Arduino Laboratory Manual

Using the Arduino Board

by

BUGHAW ELECTRONIC SOLUTIONS AND TECHNOLOGIES, INC.

Justin Oliver Lim

Toto Oppus

July 2011

Preface

This laboratory manual is aimed at undergraduate students taking up Computer Engineering, Computer Science, Electronics and Communications Engineering or other related courses. This may also be used by instructors as a reference.

Users of this manual are assumed to have basic knowledge of microprocessor architecture and C programming language.

Exercises in this laboratory manual are built on from previous exercises; thus requiring the thorough understanding of the previous exercise before proceeding to the next.

Contents

Introduction

The Arduino (Uno) Board is a micro-controller board that was created to house the ATmega328 chip. The chip is a high performance and low power 8-bit micro-controller that has 23 programmable I/O lines, 32K bytes of flash memory (of which 0.5KB is already used for the Boot loader), 1k bytes of EEPROM and 2k bytes of RAM. The Arduino Uno board provides the user with 6 analog input pins, 14 digital I/O pins of which 6 of them can also be used for PWM outputs, a power jack, a USB port, an ICSP header, a reset button, a small LED connected to digital pin 13, and a 16MHz crystal oscillator [1]. (See Arduino Uno Component View for more details about the parts.)

In comparison to earlier models, the board uses the Atmega8U2 programmed as a USB-to-serial converter rather than the FTDI USB-to-serial driver chip.

It is the latest in the Arduino USB board series and the current reference model for the Arduino platform. The board is fairly easy to use and capable of doing a lot of things.

Setup & Installation

A. Arduino IDE (For Windows)

This part will guide you through the set up and installation process of the Integrated Development Environment (IDE) that will be used throughout the exercises.

1. Open your default internet browser and access the Arduino website. Download the latest Arduino IDE version. The software is compatible with Linux, Mac and Windows so just choose the one that matches your OS. The Arduino download page is at http://arduino.cc/en/Main/Software [2].

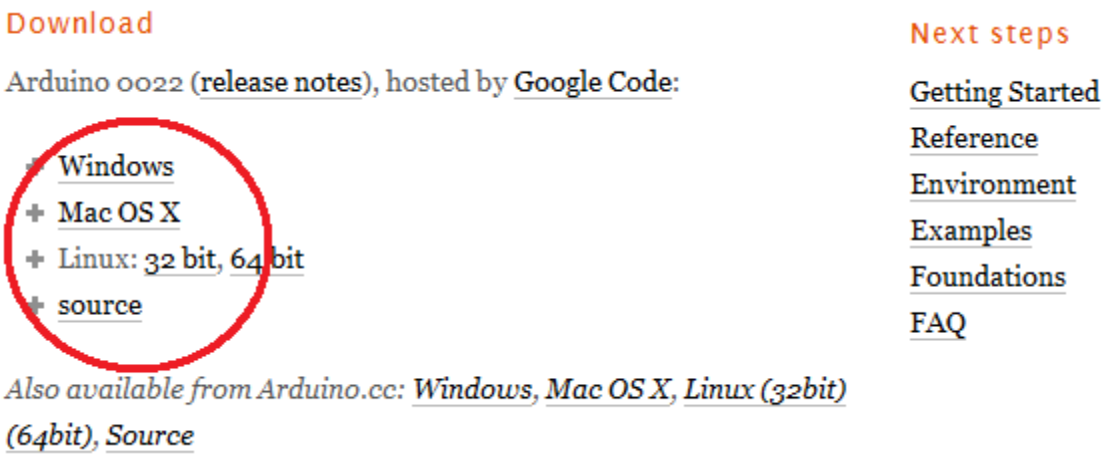


Figure 1: A part of Arduino Website's download page. The current version at this time was 0022. Arduino allows you to install its IDE on several platforms (see encircled)

2. After downloading the compressed file, extract its contents to your preferred directory (*C:\Program Files, your Desktop or etc…*). Note that the whole folder size is around 200MB when completely extracted [2].

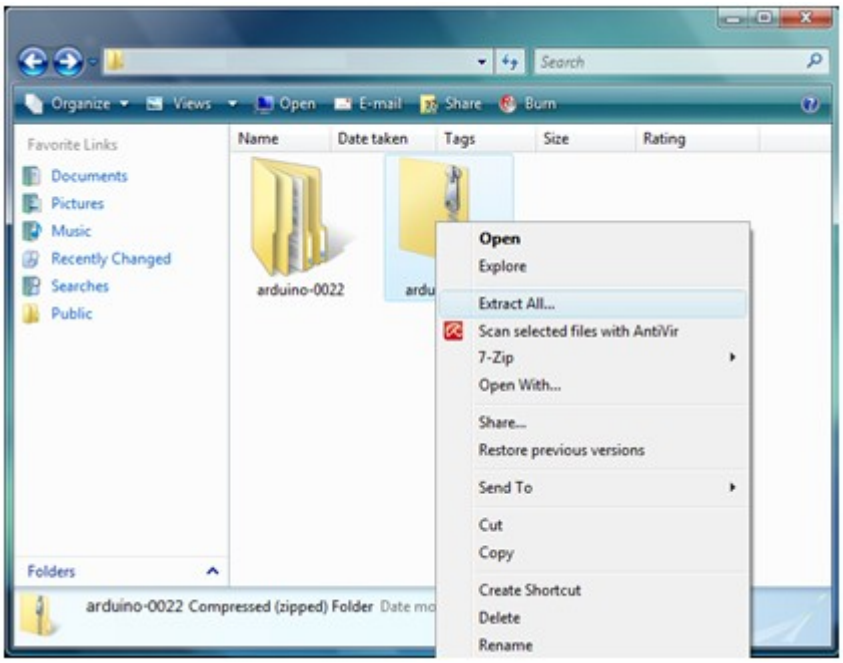


Figure 2: Screenshot of attempt to extract the zipped Arduino folder. Make sure you have an archive utility such as 7zip or WinRAR.

3. Congratulations! Arduino IDE is installed on your computer. To use it, just navigate to your main folder directory and run the **Arduino** application [2].

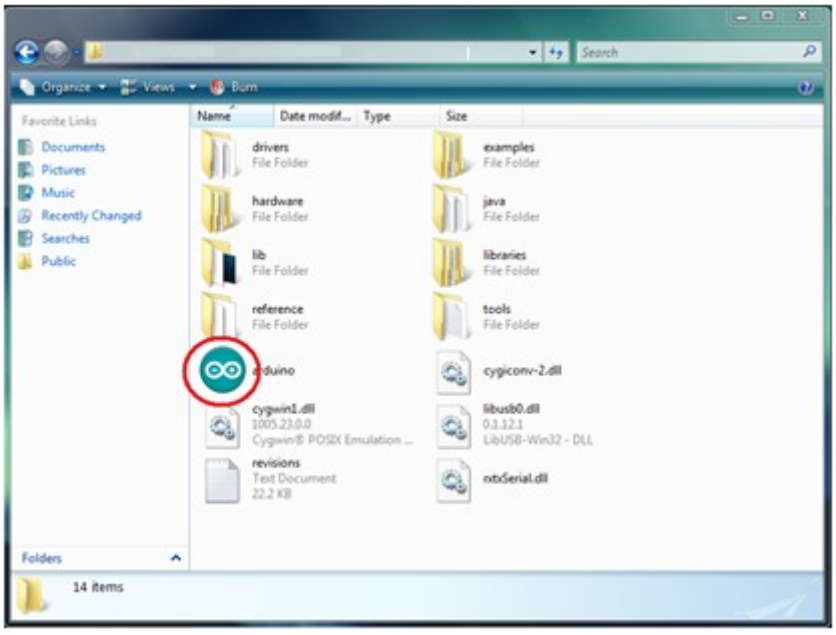


Figure 3: Screenshot of what's inside the *Arduino-0022* ***folder***. The application icon looks like an infinity symbol.

B. Arduino Uno board driver (For Windows)

This part will guide you through the set-up and installation process of the Arduino Uno board driver for the device to be recognized by the IDE.

1. Connect the Arduino UNO to the computer via USB Cable (A on fig. 4) [2]. Check if it is properly connected by observing the green LED labeled ON (B on fig. 4) on the board.

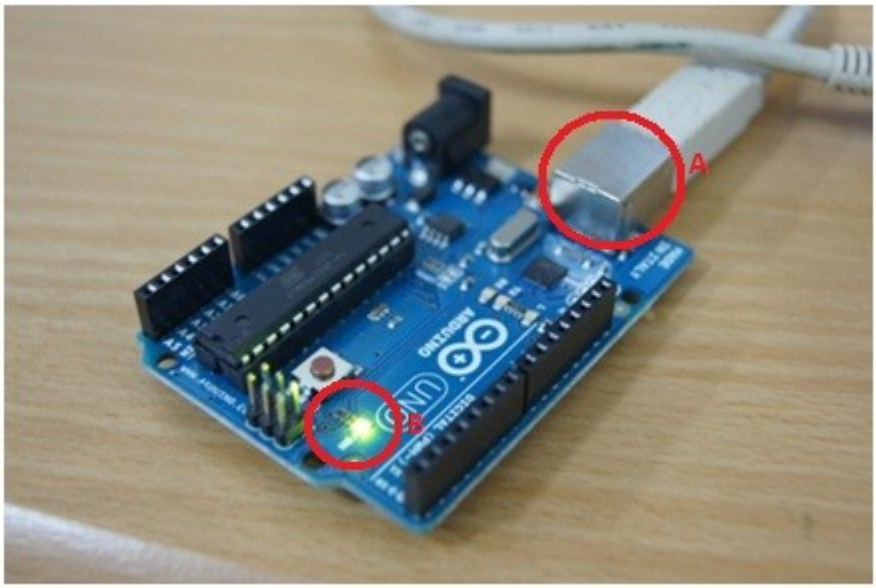


Figure 4: Photo of Arduino Uno board connected to a Computer. Note that the board's USB-B port.

2. Wait for Windows to try and install the device's driver until it fails. Navigate to the **Device Manager** through **Start > Control Panel > Device Manager**. Locate the Arduino Uno Device. Right-click it to choose **Update Driver Software** [2].

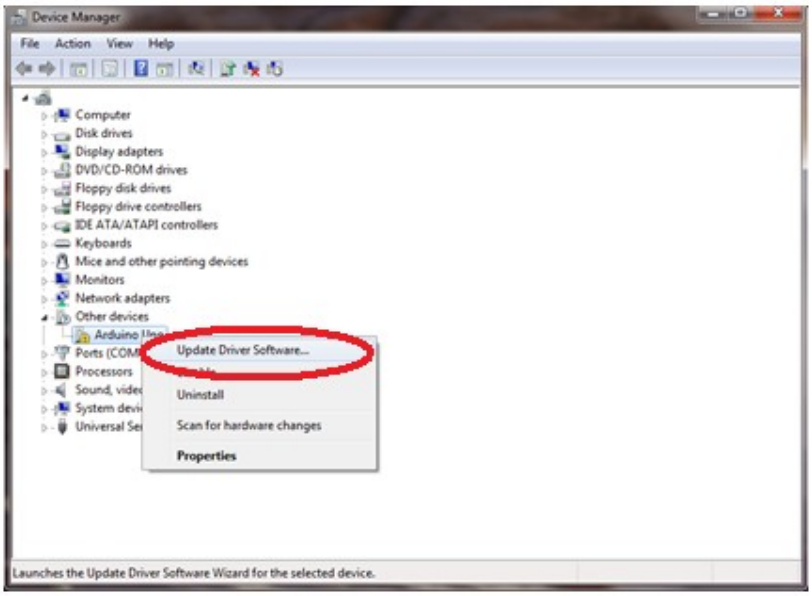


Figure 5: Screenshot of the Device Manager. The Arduino Uno should have an exclamation point.

3. Choose to browse your computer for the driver by clicking **Browse my computer for driver software** [2].

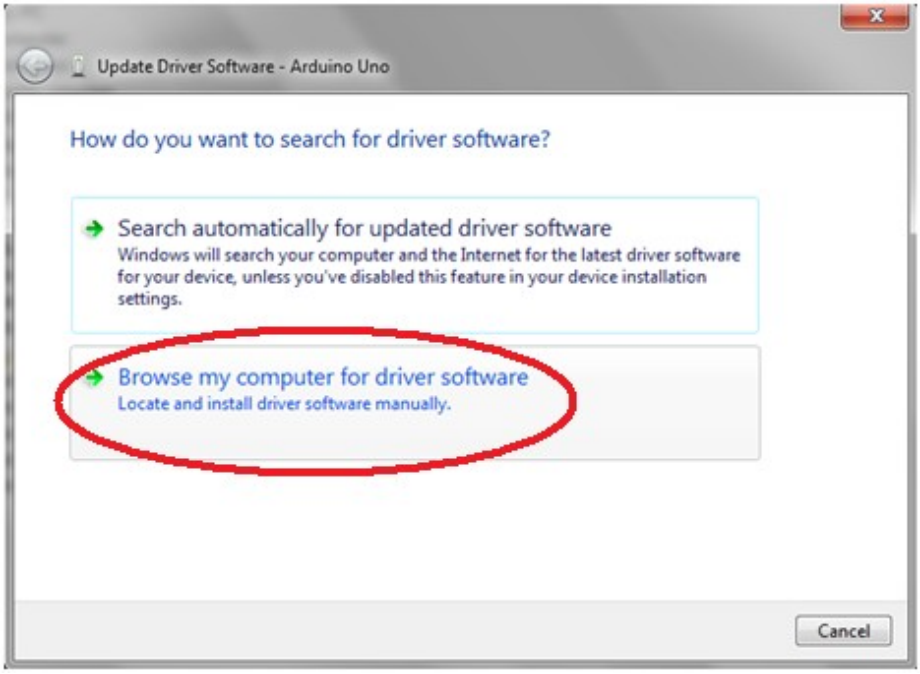


Figure 6: Screenshot of the options for searching the device driver. Choose the second option so that you can look for it in your hard disk.

4. A new window will open for you to indicate the location of the driver. Click **Browse…**

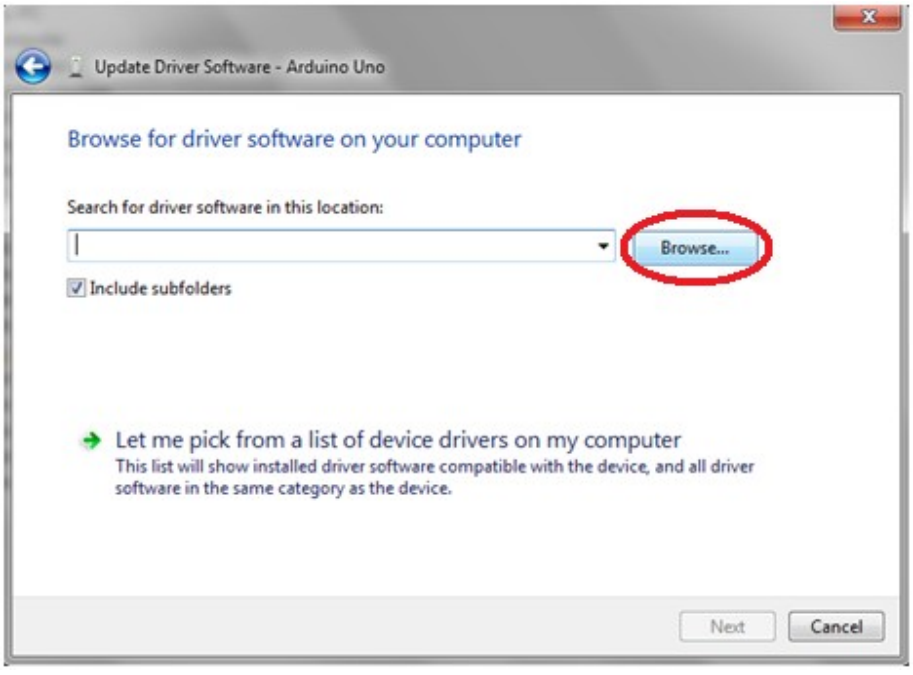


Figure 7: Screenshot of the browse option menu. Choose the first option which is to look manually for the folder that contains the Arduino Uno board's driver.

5. Navigate to your Arduino folder and choose the `drivers` folder. Click **OK** upon selection [2].

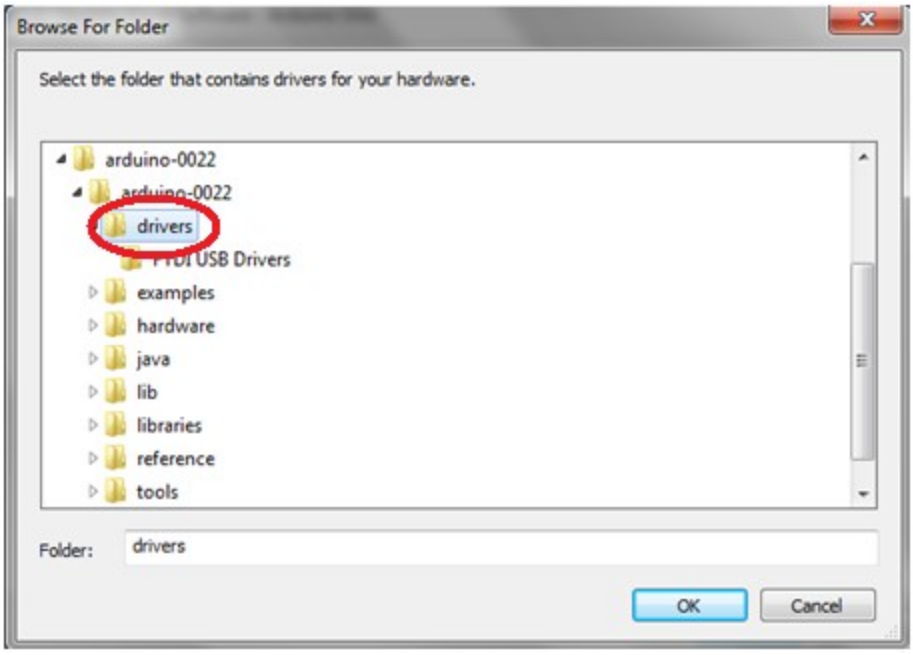


Figure 8: Screenshot of navigating through the Arduino software folder. Note that the `drivers` folder was chosen rather than the FTDI USB Drivers (It was mentioned earlier that only preceding models use this)

6. A Windows Security window sometimes pops up to confirm if you want to continue the installation. Just click, **Install this driver software anyway**.

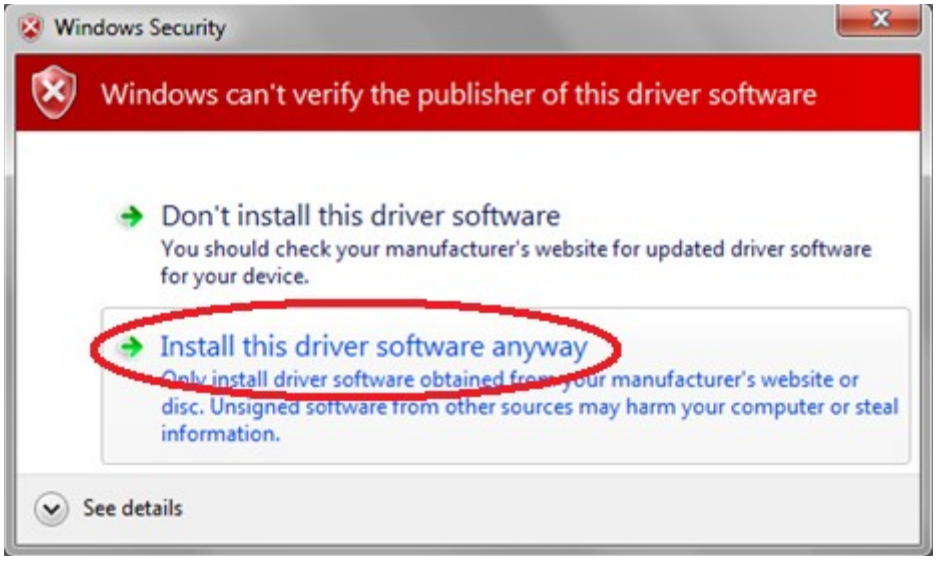


Figure 9: Screenshot of pop-up window. Windows can't verify the publisher of the device software but we know that the software's publisher is Arduino.

7. Wait for Windows to finish installing the device driver. Upon completion, you should see an installation successful message. Congratulations and click **Close**. You are ready to start programming using Arduino!

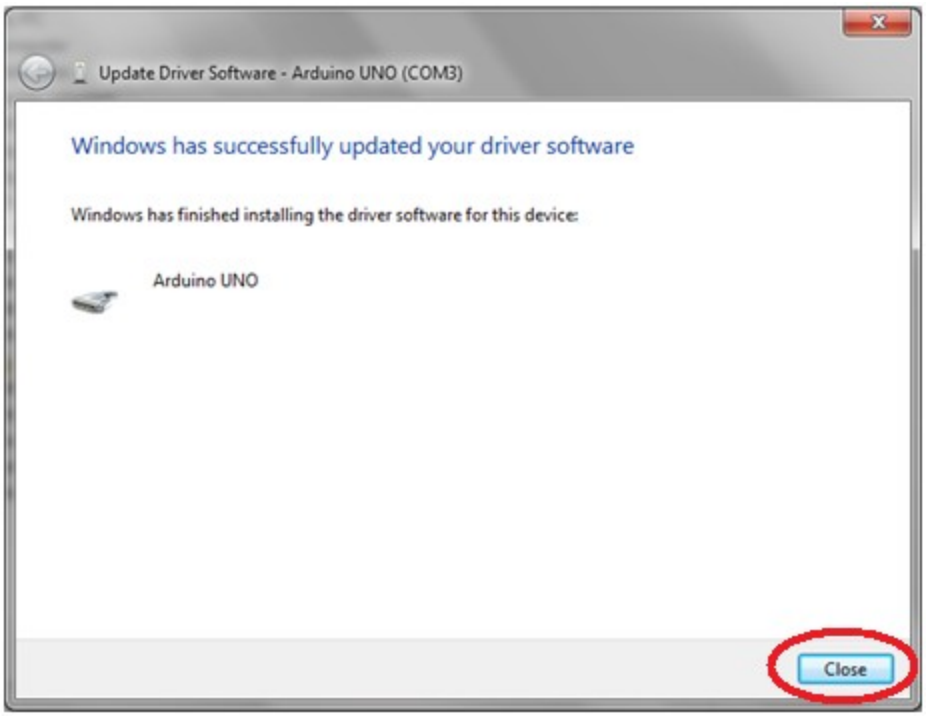


Figure 10: Screenshot of successful driver installation of the Uno board. The next step is to start doing the exercises.

Arduino Uno Board

A. Schematic Diagram

This section provides the schematic diagram of the Arduino Uno board.

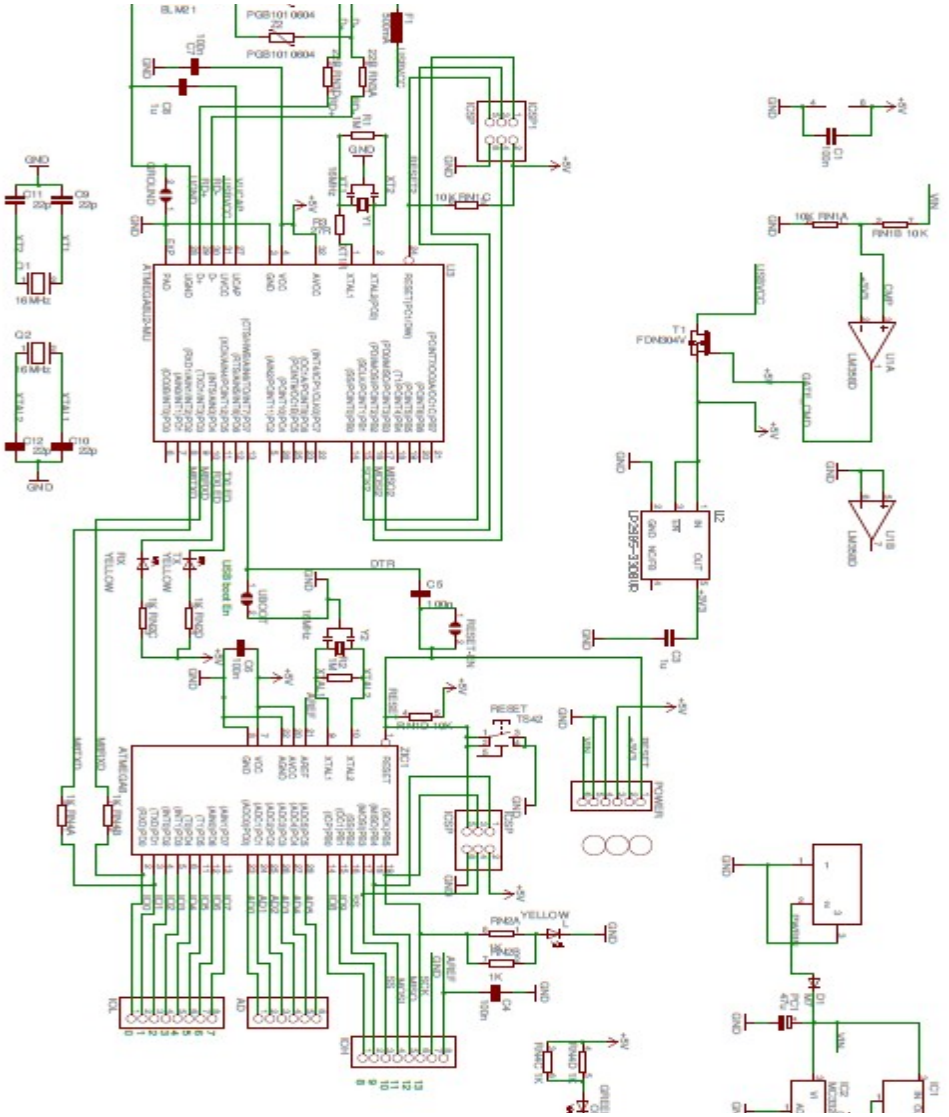


Figure 11: Schematic Diagram of the Arduino Uno board. It is possible to create the board by yourself just purchase the necessary components.

B. Component View

A pictorial view of the Arduino Uno board's peripherals can be found in this section.

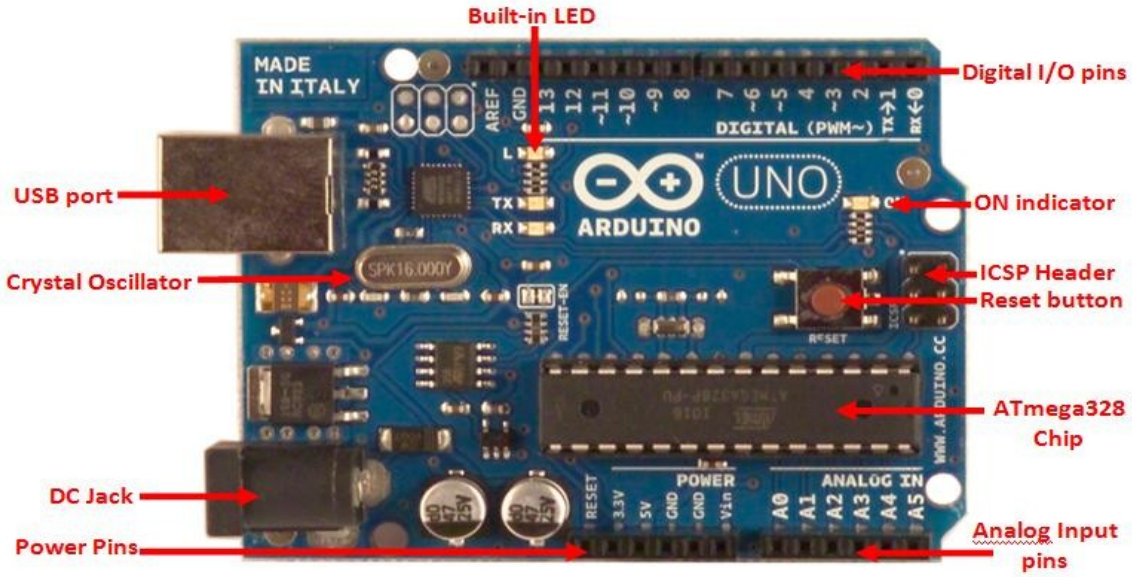


Figure 12: Photo of the Arduino Uno Board taken from [1]. The components are labelled and a brief explanation of each one is provided at the next part.

Component Explanations (taken from [1])

- ***Analog input pins*** – pins (A0-A5) that take-in analog values to be converted to be represented with a number range 0-1023 through an Analog to Digital Converter (ADC).
- ***ATmega328 chip*** – 8-bit microcontroller that processes the sketch you programmed.
- ***Built-in LED*** – in order to gain access or control of this pin, you have to change the configuration of pin 13 where it is connected to.
- ***Crystal Oscillator*** – clock that has a frequency of 16MHz
- ***DC Jack*** – where the power source (AC-to-DC adapter or battery) should be connected. It is limited to input values between 6-20V but recommended to be around 7-12V.
- ***Digital I/O pins*** – input and output pins (0-13) of which 6 of them (3, 5, 6, 9, 10 and 11) also provide PWM (Pulse Width Modulated) output by using the analogWrite() function. Pins (0 (RX) and 1 (TX)) are also used to transmit and receive serial data.
- ***ICSP Header*** – pins for "In-Circuit Serial Programming" which is another method of programming.
- ***ON indicator*** – LED that lights up when the board is connected to a power source.
- ***Power Pins*** – pins that can be used to supply a circuit with values V_{IN} (voltage from DC Jack), 3.3V and 5V.
- ***Reset Button*** – a button that is pressed whenever you need to restart the sketch programmed in the board.
- ***USB port*** – allows the user to connect with a USB cable the board to a PC to upload sketches or provide a voltage supply to the board. This is also used for serial communication through the serial monitor from the Arduino software.

Notations & Conventions

This section provides an explanation of the notations used throughout the manual.

- Code to be written in the sketch is written in `courier font.`
- Folder directories are also written in `courier font`.
- **Boldface font** may either indicate an instruction or a series of buttons and menus to access.
- *Italics* indicate tips and hints to achieve a certain goal.

NOTE: The file naming conventions used in this manual are as follows: unless otherwise specified, the filename of the final .pde files for each exercise will consist of the word “exercise” concatenated with the exercise number. Most of these exercises build on from previous ones, so it would be a good practice to use **File > Save As…** in saving your files to avoid unwanted overwriting.

An example – Introduction to the Arduino Uno board (Blinking LED)

A. Objectives

This example aims to familiarize the student with the basic operation of the Arduino Uno board, and Integrated Development Environment (IDE). By the end of the exercise, the student should be able to know the basic functionalities of the IDE.

B. Questions

- How do you compile and upload a sketch file into the Arduino Board using the IDE?
- How can the configuration of the pins be specified in the programming?
- What functions are always required in a sketch?

C. Materials Needed

- 1 x Arduino Uno board and USB cable
- 1 x 5mm/3mm Red LED
- 1 x 470Ω resistor
- 1 x breadboard

D. Using the Arduino IDE

1. Proceed to the Arduino Folder and double-click the arduino application to open the Arduino IDE which is also shown below:

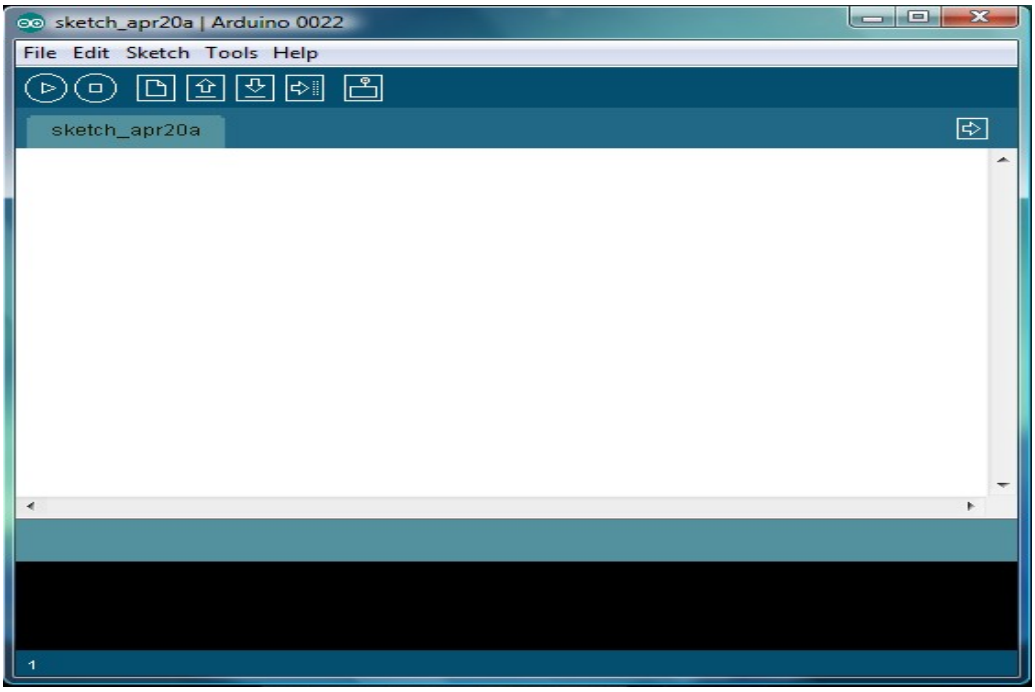


Figure 13: Screenshot of the Arduino IDE. The workspace is very simple and frequently used buttons are easy to access

2. Type the following lines of code on the front panel. The code can also be found in the examples folder of Arduino. To copy it, just navigate from **arduino-00XX > examples > Basics > Blink**. See Appendix under Structure, Functions and Time or Section F for the functions' definitions

```
1     void setup(){
2     pinMode(13,OUTPUT);
      }
3     void loop(){
4     digitalWrite(13,HIGH);
5     delay(1000);
6     digitalWrite(13,LOW);
7     delay(1000);
      }
```

3. Press **Ctrl + S** or  Icon on the IDE to save your sketch (*the term for a source code in Arduino*).

4. Click the Verify Button (A on fig. 14) to check if the sketch has no errors. At the bottom of the screen you should see if the sketch was successfully scanned free of problems (B on fig. 14). Moreover, the size of the sketch file (C on fig. 14) is also indicated on the black screen at the bottom to shows that it fits the flash memory of the Arduino Uno.

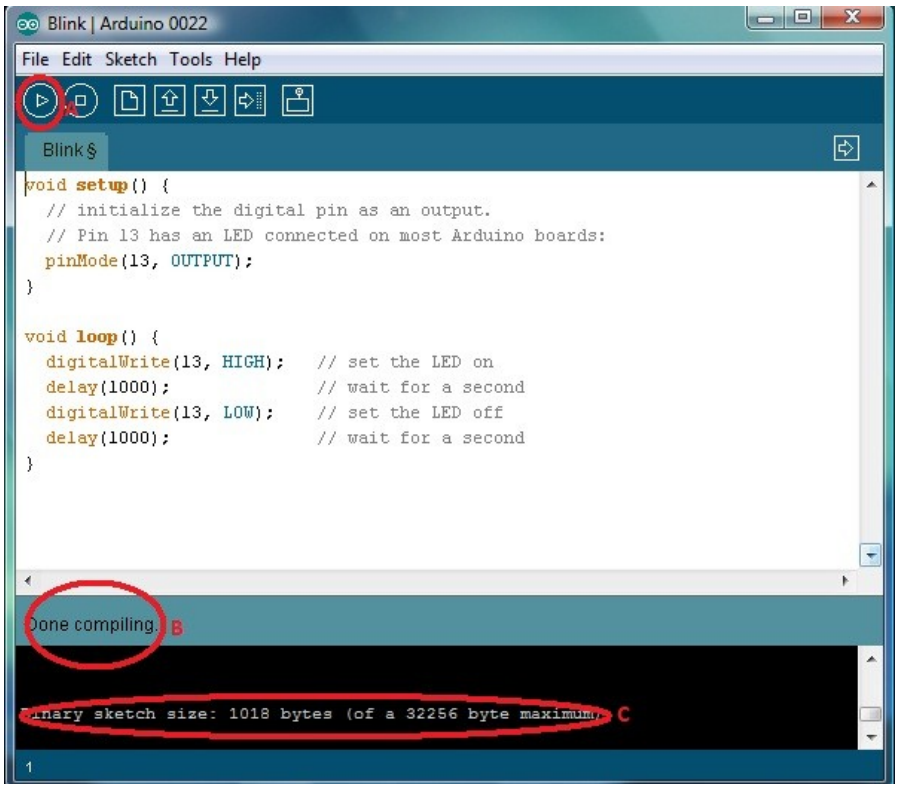


Figure 14: Screenshot of the Blink code after compilation. The verify button looks just like a play button and is located at the upper left corner. Successful compiling will be indicated at the bottom. If an error or problem occurred, it will be shown in the black screen.

E. Test your Program

1. Connect an LED to a 470Ω resistor. *Note: that the longer leg of the LED is the positive (connected to one end of resistor and the shorter one is the negative (connected to the ground)*. The other end of the resistor should then be connected to the specified pin which is 13.

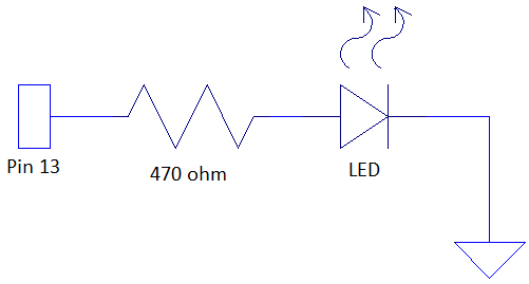


Figure 15: Schematic diagram of blink circuit.

2. Check if the board selected in the IDE is the one you are using, the Arduino Uno board.

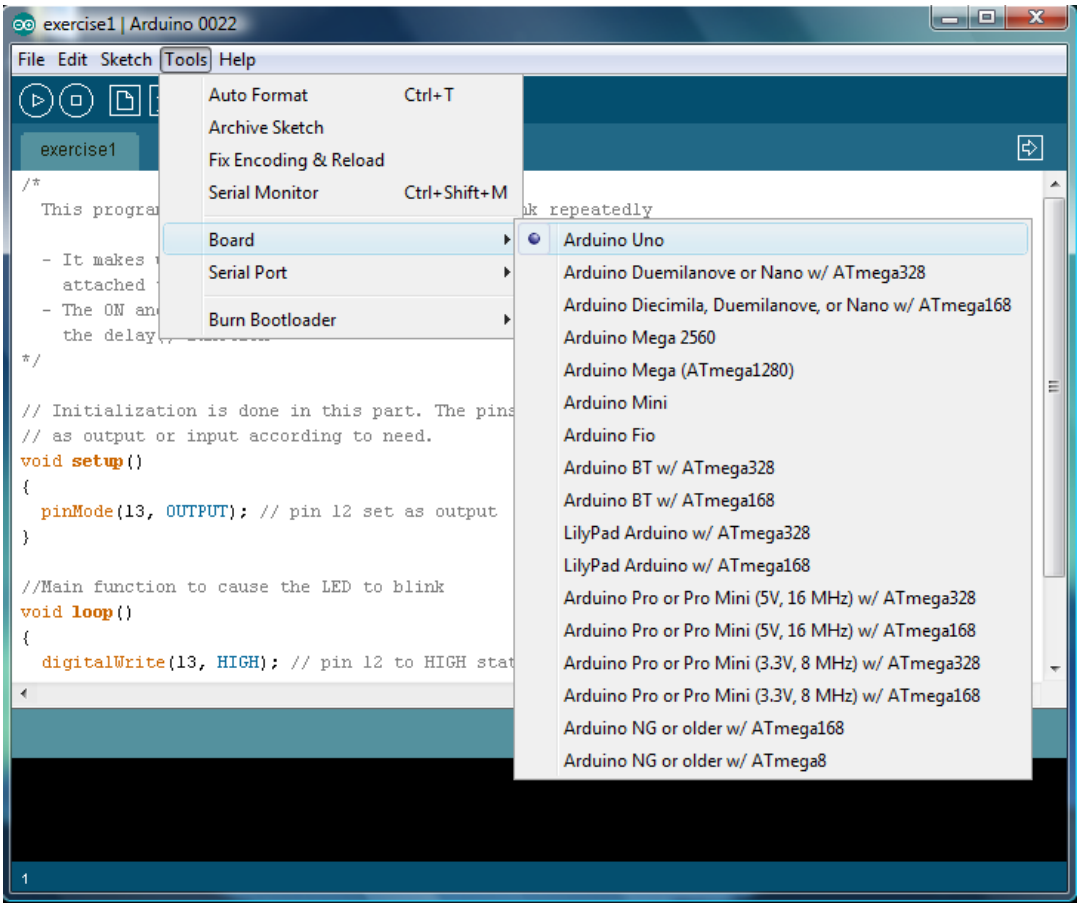


Figure 16: Screenshot showing Arduino Uno as selected board. If you have a different type of Arduino board, just select it from this list.

3. Next, make sure that the Arduino board's COM port is properly identified by the IDE. To do this, connect the board to the computer then from the IDE front panel go to **Tools > Serial Port** (A on fig. 17). A list of COM's should be there, choose the COM number associated with the microcontroller board. *Note: the microcontroller board's COM no. is found near the name of the device's driver in* ***Device Manager*** (B on fig. 17) [2].

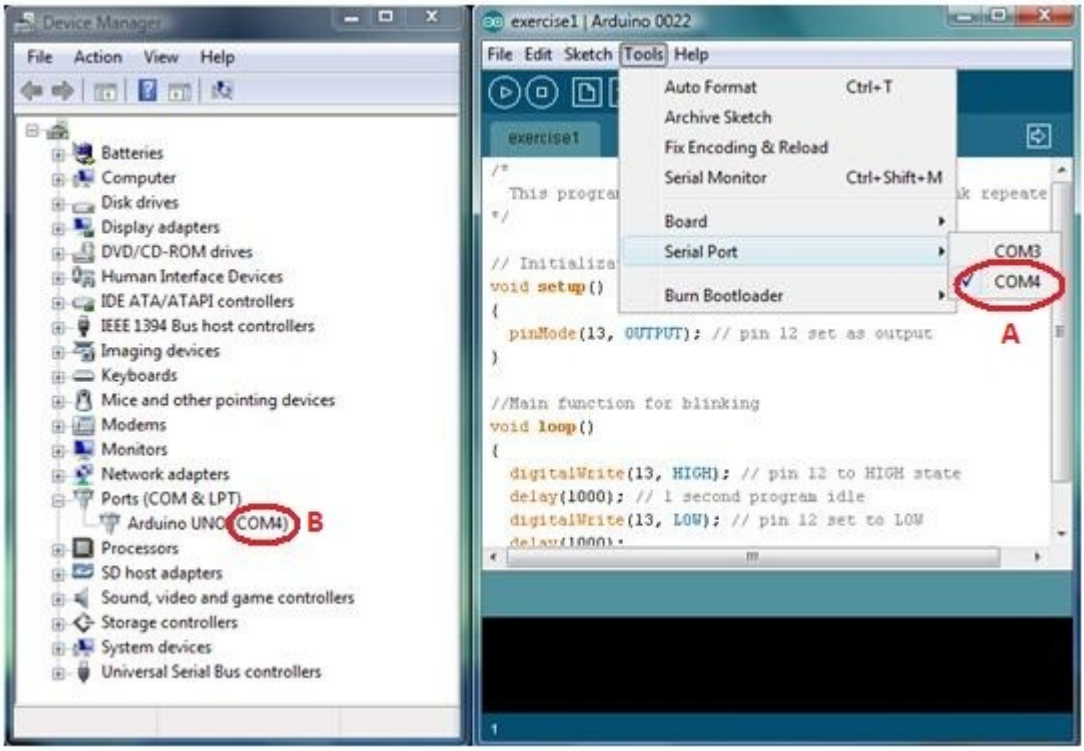


Figure 17: Screenshot of both the Device Manager and Arduino IDE windows. Note that the COM number selected as Serial Port must be the same one found beside the device's name

4. Finally, upload the sketch made earlier into the board by *clicking the*  *icon* in the IDE. The status of the upload can be found near the bottom of the program [2].

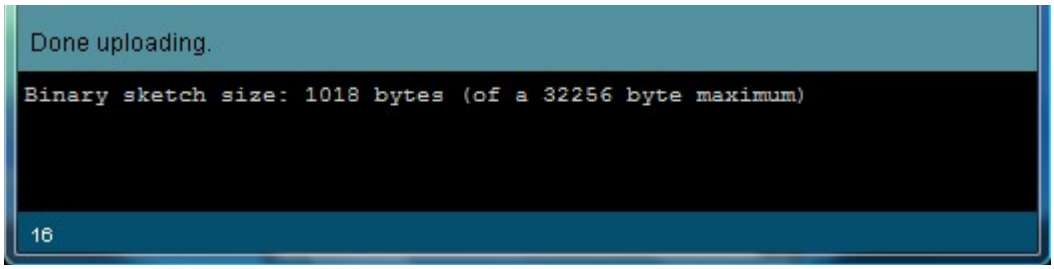


Figure 18: Screenshot showing a successful upload. Sometimes an error occurs because the COM port of the Arduino UNO was not properly specified. To solve this just follow Step 2.

5. If the Arduino Uno is connected to the PC or power source the LED should light up for 1 second and turn off for 1 second repeatedly.

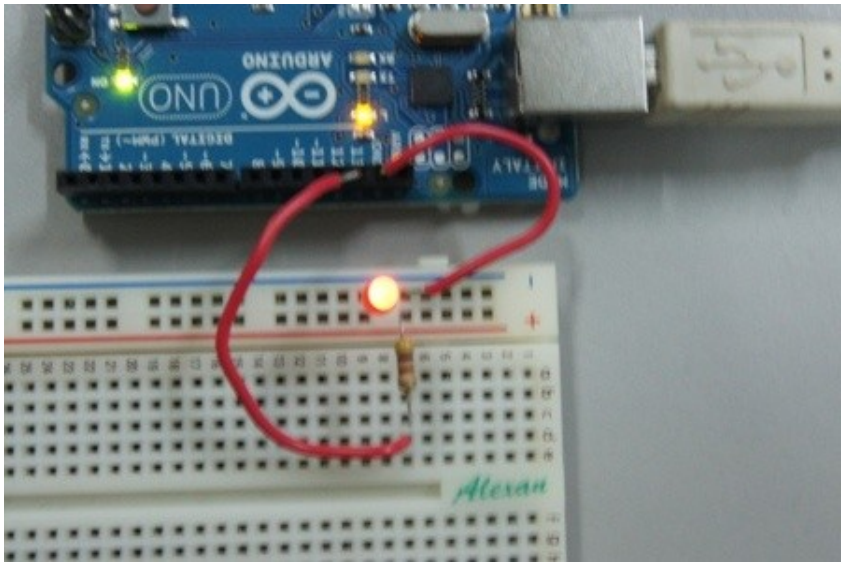


Figure 19: Photo of the LED blinking. Note that the small LED on the board is also blinking because it is also connected to pin 13.

F. Code explanations:

Line No.	Explanations
1	Creates a function called setup. The function type is void and therefore does not return any value after execution. This is only read once in a file execution and is required in a sketch.
2	The function pinMode() configures the specified pin which is 13 in this sample. It sets digital pin 13 as output.
3	Creates a function called loop. The function type is also void and therefore also does not return any value after execution. This is read repeatedly in a file execution and usually where the main program is. This is also required in a sketch.
4	The digitalWrite() function writes a high(5V) or low(0V) on a specified pin. In this case, a high state is written on pin 13 to light the LED.
5	The delay function needs a constant or variable of how long in millisecond the delay will be. This line prolongs for 1000 millisecond the LED at ON state.
6	Similar to line 4, pin 13 was set this time to low to turn off the LED.
7	This line is the same as line 5 and prolongs the LED at OFF state for 1 second.

G. What to expect

You have just written a sketch and programmed the Arduino Uno board to cause both the external LED and built-in LED connected to pin 13 to blink. You should see the

LED blink when the board is connected to the PC by a USB cable or the board is connected to a DC supply.

H. Summary

In this exercise, you have learned how to create .pde files or sketches using the Arduino IDE. You have also learned how to access and control – specifically, turn ON and OFF –an LED connected to pin 13 of the Arduino Uno Board.

Exercise 1 – Basic I/O: Dancing LED operation using a tact switch

A. Objectives

This exercise aims to further expand the student's knowledge with the basic input/output operation of the Arduino Uno board, and Integrated Development Environment (IDE) by developing the program learned for enabling an LED to blink. In the end of this exercise, by also introducing an external input (a tact switch or pushbutton), the student should have learned how to program to read inputs and to produce outputs in response to that input.

B. Questions

- In circuit theory, how is a tact switch represented when it is pressed and not?
- What function enables you to read the input from a pin?
- What function enables you to program efficiently the lighting up of several LED's?

C. Materials Needed

- 1 x Arduino Uno board and USB cable
- 4 x 5mm/3mm Red LEDs
- 4 x 470Ω resistors and 1 x 10kΩ resistor
- 1 x breadboard
- 1 x tact switch or toggle switch (which ever is available or simply a pull-up wire)

D. Instructions to File Management

1. Open the previous sketch. To do this, select **File > Open…** or the  button located in the Arduino program**.** Navigate to where you saved the .pde file, select it and click **Open**.
2. Create a copy of this sketch and rename it to correspond to this new exercise by selecting **File > Save As…**. By doing this you should now have a new .pde file that contains the code you made during the previous exercise.

E. Instructions to programming the dancing LED's.

1. Modify the existing code so that instead of only 1 pin, 4 pins are configured to accommodate the 4 LEDs. *Note that you have two main functions so choose wisely where you put the initialization and working program*. Add another configuration for a pin to be connected to the tact/toggle switch.
2. Create the part of the program that will read the switch's state. *Be sure to understand which nodes of the tact/toggle switch are already shorted and not*.
3. Make the program that would perform the dancing LED's when the program recognizes the button press. The program must not execute the light effect until the user releases the button. The dancing LED is composed of 4 LED's that will light up 1 at a time starting from LED1 to LED4 and afterwards from LED4 back to LED1 Repeatedly. *Code the program efficiently by using looping functions.*
4. Once you are done writing the code, save your file and compile it.

F. Test your program

1. Construct the circuit using the Arduino Uno, Breadboard, LEDs, resistors and tact/toggle switch.
2. Connect the board to the PC using the USB cable. Upload the sketch saved earlier and check if the program works. The LEDs should behave as previously described in Step 3 of the instructions in programming the dancing LED's. *Note that the board should not perform the blinking LED exercise anymore since it was overwritten; otherwise, the new program wasn't uploaded properly. Refer to the previous the example.*

G. What to expect

You have just written a sketch and programmed the Arduino Uno board to create dancing LEDs. You should see the LEDs turn ON one at a time forward and backward when the board is connected to the PC via USB cable or when the board is connected to a DC supply.

H. Summary

In this exercise, you have learned how to create more complicated sketches using the Arduino IDE. You have also learned how to access and control – specifically, create a running lights effect in response to a button press or a toggle switch using the Arduino Uno Board.

Exercise 2 – Basic I/O: Traffic Light System Design (Delay Construction)

A. Objectives

This exercise aims for the student to learn the use and necessity to pause programs in some applications. The exercise will simulate a traffic light system for vehicles and pedestrians. By the end of this exercise, the student should have learned how to control delay and timing to produce desired effects and prevent data flooding.

B. Questions

- How would you change the amount of time each LED is ON?
- Why is it necessary to cause some delay in particular applications like for example, reading values from a sensor?
- In how many ways or units are there to configure the delay?

C. Materials Needed

- 1 x Arduino Uno board and USB cable
- 5 x 5mm/3mm LEDs (2 Red, 2 Green, 1 Orange)
- 5 x 470Ω resistors and 1 x 10kΩ resistors
- 1 x breadboard
- 1 x tact switch or toggle switch (which ever is available or simply a pull-up wire)

D. Instructions to File Management

1. Open the previous sketch.
2. Create a copy of this sketch and rename it to correspond to this new exercise.
3. Recall the previous laboratory exercises on how to do this.

E. Instructions to Programming the Traffic Light System (Delay).

1. Modify the existing code so that each LED lights up for different time lengths similar to a traffic light system. The green LED2 (for pedestrians) and red LED1 (for vehicles) lights for 4 seconds. Then both the green LED1 (for vehicles) and red LED2 (for pedestrians) for 5 seconds. After that the orange LED turns ON for 1 second and finally, the green LED2 (for pedestrians) and red LED1 (for vehicles) lights for 4 seconds. *Reduce the number of pins to be used in the program and arrange the lights from stop light (vehicle) first to go Light (vehicle).*
2. The program should still respond or activate only after the tact/toggle switch has been pressed and released (or toggled).
3. Also, construct the program such that it stops after 3 cycles and activates only after another button press and release has been made (toggled).
4. Once you are done writing the code, save your file and compile it.

F. Test your program

1. Construct the circuit using the Arduino Uno, breadboard, LED's, resistors and tact/toggle switch.

2. Connect the board to the PC using the USB cable. Upload the sketch saved earlier and check if the program works. The LEDs should behave similar to a Traffic Light System two times and the program stops to wait for activation again.

G. What to expect

You have just written a sketch and programmed the Arduino Uno board to control a Traffic Light System. After pressing the tact switch, you should see red LED1 (for stop vehicle) and green LED2 (for go pedestrian) light up for 4 seconds. Afterwards they turn OFF, the red LED2 (for stop pedestrian) and green LED1 (for go vehicle) turns ON for 5 seconds and turns OFF. The orange LED turns ON for 1 second and finally, the red LED1 (for stop vehicle) and green LED2 (for go pedestrian) light up for 4 seconds. This cycle repeats also for 2 more times. Afterwards, when you press and release the (toggled) switch again, it should repeat the whole process.

H. Summary

In this exercise, you have learned how to use the Arduino Board and IDE to create and control a circuit that represents a system. You have also learned how to control delay and stop programs by manipulating the time of each LED and halting the program from repeatedly functioning.

Exercise 3 – Basic I/O: Light Level Indicator with Light Dependent Resistor

A. Objectives

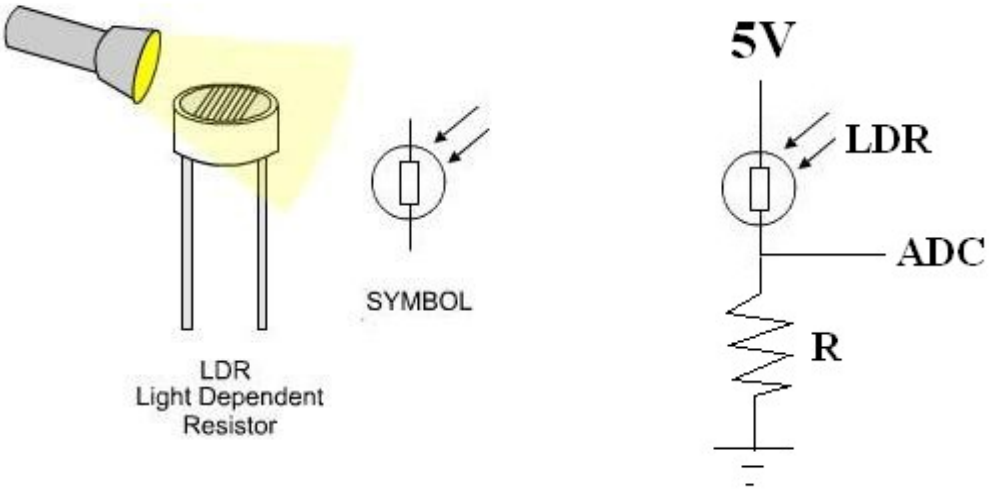
This exercise aims for the student to apply the basic I/O methods learned from the previous experiments with the ADC. As such, the exercise will also serve as an introductory to Analog-to-Digital Conversion of the Arduino Uno. Here, a Light level indicator will be made with an LDR. By the end of this exercise, the student should have a good grasp of the I/O features of the board and understand the ADC function of the Analog Input pins.

B. Questions

- How does the LDR Work? Does the resistance increase as it receives more light?
- What is the necessary function to read analog inputs?
- How does the ADC of the analog input pins process the voltages it receives?
- What estimated range values is needed for the program to determine the light level?

C. Materials Needed

- 1 x Arduino Uno board and USB cable
- 4 x 5mm/3mm LEDs
- 4 x 470Ω resistors and 1 x 2.2kΩ (= R) resistor
- 1 x breadboard
- 1 x Light Dependent Resistor (LDR)



D. Introduction to Arduino Uno's Analog-to-Digital Converter

The Arduino Uno board has exactly 6 pins that can read analog values. With the analog pins, you process values with more significant digits that are needed in applications.

Now, the analog input pins of the board are connected to an Analog-to-Digital Converter that converts the 0-5V to a range of 0-1023 unit. This means that around 4.88mV (i.e. 5V/1024) is equivalent to 1 unit. The voltage range to be compared can also be changed by using the analogReference(type) function which is explained in the Appendix. As a visual explanation:

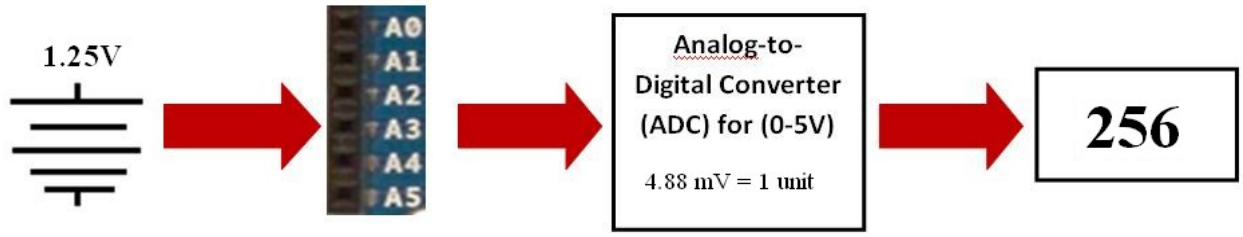


Figure 20: Diagram showing how a 1.25V input to the analog pin is read as 256 units.

When a 1.25V source is connected to one analog input pin, the ADC converts it in unit which is by the conversion ratio mentioned equal to 256 (=1.25V x 1024 / 5V). Now, to activate the ADC of the analog input pins, it is necessary to use the analogRead(pin) function. Look at the appendix for an explanation regarding this.

E. Instructions to Programming the LED lamp using an LDR as a switch

1. Open Arduino, and create first the part that would read the analog input coming from one of the analog input pins.
2. Now, construct a conditional statement for the ranges wherein the number of LEDs turned ON correspond to how less the light received is. This means 4 LEDs light up in a dark room while none in a well lit room. *Note to study the behavior of an LDR and determine the relationship of the resistance and the light it receives*. *Expect some calibrating to be done.* Also, remember that the LDR is connected to the built-in 5V source. The less the LDR's resistance the higher the voltage the analog pins receive.
3. Once you are done writing the code, save your file and compile it.

F. Test your program

1. Construct the circuit using the Arduino Uno, breadboard, LED, resistors, and LDR.
2. Connect the board to the PC using the USB cable. Upload the sketch saved earlier and check if the program works. The 4 LEDs should light up in a dark room while none in a well lit room. *If it did not work, some calibration to the ranges specified might be needed.*

G. What to expect

You have just written a sketch and programmed the Arduino Uno board to control an LED light level indicator based on the amount of light the sensor (LDR) receives. After covering the LDR with an object or your hand, you should see the 4 LEDs turn ON and upon uncovering, the number of lit LED's decrease.

H. Summary

In this exercise, you have learned how to use the Arduino Board and IDE to control outputs based on results from a sensor. You have also learned how the ADC feature of the analog input pins works by controlling the number of LED's turned ON based on the amount of light the LDR receives.

Exercise 4 – Basic Serial: “Hello World”

A. Objectives

This exercise aims for the student to learn how to use the serial feature of the Arduino Board and IDE. As such, the exercise will also serve as an introductory for the students on using the serial to output symbols and texts based on a program that later will be used to read data values from sensors. By the end of this exercise, you should be able to output a message onto the serial monitor of the Arduino IDE.

B. Questions

- How does the Arduino Board to communicate with a PC?
- What are the functions usually used in a serial communication?
- What is the necessary initialization for serial communication to take place?

C. Materials Needed

- 1 x Arduino Uno board and USB cable

D. Introduction to Arduino’s Serial COM

The Arduino Uno board can communicate with a PC, another Arduino or other microcontrollers. The board’s microcontroller chip, ATmega328, provides the Universal Asynchronous Receiver/Transmitter (UART) TTL (5V) serial communication which is accessed by digital pins 0 (RX) and 1 (TX) to communicate with other devices.

In addition, the ATmega8U2 chip on the board also channels this serial communication over USB that appears as a COM port for the board to communicate with the computer. This method is done by connecting the Arduino Uno Board via USB cable.

The Arduino IDE has a serial monitor which can be used to send to and from the Arduino Uno board textual data.

The serial communication through USB is manipulated by functions in a sketch. Note that a Serial.begin(speed) function is needed in the initialization. *The speed is matched with that found in the serial monitor of the Arduino IDE which is usually or by default 9600*. To access the serial monitor, just click the button right of the upload to I/O device button. By doing this, the serial communication is activated with a given reading speed.

Functions such as Serial.read(), Serial.available(), Serial.write() and Serial.print() are used for reading and writing in a serial communication system. Serial functions read inputs and produce outputs in bytes. For example, the Serial.available() function returns the number of bytes available to read. The serial.read() outputs the first byte of incoming data. The serial.write() outputs series of byte data but most of the time, users wants these data translated to or stay in readable text which is why Serial.print() is used more as it outputs them in ASCII text. Refer to the Appendix for explanations on these and other serial functions.

E. Instructions to programming “Hello World!”

1. Open Arduino, and create first the part that would read if there are any inputs by the user in the serial monitor.
2. Continue the code such that upon receiving exactly 3 ready to be read character (bytes) inputs that are simultaneous, a “Hello World!” sentence should appear on the serial monitor as output. *Note that the program should not output anything if the condition is not met.*
3. Construct an algorithm that would reduce the available bytes to read back to 0. *This is important to stop the program from outputting the sentence endlessly or from freezing*. *Remember that the number of bytes available to be read is reduced when bytes are read.*
4. Once you are done writing the code, save your file and compile it.

F. Test your program

1. Connect the Arduino board to the PC using the USB cable.
2. Upload the sketch saved earlier and check if the program worked. Open the Serial Monitor by clicking the icon right of the upload icon in the IDE. Type 3 characters in the new window and see if it would output a “Hello World”. Type more or less number of characters and the program should remain idle.

G. What to expect

You have just written a sketch that programmed the Arduino Uno board to use the serial monitor to allow the board and PC to communicate. Typing 3 characters in the serial Monitor should output a “Hello World!” otherwise the program remains idle.

H. Summary

In this exercise, you have learned how to communicate the Arduino Board PC. You also understood how to use the IDE’s serial monitor to read and write data which is an important skill in data analysis or microcontrollers.

APPENDIX

A. Arduino IDE Language Summary (taken from http://arduino.cc/en/Reference/HomePage)

Structure

- setup() – is called once when a sketch starts. Used to initialize variables, pin modes, start using libraries, etc.
- loop() - as its name implies, this function loops consecutively the code inside it to allow your program to respond and change.

Variables

Constants

- HIGH | LOW – when a pin is in INPUT mode, the microcontroller reports HIGH as a voltage of 3V or more and LOW if it is 2V or less. Also, the user can write HIGH while the pin is in INPUT mode and this will set the internal 20K pull-up resistor which will steer the pin to HIGH reading unless pulled LOW by external circuitry. When a pin is in OUTPUT mode, HIGH sets the pin to 5V and LOW sets it to 0V.
- INPUT | OUTPUT – configures the pins to take in input from a circuit or provide output to a circuit respectively.

Functions

Digital I/O

- pinMode(pin, mode) – configures the pin to behave as either an input or an output
  - pin – refers to the number of pin to be configured.
  - Mode – refers to what mode the pin is to be set (INPUT or OUTPUT).
- digitalWrite(pin, value) – writes a high (5V) or low (0V) value to a digital pin.
  - value – HIGH or LOW
- digitalRead(pin) – reads the value from a specified digital pin if it is high or low.

Analog I/O

- analogReference(type) – configures the reference voltage used for analog input.
  - type – can be any of the ones below:
    - DEFAULT – the default analog reference of 5V or 3.3V depending on board.
    - INTERNAL – uses the built-in reference available to some microcontrollers.
      - INTERNAL1V1 – uses built-in 1.1V reference (Arduino Mega only)
      - INTERNAL2V56 – uses built-in 2.56V reference (Arduino Mega only)
    - EXTERNAL – uses voltage applied to the AREF pin (0 to 5V only)
- analogRead(pin) – reads the value of the specified analog pin. The voltage is converted to an integer from 0 to 1023.
- analogWrite(pin, value) – writes an analog value (PWM wave) to a pin.
  - value – the duty cycle and is between 0 (always off) and 255 (always on).

Time

- millis() – returns the number of milliseconds since the Arduino board began running the current program.
- micros() – returns the number of microseconds since the Arduino board began running the current program.
- delay(ms) – pauses the program for the amount of time specified as parameter.
  - ms – the number of milliseconds to pause.
- delayMicroseconds(us) – same as delay but with a different unit of time.
  - us – the number of microseconds to pause.

Math

- min(x, y) – returns the smaller number
  - x – to be compared with y
  - y – to be compared with x
- max(x,y) – returns the larger number
  - x – to be compared with y
  - y – to be compared with x
- abs(x) – computes the absolute value of a number
  - x – the number
- constrain(x, a, b) – limits the number in the range

    - x – the variable or number to be limited
    - a – x is set to this value if x<a
    - b – x is set to this value if x>b
- map(value, fromLow, fromHigh, toLow, toHigh)
    - value – the number to be mapped
    - fromLow – minimum of the value's range
    - fromHigh – maximum of the value's range
    - toLow –minimum of range the value will be mapped
    - toHigh – maximum of the range the value will be mapped
- pow(base, exponent) – returns the result of the exponent equation
    - base – the base number
    - exponent – the exponent of the base
- sqrt(x) – returns the square root of x.
    - x – the number that will be sought of its square root.
- sq(x) – returns the square of x.
    - x – the number that will be sought of its squared value.

Communication

Serial

- Serial.begin(speed) – sets the data rate for serial transmission.
    - speed – the data rate in data bits/second (baud)
- Serial.end() – disables serial communication to allow the RX and TX pins (digital pin 0 and 1) for general I/O use.
- Serial.available() – get the number of bytes(characters) available for reading from the serial port which is data stored already in the internal serial buffer.
- Serial.read() – reads incoming data and returns the first byte of data or -1 if there's none.
- Serial.peek() – returns the next byte (character) of incoming serial data without removing it from the internal serial buffer which means that another call to this function will still output the same byte.
- Serial.flush() – flushes the buffer of incoming serial data such that the data that will be obtained is data received after all the most recent call to serial.flush().
- Serial.print() – prints data to serial port as human-readable text or ASCII text.
- Serial.println() – same as Serial.print() but output ends with a return character.
- Serial.write(input) – writes data to the serial port. Data is sent in bytes.
    - input – can be in val, str, or buf and len.
        - val – value to be send as a single byte
        - str- sring to be send as a series of bytes
        - buf – array to be send as series of bytes
        - len – length of the buffer.

Libraries

Servo

- Servo.attach(pin, min, max) – attaches the servo variable to a pin on the board.
    - min – (optional) is the PWM (in microsecond) value that will correspond to the minimum 0-degrees.
    - max – (optional) is the PWM (in microsecond) value that will correspond to the maximum 180-degrees.
- Servo.write(angle) – positions the servo motor's shaft to specified angle
    - angle – (in degrees) the angle the shaft will be positioned.
- Servo. writeMicroseconds(uS) – same as the function write() but asks an input in microseconds
    - uS – the value in microsecond to turn the shaft into a specific degree
- Servo. read() – reads the current angle value of the servo.
- Servo.attached() – checks if the servo is attached to a pin
- Servo.detach – releases the servo from specified pin