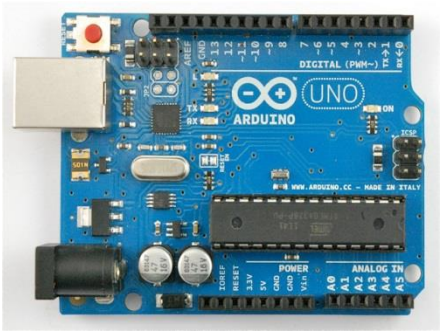
ZAGAZIG UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER & SYSTEMS ENGINEERING DEPT.

GETTING STARTED WITH ARDUINO

INTRODUCTION



Arduino is fast becoming one of the most popular microcontrollers on the market. The Arduino (shown in Figure) is a relatively inexpensive, open source physical computing platform. It is designed to facilitate interaction with the physical world via sensors while being able to perform calculations and various functions. The Arduino can be connected to a computer via a USB cable and programmed using a simplified version of the C programming language.

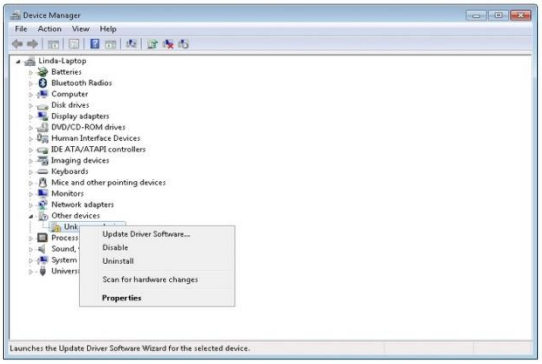
Arduino is composed of two major parts: the Arduino board, which is the piece of hardware you work on when you build your objects; and the Arduino IDE, the piece of software you run on your computer. You use the IDE to create a sketch (a little computer program) that you upload to the Arduino board. The sketch tells the board what to do.

In this lab, you will learn how to program the Arduino and use basic functions delay, pinMode, digitalWrite, digitalRead, analogRead, etc. You will also learn have other things to do such as use serial communication, Driving a DC motor and servo motor, servo motor and simulate the Arduino.

INSTALLING ARDUINO (WINDOWS)

Download the Arduino environment: There is no installer program, but rather you have to unzip a folder which gives you an Arduino folder that contains the Arduino program and a few other items. Start by downloading the zip file for Windows. The Arduino folder contains both the Arduino program itself and also the drivers that allow the Arduino to be connected to your computer by a USB cable. Before we launch the Arduino software, you are going to install the USB drivers.

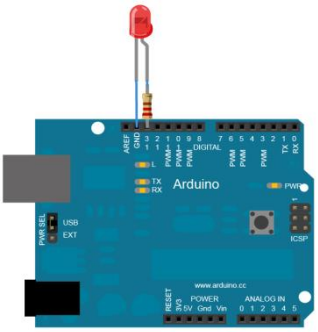
Install the drivers: Plug the Arduino board into the computer; when the Found New Hardware Wizard window comes up, ignore this message and cancel any attempts that Windows makes to try and install drivers automatically for you. The most reliable method of installing the USB drivers is to use the Device Manager. This is accessed in different ways depending on your version of Windows. In Windows 7, you first have to open the Control Panel, and then select the option to view Icons, and you should find the Device Manager in the list. Under the section ‘Other Devices’ you should see an icon for ‘unknown device’ with a little yellow warning triangle next to it. This is your Arduino. Right-click on the device and select the top menu option (Update Driver Software...). You will then be prompted to either ‘Search Automatically for updated driver software’ or ‘Browse my computer for driver software’. Select the option to browse and navigate to the arduino-1.0.2-windows\arduino1.0.2\drivers. Click 'Next' and you may get a security warning, if so, allow the software to be installed. Once the software has been installed, you will get a confirmation message.



Launch the Arduino application: You are now ready to start the Arduino Software, open the Arduino folder and open the Arduino application contained within it. This will start the Arduino IDE, but before you can get programming, you have to tell the Arduino software which type of Arduino board you are using and also select the port it is connecting to. From the 'Tools' menu, select Board and then 'Arduino Uno' . Also on the 'Tools' menu, you will find the 'Serial Port' option. Select this option and there will probably only be one option here and it will either say COM3 or COM4.

BLINKING AN LED

This example shows the simplest thing you can do with an Arduino to see physical output: it blinks an LED.

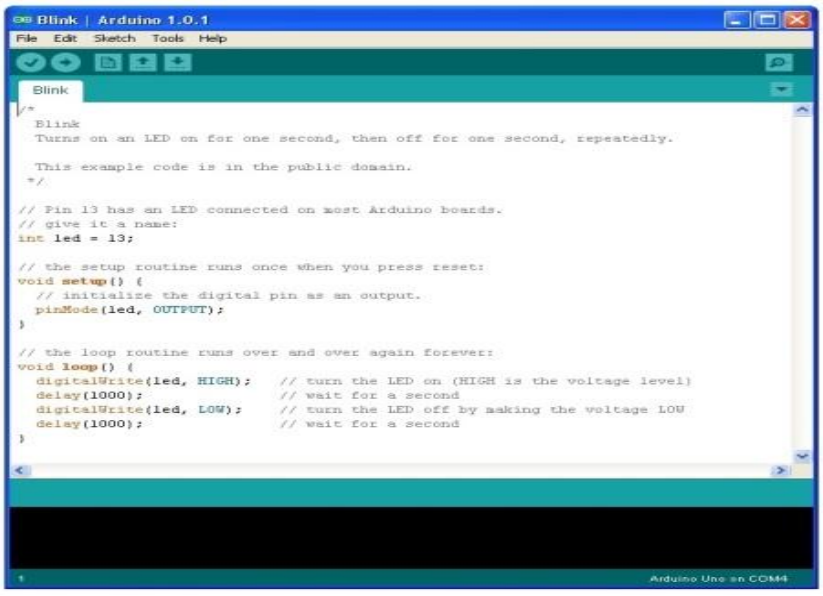


Circuit:

attach a 220-ohm resistor to pin 13. Then attach the long leg(anode) of an LED to the resistor. Attach the short leg (cathode)to ground.

Code:

Loading the 'Blink' Example: The Arduino IDE includes a large collection of example sketches that you can load up and use. This includes an example sketch for making the LED blink. Open the LED blink example sketch: File > Examples > 1.Basics > Blink.



Lets examine this sketch in detail starting with the first section:

```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */
```

This is a **comment**

Let's look at the next line:

```
int led = 13;
```

this is giving a name to the pin that the LED is attached to.

Next, we have the 'setup' function. this is run when the reset button is pressed. It is also run whenever the board resets for any reason, such as power first being applied to it, or after a sketch has been uploaded.

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

Every Arduino sketch must have a 'setup' function, and the part of it where you want to initialize pin 13 as an output pin with the line between the { }. Unlike the 'setup' function that only runs once, after a reset, the 'loop' function will, after it has finished running its commands, immediately start again.

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

the line:

```
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
```

supplies 5 volts to pin 13. That creates a voltage difference across the pins of the LED, and lights it up. Then you turn it off with the line:

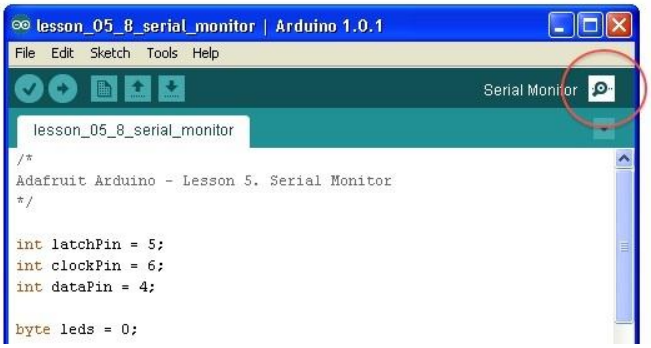
```
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
```

That takes pin 13 back to 0 volts, and turns the LED off. In between the on and the off, you want enough time for a person to see the change, so the delay() commands tell the Arduino to do nothing for 1000 milliseconds, or one second. When you use the delay() command.

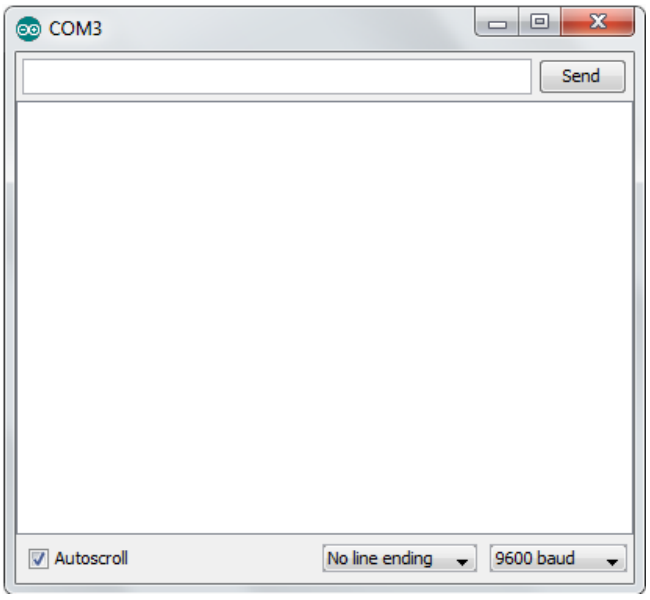
CONTROLLING AN ARDUINO OVER SERIAL

we'll learn how to communicate from the Arduino board back to the computer over the USB port. The word **serial** means "one after the other". Serial data transfer is when we transfer data one **bit** at a time. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.

In this section, you will control the LED from your computer using the Arduino Serial Monitor. The serial monitor allows you to type messages that are sent to the Arduino board as well as see any messages that the Arduino board chooses to reply with. Click on the right-most button on the toolbar in the Arduino IDE. The button is circled below.

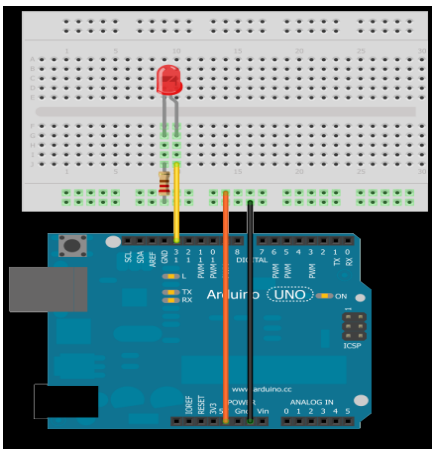


The following window will open.



As shown in Figure, The Serial Monitor has two parts. At the top, there is a field into which a line of text can be typed that will be sent to the board when you either click Send or press RETURN. Below that is a larger area in which any messages coming from the Arduino board will be displayed. Right at the bottom of the window is a drop-down list where you can select the speed at which the data is sent. Whatever you select here must match the baud rate that you specify in your script's startup message. We use 9600, which is the default, so there is no need to change anything here.

Circuit:



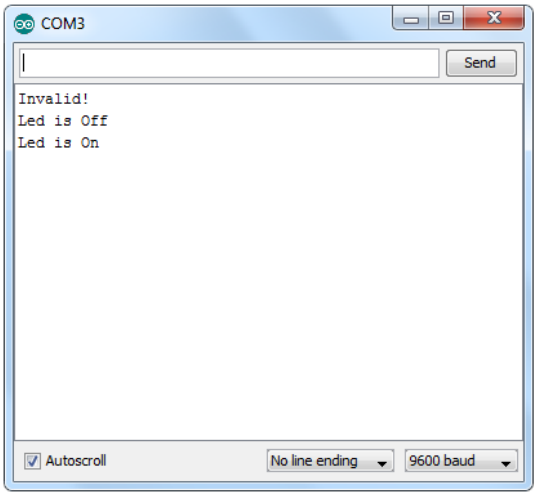
Code:

Upload the following sketch to your Arduino. Later on, we will see exactly how it works.

```
int ledPin = 13;
void setup()
{
  //Create Serial Object
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  if (Serial.available())
  {
  //Read the Input
  int val = Serial.read() - '0';
  if (val == 1)
  {
    Serial.println("Led is On");
    digitalWrite(ledPin, HIGH);
  }
  else if (val == 0)
  {
    Serial.println("Led is Off");
    digitalWrite(ledPin, LOW);
  }
  else
  {
    Serial.println("Invalid!");
  }
 }
}
```

Try typing the following commands, into the top area of the Serial Monitor that is level with the 'Send' button. Typing x, will have no effect, if the LED is already off, and you will get a confirmation message from the Arduino board that invalid number is entered. But as you enter a number 0, the LED should be off and a printout `Led is Off` message. And after entering a number 1, the LED should be on and prints out `Led is On` message, so that the Serial Monitor will appear as shown below.



Code:

Firstly, in the 'setup' function:

```
void setup()
{
  //Create Serial Object
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}
```

We have the command 'Serial.begin(9600)'. This starts serial communication, so that the Arduino can send out commands through the USB connection. The value 9600 is called the 'baud rate' of the connection. This is how fast the data is to be sent. You can change this to a higher value, but you will also have to change the Arduio Serial monitor to the same value.

The 'loop' function is where all the action happens:

```
void loop()
{
  if (Serial.available())
  {
  //Read the Input
  int val = Serial.read() - '0';
  if (val == 1)
  {
    Serial.println("Led is On");
    digitalWrite(ledPin, HIGH);
  }
  else if (val == 0)
```

```
  {
    Serial.println("Led is Off");
    digitalWrite(ledPin, LOW);
  }
  else
  {
    Serial.println("Invalid!");
  }
 }
}
```

Everything that happens inside the loop is contained within an 'if' statement. So unless the call to the built-in Arduino function 'Serial.available()' is 'true' then nothing else will happen.
Serial.available() will return 'true' if data has been send to the Arduino and is there ready to be processed. Incoming messages are held in what is called a buffer and Serial.available() returns true if that buffer is Not empty.

If a message has been received, then it's on to the next line of code:

```
  int val = Serial.read() - '0';
```

This reads the next character from the buffer, and removes it from the buffer. It also assigns it to the variable val. The variable 'val' may be declared as either a character (char) or an integer (int). If you read the serial byte as an int, the read byte will be stored as its **ASCII** decimal value. So “a” will be read as 97, 0 as 48, 1 as 49 and so on. If they are read as a characters (char), “a” will be read as ‘a’, and numbers will be read as characters as well. So 1 will be read as ’1’.
Now we are performing arithmetic on characters! We are subtracting the digit '0' from whatever digit was entered. So, if you typed '0' then '0' – '0' will equal 0. If you typed '1' then '1' – '0' will equal the number 1 because it is actually the ASCII values that are being used in the subtraction.
The 'if' statement on the next line checks to see the value of 'val' if is 0 or 1 or other value.If the 'val' equals 1, then we come to the next line:

```
    Serial.println("Led is On");
```

Which is just a shorthand for "print line" that write back a confirmation message to the Serial Monitor(from the Arduino to the computer). We use two (double quotes) to indicate the beginning and end of a line of text.

After the 'if' statement that handles the case, when the 'val' equals 1, there is a second 'if' statement that that handles the case, when the 'val' equals 0. And finally 'else' statement that handles the case, when the 'val' not equal to 0 or 1.

DRIVING A DC MOTOR

In this section, you will learn how to control a DC motor using an Arduino and a transistor. You will use an Arduino PWM to control the speed of the motor by sending a number between 0 and 255 from the Serial Monitor.

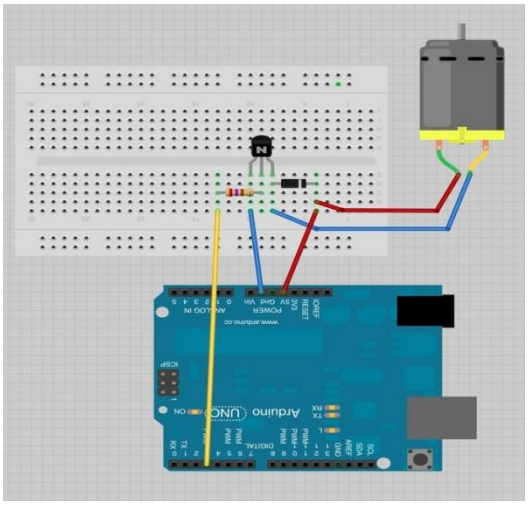
Components:

We will need the following parts to build the project:

- Small 6V DC Motor
- PN2222 Transistor
- 1N4001 diode

- 2 kΩ Resistor
- Arduino Uno
- Breadboard Layout

When you put together the breadboard, there are two things to look out for. Firstly, make sure that the transistor is the right way around. The flat side of the transistor should be on the right-hand side of the breadboard. Secondly the striped end of the diode should be towards the +5V power line.



The motor can be connected either way around

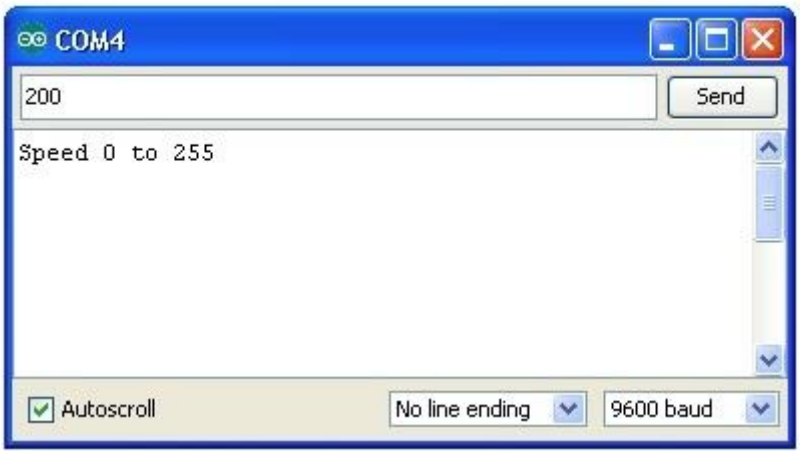
Code:

Load up the following sketch onto your Arduino.

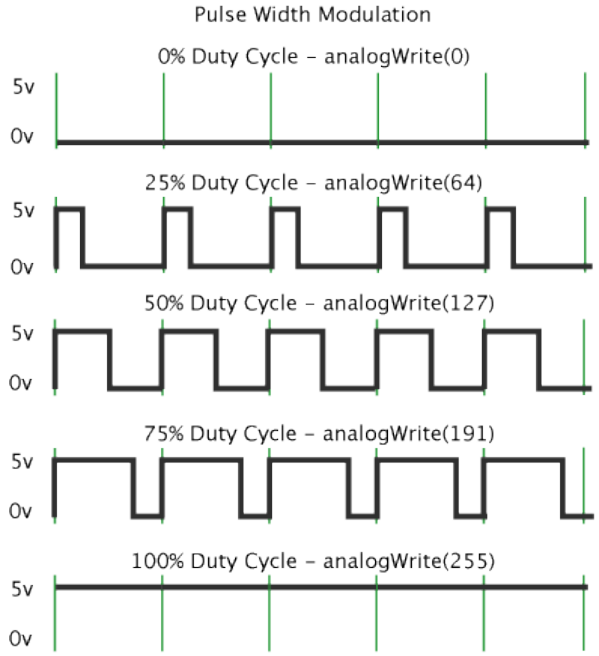
```
int motorPin = 3;
void setup()
{
  pinsMode(motorPin, OUTPUT);
  Serial.begin(9600);
  Serial.println("Speed 0 to 255");
}

void loop()
{
  if (Serial.available())
  {
    int speed = Serial.parseInt();
    if (speed >= 0 && speed <= 255)
    {
      analogWrite(motorPin, speed);
    }
  }
}
```

The transistor acts like a switch, controlling the power to the motor, Arduino pin 3 is used to turn the transistor on and off and is given the name 'motorPin' in the sketch.
When the sketch starts, it prompts you, to remind you that to control the speed of the motor you need to enter a value between 0 and 255 in the Serial Monitor.

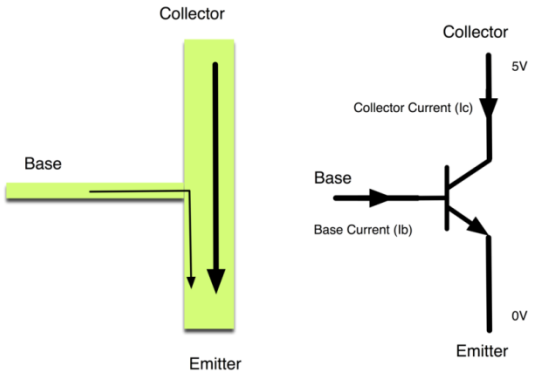


In the 'loop' function, the command 'Serial.parseInt' is used to read the number entered as text in the Serial Monitor and convert it into an 'int'.
You could type any number here, so the 'if' statement on the next line only does an analog write with this number if the number is between 0 and 255.
Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width.
In the graphic below, a call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.

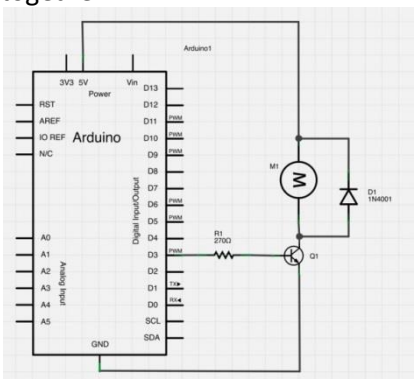


Transistors:

The small DC motor, is likely to use more power than an Arduino digital output can handle directly. If we tried to connect the motor straight to an Arduino pin, there is a good chance that it could damage the Arduino.
A small transistor like the PN2222 can be used as a switch that uses just a little current from the Arduino digital output to control the much bigger current of the motor.



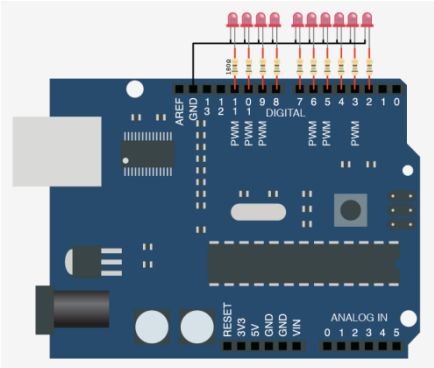
The transistor has three leads. Most of the electricity flows from the Collector to the Emitter, but this will only happen if a small amount is flowing into the Base connection. This small current is supplied by the Arduino digital output.
The diagram below is called a schematic diagram. Like a breadboard layout, it is a way of showing how the parts of an electronic project are connected together.



The pin D3 of the Arduino is connected to the resistor. Just like when using an LED, this limits the current flowing into the transistor through the base.
There is a diode connected across the connections of the motor. Diodes only allow electricity to flow in one direction (the direction of their arrow).
When you turn the power off to a motor, you get a negative spike of voltage, which can damage your Arduino or the transistor. The diode protects against this, by shorting out any such reverse current from the motor.

REPORT

1. Use the Arduino Serial Monitor to turn on 10 LEDs, each LED by its number.



2. how to control both the direction and speed of a small DC motor using an Arduino