Arduinos without (much) programming

A course for
Itchen Valley Amateur Radio Club

Brian G0UKB
Jan-Feb 2016

What on earth is an Arduino?

- It's a microprocessor
  - A single chip computer with built in timing circuit, memory, and an ability to interface to the physical world, which can be programmed (in C) from any PC
- It's more than a microprocessor
  - It is a complete prototyping platform
    - The microprocessor is on a standard board with all that's needed to get started
    - Download the development software and you are good to go
    - Standard 0.1” headers for breadboarding
  - It is readily expanded with clip on 'shields'
  - It has tremendous library support for all sorts of additional modules
- It's open source
  - So really cheap clones make it an affordable solution for many projects
- It's really popular
  - So there are LOTS of interesting projects out there.

But what about PICs?

Pros:

- Bigger range of devices
- Cheaper (for just the PIC chip)
- Small form factor
  - Although can use Atmel chip with Arduino bootloader
- There are lots of projects (but not as many as Arduino)

Cons

- Need a dedicated hardware programmer
  - PICAXE overcomes this
- Need to build a prototyping board for each project
- Prototypes boards not always reusable with different PICs
- Different form-factors (8-pin, 18 pin, 28 pin etc)
- Poor programming support for real beginners
  - Not many examples
- Hardware programmers don't always support latest chips

Generally pics are just harder to get started with but more flexible when you do!

And the Raspberry Pi?

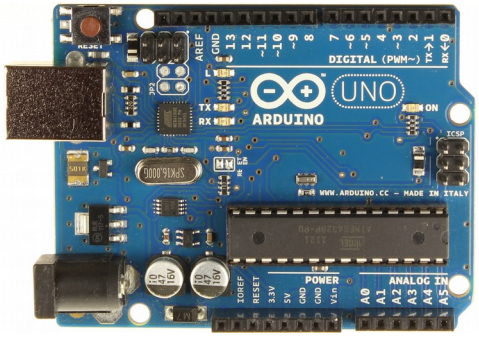
The Raspberry Pi is a fully-fledge computer rather than just a microcontroller

- Just like your normal desktop/laptop computer
- It runs an operating system (normally Debian Linux)
- It has support for a screen via HDMI, audio out, Ethernet and USB all built in
- It's operating system and files live on an SD card
  - It can also use USB pen drives and USB hard drives
- Like the Arduino it does have connectivity to the real world via GPIO pins
- It's much more expensive than the Arduino (£30) but a LOT more powerful

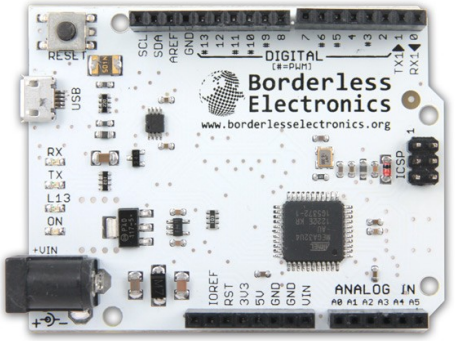
So

- PICs are the cheapest and most flexible solution but are harder to get started with
- Raspberry Pis are much more powerful and wonderful for full-blown 'computer' projects
  - I run a Webserver (serving the Club's Ladder Scoring Program) and a Media Server on my Raspberry Pis
- But the best way to get started cheaply and easily is the

Arduino

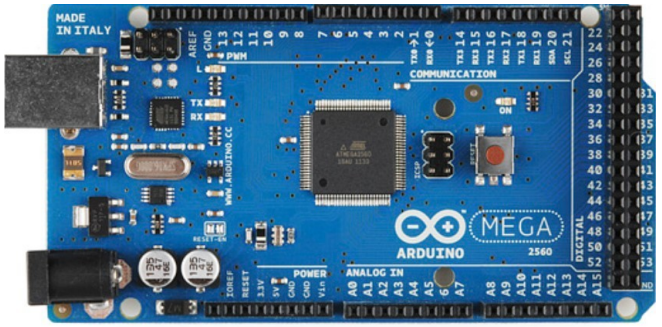


Arduino Uno

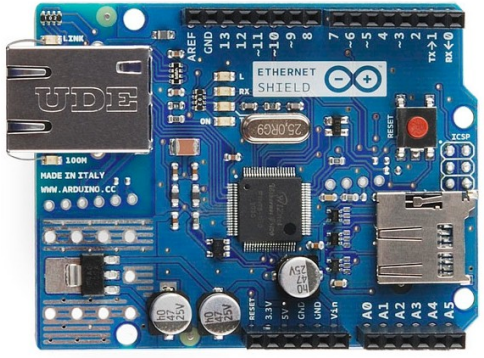


Leonardo

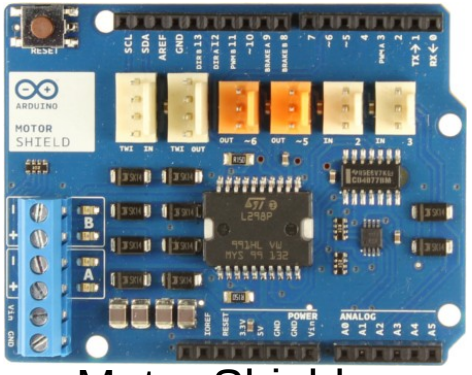
Nano



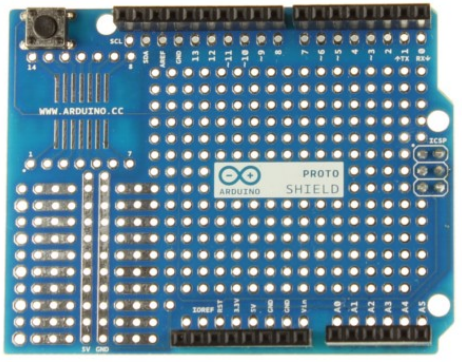
Mega



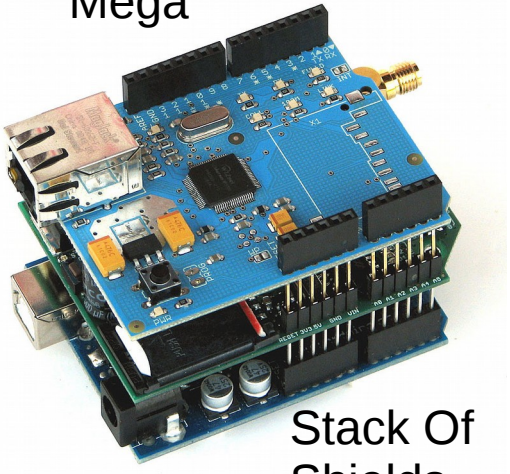
Ethernet Shield



Motor Shield



Prototype Shield



Stack Of Shields

Some of the many modules available

Temperature sensor, Vibration Switch, Hall Effect Magnetic Field Sensor
Momentary Contact Push Button, IR Emission Infrared Transmitter
Passive Buzzer, PIR Motion Sensor, Red Laser Diode, RGB 3 Color LED
Photointerrupters Light Blocking Sensor, Red & Green LED Module
Active Buzzer, Analog Temperature Sensor, Digital Pressure Barometer
Digital Temperature Humidity Sensor, RGB 3 Color LED, Mercury Tilt Switch
Photo Resistor, 5V 10A 1 Channel Mini Relay, Tilt Switch, Mini Reed Switch
Infrared Sensor Receiver, XY-axis Dual-axis Joysticks,
Linear Hall Magnetic Sensor, Reed Switch,IR Flame Detector,
1 Pair Magic Light Cups, Digital Temperature, 3mm 2 color Red and Green LED
Smoke/LPG/Butane/Hydrogen Gas Detector, Vibration Sensor,
Obstacle avoidance sensor, Tracking Sensor, Color Flashing 5MM LED,
5 Hall Effect Liner Sensor, Metal Touch Interface,
High Sensitivity ECM Sound Detector, Audio Sound Sensor Microphone
Heartbeat Sensor, Rotary Encoder

A Quick Technical Overview

(for those interested in such things)

The Arduino Uno

- Microcontroller: ATmega328P
- Clock Speed: 16 MHz
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Input Voltage (limit): 6-20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 6
- DC Current per I/O Pin: 20 mA
- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB (ATmega328P) of which 0.5 KB is used by bootloader
  - The bootloader is the code that starts the chip up and makes it into an Arduino rather than just an Atmega microprocessor
  - The remaining memory is for your program code
- SRAM: 2KB (ATmega328P)
  - Memory for your program variables
- EEPROM: 1KB (ATmega328P)
  - Memory to store values that will persist over reset
  - Not used much but handy for things like WiFi credentials, calibration data etc.

Can't code, Won't code

But surely using an Arduino means I need to learn to code!!

- No!!!!!!!!!
- The objectives of this course are:
  - To give you some idea what an Arduino can do and what projects it can be useful for
  - To help you get an Arduino connected to a computer and upload supplied programs to it
  - To give you sufficient knowledge and confidence to consider downloading an Arduino project you find on the web or in a book
  - Understand the concept of Arduinos so able to add your depth of analog and RF knowledge when discussing any Arduino based club projects
- OK – so we will have to do a little introduction to coding
  - Think of it like the Foundation Licence Morse Appreciation
  - We'll cover just enough code so you can understand the Arduino's basic processes
  - And enough to allow simple personalisation of things like Callsign on screen menus
  - Of course, you can go on and learn to code if you want!
    - And hopefully this course will have whetted your appetite.

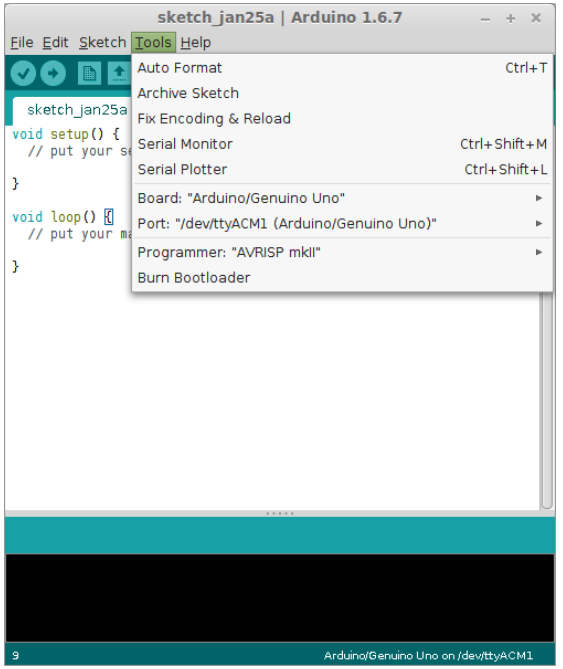
Let's get Started!

- Download the Arduino Software Integrated Development Environment (IDE)
  - From https://www.arduino.cc/en/Main/Software

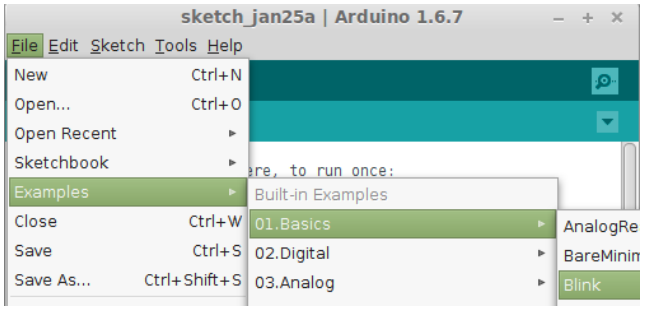
- Connect the Arduino to the computer
  - Standard USB connection
- Install drivers as needed
  - For Genuine Arduinos drivers are supplied in the IDE package
  - For Chinese clones you may need the CH340/1 driver
    - http://www.wch.cn/download/CH341SER_ZIP.html
    - Yes, it's a Chinese page and the sort I'd normally be wary about but it seems OK

We're good to go – Let's Blink

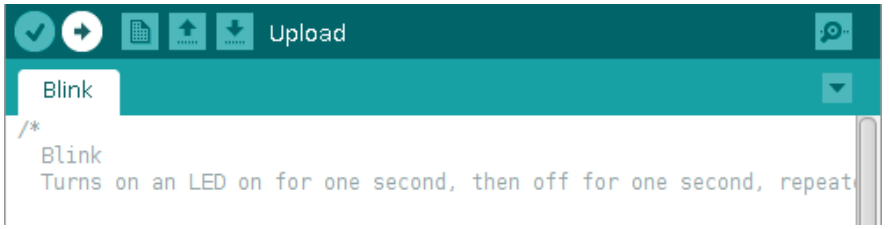
- Start the IDE
- Connect to your board



- Find the Blink example
  File/Examples/01 Basics/Blink



- Compile and Upload it
  - Click on the right arrow button
  - The compile green bar will fill
  - The Arduino's LEDs will flicker



Marvel at the LED Blinking!

Taking Stock

- So, we can flash an LED – Big deal!
- Hold on though, lets think what we've just done:
  - Connected the microprocessor to the PC
  - Designed a program on the PC
    - OK, so in this case we just used a Sample
  - Designed the hardware
    - OK, so in this case we just used an LED that was already connected
  - Compiled the program into the microprocessor's instruction code
  - Uploaded the program to the microprocessor
  - Verified that it runs OK
- If you think about it, this is all we would ever want to do!
  - All that changes is:
    - The program may be more complex
    - The hardware may be more complex
    - It may be harder to verify all is working OK or to fix things when it isn't
  - But we now have working Arduino programs

Introduction to Programming

- What is a program?
  - A program is a set of instructions, which when executed will achieve the required goal.
  - Hence we often see things like "Weight Watchers 6 Week program'
- Some Examples

Arduino C code to blink an LED once a second

```
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

Python code to print out the contents of a file

```
with open(filename,'r') as f:
    for line in f.readlines():
       print(line)
```

Any guesses?

```
Row 12 [WS]: K1, M1R, k3, p2, k2tog, k2, M1PL, p1, k1, sm, k1, p1, M1PR, k2, ssk, p2, k3, M1L, k1. 26 sts.
Row 13 [RS]: K1, M1R, p2, T4p2togR, p3, M1R, k2, p1, sm, p1, k2, M1L, p3, T4p2togL, p2, M1L, k1. 28 sts.
Row 14 [WS]: K1, M1PR, p1, k2, p2, k4, p3, k1, sm, k1, p3, k4, p2, k2, p1, M1PL, k1. 30 sts.
Row 15 [RS]: K1, M1PR, k2, T4p2togR, p3, T3incL, k2, p1, sm, p1, k2, T2L, M1L, p3, T4p2togL, k2, M1PL, k1. 32 sts.
Row 16 [WS]: K1, M1R, k1, p4, k4, (p2, k1) twice, sm, (k1, p2) twice, k4, p4, k1, M1L, k1. 34 sts.
```

Arduino Program Structure

```
/*
  Blink
  Turns on an LED on for one second, then off
for one second, repeatedly.
 */


// the setup function runs once when you press
reset or power the board

void setup() {
  pinMode(13, OUTPUT); // initialize digital pin
13 as an output.
}

// the loop function runs over and over forever

void loop() {
  digitalWrite(13, HIGH);   // turn the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off
by making the voltage LOW
  delay(1000);              // wait for a second
}
```

Header

- Document what the program does
- Declare variables
- Declare header files to link to libraries
- This simple example has no declarations, just some documentation

Setup

Code that only needs to be run once at start-up

- Such as initialising hardware
- Or displaying an initial menu screen

Loop

- This code is executed in a loop FOR EVER
- or at least while the power is on
- All the main code gets executed here

Some C Basics

```
/*
  Blink
  Turns on an LED on for one second, then off
for one second, repeatedly.
 */


// setup function runs once at startup or reset

void setup() {
  pinMode(13, OUTPUT); // pin 13 is an output
}

// the loop function runs over and over forever

void loop() {
  digitalWrite(13, HIGH);   // turn the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off
  delay(1000);              // wait for a second
}
```

Comments

- Anything between /* and */ is a comment
  - Can span several lines
  - Easy way to cut out code temporarily
- Anything follow // is a comment
  - Ends at the end of the line

Function Definitions

- We'll cover functions briefly later
- There are two important ones in all programs
  - setup and loop
  -

Code Statements

Semicolon

- All statement lines must end with a ;

Curly Braces

- Group statements together
- Each function has a { at the beginning and a } at the end
- All statements in between belong to that function
- {} are used anywhere multiple statements need to be considered as one group

Let's look at what the code does

```
/*
  Blink
  Turns on an LED on for one second, then off
for one second, repeatedly.
 */


// setup function runs once at startup or reset

void setup() {
  pinMode(13, OUTPUT); // pin 13 is an output
}

// the loop function runs over and over forever

void loop() {
  digitalWrite(13, HIGH);   // turn the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off
  delay(1000);              // wait for a second
}
```

`pinMode(13, OUTPUT);`

- Tell the Arduino that digital I/O pin 13 will be used for OUTPUT
- So the Arduino can make the pin either 5v or 0v
- We only need to do this once so we do it during setup
- Pin13 has an LED attached

`void loop() {`

- Start the main code loop which loops forever

`digitalWrite(13,HIGH)`

- Tell the Arduino to switch 5v to pin 13
- Which will turn the LED on

`delay(1000);`

- Tell the Arduino to do nothing for 1000 milliseconds
- The LED will remain lit

`digitalWrite(13,LOW);`

- Tell the Arduino to switch pin 13 to ground
- Which will turn the LED off

`delay(1000);`

- Tell the Arduino to do nothing for 1000 milliseconds
- The LED will remain off

`}`

- End of the main code loop so go back round to the beginning of the loop and do it all again, and again, and again ...

Before we play

- Some good coding practice to make life easier
  - The Blink example has
    - pinMode(13, OUTPUT); and digitalWrite(13, HIGH);
  - But what if we wanted to change pins?
    - We'd need to find every reference in the code and change it – risky
- So we can assign a symbolic name to the pin
  - const int testLED = 13;
  - This defines a variable called testLED and gives it a value of 13
  - Don''t worry about the “const int” bit, just use it
    - we'll cover C program variables later
  - So now, in our code we can write:
    - pinMode(testLED, OUTPUT); and digitalWrite(testLED, HIGH); etc.
    - And to change pins we need only change the value in one place – the definition
    - Also this automatically documents the code making it easier to understand
- Rules
  - Best place for definitions is in the header section, before the void setup { code
  - Chose meaningful names
  - Names are case sensitive
    - so if you wrote pinMode(TESTLED, OUTPUT); you would get an error

Let's Play #1

- Go ahead and change the delay values
  - Make the LED on and off for 2 seconds
  - How about one for 2 seconds, off for half a second
  - OK here's a tricky one
    - One for 2 seconds, off for two seconds, then on again for two seconds and this time off for only half a second
    - Hint – cut and paste is handy
- Change the LED pin to be a variable name
  - Still on pin 13
  - Try a real LED and resistor
  - Change pins to a different number – digital 2-12

Let's Play #2

- Take 3 LEDs and build a UK traffic light system
  - Red → Red+Amber → Green → Amber → Red
  - Rember to use variable names – it'll make the coding easier
- Too Easy?
  - How about 6 LEDs making a road junction
  - 2 sets of traffic lights
  - Out of phase so Go on one set is Stop on the other etc.
- A Challenge!
  - How about traffic lights for a Pelican Crossing
  - 3 LEDs
  - Red → Flashing amber → Green → Solid Amber → Red

In Summary

- What have we learned?
  - What an Arduino is and where it sits in relationship to a PIC or Raspberry Pi
  - How to connect an Arduino to a PC
  - How to upload a program to the Arduino
  - How to design a simple prototype and write code to support it
- Replace those LEDs with relays and you've built a very useful sequencer!
- We've come a long way this week but haver just started!

To Come...

- Read an analog input
  - The Arduino can read any voltage between 0-5v
- Attach an LCD character display
  - Now things look much 'cooler' than flashing LEDs
  - See your own callsign in lights!
- Understand Arduino library usage and examples
  - So you attach almost ANY module to the Arduino and get it working with almost no code
- A little more programming
  - Variables so you can do maths
  - Flow control so you can test things or do things multiple times