• E-Mail: mike@pwpublishing.ltd.uk

A Look at the Arduino

Mike Richards G4WNC introduces the Arduino boards and explains how to start using them to develop useful applications.

The Raspberry Pi has been dominating the computing side of radio recently so I think it's time to take a look at the Arduino platform to see how it can contribute to the modern shack.

Background

The Arduino arrived on the scene back in 2005 as a cheap and easy way for enthusiasts to take advantage of embedded computing. In this context, embedded computing is simply the inclusion of a microcontroller in a project. SmartProjects in Italy produced the original boards and it seems that the Arduino name comes from the bar in Ivera, Italy where the developers met!

In addition to using a microcontroller, the Arduino boards have the power supply, clock oscillator and other components necessary to run the processor. To make the boards as versatile as possible, the Arduino features plenty of digital input and output pins along with a few analogue inputs. These inputs and outputs are pre-connected to the microcontroller pins and can be activated by software. On its own, the Arduino board does nothing so the user has to tell it how to react through programming software that's stored in the microcontroller's flash memory. The flash memory is non-volatile so the code is retained when the power is removed.

Arduino vs. Raspberry Pi

Given the very low prices of the Model A+ Raspberry Pi, you may be wondering why anyone would buy an Arduino board. Although both are powerful development platforms, their operation and use are very different. The Raspberry Pi operates in a multitasking operating system (Linux). This type of operating system can be likened to spinning plates and relies on having a fast processing core that has enough capacity to look after several programs (spinning plates) at the same time. If you ask it to do too much (spin more plates), then the system will slow down and plates will break because the processor didn't get there in time. The Arduino, on the other hand, uses a much slower processor (typically 16MHz) that is dedicated to a single task. Using the spinning plate example, the Arduino would be excellent at spinning a single plate at a precise speed. This is because the Arduino uses what's called real-time processing. This means that the microcontroller is poised waiting for the next command, which, when received, will be carried out immediately. Conversely, in a multitasking system, a request for action may have to wait until the processor is free, in which time the plates may have crashed to the floor!

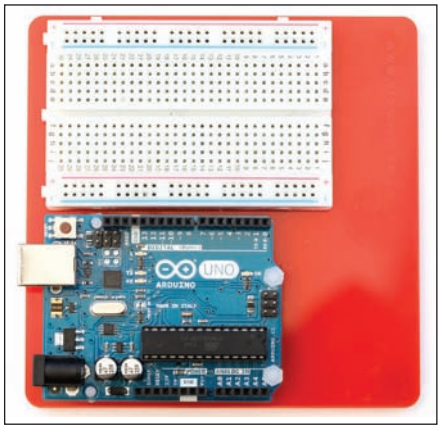


Fig. 1: Arduino R3 UNO mounted on a prototyping platform from Proto-Pic.

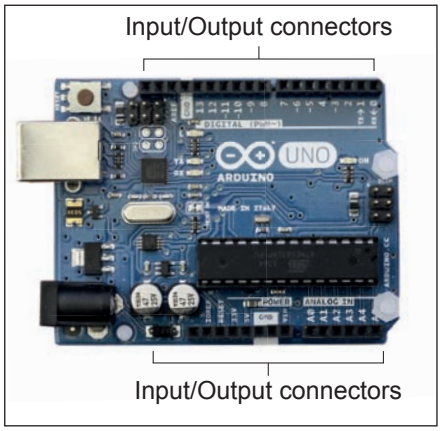


Fig. 2: Arduino Input/Output connectors.

Arduino Boards

You can be forgiven for being confused if you start looking around for Arduino boards because there are so many on the market. The official range can be seen on the main Arduino website, below. The basic board and the best place to start is with the Arduino Uno revision 3, known as UNO-R3. This uses the popular ATmega328 microcontroller running with a 16MHz clock and has plenty of power for most shack-based applications. The microcontroller uses a 28-pin DIL package and is mounted in a socket so it is very easy to replace if damaged. When developing Arduino projects, I usually start with the Uno mounted on a prototyping plate such as the one from Proto-Pic shown in **Fig. 1**. When the project is fully tested, I transfer the software to one of the smaller and cheaper boards such as the Arduino Mini, Nano or their clones.
http://arduino.cc/en/Main/Products

Shields

While the Arduino boards are very versatile, they can be further enhanced using add-on boards that plug in to the I/O connectors that run along the edge of the board, **Fig. 2**. In the world of Arduino, these boards are called shields. There is a huge range of shields available from motor controllers to Wi-Fi and Ethernet boards. If you wanted to use the Arduino to make your own antenna controller, for example, you could add the excellent motor controller shield, **Fig. 3**, that can directly drive a pair of 2A DC motors. The motor controller features two full bridge drivers so you can easily control both the speed and direction of each motor.

Compatible Confusion

You may have seen lots of development boards on the market that claim Arduino compatibility but are clearly not Arduino boards. In most cases, they refer to boards that use the same I/O connectors as Arduino shields. This means that you can use shields designed for the Arduino on these development boards. The Arduino pin layout has become something of an industry

standard so manufacturers launching new development boards often configure them with the Arduino pin layout in order to take advantage of the many existing shields out there. There are also lots of Arduino clone boards that provide a cheap alternative to the official Arduino products. If you want to check out an Arduino clone board, there is a register on the Arduino community on the website below.

http://playground.arduino.cc/Main/SimilarBoards

Using the Arduino

To realise the potential of the Arduino board, you first need to install some software to tell it what to do. Programming Arduino boards is very simple and can be done directly from your Windows, Linux or Macintosh computer. The first step is to download the Arduino software, which is available free of charge from the website below. While the software is free, I recommend a contribution to help support future development.

http://arduino.cc/en/Main/Software

The Arduino software provides what's known as an Integrated Development Environment (IDE). As the name suggests, everything you need to write and run Arduino software can be found in the IDE. When you first run the Arduino IDE, you will see a screen layout similar to that shown in **Fig. 4**. The IDE supports C and C++ programming and the initial programming template (called a sketch) shows just two functions. The first is called setup and contains the setup activities that you want to perform when the program starts. The second function is called loop and is where you put your main code that will be executed repeatedly. While this may seem daunting if you're new to programming, in practice Arduino programming can be very simple.

In addition to the excellent tutorials in the Learning section of the Arduino site, you'll also find lots of free tutorials available via YouTube and Google. If you like to learn by example, there is a huge range available via the File – Examples menu in the IDE. Let's take a look at the Blink example that you'll find in Examples – 01.Basics. The first section is a text description of the program where /* marks the start of the text and */ marks the end. The /* and */ markers are used to indicate multiline comments and I recommend using them in your own programs because it makes life so much easier when you need to make some changes to the software. The next section starts with two forward slashes that mark the start of a single line comment. I also recommend that you use these comments

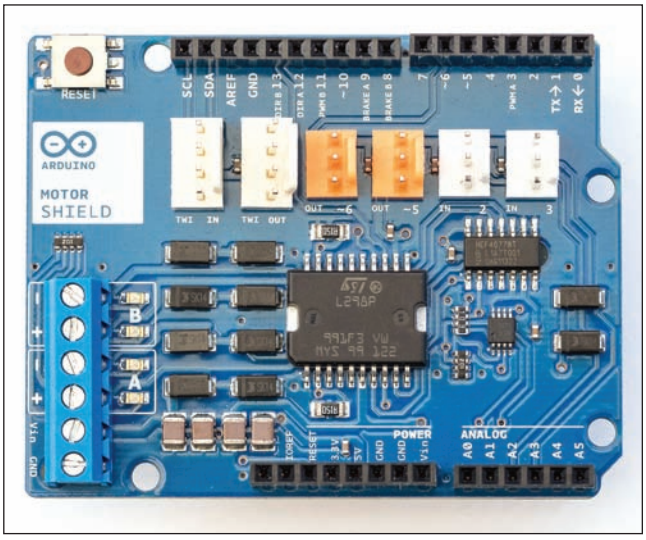


Fig. 3: Arduino Motor Shield.

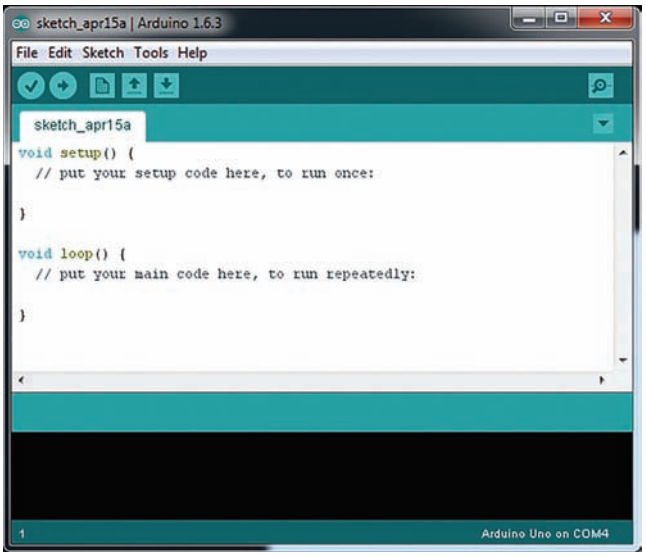


Fig. 4: Arduino IDE empty sketch.

to explain each function in your program. You will see the '***setup***' function contains a single command: ***pinMode(13, OUTPUT);***

This command tells the processor to configure Pin 13 as an output. On a standard Uno board, pin 13 has an LED connected so this command effectively puts the LED under processor control. If we move down to the '***loop***' part of the sketch, you will see that there are just four lines of code with an explanatory note at the end of each line. You should be able to see that the sketch changes the state of Pin 13 and then waits for one second before changing the state again. The result will be a flashing LED on the UNO board.

Programming the UNO

Let's now take a look at how we get the sketch from your computer to the UNO board. This is one area that can be tricky with some systems but not with the Arduino. The secret to the simple upload is the provision of bootloader software in the ATmega328 chip. This software is preinstalled on the Arduino ATmega328 chip and causes it to go into special mode where it can handle the upload of software via the USB port.

The first step is to connect the UNO USB port to your computer. As soon as it's recognised, you should see drivers being installed. On a Windows system, you will also see the port allocated to the UNO. Moving back to the Arduino IDE, use the Tools menu to set the board to UNO and the port to the one allocated by Windows. You are now ready to send the example blink program to the board. At the top of the Arduino IDE, you will see a right-arrow, which is the Upload button. Press this and your software will be transferred to the UNO. If all is well, you should see the LED flashing slowly. To see how easy it is to adapt and develop software, go back to the Arduino IDE and change delay(1000) to delay(250) on both lines. Press the Upload button again and you should find that the LED is flashing more quickly. The facility to change and test the software so easily is one of the features that makes the Arduino so popular.

Libraries

Libraries are blocks of prewritten code that go a long way to making Arduino programming easier. A good example is to be found when you want to use

a display to show some output from the Arduino. The most popular display devices are the cheap two- or four-line LCD units that are available for around £5 on eBay, **Fig. 5.** While they are very attractive, they do require quite a bit of programming to get readable text to appear on the screen. This is a classic case for a library to do all the heavy lifting. In my projects, I often use a four-line 20-character LCD from eBay and combine it with the i2c/SPI LCD Backpack available from Adafruit for under £10, **Fig. 6**. The end result is a useful, four-line, LCD display that only requires four wires to connect to the Arduino and costs around £15. Adafruit have very kindly produced an Arduino library for this combination that makes displaying messages very easy. Here's the code to start the display and then print a message:

```
LiquidCrystal lcd(0);
lcd.begin(20, 4); // initialise the LCD and let the library code know the size
lcd.print("Hello"); // send the text to the display
lcd.setBacklight(HIGH); //turn the backlight on
```

As you can see, the code is very readable and easy to understand. Because the Arduino system is so well established, there are libraries available for most of the more complex tasks. In many cases, you can build the bulk of your application simply by linking libraries together.

Alternative IDEs

A comparatively new programming tool that's growing in popularity is codebender, **Fig. 7**. This is a free, web-based IDE that can be used to program many types of development board including the Arduino series. Go to the codebender site (below) and click the Try Now link at the bottom of the page to run a quick online tutorial. If you like it, the next stage is to register at which point you will be allocated your own account and a codebender plug-in for your browser will be installed. Once you're registered, you can use the search facility to find interesting projects that you can download to your Arduino board or maybe use as the basis for your own project.

http://codebender.cc

If you like the idea of visual programming, then Scratch for Arduino may be for you, **Fig. 8**. Scratch is a simple visual programming language that's been designed by the Massachusetts Institute of Technology (MIT) in the USA to enable youngsters to develop interactive stories, games and animations. It's proved very popular with schools on the Raspberry Pi and there is now a variant (S4A) developed for use with the Arduino. You can download the software free of charge but as always, I suggest you make a donation to encourage the developers. I'll cover the use of S4A in a bit more detail next time.

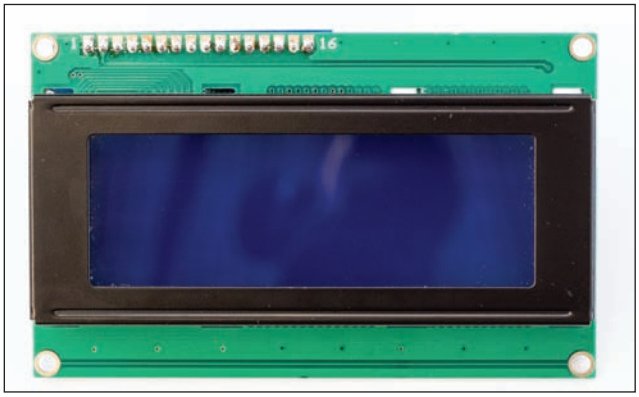


Fig. 5: A four-line 20-character LCD purchased on eBay.

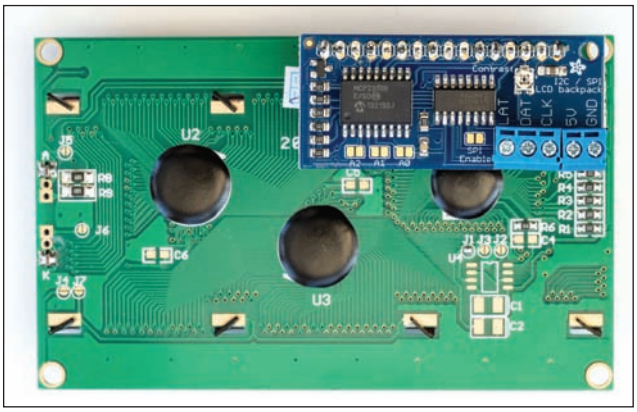


Fig. 6: The Adafruit I2C/SPI LCD Backpack mounted on the display.

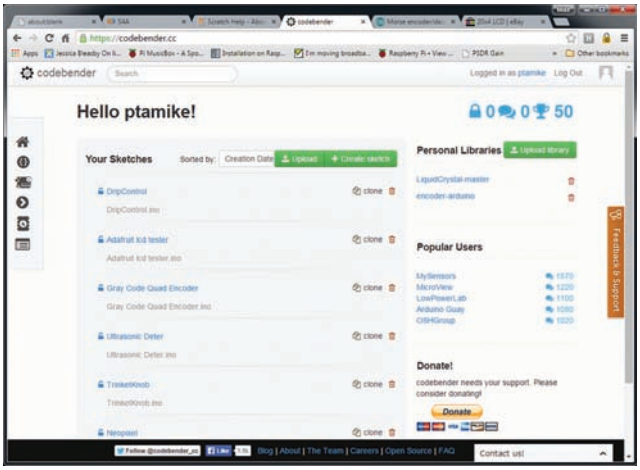


Fig. 7: The codebender logged-in screen.

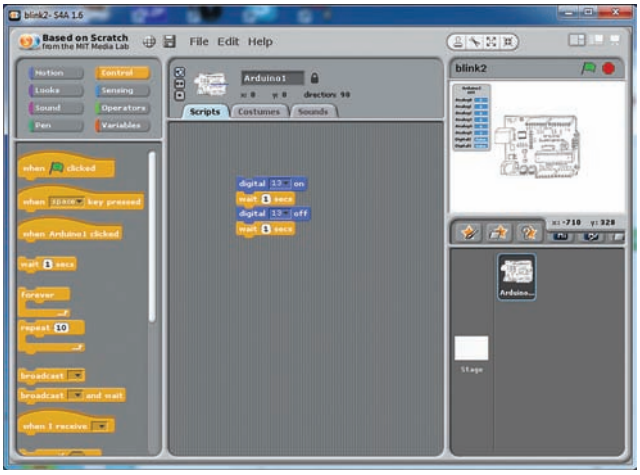


Fig. 8: Scratch for Arduino S4A showing a basic blink program.

• E-Mail: mike@pwpublishing.ltd.uk

An Arduino Project

Mike Richards continues his look at the Arduino platform by describing how to program a simple digital voltmeter.

This month I'll continue with my look at the Arduino with some simple guidance on getting started. One of the features of the Arduino that makes it particularly attractive is the tremendous support that's available. There are countless tutorials on YouTube and on a variety of other sites including Arduino's home site. If you encounter a specific problem when programming, there are plenty of forums with experts on hand to help you out. The official Arduino forums at the URL below are a good place to start.

http://forum.arduino.cc

These forums cover most aspects of using the Arduino and there are many years of Arduino experience on tap. The best way to make use of the forums is to start by searching on your specific problem. For example, if you're trying to run a stepper motor, try searching for 'stepper motor driver'. I did that and one of the first few results was a very useful tutorial on stepper motor basics. To see if there's a pre-written library available, search for 'stepper motor library'. I also tried that search and discovered there was a wide range of stepper motor routines to hand. These covered all manner of common motor driver chips, making it very easy to incorporate a stepper motor into a project. The availability of such extensive support reduces the coding task down to little more than some decision-making and configuration to make use of these libraries.

Simple Project

To show you how easy it can be to program the Arduino, I'll take you through the construction and programming of a simple digital voltmeter. I'll base this project on the Arduino Uno because this is probably the most common Arduino and certainly the best one to start with if you're new to the world of Arduinos.

Our first requirement is to be able to measure a DC voltage with the Arduino. Because the Arduino processor, like all computers, can only deal in numbers, we need to convert the analogue voltage we are measuring into a digital format. This is done using an analogue to digital converter (ADC). The technique used for the conversion is to measure the voltage on the input pin and change the result into binary format number as illustrated in **Fig. 1**. The speed and frequency capabilities of the measurement process are important because we want to be able to track changes in the measured voltage. For DC measurements, a capability of several thousand measurements per second would be more than adequate.

In many systems, the analogue to digital conversion requires an external ADC but the Uno has a six-channel ADC built in so we can complete the analogue to digital conversion without any additional hardware. In the Uno, the analogue input ports store the measurement results in a register, which is rather like a special memory. To measure the voltage on any of the six analogue pins, we simply have to issue the command to read the appropriate pin. The command 'analogueRead(A0);' will read the voltage value of analogue pin A0. Before we can make some practical use of this, we need to consider the measurement range of the Uno's ADCs. The Uno's converters employ a 10-bit binary numbering scheme, which means that there are 1024 discrete steps between 0V and the full scale reading, **Fig. 2**. As with an analogue voltmeter using a moving coil meter, we need to know the sensitivity of the instrument, in other words what is its full-scale voltage. Whereas a moving coil meter will be fixed, we have some choices available with our Arduino-based instrument. In its default configuration, the full-scale voltage will be the same as the Arduino's 5V supply. While this sounds simple enough, there are a few issues. If you are powering the Uno via a USB port, you may well find that the 5V rail is anywhere between about 4.8 and 5.2V and, what's worse, it may vary due to the changing demands of other devices on the same USB port. With an inaccurate and unstable reference voltage, all our measurements will be compromised. A simple solution is to power the Uno using an external 7 to 12V DC power unit. This improves the situation because the application of external power causes the Uno automatically to disconnect the USB supply and to use its on-board regulator as a power source. The

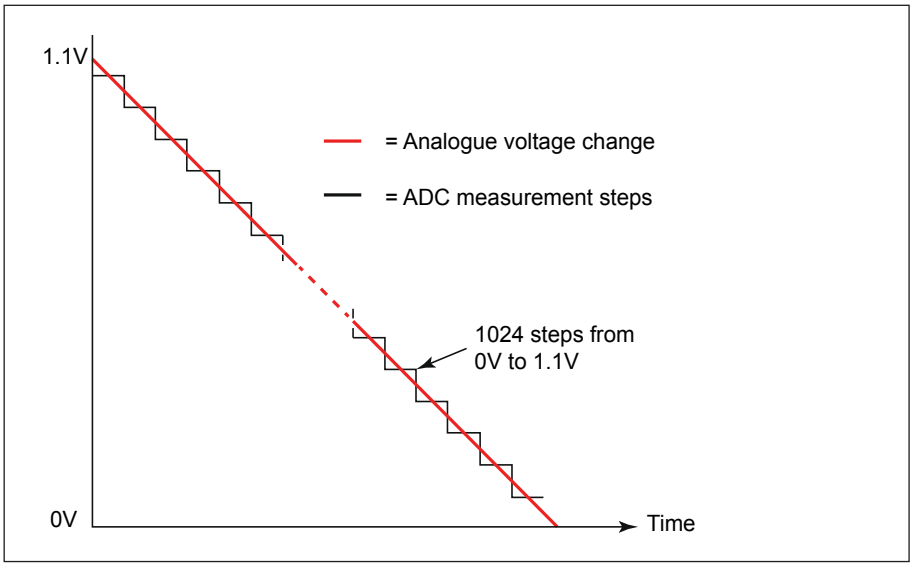


Fig. 1: Illustration of the analogue to digital conversion process.

Decimal weighting given to each digit

10 binary digits

Weighting >	512	256	128	64	32	16	8	4	2	1
99	0	0	0	1	1	0	0	0	1	1
25	0	0	0	0	0	1	1	0	0	1
1000	1	1	1	1	1	0	1	0	0	0
452	0	1	1	1	0	0	0	1	0	0
1023	1	1	1	1	1	1	1	1	1	1
77	0	0	0	1	0	0	1	1	0	1
296	0	1	0	0	1	0	1	0	0	0

Fig. 2: 10-bit binary numbering scheme showing examples.

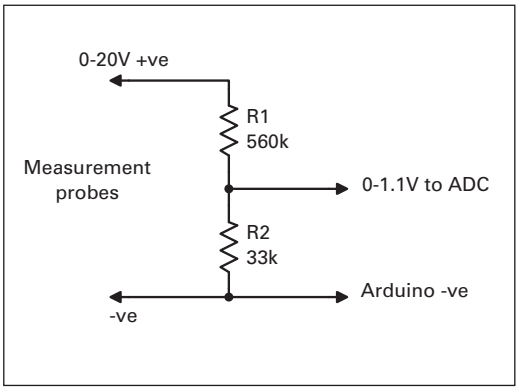


Fig. 3: A simple voltage divider chain.

```
void setup() {
  // Use the internal 1.1V ADC reference
  analogReference(INTERNAL);

  // Initialise the serial port
  Serial.begin(9600);

}

void loop() {
  // Read the ADC value and store the result in adcValue
  int adcValue = analogRead(A0);

  // Convert the ADC value to the true value
  float trueValue = adcValue/51.15;

  // Print the result via the serial port
  Serial.println(trueValue);

}
```

Fig. 4: The complete code for the simple digital voltmeter.

on-board regulator provides a much more stable and accurate 5V reference than a typical USB port. A second alternative is to make use of the internal 1.1V reference supplied by the ATmega328 processor. When using this dedicated reference, the full-scale range is 1.1V. It's also possible to supply your own reference voltage for the ADC conversion. This must be between 1.1 and 5V and should be connected to the AREF pin of the Uno. For our example, we will use the internal 1.1V reference because this allows you to power the Uno over the USB connection without compromising the accuracy. To switch to the internal reference, we use the command: 'analogReference(INTERNAL);'

Practical Measurement

So far we have settled on using the internal 1.1V reference and have six 10-bit ADCs at our disposal so we might ask what results will we get and how we can use them. Let's begin by using just one ADC channel. From earlier, you will note that the 10-bit resolution of the Uno provides 1024 steps from 0V to our full-scale voltage that we've set at 1.1V. From that, we can see that each step in the ADC output represents 1.1V/1024, which is 1.074mV. Therefore, to convert the ADC's result to millivolts, we need to multiply the ADC reading by 1.074.

We now have a reasonably accurate millivoltmeter that can measure from just over 1mV to 1.1V. To make this into a practical measurement system, we need to add some range-dividing resistors. A useful range for use around the shack would be 0-20V so we need a divider chain that will reduce 20V to the full-scale of our Arduino ADC, which is 1.1V. After playing with an online calculator, it looks as though 560kΩ and 33kΩ combined as shown in **Fig. 3** provides a simple solution that will reduce 20V down to 1.11V and provide a measurement input resistance of around 600kΩ. You can, of course, build a more accurate divider by combining resistors in series and parallel but I'll stick with the simple solution for this exercise.

Start Programming

Programming the Arduino is very simple and we'll approach it in easy steps. For this exercise, I'll assume that you've installed and tested the IDE as per last month's *Data Modes*. If you don't have that to hand, you can find the official 'Getting Started' tutorial at the URL below.

http://goo.gl/n3dR7i

Start the Arduino IDE and you should see the default sketch template with setup and loop sections. The first program that we'll write will repeatedly read the voltage from the analogue input A0 and print the result on your computer screen. As discussed earlier, we will be using the Uno's internal 1.1V reference so we need to add a line of code in the setup section to connect the reference:

analogReference(INTERNAL);

Because we want to display the measurement result on the computer, we also need to prepare the serial connection with the command:

Serial.begin(9600);

This prepares the serial port to operate at 9600 baud.

That completes the setup so we can move on to the main program. As the name of this section implies (loop), this part of the program will be repeated at full speed until the power is removed. In this section we want to achieve the following:

1. Read the voltage from the ADC.
2. Convert it to take account of the voltage divider and ADC steps.
3. Print the result on the computer screen.

To perform step 1, we need to read

the value from the ADC and store it in memory so that we can convert it. We do this by using what's known as a variable. By creating a variable, we reserve some space in the microcontroller's memory to store the ADC value. For this to work efficiently, we have to tell the system the type of variable we need. In this case, we need an integer because the result from the ADC will always be an integer. To simplify the programming, we can declare the variable and store the ADC reading in a single line of code as below:

int adcValue = analogRead(A0);

The next step is to convert the ADC reading to the voltage applied to the resistive divider. If everything works as planned, applying 20V DC to the divider should give an ADC reading of 1023. To convert that back to 20V, we would need to divide the ADC value by 51.15. To do this, we create another variable but this time we need a floating-point number:

float trueValue = adcValue/51.15;

When giving names to variables, you should choose a name that describes the value and use what's known as camel case. This is where words are joined without spaces but a capital letter is used to denote the start of a new word. In Arduino programming, it's common practice to start with a lower case as in this example. It's also good practice to add plenty of comments because it makes life so much easier when you revisit the code at a later date. Code that appears logical now can be a nightmare later if not properly commented.

All that remains is to send the result over the USB serial connection to the computer:

Serial.println(trueValue);

The Serial.println part sends the variable 'trueValue' over the serial port and adds a new line so that the result will scroll on the screen.

That completes the programming and I've shown the complete sketch in **Fig. 4**.

Checking and Running

Once you've finished typing and entering the code, you can check it by clicking the tick icon at the top left of the Arduino IDE, **Fig. 5**. When you click this, the IDE will run through the program checking that the syntax is correct. If there are any errors, they will be listed in the lower panel. The most common mistake in Arduino programming is forgetting to add the semicolon at the end of each line. If the code is correct, you can press the right arrow button on the IDE to check and transfer the code to the Uno. You will see a green process bar towards the bottom of the screen that will be followed by a message 'Done uploading' and a summary of the memory usage will be shown in the status panel. At this point, your Uno is now working as a digital voltmeter passing results over the serial port. To see the results, you can use the serial monitor that's included with the IDE. Go to the Tools menu and Serial Monitor. A new window will open that should show the results scrolling down the screen, **Fig. 6**.

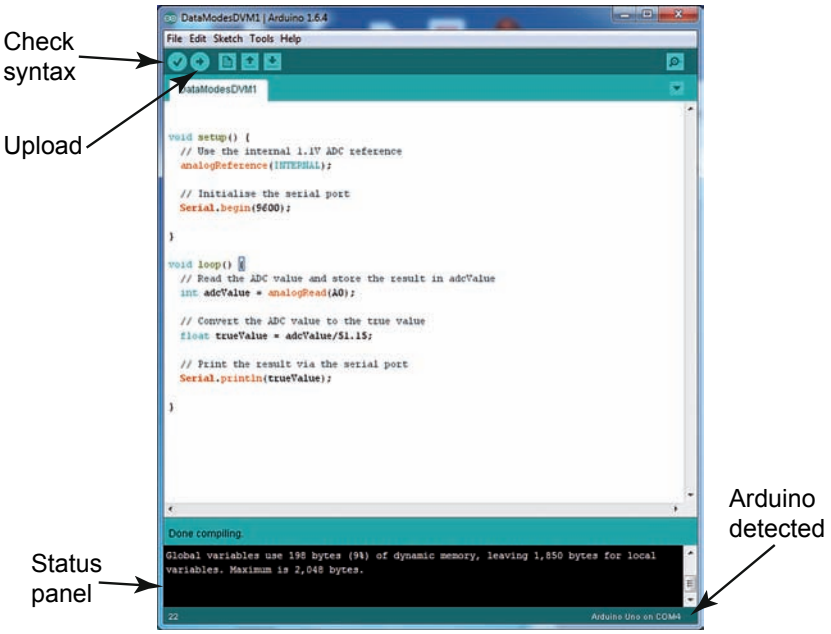


Fig. 5: The Arduino IDE.

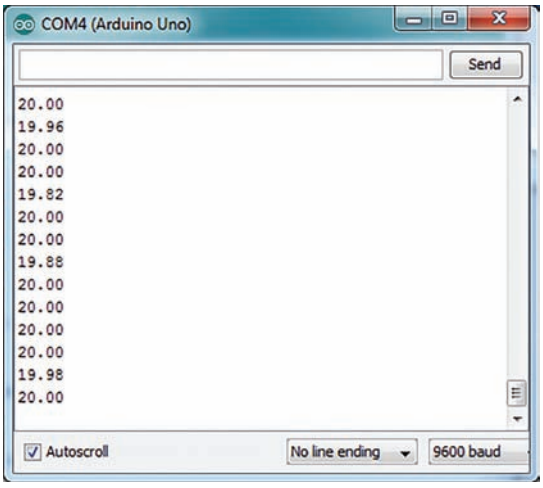


Fig. 6: The computer serial screen showing the measurement results.

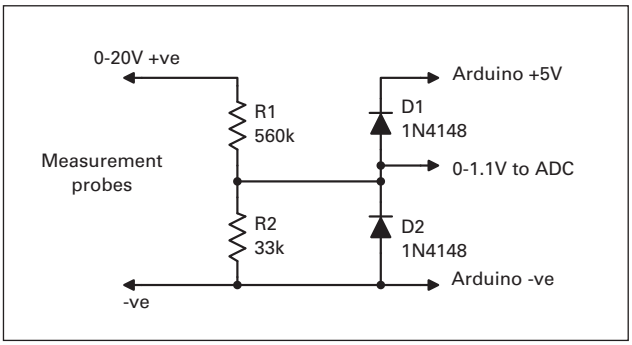


Fig. 7: The voltage divider enhanced to provide extra protection with clamp diodes.

Protecting Your Arduino

When you start connecting your Arduino to the outside world, you expose it to all manner of potential risks of which the greatest is probably static discharge. The safest way to protect the Arduino's input pins is to add a couple of clamp diodes. 1N4148 diodes are fine for this and should be connected as shown in **Fig. 7**. This configuration limits the voltage on the input pin to −0.6V to +Vcc + 0.6V. The ATmega328 chip does include low current clamp diodes so the additional clamp diodes may be a case of 'belt and braces'.

Next month, I'll take this program a step further to make a simple, wide-range RF power meter.

• E-Mail: mike@pwpublishing.ltd.uk

An Arduino Power Meter

Mike Richards G4WNC concludes his Arduino coverage with a power meter project.

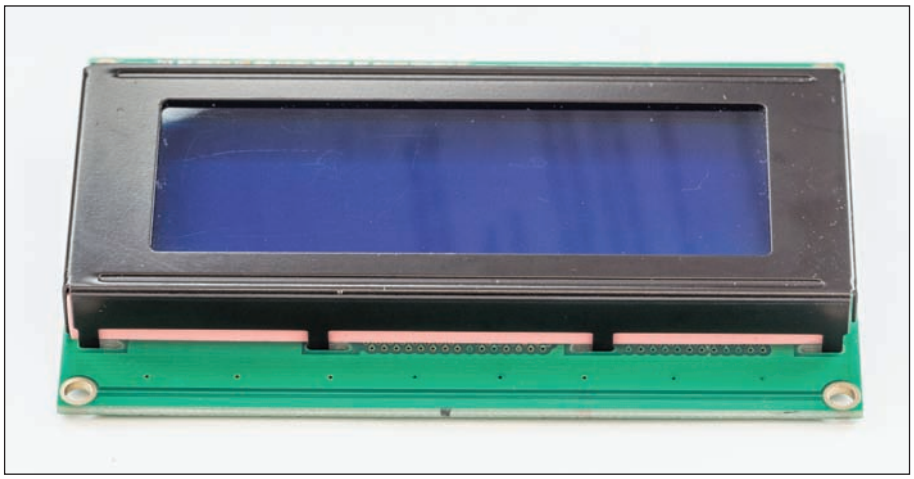


Fig. 1: Four-line 20-character LCD display.

This month I'm concluding my coverage of the Arduino microcontroller with a simple project for a DC to 500MHz, 1nW to 100W, in-line, power meter. I know that sounds like a tall order but it's surprisingly easy, thanks to a wonderful chip from Analog Devices.

Show Time

In the first part, I showed you how to make a simple digital voltmeter using an Arduino Uno with the results displayed on your PC. While that was a useful introductory project, displaying the results via a serial print command on your computer is a bit clumsy and impractical.

A better solution would be to use a dedicated display attached to the Uno. Probably the easiest and cheapest way to do that is to use one of the multi-line LCD modules that are available from many electronic suppliers. If you're just running a simple voltmeter, the two-line, 16 or 20 character units would probably do the trick. However, for the power meter project, I need more lines because I want to show the output voltage from the measurement device along with the RF power in dBm and in watts. I therefore chose a four-line 20-character unit, **Fig. 1**. On eBay these are available for as little as £3-£6 or about £12 from UK suppliers. The important point is to make sure that the unit is Hitachi HD44780 compatible. This is by far the most common standard so you shouldn't have any problems finding one.

The basic connection method is shown in **Fig. 2** and uses nine connection wires. An alternative and simpler connection system is to use an Adafruit LCD Backpack. These neat devices plug into the back of the LCD unit and use only two data connections plus the power supply and employ the I2C signalling protocol to communicate between the Uno and the LCD. I've shown the diagram in **Fig. 3**.

LCD Library

Last month I introduced the use of code libraries as a way to minimise the code we have to write. The LCD panels are a classic instance of where a code library can simplify the programming and reduce the amount of code you need to write. In this case, the library does all the heavy lifting and all we have to do is to tell it the text or results that we want it to print.

The Arduino LiquidCrystal library is included in the standard IDE and is fine if you're using the multi-line connection systems shown in **Fig. 2**. However, if you're using the Adafruit LCD backpack, you will

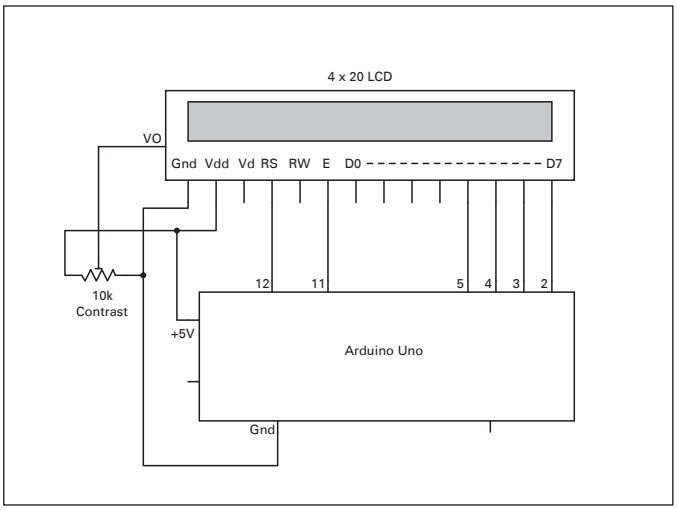


Fig. 2: Connecting an LCD to the Uno.

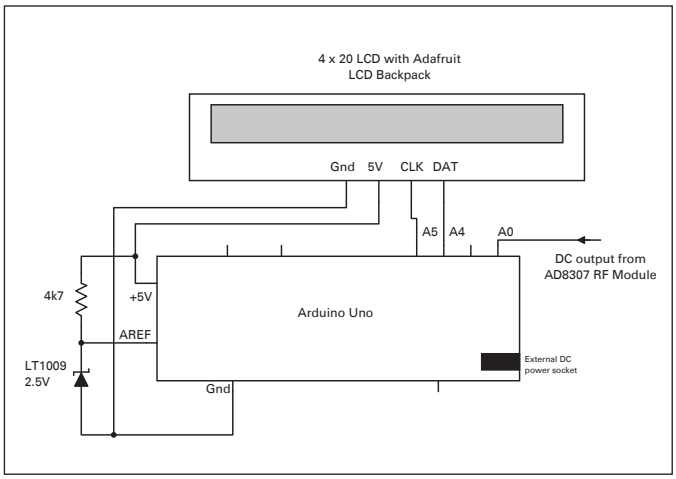


Fig. 3: Alternative and simpler connections using the Adafruit LCD Backpack.

Fig. 4: Power meter display.

need to download an updated version of the LiquidCrystal library from the Adafruit github site, below.

https://github.com/adafruit/LiquidCrystal

Use the option to download a zip file to your hard drive. Next you open the Arduino IDE and choose Sketch – Include Library – Add Zip Library. Navigate to the downloaded zip file and click 'OK' to install it. You can then re-start the Arduino IDE and the new library will be available for use. The Adafruit version of the LiquidCrystal library is fully compatible with the standard library but adds the code to support the backpack.

To use the LCD library in our program, all we have to do is add the following line at the beginning of the sketch:

#include "LiquidCrystal.h"

We also need to use the Wire library to support I2C communications as follows:

#include "Wire.h"

The next task is to create an instance of the display so that we can start using it. To do this, we add another line with the name of the library followed by the name of our instance like this:

LiquidCrystal lcd(0);

In this case, the (0) tells the library that we're using an I2C connection. If you're using the seven-wire system shown in Fig. 2, the line changes to:

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

In this case, the numbers in brackets are the Arduino pin numbers of the connecting wires to the display unit. With the preparation work complete, putting text on the screen is completed using the lcd.print command. To print 'hello, we use the line: *lcd.print("Hello");* This will print the text at the current cursor position. While this may be fine for some projects, the RF power meter needs to look like **Fig. 4**. Here you can see that the labels for the measurement units are fixed and we need to be able to display and update the measured values while the meter is operating. The library comes to the rescue here because there is a setCursor(Column, Row) command that enables us to put the cursor anywhere on the display and then to print from that point. For our purposes that requires two code lines, one to position the cursor and the other to display the text as follows:

lcd.setCursor(0, 1);

lcd.print("AD8307 Volts: ");

The first line puts the cursor at the beginning of the second line and the second line displays the text. One point to remember is that the display row and column numbering both start from zero. The text you send to the LCDs is persistent text and will stay on the screen until it is overwritten or the power is removed. This is quite a useful feature because it means that we can print the measurement labels to the screen as part of the setup process so we don't have to keep rewriting them. This results in less work to be done in the main program loop so the measurements will refresh more often. While the display persistence is a very useful feature for our measurement labels, it's not so helpful when displaying the measurement results. As an example, if the power reading changed from 99.9W to 1.0W, the display would show 1.09. This is because the final 9 would not have been overwritten by the new measurement because it is shorter. This is both irritating and leads to erroneous results. The simple solution, used in this project, is to clear the result field by printing a string of spaces just before we display the updated result. Here's an example showing the four lines of code:

lcd.setCursor(15,3); // Position the cursor

lcd.print(" "); // Print spaces to clear the last result

lcd.setCursor(15,3); //Position the cursor

lcd.print(pWatts,1); //Print the measurement result

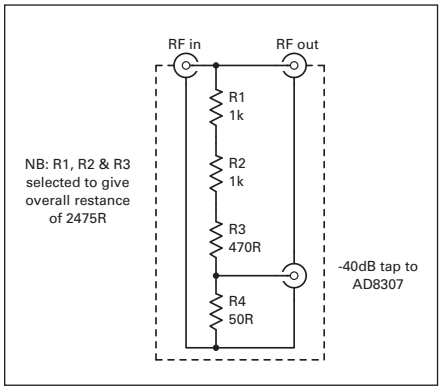


Fig. 5: 40dB loss power tap (see photo).

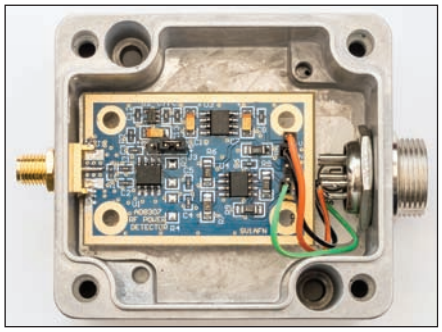
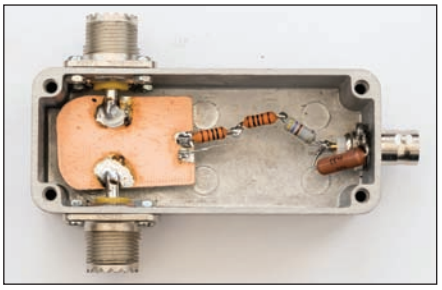


Fig. 6: AD8307 module from SV1AFN.

RF Measurements

Now that we have a practical display system, we can move on to look at the RF measurement system. Analog Devices produce an impressive range of specialist chips for all manner of RF measurements and the device I'm using here is the AD8307 logarithmic amplifier and detector. This



The 40dB tap as in the circuit of Fig. 5.

device has been used in lots of amateur power meter designs, is particularly easy to use and has a very wide measurement range. The AD8307 converts the incoming RF signal into a DC voltage that is directly proportional to the power input in dBm (decibels relative to a milliwatt). The ±0.3dB bandwidth of the AD8307 is DC to 500MHz and the linear input range is an impressive 88dB. The maximum input level to the AD8307 is +10dBm so we need to reduce the incoming power level from the transmitter using a simple resistive divider. The circuit shown in **Fig. 5** provides 40dB attenuation to the measurement feed while providing a through path for the main transmitter feed. The 40dB attenuator reduces 100W (+50dBm) RF to an ideal +10dBm for the instrument. You will find lots of designs for 40dB power taps on the internet and most use the Hammond 1590A die-cast box to make a very neat and robust unit. If you want to be able to use the unit up to 500MHz, you should use BNC, SMA or N-type connectors and follow the construction notes carefully. For HF use, construction is far less critical. While you can build your own PCB for the AD8307, **Makis Katsouris SV1AFN** produces an excellent, ready-built, board for just $22 (about £14), including postage, **Fig. 6**. You can see and order the boards at the website below.

https://goo.gl/5Nk3Gt

As with the power tap, it's important to house the AD8307 module in a good quality screened enclosure to reduce the risk of false readings due to stray RF. In my prototype, I used a Hammond 1550Z (35 x 58 x 64mm) die-cast box for the AD8307 module. The only soldering required on the AD8307 board is a header strip for the output/DC power and an SMA jack for the signal input. The supplied SMA jack is not a panel mount version so it's worth changing that for the longer panel mount type so that it can fit directly to the side panel of the die-cast box. For the DC power supply and measurement output, I used a six-pin microphone connector but with only three connections you could use a stereo jack.

Measurements

The AD8307 converts the input RF power to a proportional DC voltage, so the Arduino's role in this project is to accurately measure

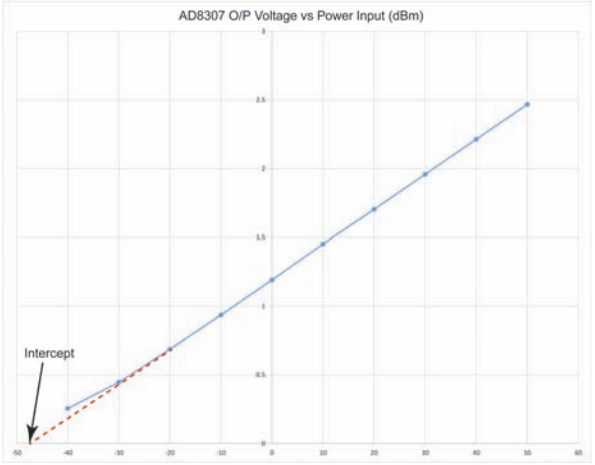


Fig. 7: Plot showing calculation of the intercept point for the AD8307.

the voltage and then to calculate the power in dBm and watts. The input voltage range from the AD8307 board is 200mV for no signal through to just under 2.5V at +10dBm (100W at the 40dB power tap). For this we need to configure the Arduino's analogue to digital converter (ADC) to operate as a 2.5V FSD (full scale deflection) voltmeter. We discussed this last month and in order to get the best resolution from the ADC, we need to use an external 2.5V reference. This is very easy to do because precision shunt regulators are readily available. I used the Texas Instruments LT1009, which has 5mV accuracy and costs just £1.25. As I mentioned earlier, the AD8307 converts the incoming RF signal into a DC voltage that is directly proportional to the power in dBm. In order to use this DC voltage we need some sort of reference. That is achieved by using the manufacturer's intercept point.

If you look at **Fig. 7**, this shows the output graph of the AD8307. You'll note that it is a straight line until the bottom end where it tails off and stops at about 0.2V output. The intercept point is the theoretical point where the straight-line part of the graph would reach 0V output if it was extended – note the dotted line. The nominal intercept for the AD8307 is −84dBm (−44dBm with the power tap). The other important characteristic of the AD8307 output is the slope of the graph, in other words, what voltage change to expect when the signal input is changed. The nominal setting for the AD8307 is 25mV per decibel so a 20dB level change would alter the output voltage by 0.5V (0.025 × 20). The slope is normally converted to a single number, which is the dBm change/voltage change. In this case, the nominal value is 40 (1÷0.025). The intercept and slope is all we need to be able to calculate the power in dBm of any signal within the measurement range. The simple formula is: power (dBm) = (slope × voltage output) − intercept. In our case we have an additional 40dB power tap in the signal path so the intercept changes from −84dBm to −44dBm.

Let's now go through the Arduino calculation process in the sketch. The first task is to read the voltage from the ADC:

value = analogRead(0);

This reads the voltage and stores it in the variable 'value'.

The output from the ADC is just a number between 0 and 1023 so we need to convert that to the true voltage. Here's' the code:

*vout = (value*2.5)/1023;*

Next we need to convert the voltage to dBm using the slope and intercept.

*powerdB = (40*vout)-44;*

Finally we need to convert from dBm to watts:

pWatts = pow(10.0,(powerdB -30)/10.0;

You can see the entire Arduino sketch in **Fig. 8** and I have made the sketch available on Codebender and on my website (below) where you can copy and paste it into your Arduino sketch. The sketch is fully commented to help you to understand the purpose of each code line.

http://goo.gl/BaMuaC

Improving Accuracy

There are three simple things that can be done to improve the overall accuracy of the measurements. The first is to measure the conversion slope of your AD8307. To do this, first apply an RF signal and note the AD8307 voltage reading from the Arduino. Now change the RF signal using an attenuator of known accuracy – 20dB is ideal. Make a note of the revised AD8307 voltage reading. The slope for your device is calculated by dividing the attenuator value by the change in output voltage. You should get a number close to 40.

Next, measure the value of the 2.5V reference voltage with an accurate voltmeter. Finally, you can trim the intercept value by applying an RF signal of known accuracy and trimming the intercept value for the correct power reading from the Arduino. Do note, though, that you should only trim the intercept after you have trimmed the slope and reference voltage. In the final sketch you'll see that I've created dedicated variables for the slope, intercept and refVolts. You can insert your measured values into these variables.

```
/*
This sketch uses the Adafruit i2c/SPI LCD backpack
and shares some code from Adafruit
(http://www.ladyada.net/products/i2cspilcdbackpack/index.html)

Sketch produced by Mike Richards (G4WNC) for publication in the
August issue of Practical Wireless magazine in the UK
www.g4wnc.com

The LCD Backpack connections:
* 5V to Arduino 5V pin
* GND to Arduino GND pin
* CLK to Analog #5
* DAT to Analog #4

* The AD8307 voltage output connects to the Arduino A0 pin

*/

// include the library code:
#include "Wire.h"
#include "LiquidCrystal.h"

// Initialise the variables
int value = 0; // Used to store the raw reading from the ADC
float vout = 0.000; // Holds the true value of the ADC output voltage
float powerdB = 0.00; // Calculated power in dBm
float pWatts = 0.00; // Calculated power in watts
float slope = 39.44; // Slope of the AD8307 log output (Default = 40)
float intercept = 47.0; // 0V intercept point (Default = 44)
float refVolts = 2.499; // Measured value of the 2.5V external reference

// Connect to the LCD via i2c, default address #0 (A0-A2 not
jumpered)
LiquidCrystal lcd(0);

void setup() {
// Start by setting-up the pin configurations
analogReference(EXTERNAL); // Set the Arduino to use an external
reference for the ADC
pinMode(4, OUTPUT); // Digital pin 4 is used to supply power to the
voltage reference
digitalWrite(4, HIGH); // Make sure pin 4 is high
pinMode(A0, INPUT); // Enable the first (A0) input to the ADC
// set up the LCD's number of rows and columns:
lcd.begin(20, 4); // Set 4 lines of 20 characters
// Print the title to the LCD.
lcd.setCursor(0,0); // set the cursor to the top left, line 0 column 0
lcd.print("Arduino Power Meter"); // Message on the top line of the
display
lcd.setBacklight(HIGH); // Turn the backlight on to make the display
visible
//Now print the measurement labels
lcd.setCursor(0,1); // move the cursor to the first position on the
2nd line
lcd.print("AD8307 Volts: "); // print the label
lcd.setCursor(0, 2); // Move the cursor to the start of the 3rd line
lcd.print("Power (dBm): "); // Print the label
lcd.setCursor(0, 3); // Position the cursor
lcd.print("Power (Watts):"); // Print the label
}

void loop() {

value = analogRead(0); //read the ADC and store the result in value
vout = (value*refVolts)/1023; // Convert the ADC result to volts in vout

powerdB = (slope*vout)-intercept; // convert the voltage to dBm in
50 ohms
pWatts = pow(10.0,(powerdB -30)/10.0); // convert dBm to watts
lcd.setCursor(13, 1); //Move the cursor to the 13th position
lcd.print(vout,3); // Display the AD8307 voltage
lcd.setCursor(13, 2); // Position the cursor
lcd.print("      "); // Print spaces to clear the last result
lcd.setCursor(13, 2); // Position the cursor
lcd.print(powerdB,1);// Display the power in dBm
lcd.setCursor(15,3); // Position the cursor
lcd.print("    "); // Print spaces to clear the last result
lcd.setCursor(15,3); //Position the cursor
lcd.print(pWatts,1);// Display the power in watts
}
```

Fig. 8: The complete, fully commented Arduino sketch for the power meter.