Microcontroller Programming with the Arduino

Contents

Intro

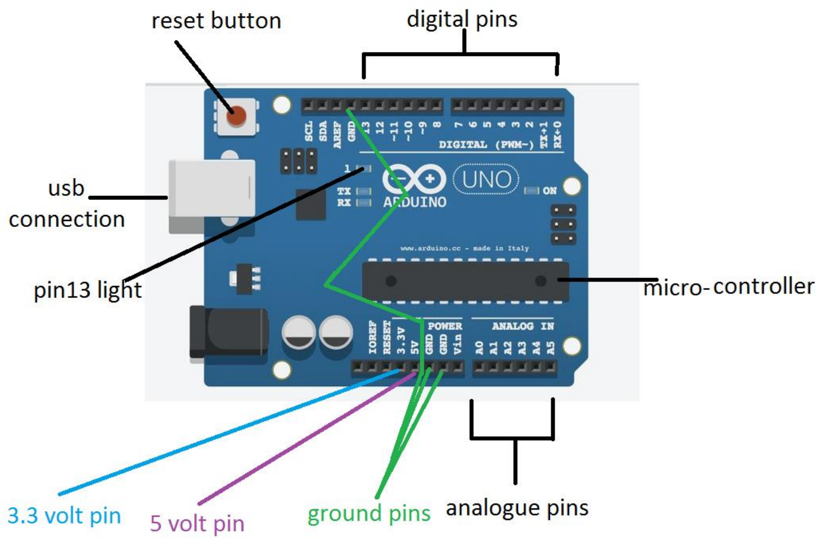
Audience

This tutorial pdf is intended for those students currently studying the first year programming course in C. It will teach you the basics of programming the arduino microcontroller without any assumptions that the reader knows anything about electronics, all basic concepts are learnt along the way.

Your kit

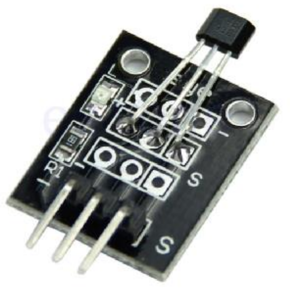
As part of your course you should have access to the arduino and the ICs (integrated circuits allow you to extend the functionality of your arduino), ask your teacher about these if you haven't got a hold of them yet (you may have to pay a small fee for the kit).

Here is a small decription of each element (don't worry if you don't have a particular item, it may have been taken out for repairs or been removed from the course).



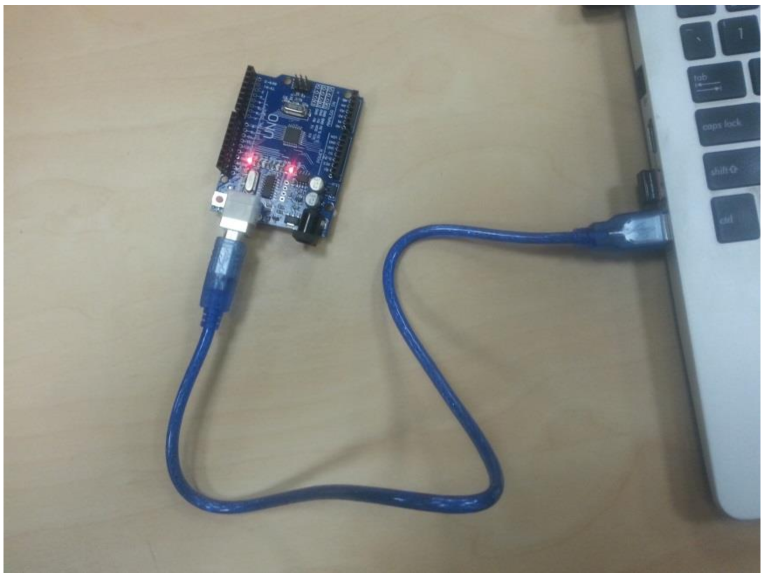
The Arduino microcontroller

- The USB connection is where you will plug your Arduino into your desktop or laptop.
- The reset button resets the program currently running on the Arduino.
- The digital pins can read and write output/input 1s and 0s (bits) as high (5 volt) and low (less than 3 volts) voltages. They can also simulate analogue output in the form of something called pulse width modulation, which is a fancy name for flipping between high and low voltage at different speeds so that the output is high for some percentage of time to simulate analogue output.
- The pin13 light is an LED (light emitting diode), if you write a 1 to pin13, the light will turn on.
- The micro-controller processes your program to control the pins on the Arduino.
- The 3.3 volt and 5 volt pin supply voltage to devices connected to the Arduino.
- The ground pins complete the circuit from devices connected to the digital/analogue output or 5v/3.3v pins.
- The analogue pins can read analogue input. They can also do general I/O too.

 digital multi-function shield • Is connected to the top of your Arduino • Has a reset button too • Has 3 buttons which you can read input • Has 4 LED lights • Has a 7 segment (digital alarm clock) display	 LCD shield • Can write text to lCD screen • Has several buttons • Turn the screw on the blue box to adjust screen brightness	 Buzzer • Can make annoying sounds at different pitch-es
 Sound Sensor • Can detect sound • Turn the screw to adjust sensitivity	 Obstacle sensor • Detects if an object is within a defined range • Turn the screw to change the range	 Ultrasonic sensor • Detects the distance be-tween the object it faces and itself
 Digital compass • Detects the axis it is rotat-ed at	 IMU – Internal Measurement Unit • Detects the axis it is rotat-ed at • Detects accelerations it is put through	 Temperature sensor • Can detect the tempera-ture of the current envi-ronment

Development Environment

Connecting the Arduino



The first step is to get your cable, connect one end to the Arduino and the other to the USB port of your laptop of Desktop. Use the picture above as a reference, some lights on the Arduino should light up.

Setting up your programming environment

We will be using the Arduino IDE (integrated development environment) to write and upload out programs to the Arduino device. The website for this IDE is: https://www.arduino.cc/en/Main/Software .
Installation instructions are as follows:

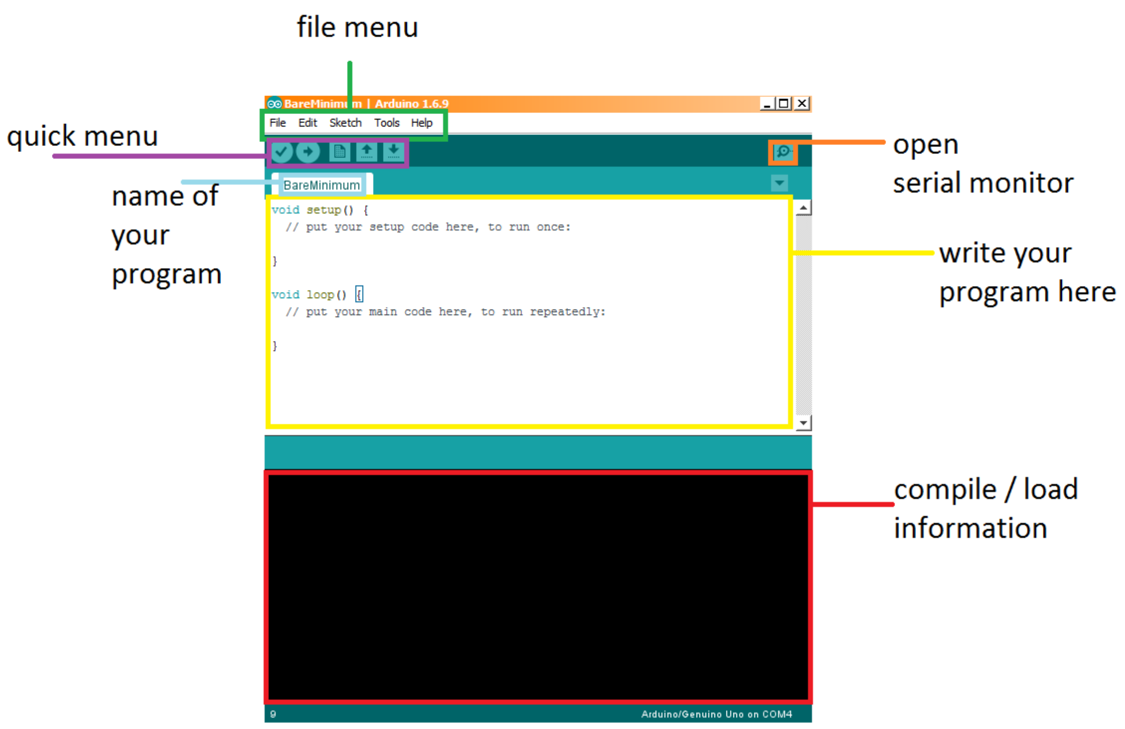
Ubuntu

1. Open the terminal (Ctrl + t)
2. Paste in: sudo-apt get install arduino arduino-core
3. Follow prompts

Windows 7/8/10, OS X, Linux (other)

1. Go to https://www.arduino.cc/en/Main/Software
2. Download the correct installer
3. follow prompts

Interface basics


file menu
quick menu
name of your program
open serial monitor
write your program here
compile / load information
BareMinimum | Arduino 1.6.9
File Edit Sketch Tools Help
BareMinimum
9
Arduino/Genuino Uno on COM4

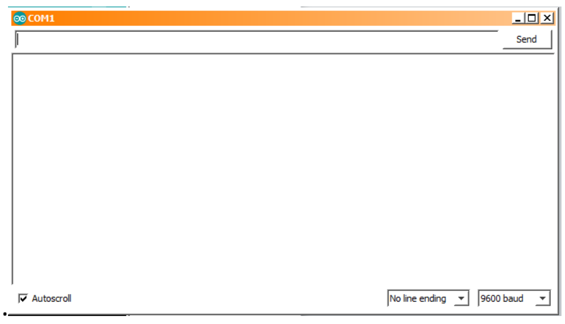
```
void setup() {
  // put your setup code here, to run once:

}

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

}
```

- In the file menu (green) you can do various operations such as:
  - File -> open/save your program (called a sketch)
  - Edit -> find/replace/copy and paste
  - Sketch -> compile/upload program to Arduino
  - Tools -> set the port the Arduino is connected to, set the board type (we are using the UNO)
- The text-editor (yellow) is where you will write your programs
- The compile/load information section (red) displays compile time information
- The quick menu has 5 buttons
  - The tick button compiles your program
  - The arrow button uploads the program to the Arduino (only if it compiles first)
  - The rest of the buttons are: Open a new file, open a specific file, save
- Serial Monitor open button (orange)
  - Picture below
  - Allows the developer to send information to the Arduino
  - Allows the developer to read information sent from the Arduino
  - If Auto scroll is checked, the latest message from the Arduino is focused on
  - 9600 baud rate is the rate at which the Arduino is communicating with your IDE, 9600 baud is 9600 bits per second



Programming the Arduino

Most basic Program (Sketch)

The language we will be using to program the Arduino will be a variation of C/C++.

Take a look at the most basic program for Arduino, it does nothing. There are two functions setup and loop. The setup function is called when the program begins and the loop function is called continually whilst the Arduino is running.

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

}

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

}
```

Figure 1 Most basic program

Essential the Arduino executes the following C program: “setup(); while(1) loop();”

Running the program

1. Connect your Arduino to your laptop/desktop
2. Enter the most basic program into the editor (should appear by default)
3. Click the compile button (the tick) to verify the program is legitimate C
4. Click the upload button (you may have to save the program first)
5. Stand back and watch as the Arduino blinks a few times to let you know a program is being uploaded before doing… nothing

Note: if the upload fails try:

1. going to tools->board and make sure the UNO is selected
2. going to the tools->port and select the right board (you may have a few options, try them until you find which USB port your Arduino is connected to.

Blinking Pin 13

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

// the loop function runs over and over again forever
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
}
```

The digital pin 13 is special, not only can you connect other pins to it for input/output but it is also connected to an LED on the board (see the Your Kit section with the description for the Arduino board).

We can set this pin HIGH to light up the LED, and low to turn the LED off.

1. pinMode - The first thing we do is initialize pin 13 as an output pin using the pinMode function. This takes in the pin selected as the first argument and OUTPUT or INPUT as the second argument. We are going to be setting the output for this pin so we use OUTPUT.
2. Loop function:
   a. digitalWrite: this function takes the pin id as the first argument and whether the pin should output HIGH (digital 1) or LOW (digital 0). Note: HIGH and LOW are pre-processor defines for 0x1 and 0x0 respectively.
   b. delay causes the program to hang waiting for delay to return, for x milliseconds where x is the first argument.

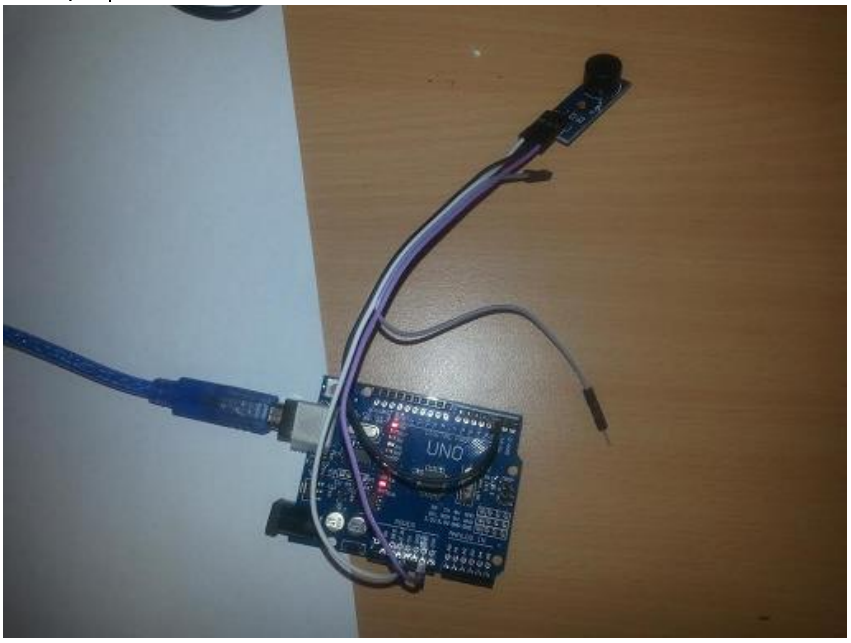
Compile and upload your program to your Arduino, the pin 13 LED should blink at a rate of 0.5 times per second (turn on, wait for 1 second, turn off, wait for 1 second).

Your first module: The buzzer

The buzzer module makes an awful sound, yay!

This IC has 3 pins to attach to the Arduino, a VCC pin which connects to any of the 5 volt pins on the Arduino, a GND pin which connects to any of the ground pins on the Arduino and an input pin (named I/O on the device) for sending through a 1 or a 0 as input data.

A picture below shows my wiring, not the colours of the pin cables (called male to female wire connectors) do not mean anything. The buzzer pins are marked on the IC (VCC, I/O, GND), for this tutorial I am using digital pin 3 for the I/O pin.



The way the IC works is as follows, the I/O pin on the buzzer must receive a 1 followed by a 0 at some frequency f. The frequency defines the frequency of the sound the buzzer makes.

```
int ioPin, frequency = 400;
void setup() {
  ioPin = 3;
  pinMode(ioPin, OUTPUT);
}
void loop() {
  digitalWrite(ioPin, LOW);
  delay(500 / (double)frequency);
  digitalWrite(ioPin, HIGH);
  delay(500 / (double)frequency);
}
```

In this program we setup 2 variables, ioPin specifies the pin we connected I/O to, and frequency is the frequency we change our input signal at. Note: as you can see we can initialize frequency/ioPin/any global variable in global space or in setup(). Then in our loop function, using the digitalWrite function we learnt, we set the pin LOW, then wait for 500 / frequency milliseconds, this equation converts our frequency variable to how long we should delay to write both a 0 and a 1 at frequency times per second (delay per 0 and 1 send = 1000ms/frequency, delay per 0 or 1 send = 1000ms/frequency/2 = 500/frequency).

Talking with your Arduino

Exchanging messages between yourself (through the Serial Monitor) and your Arduino does not require any modules. You can do it if you want to alter the logic or control of your program, view sensors' input and/or debug your program (see what's going on and why your program may not be working).

```
int loopCounter; //counts the number of loop() function calls
void setup() {
  Serial.begin(9600); //setup our Serial i/o @ 9600 bits per second
  loopCounter = 0;
}

void loop() {
  Serial.print("hello"); //prints hello with no new line character
  String arduinoString = " world"; //an Ardiono string
  Serial.println(arduinoString); //prints the ardiono string with a new line character

  //C way to print-out loopCounter
  Serial.print("counter: ");
  char buffer[100];
  char * ptr = itoa(loopCounter, buffer, 10);
  char buffer2[100] = "C count: ";
  strcat(buffer2, ptr);
  Serial.println(buffer2); //can also do: Serial.print(buffer2); Serial.println(loopCounter);

  //Simpler C++/Arduino way
  Serial.println("C++/Arduino count: " + String(loopCounter));

  loopCounter++;
  delay(2000);
}
```

In the code above, we have an int variable called loopCounter which we will be using to count the number of times loop is completed. Notice in our setup function we have a line Serial.begin(9600), this sets up our serial communication to/from the Arduino and computer at a rate of 9600 bits per second (a frequently used comm rate for Arduino). According to Arduino supported baud rates include: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600, and 115200.

In our loop function we use the function Serial.print() and Serial.println() a few times. Serial.println() prints out its first argument with a trailing new line character, Serial.print() omits an end of line character. Both functions cat take in c-strings, ints, floats, doubles etc… as well as the Arduino String (technically a C++ wrapper for c-strings). In the first few lines of loop we use Serial.print to print out a c-string "hello" and then we declare an Arduino String arduinoString to " world", and print the Arduino string with a trailing new line.

We then expose a C-style way to print out loopCounter using a buffer to write loopCounter in with the C itoa function, then we use another c-string buffer to hold: "C count: " before doing a string concatenation with the C function strcat to combine the two, then print the result.

The Arduino (technically C++ style) way of printing out our loopCounter is far more simpler and intuitive.

Note: to convert an Arduino String to an int, you can use the built in method toInt(), like String s = "5"; int sInt = s.toInt();

Run this Arduino program and click the Serial Monitor button to view the console.

In the next program the Arduino will be receiving data from our Serial Monitor window. If you open your monitor, you should see a text input widget next to a send button. You can send information to the Arduino using this.

Armed with that knowledge, check out the following program source code:

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  if (Serial.available()){ //Serial.available() returns true if there is some data to read
    String input = Serial.readString(); //readString reads the line of input the user typed
    Serial.println("got a message: " + input);
    if (input.length() >= 7){
      if(input.substring(0, 5) == "print"){
        int count = input.substring(6, input.length()).toInt();
        for(int i = 0; i < count; i++)
          Serial.println(i);
        Serial.println("executed counting operation");
      }
    }
  }
  delay(10);
}
```

Look at the loop function, here we are using a function Serial.available() which returns true if the user hit send with some data. If we do have some information coming in, we print it out. Using the length() method of our String, we check if the length is above 7. In this simple application we use the substring method to detect if the first 4 letters (sub-string indices 0 to 5 exclusive) say “print”, if they do, we convert the rest of the string into an int, and print out the numbers from 0 to that input integer.

See if you can create some interesting applications by combining user i/o with the modules seen so far.