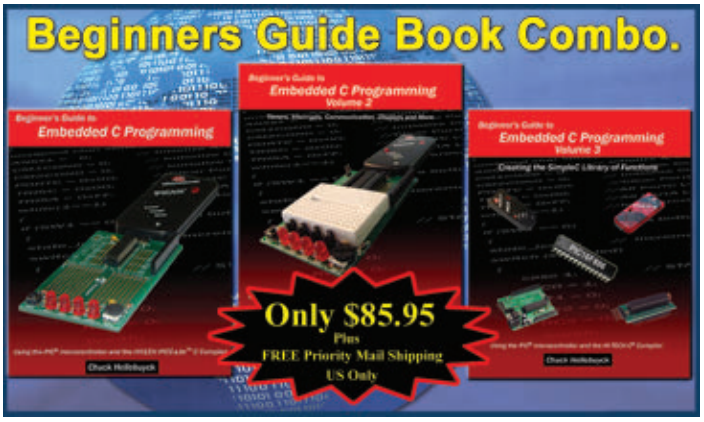
If you wanted to learn how to program microcontrollers, then you've found the right book! Microchip PIC microcontrollers are being designed into electronics throughout the world and none is more popular than the eight-pin version. Now the home hobbyist can create projects with these little microcontrollers using a low cost development tool called the CHIPAXE system and the Basic software language. Chuck Hellebuyck introduces how to use this development setup to build useful projects with an eight-pin PIC12F683 microcontroller. **$14.95**

Programming Arduino Next Steps: Going Further with Sketches
by Simon Monk

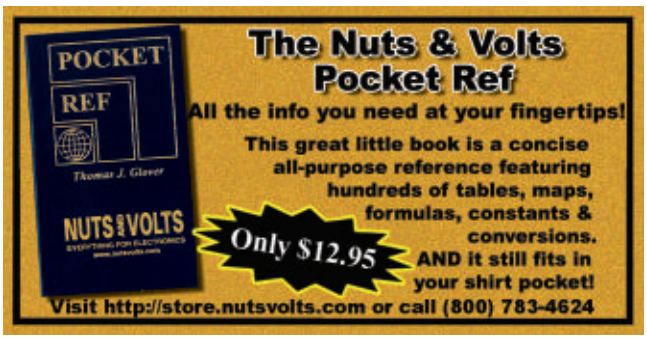


In this practical guide, electronics guru Simon Monk takes you under the hood of Arduino and reveals professional programming secrets. Also shows you how to use interrupts, manage memory, program for the Internet, maximize serial communications, perform digital signal processing, and much more. All of the 75+ example sketches featured in the book are available for download. **$20.00**

EDUCATIONAL



CD-ROM SPECIAL



>>> QUESTIONS

Thirst For Knowledge

I've been an *NV* subscriber for years. I have an interest, but not a proficiency — yet. Some articles I understand, some I do not, and I really would like to get to the point where I can run with most things presented in each issue, as well as in other magazines. Coming up on retirement, I will have more time to invest in areas of interest. So, I'm looking for guidance in a couple of directions.

1. I want to get proficient building things using small microprocessors (home automation, data loggers for temp and humidity, robotics, small handheld computers, hacking appliances, other??), but I'm not sure where to start and the best way to learn. I've built a few PCs over the years, and while I would have to dust off my memory, I have programmed a little in the past with Visual Basic and SQL.

2. I want to get more knowledgeable in AC/DC electronics. I've taken a basic fundamentals course at community college a number of years ago, but would like to refresh that and get on a path to continue building my knowledge base in all aspects, be it AC/DC electronics, computers, communications, etc.

3. I have a two year old grandson and I want to get him excited about technology, programming, math, engineering, etc., at age appropriate levels, and see what — if anything — he might want to do with the knowledge as he gets older. Figuring that almost any profession he chooses will benefit from exposure and proficiency in this area, and it can't be accomplished overnight.

Any insights, thoughts, sources, books, coursework, etc., anyone can share would very much be appreciated, as well as any thoughts on where you see all this going in the next 20 years, so maybe I can guide my grandson to get out in front of things.

Thank you in advance for anything you can share.

#2161 **Mark McCurdy**
Grapevine, TX

68HC11 To DS1302 RTC

I recently purchased what some would consider a dinosaur when it comes to microcontrollers: a 68HC11. I purchased it at an estate sale because it was one I used while attending school. My question is where can I find info on creating a program that would allow me to interface the SPI to a DS1302 RTC?

#2162 **Michael Bennett**
Kankakee, IL

Cranky Flashlight

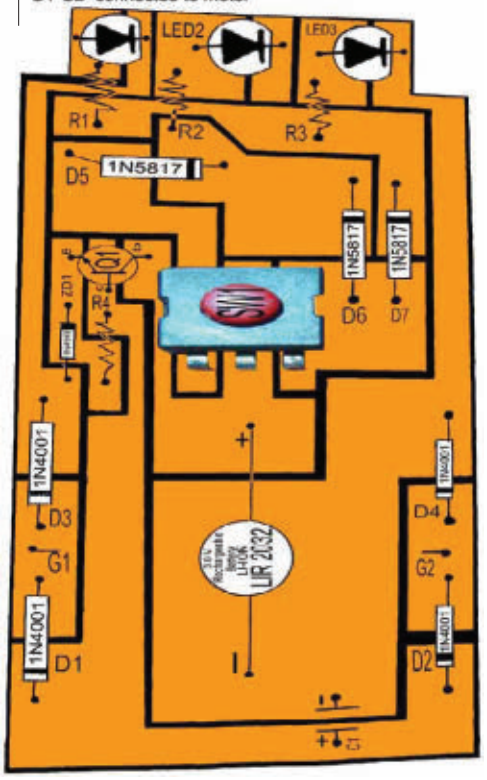
I have a three-LED hand-crank flashlight. I've included a copy of the PCB (orange) hoping to find out where the problem may be. The switch (SW1) provides two modes of lighting: one LED only (LED2) and all LEDs (LED1-LED3); in the OFF mode, the hand-crank is to be used in order to recharge the 3.6V rechargeable battery.

The hand-crank flashlight works in either of the two modes ONLY WHEN the hand-crank is being used. When I stop hand-cranking it, that's when lighting stops. I assume the li-on battery is not being charged (no white stuff around it for poor connection).

The black/plastic transistor is marked as SC8050. NTE (**NTEinc.com**) doesn't have a replacement for it. Can someone suggest a replacement and if it's NOT the transistor, suggest where the problem might be?

#2163 **Nate Franklin**
Schererville, IN

ZD1=1N4148
D1-D4=1N4001 (50V-1Amp.)
D5-D7=1N5817 (1Amp.-20PIV)
C1=47uF-16V
SW1= SPST switch (ON-ON-OFF)
Q1=SC8050,NPN Silicon, TO-92
R1-R3=10-OHM,1/8W Resistors(Brown,Black,Black)
R4=47-OHM,1/8W (Yellow,Violet,Brown)
LED1-LED3=T-1 3/4, white,water clear in off-state
G1-G2=connected to motor



All questions *AND* answers are submitted by *Nuts & Volts* readers and are intended to promote the exchange of ideas and provide assistance for solving technical problems. All submissions are subject to editing and will be published on a space available basis if deemed suitable by the publisher. Answers are submitted by readers and **NO GUARANTEES WHATSOEVER** are made by the publisher. The implementation of any answer printed in this column may require varying degrees of technical experience and should only be attempted by qualified individuals.

Always use common sense and good judgment!

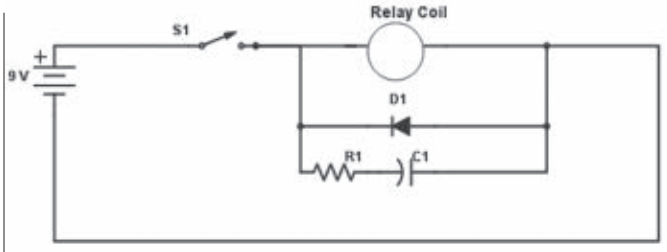
Send all questions and answers by email to **forum@nutsvolts.com**
or via the online form at **www.nutsvolts.com/tech-forum**

>>> ANSWERS

[#1161 - January 2016]
Clipping The Spike

I'm using an Arduino to control a set of relays, and both the microcontroller and relays share a 9V power bus. I plan to put a diode across the relays to clip any reverse voltage spikes. Is there anything else I should consider to prevent a spike from shutting down the processor?

#1 The inductive reaction from opening a circuit which contains an inductor (relay coil) is due to the reverse voltage generated by the collapsing magnetic field around the coil when the current is interrupted by opening the contact. This high voltage can cause arcing across the contacts or spike on electronic components which leads to their demise.



For a DC powered circuit, you could add a snubber circuit (series resistor and capacitor) in parallel with both the diode and relay coil as shown in **Figure 1**. Just make sure the resistor and capacitor can handle the power discharged from the diode coil when the contact opens. For an AC circuit, you can use a properly sized Metal Oxide Varistor (MOV) to snub the spike.

Tim Brown
via email

#2 When controlling inductive loads, I like to use an N-channel MOSFET such as a 2N7000, BS107, or equivalent to control the device in addition to the flyback diode. This provides further isolation and reduces the current requirement of the Arduino just in case you want to control more that a couple of relays.

Gene Sellier
Fairhope, AL

[#1162 - January 2016]
Capacitor Forming

I'm trying to resurrect an old

Halicrafter's communications receiver from at least the '50s. I'm planning to replace the electrolytic capacitors in the power supply with capacitors I salvaged from a more recent TV set.

However, I've read that electrolytic capacitors — once formed at a certain voltage — can take months, if not years, to reform at a new voltage. Until then, the capacitor value can be significantly off from what's on the label.

Can anyone shed some light?

#1 The thin aluminum oxide dielectric (energy-storing) layer in electrolytic capacitors is formed on the specially-treated anode metal, and the electrolyte contacts the outer can (on early parts) or the cathode foil (on later "dry" parts). When radios have been unpowered for a very long time, electrolytic capacitors tend to lose their dielectric layer and their voltage rating, but not usually much capacitance. They may destroy themselves and other parts when re-powered, unless slow-start techniques are used to renew their dielectric.

When restarting long-idle sets, it's best to use a variable-voltage (variac) transformer to increase voltage slowly over several days. Note that variacs are usually not isolated. An alternative method is to power up with an incandescent lamp socket in series with the AC line, starting with a 60 watt bulb and increasing the wattage gradually. Turn the set off occasionally and check for excessive heating.

I have repaired many radios using higher voltage TV capacitors, and have never known one to take more than a week, unless it was defective. If the TV parts aren't ancient, they may still be sufficiently formed for your new (lower) voltage, but for safety, follow the above procedures.

There is a strong shock hazard posed by AC/DC sets. Many early versions of these sets had all B(-) connections grounded to the chassis, including one side of the switched line cord, no matter which way it's plugged in. Missing or wrong size screws or knobs or rotting rubber mounting grommets could make outer metal cabinets lethal. Some otherwise well-built later radios, such as the Hallicrafters SX-41, had a live chassis. Most later sets used an isolated internal ground system to minimize the hazard. It's safest to work on AC-DC sets using an isolation transformer — especially if you'll be connecting any AC-powered test gear. You can make your own using back-to-back filament or power transformers of appropriate power. Good luck with your repair!

Stanley Pitman
via email

#2 Let me start with a rule of thumb: Electrolytic caps drop some value if formed to higher voltages (within working range). This is because of thickening of the barrier layer, BUT this can vary. That's why most are rated plus or minus 20%. They generally only need to be big enough (power filter or bypass). Don't expect much value drift moving to a lower voltage. Other concerns are with internal resistance (can screw with bypass performance in audio stages, and heats cap at high duty cycle).

Nick Vitinaros
Grand Blanc, MI

SERVOCITY.com

WE HAVE THE PARTS FOR YOUR IDEAS

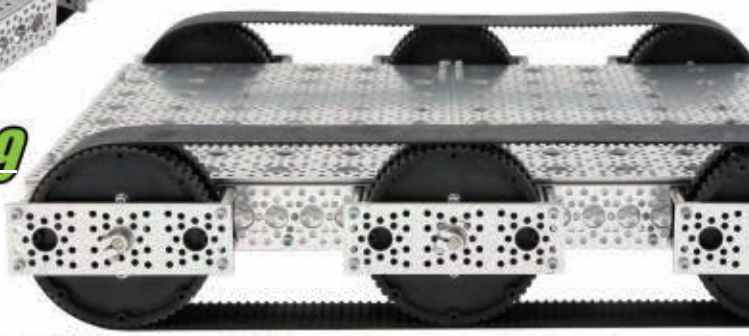
NEW

TRACKED ROBOT PLATFORM

AGENT 390

A smooth, ball bearing supported, drive track system with high maneuverability and omnidirectional control

just $389.99 (motors included!)



Capable of carrying over 50 lbs!

Easily re-configurable

18"L x 16.42"W | Powerful 313rpm Gearmotors | Easy Assembly

NEW

HiTEC D-SERIES SERVOS

- Smoother operation
- More precise
- 25T spline
- Digital & Programmable

D625MW $39.99
Max. Torque: 138.87 oz-in
Max. Speed: 0.13 sec/60°

D645MW $39.99
Max. Torque: 180.1 oz-in
Max. Speed: 0.17 sec/60°

D940TW $147.99
Max. Torque: 229.14 oz-in
Max. Speed: 0.07 sec/60°

D945TW $147.99
Max. Torque: 319.40 oz-in
Max. Speed: 0.10 sec/60°

D950TW $147.99
Max. Torque: 486.05 oz-in
Max. Speed: 0.14 sec/60°

D980TW $169.99
Max. Torque: 611.0 oz-in
Max. Speed: 0.17 sec/60°

M = Metal Gears T= Titanium Gears W= Wide Voltage (4.8 ~ 8.4V)

MORE NEW PRODUCTS

6-12V HD Precision Planetary Gearmotors w/ Encoders *$59.99*

X2 AC Pro Multi-Charger w/ Soldering Station *$199.99*

Futaba T6K 6-channel 2.4GHz Transmitter *$199.99*

Mantis™ 4WD & 6WD Off-Road Rovers *$209.99 - $479.99*

Actobotics® Dual Servo Driver *$79.99*

Flat Aluminum Channel Plates (3 sizes) *$8.99 - $16.99*

3.8" High Traction Paddle Tires *$19.99 / pair*

Mini and Heavy Duty Toggle Switches *$4.99 - $16.99*

90° Tube Clamps 3/8" thru 1" Bore *$7.99 - $8.99*

Mini and Large Bread-board Snap Mounts *$2.99 - $3.49*

Gear Drive Pan Kit A (4.2 : 1 Ratio) *$79.99*

Servo Drive Gear Rack Linear Slide Kits *$89.99 - $99.99*

sales@servocity.com | 620.221.0123 | www.servocity.com

Learn more

JOIN THE EVOLUTION.

Get "mobile smart" in 3 easy steps:

1. Get your ***AIR for Wiced Smart*** dev kit at your distributor of choice. (See our website for a current list.)

2. Develop your wireless link and basic app using our exclusive ***Atmosphere*** development tool.

3. With our AIR for Wiced Smart module on board, proceed in record time to a prototype and final, mobile-app development!

AIR Anaren Integrated Radio for WICED by BROADCOM

Evolve to app-based control with AIR for Wiced Smart!

If you're ready to evolve from fixed control panels populated with dials, buttons, keypads, and LCD displays to mobile-app based control of your embedded product – check out Anaren's AIR for Wiced Smart module, featuring Broadcom's Wiced Smart *Bluetooth*® chip (BCM20737). Not only does our small-footprint, SMT, and pre-certified all-in-one module save you the time, effort, and trouble of designing your own radio... It's supported by our industry-exclusive *Atmosphere* development ecosystem that lets you develop your basic embedded code and app code in one, easy-to-use development tool – for a far speedier product development cycle and time-to-market. **Follow the steps at left to join the evolution, right now!**

www.anaren.com/AIRforWiced
800-411-6596
In Europe: 44-2392-232392

Anaren®
What'll we think of next?™

Bluetooth®