CDT-PV Arduino Workshop: Basics and Solar Module Field-Test

Alexander Smith

Outline

Today, you will be learning how to use an Arduino to control electronic circuits and to use both to measure the power output from a silicon solar module. We will start small – getting an LED to blink on and off – and we will end-up with an Arduino connected to a circuit which can measure the current and voltage output of a solar module and save the data to SD card. We will also be using LCDs and transmitting data over radio. We will then take a measurement of the power generated by the module on top of Bath Abbey tomorrow afternoon (if weather permits).

Learning Objectives:

- To appreciate the Arduino ethos and the value of DYI in scientific applications.
- To understand the electronics required for a voltmeter and ammeter.
- Become familiarised with the Arduino language and the IDE.
- To use an Arduino for useful data acquisition.
- To estimate the solar generation potential of Bath Abbey's roof.

Also:

- We will get a visit from Richard Bowman who will demonstrate his Raspberry Pi microscope (with sub-micron positioning control) http://www.np.phy.cam.ac.uk/people/rwb27 at 2pm.
- We will be going onto the roof of Chancellor's Building to see the solar module and Arduino logger which I deployed earlier this week. We will then try to transmit that data back here to East Building.

Introduction

Arduino is an Italian open-source project which aims to bring cheap microcontrollers into the hands of hobbyists and teachers. They develop a series of microcontroller boards, Arduinos, which come in all shapes and sizes and are usually tailored for a specific purpose (hand-held games consoles / interfaces, internet-clients, etc.). Most can be repurposed for anything but some are better at certain things. Today we will be using the Arduino Leonardo to develop a current-voltage data-logger for a Silicon solar module, step-by-step and with a few detours along the way.

The Arduino Uno is the flagship device, which is normally used to learn on. The Leonardo is much the same, but with a microUSB port - so your phone-charger USBs should be able to power and interface them (but I'll provide the cables). There are also a few added extras (such as memory which doesn't wipe when it loses power) which make this attractive for measurement purposes.

Often people put Arduino, Raspberry Pi and STM32s in the same basket. This isn't a bad comparison but there are differences between them. For example, Arduino is the only open-source one, so you could build that yourself from just a few components, giving you greater flexibility in anything you intend to produce. The open-source ethos also means there is a great (and sometimes helpful) community out there who provide quick answers to problems. There are also a lot of community libraries which mean you don't need to program the protocols for interfacing with devices (we'll get to that). In principle all three could do the job, but the Arduino is cheaper and incredibly easy to learn.

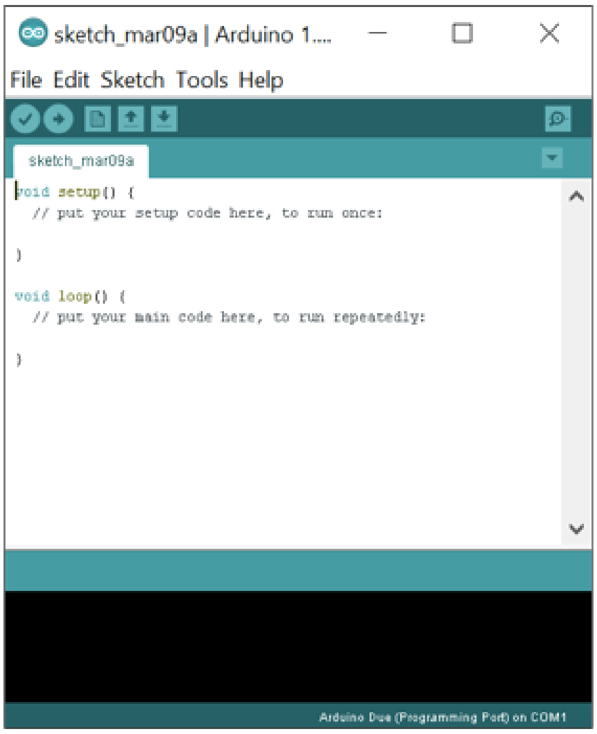
The Raspberry Pi and STM32 are also distinct from Arduino as they are technically small computers – they run an operating system and you can install software on it – however an Arduino is a microcontroller; it repeats a set of tasks and interfaces with the outside world more readily. This is why the Arduino is starting to find application in prototyping, interactive design, and data acquisition.

Arduino IDE

Before you get started you will want to download the Arduino IDE. While in-principle you don't need this, it makes programming the Arduino incredibly easy and comes with some of the built-in libraries we'll be using. As of November 2016 you can also use their web editor. Today we'll use the IDE as we'll need to use it offline.

You can download it from https://www.arduino.cc/en/Main/Software. (Windows, Mac, Linux).

The IDE (Integrated Development Environment) is where you'll be writing your Arduino code (called sketches). From here you can interface with your Arduino through Ethernet, USB, WiFi, or Bluetooth, as well as upload new sketches and check the code compiles.


sketch_mar09a | Arduino 1....
File Edit Sketch Tools Help
sketch_mar09a
Arduino Due (Programming Port) on COM1

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 1 - The Arduino IDE. Note the C/C++-like syntax.

To connect your Arduino, you'll need to select the right model and port. Under tools there is a sub-menu marked 'Board:' and the currently selected model. Make sure this is the Arduino Leonardo. To select the correct port: under tools there is a sub-menu labelled as 'Port' you'll need to select the identified port (but this is different on each OS).

Windows: under Control Panel > Device Manager you should be able to look at your USB (or Ethernet) ports. The Arduino should show up. If you go under device properties it should display the port listed as COM## with the hashes replaced as numbers. In the IDE you will want to select this as your port.

Mac: If all goes well you should be able to select your model and then under port the serial ports should automatically refresh when an available device is present. If none are present 'serial' will be displayed, but greyed-out.

Linux: I haven't tried for Linux, but I recommend you follow the Mac procedure.

Luckily Arduino has a large community. So any problems connected can usually be troubleshooted with a Google search.

If you're worried your Arduino board isn't working, check that the green-light is on next to the reset button. On a Leonardo there is also an orange LED next to that green one. If it doesn't come on as you plug the Leonardo in to the USB-port there may be a bigger problem. Let me know.

Programming

Arduino originally ran off a language called Processing which was in-part developed by their group. The syntax is very similar to C++ but it has a few quirks. Most of the hard work has been done for you, as Arduino is a library driven project – this means that instead of writing something which takes information, converts to 0's and 1's and does something with it at a certain rate with certain metadata, you can just write myFile.print(String(voltage)); to store a value on a file on an SD card for example.

There is a slight limitation with Arduino, which you wouldn't have found on other systems. The size of your program is limited, quite severely, by the hardware. This means on a Leonardo the largest sketch can be 28KB after compiling. Also, the amount of memory you can use for variables is limited to 2.5KB. However, this usually isn't a problem and it shouldn't be today.

To create a new sketch, you should use the button provided on the IDE. The IDE is adamant that you should create a new folder (and it will do it for you) and it will make a new file in that folder with the same name. The IDE will then populate a blank sketch with the two control structures required for all Arduino sketches: setup and loop.

Function 'void setup()' is where all of your initialisation takes place. This is only executed once, so it's good for small setup commands such as SD.open(char*[]) - which opens a chosen file on an SD card- and Serial.begin(int) – which begins a serial connection as the designated bit rate. Here you'll want to initialise all of your variables (assign values to them) and turn on any hardware you want to use.

In ‘void loop()’ you execute your main code. This is repeated indefinitely (or until your Arduino eats through your battery pack). Here you will want to take sensor readings over time intervals, write a line of text to a file, wait for an RF message (or send one), turn on (and off) an LED. You can set a delay between commands (and between each cycle) by putting the line: “delay(int milliseconds);” where you want the pause.

You will probably want to do a lot of your Serial.print() and Serial.println() commands in both setup and loop. Some in setup() to say you’ve successfully initialised or a file exists, and some in loop() to write data back to you.

You might want to include some libraries. This is done using the #include<library_name.h> command at the top of the file (like in C/C++). You can get the IDE to do this for you if you want.

And often you will want some global variables – variables which are known throughout the entire program. These are declared above the setup loop and don’t need to be given a value straight away. Unless you start building your own functions, this is essentially just a semantic, as there is almost no drawback to doing this in a program which contains only loop and setup. However, it might become worthwhile later if you pass variables in and out of functions. I do recommend that you declare your sensor input pins as global variables - and normally your output pins too (unless you’re doing something clever with LEDs and alternating which one lights up).

Generally speaking, if you find a bug: Google it! Someone has probably faced the same struggle and solved it. However, I will be here throughout the day to help you along.

I recommend you read ‘Getting Started with Arduino’ by Massimo Banzi (co-founder of Arduino). It explains everything very simply and in a very chatty manner. Speak to me about getting a copy.

Libraries

I've said this a lot, but Arduino is a library and community driven platform. If you want to do something which interfaces with a bit of hardware, there's a very good chance that someone has written a library for it which saves you the bother. If you like a challenge it can be a fun process making your own code and it teaches you how much you really should appreciate some of the standards created (I recently learned about the HTTP request standards – the ones Google Chrome sends when you want to load twitter.com). But today we will stick to the libraries.

For the LCD segment we will be using the LiquidCrystal library. This one comes built-in to the IDE and can be accessed under Sketch > Include Library. Although once you become C/C++ whizzes, you'll see that all you need to do is include the library at the very top of your file using #include<LiquidCrystal.h>

For the radio segment we will be using the VitualWire library. This will need to be downloaded as it is a community library. It was created by Mike McCauley and he hosts it (and a few videos) on his website and on YouTube. For the download, visit http://www.airspayce.com/mikem/arduino/VirtualWire/VirtualWire-1.27.zip.
This library has actually been replaced by the RadioHead library, but that shouldn't stop us. These files will need to be saved in your Arduino library. You'll probably need to right-click on the Arduino icon in your folders and click expand or show contents to get the Arduino files up. Then you can go to Contents > Java > Libraries and you'll want to unzip the download in that folder. You should then close the IDE and relaunch it. The library should now be in the drop-down and an example piece of code should be under examples. Make sure you have "#include<VirtualWire.h>" at the top of your program.

Control

The Arduino is not a computer – we've seen that with the single indefinite loop. However, it can still do quite a lot and come quite close if you program it cleverly. If you try hard enough, you can get it to process HTTP requests and fall back on the right function to handle the request (get file, set file, remove file, add file) – that's all a browser asks for and all a (primitive) web-sever does (plus a bit of authentication). If you want to do that you would want to use the larger Arduino, the Arduino Mega, or a Raspberry Pi, which can do a lot more on that front and doesn't get hit by the code-size limit.

But for making things happen repeatedly, separated by a precise interval, at low-power and with low-cost, then an Arduino Uno/Leonardo/Nano is probably the right tool for the job. This is handy for acting as a sensor (incl. data acquisition), displaying information which updates, sending data, controlling motors. Things are still mechanically automated, just with a smaller scope.

In this segment we will power and display a message on an LCD and, later, display radio messages we've received. This might seem like a small step, but it means we're on our way to having a stand-alone device which can be useful without being hooked-up to a computer. Alternatively, we could use a "shield" (an add-on which fits on-top of the Arduino) with an SD card reader built-in. Then we could write to an SD card. But there would still be no way of checking the data without stopping the acquisition.

If you were to add a keypad (just buttons connected to an input pin) then you could have created a security-gate or door which only unlocked when the right code was entered. The LCD would simply provide feedback to the user.

Measurement

Now we've managed to interface with the Arduino, it's time to use it to take some measurements.

Arduino is starting to become an option for data logging and even started to be used for important measurements. The IAEA commissioned a UAV-mounted Geiger counter to survey the Fukushima area and measure dose levels. This GM tube could be powered, and pulses counted with an Arduino, with data saved to SD. (Other UAVs have also been operating using Arduinos, but generally small ones).

For us, we're interested in something quite similar. We want to measure the voltage of a signal which we send to one of the input pins. The Arduino's analog pins will provide an integer value between 0 and 1023 (which correspond to 0V and 5V). This can be measured using the function analogRead(int pinID). For most situations this is enough and you won't need to to higher voltages but if you are expecting to go out of that range then you may want to use a voltage divider (see later) or something similar to bring the voltage back in range. You should be aware of the resolution of analogRead(int). Each integer values represents about 0.005V. A voltage divider would lower the resolution. For a thermistor, we should be okay, as ours has a linear response of 10mV per C.

Arduino is also good at receiving digital inputs – for the GM circuit this meant counting pulses when a gamma ray hit the tube. The Arduino's digital pins can be used as sensors with the command digitalRead(int pinID). This will return a HIGH or LOW (1 or 0) depending on whether there is a signal at the pin at the moment it is measured. For a 5V supply to the Arduino, the threshold is 3V. Anything above this will return HIGH and anything below will return LOW. With an 'if' statement this can be used to count pulses. However, calibration is probably required – the Arduino will miss some pulses while it does other things (it has its own 'dead-time').

Sending data is relatively easy. Serial.println() does a lot of work for you – if you pass a value only, it will format it as a string (actually a character array), add a line break, convert to the right format and send over USB/Ethernet. But you will probably want to send more complicated things than a single value. Perhaps two values!

To do this you can either send many Serial.print() commands or you can create a string and send that as a single line. This has the added advantage of being re-usable (you can send it over serial, to SD, over the internet, and to LCD all at once if you really wanted). To do this you need to declare a new variable as a string and then cast your values to a string as you append them to the new variable. E.g. String newLine = "Data: " + String(valueA) + " " + String(valueB);

At the other end, you will want to be able to extract the data after it has been sent. If you want to use Python you will either read the data line by line as it comes in from serial or read it from a textfile (and we all know how to do the latter now, don't we!). This is just string manipulation, separating the line by a delimiter and appending the values to a list. To interface with serial in python you will want to include the line "import serial" at the top of your code. I will demonstrate this during the workshop.

Radio

For about one hundred years' radio has revolutionised telecommunications. Nowadays data transfer if mostly done over the internet, but this is only suitable if there is the correct infrastructure. For portable devices you would need to use the mobile data networks (which costs money) or alternatively radio. This is still done by many people (if you tune an amateur radio to the correct frequencies you can sometimes here the data transfers – it sounds like the old dial-up modems).

Radio is especially good in rural areas or big open-spaces, as messages can be sent over very long distances at a reasonably low cost, if the correct equipment is used. See https://www.youtube.com/watch?v=zddC8rQasrg for a demonstration.

If we consider a solar farm, or a solar module deployed on a rooftop, we might – from time to time – want to be able to measure the power generated without going to inspect the module. This could be done using wires (or even WiFi), but in principle this could also be done using radios. All we need is a transmitter unit and a receiver unit and to know how to set up the communication.

On an Arduino this is rather straight forward. Again, this interfacing with hardware is well served by libraries created by the Arduino community. We will be using the VirtualWire library and a demonstration of how to set it up is at https://www.youtube.com/watch?v=U839NZ78EOo [for reference]. Remember to use the 5V port rather than a digital output pin to power the transmitter/receiver. Signals sent to/from the units should be considered 5V digital pulses.

Electronics

Arduinos are only good at interfacing with some kind of electronic circuit. A lot of these come pre-built (radio, LCD, some sensors) but from time-to-time you may wish to build your own circuits tailored to your needs. This is especially true if you want to do some signal processing, or increase the voltage of the supply. In this case you will need to know a reasonable amount of basic electronics, probably up-to and including BJT transistors. However, for today's circuits you shouldn't need much more than knowledge of Ohm's law, combining resistors in series and in parallel, and how a voltage divider works.

Ohm's Law $V = I\ R$

Resistors in Series $R = R_1 + R_2$

Resistors in Parallel $\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$

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

(For more electronics information and for those truly interested in studying it, I recommend you read 'The Art of Electronics' by Horowitz and Hill. It's big and technical but it appears to have everything).

Missions

Boot Camp:	Connect the Leonardo and print “Hello, World!” to serial.
Mission One:	Connect the Leonardo to an LED (via a resistor). Write a sketch which turns on the in-built LED (pin 13). Then connect a button to the breadboard and control the external LED using the button. Then get both LEDs to blink repeatedly.
Mission Two:	Connect an LCD to the Arduino. This can be trickier than it looks, especially when using external libraries and shields (they sometimes reserve pins). Get the LCD to display a string. (Play with the contrast and make sure the letters are legible).
Mission Three:	Connect a thermistor to the Arduino and use it to take temperature measurements. Use a datasheet if required. The thermistor is an LM35. (Make sure the sensor is working).
Mission Four:	Use Python to interface with the Arduino’s serial port and read data the Arduino sends. Use this to store the temperature data to a file. (Modify the sketch to only take a measurement after receiving a command from another computer via serial. Test it with the Arduino IDE, then get the python script to send the command).
Mission Five:	Use the VirtualWire library to interface with the RF receiver and display the received messages on the LCD. (Modify your sketch so that only messages which begin with your IDs are displayed).
Mission Six:	Now send messages to other groups using the transmitter. You will be told who your target is over radio.
Mission Seven:	Measure the voltage produced by the voltage divider, powered from the Arduino’s 5V pin. (Then try and measure the voltage of a solar cell).
Mission Eight:	Now measure the current across a known-resistance. Start with the 5V input and move to a solar cell. Calculate the power and transmit this to base station (ID 0).
Mission Nine:	Split-up. Each group will become a beacon spread across the east-end of campus. We will now attempt to transmit the current-voltage data of a Silicon solar module (which you’ll make

at Loughborough) back to the classroom. We will use the same procedure as before and only receive messages starting with our IDs and send to the next ID up. This is ambitious. Make Commander Rob proud.