LAB MANUAL

DEPARTMENT: ARTIFICIAL INTELLIGENCE & DATA SCIENCE

CLASS: S.E. SEMESTER: II

Subject Name: Internet of things Laboratory (217529)

INDEX OF LAB EXPERIMENTS

PART-I

Lab Expt .No	Problem Definition/Statement	Date Of Completion
	Group A	
1.	Study of Raspberry-Pi/ Beagle board/ Arduino and other microcontroller (History & Elevation)	
2.	Study of different operating systems for Raspberry-Pi /Beagle board/Arduino. Understanding the process of OS installation	
3	Study of different GATES (AND, OR, XOR), Sensors and basic binary operations.	
4	Study of Connectivity and configuration of Raspberry-Pi /Beagle board/Arduino circuit with basic peripherals like LEDS. Understanding GPIO and its use in the program.	
	Group B	
5	Write a program using Arduino to control LED (One or more ON/OFF). Or Blinking	
6	Create a program that illuminates the green LED if the counter is less than 100, illuminates the yellow LED if the counter is between 101 and 200 and illuminates the red LED if the counter is greater than 200	
7	Create a program so that when the user enters ‘b’ the green light blinks, ‘g’ the green light is illuminated ‘y’ the yellow light is illuminated and ‘r’ the red light is illuminated	
8	Write a program that asks the user for a number and outputs the number squared that is entered	
9	Write a program to control the color of the LED by turning 3 different potentiometers. One will be read for the value of Red, one for the value of Green, and one for the value of Blue	
10	Write a program read the temperature sensor and send the values to the serial monitor on the computer	
11	Write a program so it displays the temperature in Fahrenheit as well	
	as the maximum and minimum temperatures it has seen	
12	Write a program to show the temperature and shows a graph of the recent measurements	

13	Write a program using piezo element and use it to play a tune after someone knocks	
14	Understanding the connectivity of Raspberry-Pi /Beagle board circuit / Arduino with IR sensor. Write an application to detect obstacle and notify user using LEDs	
	Group C	
15	Study of ThingSpeak – an API and Web Service for the Internet of Things (Mini Project: Same can be done parallel with PBL)	
16	Write an application to control the operation of hardware simulated traffic signals. (Mini Project: Same can be done parallel with PBL)	
17	Develop a Real time application like smart home with following requirements: When the user enters into the house the required appliances like fan, light should be switched ON. Appliances should also get controlled remotely by a suitable web interface. The objective of this application is that students should construct complete Smart applications in groups. (Mini Project: Same can be done parallel with PBL)	
18	Write an application for stopwatch or countdown timer. (Mini Project: Same can be done parallel with PBL)	

Subject Coordinator **Head of Department**

Group A

EXPERIMENT NO. 1 (Group A)

Aim: Study of Raspberry-Pi/ Beagle board/ Arduino and other microcontroller (History & Elevation)

Outcome: At end of this experiment, student will be able to study of different microcontrollers like Raspberry-Pi/ Beagle board/ Arduino

Hardware Requirement: Arduino

Software Requirement: Arduino IDE

Theory:

Internet of Things: - IoT is short for Internet of Things. The Internet of Things refers to the evergrowing network of physical objects that feature an IP address for internet connectivity, and the communication that occurs between these objects and other Internet-enabled devices and systems. The Internet of Things (IoT) refers to the use of intelligently connected devices and systems to leverage data gathered by embedded sensors and actuators in machines and other physical objects. In other words, the IoT (Internet of Things) can be called to any of the physical objects connected with network.

Examples of IoT: -

1) Apple Watch and Home Kit.
2) Smart Refrigerator.
3) Smart Refrigerator.
4) Smart cars.
5) Google Glass.
6) Smart thermostats.

A) Raspberry-Pi:- The **Raspberry Pi** is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote teaching of basic computer science in schools and in developing countries. It does not include peripherals (such as keyboards and mice). The **Raspberry Pi** is a low cost, credit-card sized computer that plugs intoa computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. The Raspberry Pi is a credit-card-sized computer that costs between $5 and $35. It's available anywhere in the world, and can function as a proper desktop computer or be used to build smart devices. A Raspberry Pi is a general-purpose computer, usually with a Linux operating system, and the ability to run multiple programs. Raspberry Pi is like the brain. Its primary advantage comes in processing higher level processing capability. It's a single board computer.

Fig.A.1: - **Raspberry-Pi**

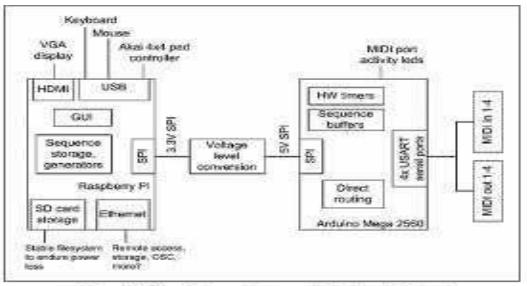


Fig. A.2: -**Raspberry-Pi Architecture**

Here are the various components on the Raspberry Pi board:

· **ARM CPU/GPU** -- This is a Broadcom BCM2835 System on a Chip (SoC) that's made up of an ARM central processing unit (CPU) and a Video core 4 graphics processing unit (GPU). The CPU handles all the computations that make a computer work (taking input, doing calculations and producing output), and the GPU handles graphics output.

· **GPIO** -- These are exposed general-purpose input/output connection points that will allow the real hardware hobbyists the opportunity to tinker.

· **RCA** -- An RCA jack allows connection of analog TVs and other similar output devices. · Audio out -- This is a standard 3.55-millimeter jack for connection of audio output devices such as headphones or speakers. There is no audio in.

· **LEDs** -- Light-emitting diodes, for your entire indicator light needs.

· **USB** -- This is a common connection port for peripheral devices of all types (including your mouse and keyboard). Model A has one, and Model B has two. You can use a USB hub to expand the number of ports or plug your mouse into your keyboard if it has its own USB port.

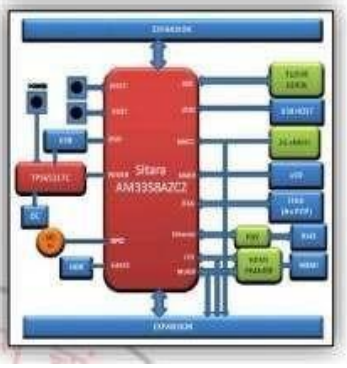
· HDMI -- This connector allows you to hook up a high-definition television or other compatible device using an HDMI cable.

· **Power** -- This is a 5v Micro USB power connector into which you can plug your compatible power supply.

· **SD card slot** -- This is a full-sized SD card slot. An SD card with an operating system (OS) installed is required for booting the device. They are available for purchase from the manufacturers, but you can also download an OS and save it to the card yourself if you have a Linux machine and the wherewithal.

· **Ethernet** -- This connector allows for wired network access and is only available on the Model B.

B) Beagle board: - The Beagle Board is a low-power open-source single-board computer produced by Texas Instruments in association with Digi-Key and Newark element14. The BeagleBoard was also designed with open source software development in mind, and as a way of demonstrating the Texas Instrument's OMAP3530 system-on-a-chip.] The board was developed by a small team of engineers as an educational board that could be used in colleges around the world to teach open source hardware and software capabilities. It is also sold to the public under the Creative Commons share-alike license. The board was designed using Cadence OrCAD for schematics and Cadence Allegro for PCB manufacturing; no simulation software was used. Beagle Bone Black is a low-cost, open source, community-supported development platform for ARM® Cortex™-A8 processor developers and hobbyists. Boot Linux in under 10-seconds and get started on Sitara™ AM335x ARM Cortex-A8 processor development in less than 5 minutes with just a single USB cable.



Here are the various components on the Beagle board:

Processor: AM335x 1GHz ARM® Cortex-A8

· 512MB DDR3 RAM
· 4GB 8-bit eMMC on-board flash storage
· 3D graphics accelerator
· NEON floating-point accelerator
· 2x PRU 32-bit microcontrollers

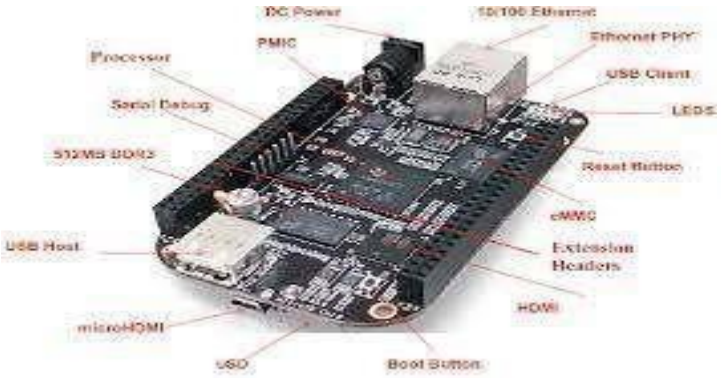
Connectivity

· USB client for power & communications
· USB host
· Ethernet
· HDMI
· 2x 46 pin headers

Software Compatibility

· Debian
· Android
· Ubuntu
· Cloud9 IDE on Node.js w/ BoneScript library
· plus, much more

C) Arduino: - Arduino is an open-source hardware and software company, project and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices and interactive objects that can sense and control objects in the physical and digital world. Arduino board designs use a variety of microprocessors and controllers. The boards are equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards or breadboards (shields) and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs from personal computers. The microcontrollers are typically programmed using a dialect of features from the programming languages C and C++. In addition to using traditional compiler tool chains, the Arduino project provides an integrated development environment (IDE) based on the Processing language project. Arduino is open-source hardware. The hardware reference designs are distributed under a Creative Commons Attribution Share-Alike 2.5 license and are available on the Arduino website. Layout and production files for some versions of the hardware are also available.



Here are the various components on the Arduino board:

Microcontrollers

ATmega328P (used on most recent boards)

ATmega168 (used on most Arduino Diecimila and early Duemilanove)

ATmega8 (used on some older board)

Digital Pins

In addition to the specific functions listed below, the digital pins on an Arduino board can be used for general purpose input and output via the pinMode(), digitalRead(), anddigitalWrite() commands. Each pin has an internal pull-up resistor which can be turned onand off using digitalWrite() (w/ a value of HIGH or LOW, respectively) when the pin is configured as an input. The maximum current per pin is 40 mA.

Analog Pins

In addition to the specific functions listed below, the analog input pins support 10-bit analog-to-digital conversion (ADC) using the analogRead() function. Most of the analog inputs can also be used as digital pins: analog input 0 as digital pin 14 through analog input 5 as digital pin 19. Analog inputs 6 and 7 (present on the Mini and BT) cannot be used as digitalpins.

Power Pins

· VIN (sometimes labelled "9V"). The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated powersource). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin. Note that different boards accept different input voltages ranges, please see the documentation for your board. Also note that the LilyPad has no VIN pin and accepts only a regulated input.

· AREF. Reference voltage for the analog inputs. Used with analogReference().

· Reset. (Diecimila-only) Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board. · Analog Reference pin (orange) · Digital Ground (light green)

· Digital Pins 2-13 (green)

· Digital Pins 0-1/Serial In/Out - TX/RX (dark green) - These pins cannot be used for digital i/o (digitalRead and digitalWrite) if you are also using serial communication (e.g. Serial.begin).

· Reset Button - S1 (dark blue)In-circuit Serial Programmer (blue-green)

· Analog In Pins 0-5 (light blue)

· Power and Ground Pins (power: orange, grounds: light orange)

· External Power Supply In (9-12VDC) - X1 (pink)

· Toggles External Power and USB Power (place jumper on two pins closest to desired supply) -SV1 (purple)

· USB (used for uploading sketches to the board and for serial communication between theboard and the computer; can be used to power the board) (yellow)

Conclusion: At end of this experiment, we have studied different microcontrollers likeRaspberry-Pi/ Beagle board/ Arduino with diagram.

EXPERIMENT NO. 2 (Group A)

- **Aim:** Study of different operating systems for Raspberry-Pi/Beagle board/Arduino. Understanding the process of OS installation.

- **Outcome**: To study operating systems for platforms such as Raspberry-Pi/ Beagle board/Arduino.

- **Hardware Requirement:** Raspberry-Pi

- **Software Requirement:** Raspbian OS

- **Theory:**

1) Raspberry-Pi: - The Pi can run the official Raspbian OS, Ubuntu Mate, Snappy Ubuntu Core, the Kodi- based media centers OSMC and LibreElec, the non-Linux based Risc OS (one for fans of 1990s Acorn computers). It can also run Windows 10 IoT Core, which is very different to the desktop version of Windows, as mentioned below.

- OS which install on Raspberry-Pi: Raspbian, Ubuntu MATE, Snappy Ubuntu, Pidora, Linutop, SARPi, Arch Linux ARM, Gentoo Linux, etc.

How to install Raspbian on Raspberry-Pi:

Step 1: Download Raspbian

Step 2: Unzip the file. The Raspbian disc image is compressed, so you'll need to unzip it. The file uses the ZIP64 format, so depending on how current your built-in utilities are, you need to use certain programs to unzip it.

Step 3: Write the disc image to your microSD card. Next, pop your microSD card into your computer and write the disc image to it. The process of actually writing the image will be slightly different across these programs, but it's pretty self-explanatory no matter what you're using. Each of these programs will have you

select the destination (make sure you've picked your microSD card!) and the disc image (the unzipped Raspbian file). Choose, double-check, and then hit the button to write.

Step 4: Put the microSD card in your Pi and boot up. Once the disc image has been written to the microSD card, you're ready to go! Put that sucker into your Raspberry Pi, plug in the peripherals and power source, and enjoy. The current edition to Raspbian will boot directly to the desktop. Your default credentials are username pi and passwordraspberry.

2) BeagleBone Black: - The BeagleBone Black includes a 2GB or 4GB on-board eMMC flash memory chip. It comes with the Debian distribution factory pre-installed. Youcan flash new operating systems including Angstrom, Ubuntu, Android, and others.

1. Os which install on BeagleBone Black: Angstrom, Android, Debian, Fedora, Buildroot, Gentoo, Nerves Erlang/OTP, Sabayon, Ubuntu, Yocto, MINIX 3

How to install Debian on BeagleBone Black:

Step 1: Download Debian img.xz file.

Step 2: Unzip the file.

Step 3: Insert your MicroSD (uSD) card into the proper slot. Most uSD cards come with a full-sized SD card that is really just an adapter. If this is what you have then insert the uSD into the adapter, then into your card reader.

Step 4: Now open Win32 Disk imager, click the blue folder icon, navigate to the debian img location, and double click the file. Now click Write and let the process complete. Depending on your processor and available RAM it should be done in around 5 minutes.

Step 5: Alright, once that's done, you'll get a notification pop-up. Now we're ready to get going. Remove the SD adapter from the card slot, remove the uSD card from the adapter. With the USB cable disconnected insert the uSD into the BBB.

Step 6: Now, this next part is pretty straight forward. Plug the USB cable in and wait some more. If everything is going right you will notice that the four (4) leds just above the USB cable are doing the KIT impression. This could take up to 45 minutes; I just did it again in around 5 minutes. Your mileage will vary. Go back and surf reddit some more.

Step 7: If you are not seeing the leds swing back and forth you will need to unplug the USB cable, press and hold down the user button above the uSD card slot (next to the 2 little 10 pin ICs) then plug in the USB cable. Release the button and wait. You should see the LEDs swinging back and forth after a few seconds. Once this happens it's waiting time. When all 4 LEDs next to the USB slot stay lit at the same time the flash process has been completed.

Step 8: Remove the uSD card and reboot your BBB. You can reboot the BBB by removing and reconnecting the USB cable, or hitting the reset button above the USB cable near the edge of the board.

Step 9: Now using putty, or your SSH flavor of choice, connect to the BBB usingthe IP address 192.168.7.2. You'll be prompted for a username. Type root and press Enter. By default, there is no root password. I recommend changing this ASAP if you plan on putting your BBB on the network. To do this type password, hit enter, then enter your desired password. You will be prompted to enter it againto verify.

3) Arduino: - The Arduino itself has no real operating system. You develop code for the Arduino using the Arduino IDE which you can download from Arduino - Home. Versions are available for Windows, Mac and Linux. The Arduino is a constrained microcontroller. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. You are literally writing the "firmware" when you write the code and upload it. It's both good and its bad.

Conclusion: Here studied different operating systems for platforms such as Raspberry-Pi/Beagle board/Arduino

EXPERIMENT NO. 3 (Group A)

Aim: Study of different GATES (AND, OR, XOR), Sensors and basic binary operations.

Outcome: To study different GATES (AND, OR, XOR), Sensors

Hardware Requirement: Logical Gates,Sensors etc.

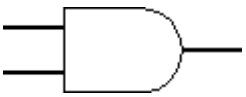
Software Requirement: Raspbian OS

Theory:

A logic gate is a device that acts as a building block for digital circuits. They perform basic logical functions that are fundamental to digital circuits. Most electronic devices we use today will have some form of logic gates in them. For example, logic gates can be used in technologies such as smartphones, tablets or within memory devices.
In a circuit, logic gates will make decisions based on a combination of digital signals coming from its inputs. Most logic gates have two inputs and one output. Logic gates are based on Boolean algebra. At any given moment, every terminal is in one of the two binary conditions, *false* or *true*. False represents 0, and true represents 1. Depending on the type of logic gate being used and the combination of inputs, the binary output will differ. A logic gate can be thought of like a light switch, wherein one position the output is off -- 0, and in another, it is on -- 1. Logic gates are commonly used in integrated circuits (IC).

Basic logic gates

There are seven basic logic gates: AND, OR, XOR, NOT, NAND, NOR, and XNOR.
The *AND gate* is so named because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator. The following illustration and table show the circuit symbol and logic combinations for an AND gate. (In the symbol, the input terminals are at left and the output terminal is at right.) The output is "true" when both inputs are "true." Otherwise, the output is "false." In other words, the output is 1



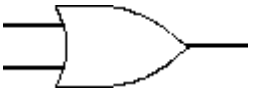
only when both inputsone AND two are 1.

AND gate

Input 1	Input 2	Output

	1	
1		
1	1	1

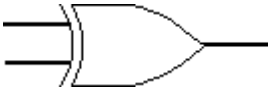
The *OR gate* gets its name from the fact that it behaves after the fashion of the logical inclusive "or." The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false." In other words, for the output to be 1, at least input one ORtwo must be 1.



OR gate

Input 1	Input 2	Output
	1	1
1		1
1	1	1

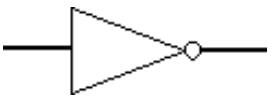
The *XOR* (*exclusive-OR*) *gate* acts in the same way as the logical "either/or." The output is "true" if either, but not both, of the inputs are "true." The output is "false" if both inputs are "false" or if both inputs are "true." Another way of looking at this circuit is to observe that the output is 1 if the inputs are different, but 0 if the inputs arethe same.



XOR gate

Input 1	Input 2	Output
	1	1
1		1
1	1	

A logical *inverter*, sometimes called a *NOT gate* to differentiate it from other types of electronic inverter devices, has only one input. It reverses the logic state. If the input is 1, then the output is 0. If the input is 0, then the output is 1.



Inverter or NOT gate

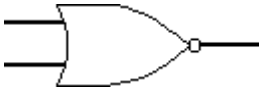
Input	Output
1	
	1

The *NAND gate* operates as an AND gate followed by a NOT gate. It acts in the manner of the logical operation "and" followed by negation. The output is "false" if both inputs are "true." Otherwise, the output is "true."

NAND gate

Input 1	Input 2	Output
		1
	1	1
1		1
1	1	

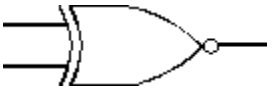
The *NOR gate* is a combination OR gate followed by an inverter. Its output is "true" if both inputs are "false." Otherwise, the output is "false."



NOR gate

Input 1	Input 2	Output
		1
	1	
1		
1	1	

The *XNOR (exclusive-NOR) gate* is a combination XOR gate followed by an inverter. Its outputis "true" if the inputs are the same, and "false" if the inputs are different.



XNOR gate

Input 1	Input 2	Output
		1
	1	
1		
1	1	1

Complex operations can be performed using combinations of these logic gates. In theory, there is no limit to the number of gates that can be arrayed together in a single device. But inpractice, there is a limit to the number of gates that can be packed into a given physical space. Arrays of logic gates are found in digital ICs. As IC technology advances, the required physical volume for each individual logic gate decreases and digital devices of the same or smaller size become capable of performing ever-more-complicated operations at ever-increasing speeds.

Composition of logic gates

In most logic gates, the low state is approximately zero volts (0 V), while the high state is approximately five volts positive (+5 V).

Logic gates can be made of resistors and transistors or diodes. A resistor can commonly be used as a pull-up or pull-down resistor. Pull-up and pull-down resistors are used when there are any unused logic gate inputs to connect to a logic level 1 or 0. This prevents any false switching of the gate. Pull-up resistors are connected to Vcc (+5V), and pull-down resistors are connected to ground (0 V).

Commonly used logic gates are TTL and CMOS. TTL, or Transistor-Transistor Logic, ICs will use NPN and PNP type Bipolar Junction Transistors. CMOS, or ComplementaryMetal-Oxide-Silicon, ICs are constructed from MOSFET or JFET type Field Effect Transistors. TTL IC's may commonly be labeled as the 7400 series of chips, while CMOS ICs may often be marked as a 4000 series of chips.

Types of Sensors

There are many different types of sensors. Here at Variohm, we offer a full range of sensors for industrial and commercial use.

Sensors are used throughout almost every industry for applications which we come into contactwith on a daily basis as well as more industrial and specialist applications.

Sensors can be found in the home, the office, in our cars, buses, trains, trams, computers, medical facilities, labs, power plants, restaurants, food processing factories, production lines etc A Sensor is used to take a measurement, the measurement will be processed and the result of the process, an output will be given. The output will then cause something to change or move. A simple example is the temperature sensor in a thermostat. The temperature sensor is constantly monitoring the temperature, once the measurement taken reaches the desired temperature, the measurement is processed and the output causes the boiler to switch off.

Types of Sensors

There are many different types of sensors, the main categories are;

- Position Sensors

- Temperature Sensors
- Force Sensors
- Vibration Sensors
- Piezo Sensors
- Fluid Property Sensors
- Humidity Sensors
- Strain gauges
- Photo Optic Sensors
- Flow and Level Switches

These categories can all be split further into subcategories for example, within position sensors there are the following types;

- Contacting
- Non-contacting
- Rotary
- Linear

And these types of sensors can be split even further, within non-contacting you have the following types of sensors;

- Hall effect
- Capacitive
- Eddy Current
- Ultrasonic
- Laser
- Proximity

By splitting one category – Position Sensors it is clear to see that the number of sensors present in today's world is so vast that one blog post could not cover every type of sensor. However, here is an overview of different types of sensors Variohm can offer.

Types of Sensors – Position Sensors

As discussed above there are many varieties of position sensor; linear, rotary, contacting, non-contacting and use a variety of different technologies. Position sensors are used to measureand monitor the position or displacement of an object.

We have been supplying position sensors for over 40 years and have developed our own range of position sensors which have been added to the comprehensive range from our suppliers and partners. Our own range includes;

Linear position Sensors

- VLP
- VXP
- ELPM
- VLPSC

Rotary Position Sensors

- Euro-X Hall Effect
- Euro-XP Puck – 2 part puck and magnet design
- Euro – XPD – D shaft
- CMRS
- CMRT
- CMRK

Further reading on Position SensorsWhat is a Position Sensor?

Position Sensor Applications

Types of Sensors – Pressure Sensors

Pressure sensors are often split into the following two categories; Pressure transducers and pressure switches. The main difference is that pressure transducers give accurate feedback on real-time pressure and pressure switches have a set limit which causes them to switch. Both pressure switches and pressure transducers have mechanisms which use the formula – Pressure
= force divided by area to detect pressure.
Pressure sensors can measure the pressure in gases, liquids or solids and are used in a variety of industries. Underwater pressure transducers are referred to as level meters as thepressure they measure is directly related to the level of the water.
Pressure can be gauge, differential, absolute or vacuum and can be measured in Bar or PSI. Many of our pressure sensors come from our trusted suppliers and we also have our own range of;
Pressure Transducers – EPT range Pressure Switches - EPS range Further reading on Pressure Sensors Pressure Transducer Applications What is a Pressure Sensor?

Types of Sensors – Load Cells and Force Sensors

Load Cells are available in a wide variety of shapes and sizes. They are used to measure various types of force, the main one being weight. Load cells are used in all types of scales; from bathroom scales to counting scales, industrial scales, truck scales, hopper scales and everything in between.
Most load cells use internal strain gauges to monitor force based on the level of distortion on thestrain gauge.
Our load cells come from our trusted suppliers. And can be seen on our website.
Further reading on Load Cells
Load Cells for Weighing
Load Cell Types and Applications

Types of Sensors – Temperature Sensors

Temperature Sensors are another very common type of sensor – they are all around us. Temperature sensors are used to measure and monitor temperature, whether this is the main variable requiring measuring or a secondary variable which requires monitoring as a safety precaution within another application.

Different types of temperature sensors will require different approvals. Medical approvals will be required for temperatures used for patient monitoring or within medical devices. Other certifications will be required for temperature sensors in food and beverage applications.

Temperature sensors come in many different shapes sizes and types; thermistors, probes, thermocouples, RTDs and temperature transducers to name a few. Temperature Sensors is another type of sensor which we carry our own range of;

2. Further reading on temperature sensors;
3. Temperature Sensor Applications
4. Types of Temperature Sensors

Conclusion: studied different GATES (AND, OR, XOR), Sensors

EXPERIMENT NO. 4 (Group A)

- **Aim:** Study of Connectivity and configuration of Raspberry-Pi /Beagle board/Arduinocircuit with basic peripherals like LEDS. Understanding GPIO and its use in the program.

- **Outcome:** Connectivity and configuration of Raspberry-Pi /Beagle board/Arduino circuitwith basic peripherals like LEDS

Hardware Requirement: Raspberry-Pi /Beagle board/Arduino, LED etc.

Theory:

Connecting Hardware Peripherals to Raspberry-Pi board. Raspberry-Pi setup through SSH (Headless Configuration of Raspberry-Pi-3 (VNC, Putti))

How to Connect a Raspberry-Pi to the Laptop or PC Display

Required devices:

Raspberry Pi
Ethernet Cable Laptop/ PC
SD Card with Raspbian Micro USB Cable

How does it Work?

To connect a Raspberry Pi to a laptop or PC display, you can simply use an Ethernet cable.
The Raspberry Pi's desktop GUI can be viewed through the laptop or PC display using a 100mbps Ethernet connection between the two.
We used VNC server software to connect the Raspberry-Pi to our laptop or PC.
Installing the VNC server on your Raspberry-Pi allows you to see the Raspberry Pi's desktop remotely.

Setting up your Raspberry Pi
Install Raspbian OS on blank SD card.
Insert this SD card into Raspberry-Pi board.
Connect micro USB cable to power the Raspberry-Pi.
Sharing Internet Over Ethernet in Window OS

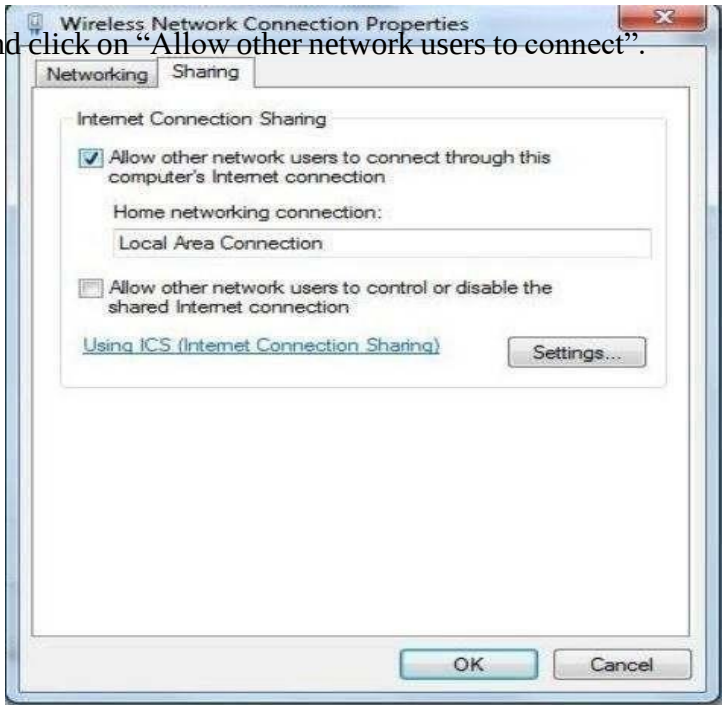
This step explains how you can share your laptop or PC with the Raspberry Pi via Ethernet cable.


Control Panel ▸ Network and Internet ▸ Network and Sharing Center
Control Panel Home
Manage wireless networks
Change adapter settings
Change advanced sharing settings
View your basic network information and set up connections
See full map
DELL-PC
(This computer)
jaysolabs
Internet
View your active networks
Connect or disconnect
jaysolabs
Home network
Access type: Internet
HomeGroup: Joined
Connections: Wireless Network Connection (jaysolabs)
Change your networking settings
Set up a new connection or network
Set up a wireless, broadband, dial-up, ad hoc, or VPN connection; or set up a router or access point.
Connect to a network
Connect or reconnect to a wireless, wired, dial-up, or VPN network connection.
Choose homegroup and sharing options
Access files and printers located on other network computers, or change sharing settings.
Troubleshoot problems
Diagnose and repair network problems, or get troubleshooting information.

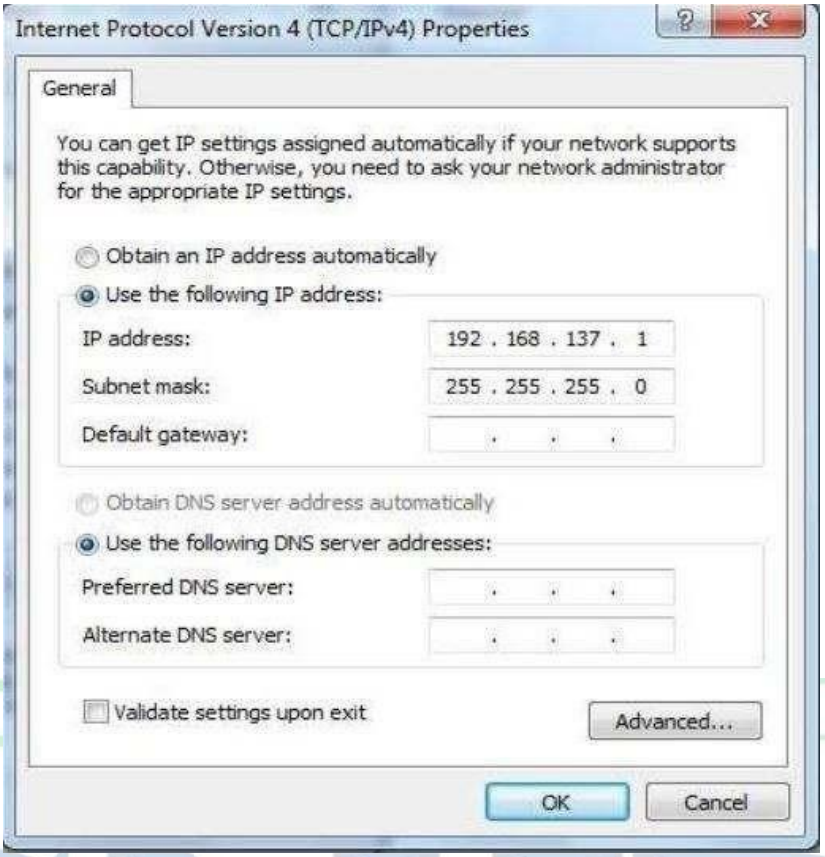
To share internet with multiple users over Ethernet, go to Network and Sharing Center. Then click on the WiFi network

Double click on Wireless area connection Click on Properties (shown below)

Go to “Sharing” tab and click on “Allow other network users to connect”.



Now, to check the IP assigned to the network established, click on the new local area connection link created:



Setting up the VNC Server to Connect Your Raspberry Pi to the Laptop or PC

Display First install VNC server and Putty on your laptop/ PC

Open Putty Software, and enter login ID: pi and Password: raspberry.

After that, enter commands into Putty i.e,

$ sudo apt-get update

$ sudo apt-get install tightvncserver

$ vncserver :1

You will be prompted to enter and confirm a password.

This will be asked only once, during first time setup.

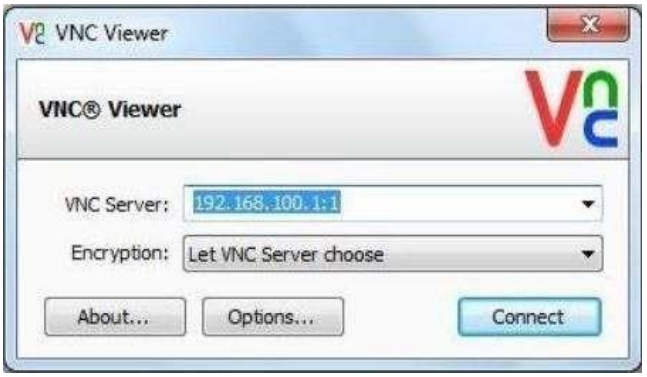
Enter an 8 digit password.

Note that this is the password you will need to use to connect to your Raspberry Pi remotely.

Setting Up the Client Side (Laptop or PC)

Download VNC client and install it.

When you first run VNC viewer, you will see following screen

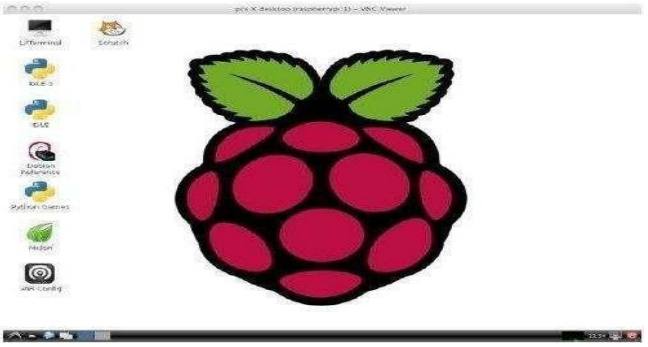


Enter the IP address of your Raspberry Pi given dynamically by your laptop and append with : 1 (denoting port number) and press connect.
You will get a warning message, press 'Continue':

Enter the 8 digit password which was entered in the VNC server installation on your Raspberry Pi:

Finally, the Raspberry Pi desktop should appear as a VNC window.
You will be able to access the GUI and do everything as if you are using the Pi's keyboard, mouse, and monitor directly.

Raspberry-Pi setup using mouse, keyboard and monitor
To work with Raspberry-Pi, we have to connect some peripherals to it. They are as follows



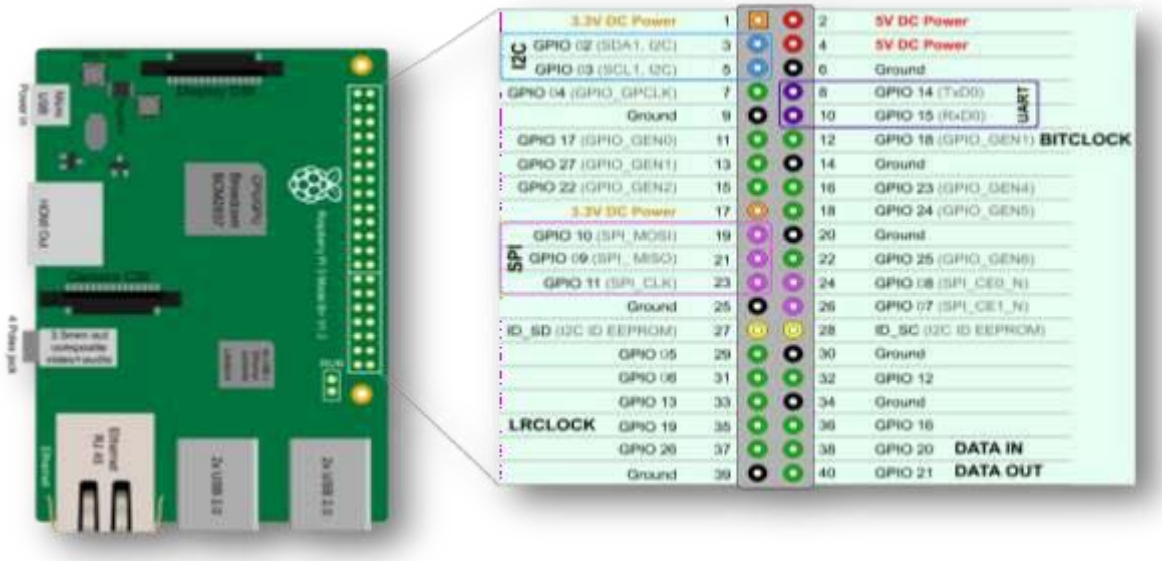
Most modern television sets and monitors have an HDMI port, and are the easiest to get working with the Raspberry Pi.
You can use an HDMI cable to connect the Raspberry Pi directly to the television or monitor.
VGA
Some old monitors have a VGA port.
These can be trickier to use as you'll need an HDMI-to-VGA converter, which can change digital video to analogue video.
A simple port adapter won't work.

Power supplies

For Raspberry Pi 3, it's recommended to use a 5V, 2.5A power supply.Mobile device charger

Many mobile devices are powered using a 5V micro USB charger.

These can often be used to power the Raspberry Pi, although it's worth checking that they provide sufficient voltage and current (5V / 1.2 - 2.5A).



Keyboard and mouse

Wired keyboard and mouse

Any standard USB keyboard and mouse can be used with the Raspberry Pi. These plug and play devices will work without any additional driver. Simply plug them into the Raspberry Pi and they should be recognized when it starts up.

Bluetooth keyboard and mouse

Bluetooth keyboards and mice can work with the Raspberry Pi, but your success rates will vary depending on the model and manufacturer. It's best to consult the manufacturer's documentation to see whether or not a device is compatible with the Raspberry Pi.

SD cards

The latest version of Raspbian, the default operating system recommended for the Raspberry Pi, requires an 8GB and some have higher

than others. Any 8GB SD card will work, although you'll need to follow the software setup guide to learn how to load an operating system onto the card.

Understanding GPIO pins on Raspberry Pi board and its use in programAim/Objectives:

To understand the GPIO pins of Raspberry-Pi 3

To program the GPIO pins of Raspberry-Pi 3 using Python

Introduction:

Raspberry Pi 3 Model B is the latest version of raspberry pi board.It is released on 29 February.

The above figure shows the Raspberry Pi 3 Model B and It's GPIO pins

General-purpose input/output (GPIO) is a generic pin on an integrated circuit or computer board whose behavior—including whether it is an input or output pin—is controllable by the user at run time.

There are 40 pins available on board of Raspberry pi 3 model B.

The Pins are arranged in a 2×20 fashion as shown in the figure above

Out of these, 26 pins are GPIO pins

As you can observe, the numbers to the pins are given in zigzag manner.

The first (bottom) row starts with number '1'. So the pins in this row have odd numbers i.e. from 1 to39.

The 2nd (Top) row starts with number '2'. So the pins in this row have even numbers i.e. from 2 to 40. Out of 40 pins, 26 pins are GPIO pins,8 pins are Ground (GND) pins,2 pins are 5V power supply pins 2pins are 3.3V power supply pins, 2 pins are not used

Now if you're coming to the Raspberry Pi as an Arduino user, you're probably used to referencing pins with a single, unique number.

In Raspberry Pi there are two different numbering schemes for referencing Pi pin numbers:

Broadcom chip-specific pin numbers (BCM)

Physical pin numbers (BOARD)

You're free to use either number-system.

The programs require that you declare which scheme you're using at the very beginning of your program.

In a program, at a time, you can use only one number scheme.

Broadcom chip-specific pin numbers (BCM)

BCM - Broadcom pin number, commonly called "GPIO", these are the ones you probably want to use with RPi.GPIO

The parameter used for this system is (GPIO.BCM).

This is a lower level way of working - it refers to the channel numbers on the Broadcom SOC.

To use this system, you have to always work with a diagram describing which channel number goes to which pin on the RPi board.

Your script could break between revisions of Raspberry Pi boards.

In this system 26 GPIO pins are named as GPIO 01 to GPIO 26

Physical Numbering System (BOARD)

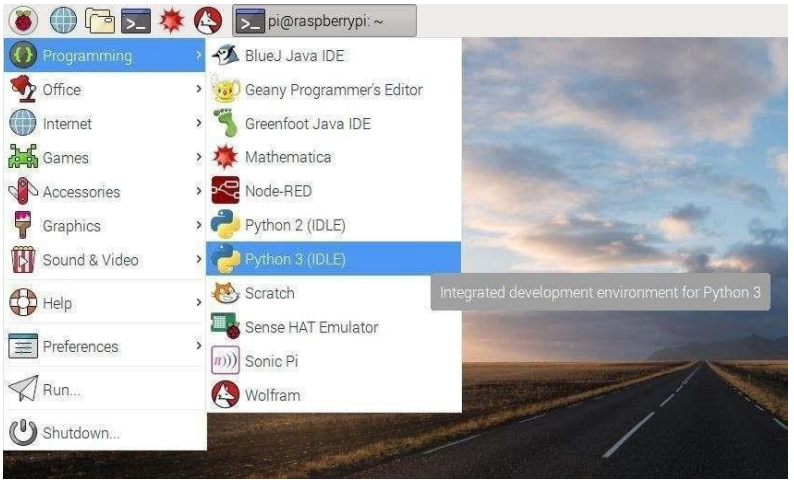
This system uses physical - Numbers corresponding to the pin's physical location on the header
The numbers printed on the board are physical numbering system.
The parameter used for this system is (GPIO.BOARD).
The advantage of using this numbering system is that your hardware will always work, regardless of the board revision of the RPi.
You will not need to rewire your connector or change your code.In this system
26 GPIO pins are named between 0 to 40

The below table summarizes the pinout of Raspberry-Pi in both the number systems.

Sr No.	Pins		BOARD Pin No	BCM Pin No
1	3.3V		1, 17	1, 17
2	5V		2, 4	2, 4
3	Ground		6,9,14,20,25,30,34,39	6,9,14,20,25,30,34,39
4	UART	TXD	8	GPIO 14
		RXD	10	GPIO 15
5	I2C1	SDA1	3	GPIO 2
		SCL1	5	GPIO 3
	I2C0	SDA0	27	ID_SD
		SCL0	28	ID_SC
6	SPI0	MOSI 0	19	GPIO 10
		MISO 0	21	GPIO 9
		SCLK 0	23	GPIO 11
		CE 0	24	GPIO 8
	SPI1	MOSI 1	38	GPIO 20
		MISO 1	35	GPIO 19
		SCLK 1	40	GPIO 21
		CE 1	26	GPIO 7

The Python IDLE shell and command line
To use the Python IDLE IDE for programming in Raspberry-Pi use the

following Open Python 3 from the main menu:



Or open terminal window and type the command sudo idle 3.5 and press enter Install all libraries required for Buzzer as given above.

Write the program as per algorithm given below

Save with Ctrl + S and run with F5.

See output on Python Shell or Terminal Window.

Raspberry Pi GPIO programming using Python

The Raspberry Pi is often used in conjunction with other hardware to create interesting electronic projects.

The Pi 3 comes with 40 GPIO pins that you can use to interface with various hardware devices—for both receiving data from them or for writing data to them.

To do this, we have to program the GPIO pins. To do this, special libraries in Python are used.

To include these libraries in the program, the command used is 'import'

This way, we can write applications to both read and also to control devices, i.e., turn them on and off, etc.

The default operating system used in Raspberry-Pi is Raspbian.

The Python package used for Raspberry Pi GPIO programming is RPi.GPIO. It is already installed in Raspbian.

If you are using any other operating system, the package can be installed by using the following command:

$ sudo pip install RPi.GPIO

There are important 8 steps in the programming of Raspberry-Pi using Python as follows Import the RPi.GPIO library using the following command import RPi.GPIO as GPIO Import the Time library using the following command

import time

Set numbering scheme to be used. The method used for this is GPIO.setmode(). We will use physical number scheme. So the method is written as

GPIO.setmode(GPIO.BOAD)

Set the pin mode as INPUT or OUTPUT using the commands GPIO.setup(channel, GPIO.IN)

GPIO.setup(channel, GPIO.OUT)

Read input using following command

GPIO.input(pin no)

Write output using following comman

GPIO.output(pin no, state)

Give delay using command using following command time.sleep(1) # delay for 1

print("Exiting...")

You must clean up the pin set-ups before your program exits otherwise those pin settings will persist, and that might cause trouble when you use the same pins in another program.
The Pi 'expresses its displeasure' with a warning.
To clean up the entire set of pins, invoke GPIO.cleanup().
If you want only a few pins to be cleaned up, then the pin numbers should be provided as GPIO.cleanup (channel_list).
Anyway, you can suppress the warning messages by calling GPIO.setwarnings (False). Save the program with proper name. The file is saved with extension '.py'.
The IDE named 'IDLE' used for programming is an interpreter and not a compiler. So to run the python program, we need to give the super user permission as follows.

Studying Connectivity and Configuration of Raspberry Pi board with basic peripherals (LEDs)

Aim/Objectives:

To understand the concept of Led bar
To understand the common anode & common cathode configuration.
To interface LED bar with Raspberry Pi.
Generate various patterns on LED bar.

Software: Raspbian OS , IDLE editor
Hardware Modules: Raspberry Pi Board, LED bar, Monitor

Introduction to "LED"

LED is a Light Emitting Diode. Light emitting diode is a two lead semiconductor light source. It is a p-n junction diode, which emits light when it is activated. When a suitable voltage is applied to the leads, electrons are able to recombine with electron holes within the device, and the color of light (corresponding to the energy of photon) is determined by the energy band gap of the semiconductor. It has two terminals named as 'anode (+ve)' and 'cathode (-ve)'. Battery is connected to these two terminals. When LED is forward biased, it emits light. In LED bar number of LEDs are connected in series(in our case 8 LEDs are connected) LED bar has two configurations as Common Anode: In this, anode terminal of all the LEDs are made common and connected to the VCC (+5v). By controlling cathode terminal we can make LED ON or OFF (current sourcing). Common Cathode: In this, cathode terminal of all the LEDs are made common and connected to the Ground (0v). By controllinganode terminal we can make LED ON or OFF (current sinking).

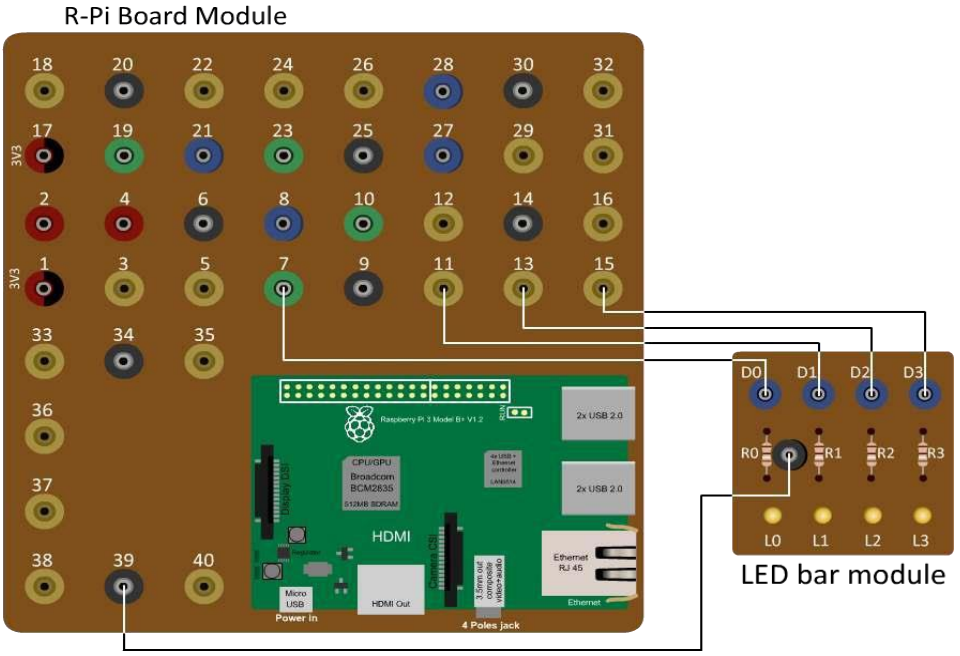
Safety precautions:

Raspberry-Pi provides 3.3V and 5V VCC
pinsRaspberry-Pi operates on 3.3V.
Various sensors and actuators operate on different voltages.
Read datasheet of a given sensor or an actuator and then use appropriate VCC pin to connect a sensor or an actuator.
Ensure that signal voltage coming to the Raspberry-Pi from any sensor or actuator does not

If signal/data coming to Raspberry-Pi is greater than 3.3V then use voltage level shifter module to decrease the incoming voltage.

The Raspberry-Pi is a costly device, hence you should show the circuit connections to your instructor before starting your experiment.

Interface diagram:



Steps for assembling circuit:

Connect led bar module pins from D0- D3 to Raspberry Pi GPIO pins 7, 11, 13, 15 respectively. Connect led bar module pin COM to the GND pin of Raspberry-Pi module.
Procedure:

Write the program as per the algorithm given below.

Save program.
Run code using Run module.Algorithm:
Import GPIO and Time library
Set mode i.e. GPIO.BOARD
Set GPIO 8 pins as a Output pinPrint message “ON”
After 1 second time delay, Make all the leds ON one by one
Print message “OFF”
After 1 second time delay, Make all the leds OFF one by one

Conclusion:We have implemented this experiment using LED glowing with Arduino IDE and arduino board and LED

Questions:

1) Explain GPIO pins connectivity in details.
2) Explain programming of Raspberry-Pi using Python.

EXPERIMENT NO. 5 (Group B)

Aim: Write a program using Arduino to control LED (One or more ON/OFF). Or Blinking

Outcome: Connectivity and configuration of Raspberry-Pi /Beagle board/Arduinocircuit with basic peripherals like LEDS

Hardware Requirement: Arduino, LED, 220 ohm resistor etc.

Software Requirement: Arduino IDE

Theory:

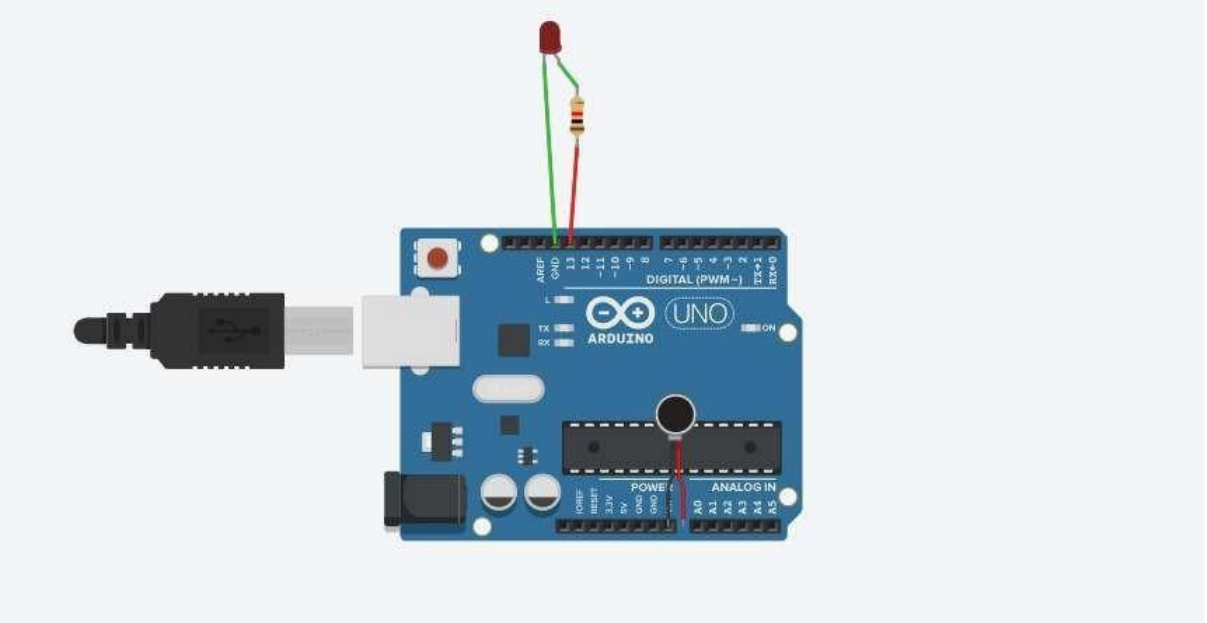
This example shows the simplest thing you can do with an Arduino to see physical output: it blinks the on-board LED.

This example uses the built-in LED that most Arduino boards have. This LED is connected to a digital pin and its number may vary from board type to board type. To make your life easier, we have a constant that is specified in every board descriptor file. This constant is *LED_BUILTIN* and allows you to control the built-in LED easily. Here is the correspondence between the constant and the digital pin.

- D13 - 101
- D13 - Due
- D1 - Gemma
- D13 - Intel Edison
- D13 - Intel Galileo Gen2
- D13 - Leonardo and Micro
- D13 - LilyPad
- D13 - LilyPad USB
- D13 - MEGA2560
- D13 - Mini
- D6 - MKR1000
- D13 - Nano
- D13 - Pro
- D13 - Pro Mini
- D13 - UNO
- D13 - Yún
- D13 - Zero

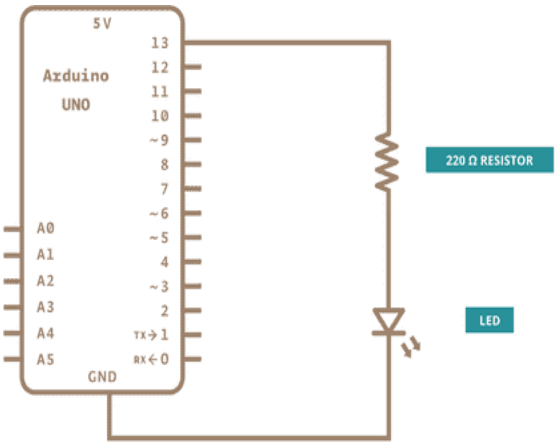
If you want to lit an external LED with this sketch, you need to build this circuit, where you connect one end of the resistor to the digital pin correspondent to the *LED_BUILTIN* constant. Connect the long leg of the LED (the positive leg, called the anode) to the other end of the resistor. Connect the short leg of the LED (the negative leg, called the cathode) to the GND. In the diagram below we show an UNO board that has D13 as the LED_BUILTIN value.

The value of the resistor in series with the LED may be of a different value than 220 ohm; the LED will lit up also with values up to 1K ohm.




```
// C++ code
//
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000); // Wait for 1000 millisecond(s)
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
}
```

Schematic



Code

After you build the circuit plug your Arduino board into your computer, start the Arduino Software (IDE) and enter the code below. You may also load it from the menu File/Examples/01.Basics/Blink . The first thing you do is to initialize LED_BUILTIN pin as an output pin with the line

pinMode(LED_BUILTIN, OUTPUT);

In the main loop, you turn the LED on with the line:

digitalWrite(LED_BUILTIN, HIGH);

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

digitalWrite(LED_BUILTIN, LOW);

That takes the LED_BUILTIN pin back to 0 volts, and turns the LED off. In between the on and the off, you want enough time for a person to see the change, so the **delay()** commands tell the board to do nothing for 1000 milliseconds, or one second. When you use the **delay()** command, nothing else happens for that amount of time. Once you've understood the basic examples, check out the BlinkWithoutDelay example to learn how to create a delay while doing other things.

Once you've understood this example, check out the DigitalReadSerial example to learn how read a switch connected to the board.

Conclusion: -

EXPERIMENT NO. 6 (Group B)

Aim: Create a program that illuminates the green LED if the counter is less than 100, illuminates the yellow LED if the counter is between 101 and 200 and illuminates the red LED if the counter is greater than 200

Outcome: Connectivity, configuration and control of LED using Arduino circuit under different conditions.

Hardware Requirement: Arduino, LED, 220 ohm resistor etc.

Software Requirement: Arduino IDE

Theory:

The problem statement is like Arduino traffic light, a fun little project that you can build in under an hour. Here's how to build your own using an Arduino, and how to change the circuit for an advanced variation.

What You Need to Build an Arduino Traffic Light Controller

Apart from the basic Arduino, you'll need:

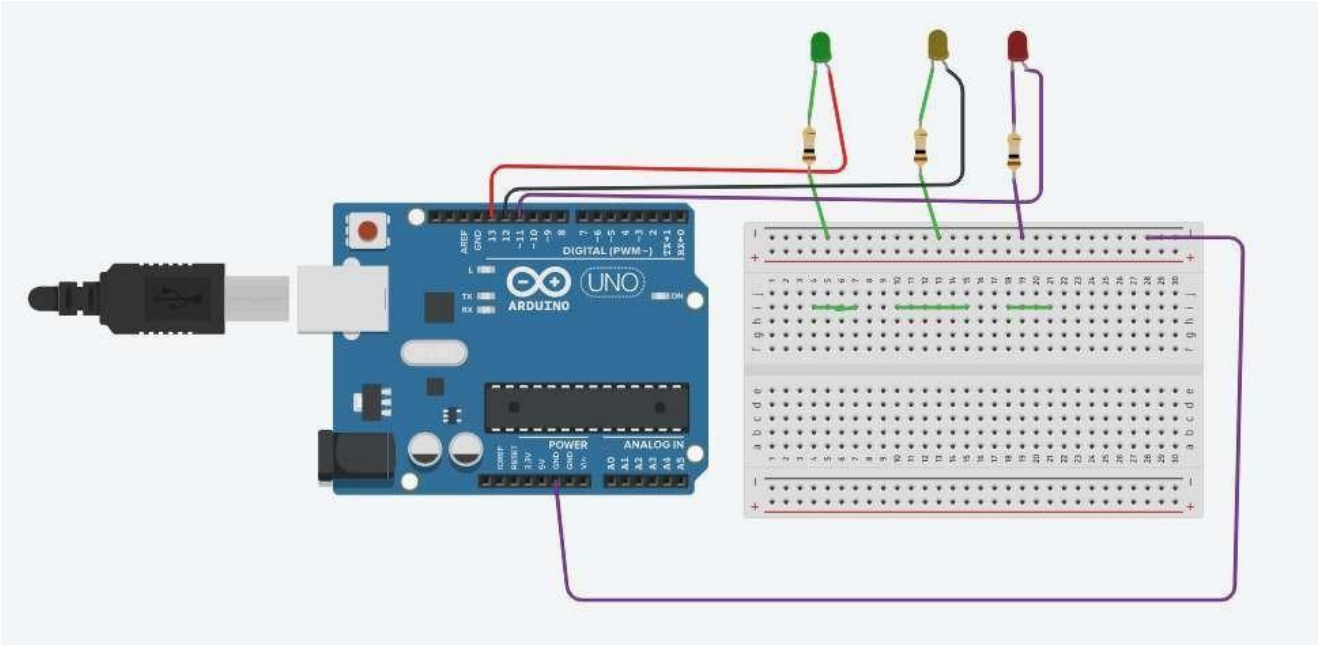
- 1 x 10k-ohm resistor
- 1 x pushbutton switch
- 6 x 220-ohm resistors
- A breadboard
- Connecting wires
- Red, yellow and green

LEDs Arduino Traffic Light: The

Basics

Let's start small. A basic, single traffic light is a good place to start. Here's the circuit:

Connect the anode (long leg) of each LED to digital pins eight, nine, and ten (via a 220-ohm resistor). Connect the cathodes (short leg) to the Arduino's ground.




```
// C++ code
//
int counter1=0;
void setup()
{
  //pinMode(LED_BUILTIN, OUTPUT);

}

void loop()
{
  if (counter1<10)
  {
    digitalWrite(13,HIGH);
    digitalWrite(12,LOW);
    digitalWrite(11,LOW);
    delay(1000);
  }
  else
  {
    if(counter1<20)
    {
    digitalWrite(13,LOW);
       digitalWrite(12,HIGH);
       digitalWrite(11,LOW);
       delay(1000);
    }
    else
    {
      digitalWrite(13,LOW);
      digitalWrite(12,LOW);
      digitalWrite(11,HIGH);
      delay(1000);
    }
  // Wait for 1000 millisecond(s)

  }
  counter1= counter1+1;
}
```

Code for the Arduino Traffic Light

Start by defining variables so that you can address the lights by name rather than a number. Start a new Arduino project, and begin with these lines:

```
int red = 10; int
yellow = 9; int
green = 8;
```

Next, let’s add the setup function, where you'll configure the red, yellow and green LEDs to be outputs. Since you have created variables to represent the pin numbers, you can now refer to the pins by name instead:

```
void setup(){ pinMode(red,
   OUTPUT);
pinMode(yellow, OUTPUT);
   pinMode(green, OUTPUT);
}
```

The **pinMode** function configures the Arduino to use a given pin as an output. You have to do this for your LEDs to work at all. Now for the actual logic of the traffic light. Here's the code you need. Add this below your variable definitions and setup function:

```
void loop(){
   changeLights
   ();
   delay(15000);
}
void changeLights(){
   // green off, yellow on for 3
   seconds digitalWrite(green, LOW);
   digitalWrite(yellow, HIGH);
   delay(3000);
```

```
  // turn off yellow, then turn red on for 5
  seconds digitalWrite(yellow, LOW);
  digitalWrite(red,
  HIGH); delay(5000);
  // red and yellow on for 2 seconds (red is already on though)
  digitalWrite(yellow, HIGH);
  delay(2000);
  // turn off red and yellow, then turn on
  green digitalWrite(yellow, LOW);
  digitalWrite(red, LOW);
  digitalWrite(green,
  HIGH); delay(3000);
}
```

Upload this code to your Arduino, and run (make sure to select the correct board and port from the **Tools** > Board and **Tools** > **Port** menus). You should have a working trafficlight that changes every 15 seconds, like this (sped up):

Let's break down this code. The **changeLights** function performs all the hard work. This rotates the traffic light through yellow and red, then back to green. As this gets called inside the **loop** function, the Arduino will run this code forever, with a 15-second pause every time.

The **changeLights** function consists of four distinct steps:

- Green on, yellow off
- Yellow off, red on
- Yellow on, red on
- Green on, red off, yellow off

These four steps replicate the process used in real traffic lights. For each step, the code isvery similar. The appropriate LED gets turned on or off using **digitalWrite**. This is an Arduino function used to set output pins to HIGH (for on), or LOW (for off).

After enabling or disabling the required LEDs, the **delay** makes the Arduino wait for a given amount of time. Three seconds in this case.

Going Deeper: Arduino Pedestrian Crossing

Now that you know the basics, let's improve it. Add in a pushbutton for pedestrians to change the light whenever they like:

Notice how the traffic light is exactly the same as the previous example. Connect the button to digital pin 12. You'll notice that the switch has a high-impedance 10k-ohm resistorattached to it, and you may be wondering why. This is a pull-down resistor.

A switch either lets the current flow or doesn't. This seems simple enough, but in a logic circuit, the current should be always flowing in either a high or low state (remember, 1 or 0, HIGH or LOW). You might assume that a pushbutton switch that isn't actually pressed wouldbe in a LOW state, but in fact, it's said to be 'floating', because no current gets drawn at all.

In this floating state, it's possible that a false reading will occur as it fluctuates with electrical interference. In other words, a floating switch is giving neither a reliable HIGH nor LOW reading. A pull-down resistor keeps a small amount of current flowing when the switch gets closed, thereby ensuring an accurate low state reading.

In other logic circuits, you may find a pull-up resistor instead, and this works on the same principle, but in reverse, making sure that particular logic gate defaults to high.

Now, in the loop part of the code, instead of changing the lights every 15 seconds, you're going to read the state of the pushbutton switch instead, and only change the lights when it's activated.

Code for the Arduino Pedestrian Crossing

Start by adding a new variable to store your button pin:

```
int button = 12; // switch is on pin 12
```

Now, in the setup function, add a new line to declare the switch as an input. Add a line to set the traffic lights to the green stage. Without this initial setting, they would off until the first time **changeLights** runs.

```
pinMode(button, INPUT);
digitalWrite(green, HIGH);
```

Change the entire loop function to the following instead:

```
void loop() {
   if (digitalRead(button) == HIGH){
      delay(15); // software debounce if
      (digitalRead(button) == HIGH) {
// if the switch is HIGH, ie. pushed down - change the lights!
         changeLights();
delay(15000); // wait for 15 seconds
}
}
}
```

That should do it. You may be wondering why the button checking happens twice (**digitalRead(button)**), separated by a small delay. This is debouncing. Much like the pull-down resistor for the button, this simple check stops the code detecting minor interference as a button press.

By waiting inside the **if** statement for 15 seconds, the traffic lights can't change for at least that duration. Once 15 seconds is over the loop restarts. Each restart of the loop, it reads the state of the button again, but if it isn't pressed, the **if** statement never activates,the lights neverchange, and the program restarts again.

Here's how this looks (sped up):

Arduino Traffic Light with Junction

Let's try a more advanced model. Instead of a pedestrian crossing, change your circuit tohave two traffic lights:

Connect the second traffic light to digital pins 11, 12, and 13.

Code for the Arduino Traffic Light with Junction

First, assign your new traffic light pins to variables, and configure them as outputs, like in the first example:

```
// light one int
red1 = 10;
int yellow1 = 9;
int green1 = 8;
// light two int
red2 = 13;
int yellow2 = 12;
int green2 = 11;
void setup(){
// light one pinMode(red1,
   OUTPUT);
pinMode(yellow1, OUTPUT);
   pinMode(green1, OUTPUT);
// light two pinMode(red2,
   OUTPUT);
pinMode(yellow2, OUTPUT);
   pinMode(green2, OUTPUT);
}
```

Now, update your loop to use the code from the first example (instead of the pedestrian crossing):

```
void loop(){
   changeLights();
   delay(15000);
}
```

Once again, all the work is carried out in the **changeLights** function. Rather than going**red** > **red** & **yellow** > **green**, this code will alternate the traffic lights. When one is on green, the other is on red. Here's the code:

```
void changeLights(){
    // turn both yellows on
    digitalWrite(green1, LOW);
    digitalWrite(yellow1,
    HIGH);
    digitalWrite(yellow2,
    HIGH); delay(5000);
    // turn both yellows off, and opposite green and red
    digitalWrite(yellow1, LOW);
    digitalWrite(red1, HIGH);
    digitalWrite(yellow2,
    LOW); digitalWrite(red2,
    LOW);
    digitalWrite(green2,
    HIGH);
```

```
    delay(5000);
    // both yellows on again
    digitalWrite(yellow1,
    HIGH);
   digitalWrite(yellow2,
   HIGH);
   digitalWrite(green2, LOW);
   delay(3000);
    // turn both yellows off, and opposite green and red
   digitalWrite(green1, HIGH);
   digitalWrite(yellow1,
   LOW); digitalWrite(red1,
   LOW);
   digitalWrite(yellow2,
   LOW); digitalWrite(red2,
   HIGH); delay(5000);
}
```

Conclusion:

EXPERIMENT NO. 7 (Group B)

Aim: Create a program so that when the user enters "B" the green light blinks, "g" the green light is illuminated "y" the yellow light is illuminated and "r" the red light is illuminated

Outcome: Connectivity, configuration and control of LED using Arduino circuit under different conditions.

- **Hardware Requirement:** Arduino, LED, 220 ohm resistor etc.
- **Software Requirement**: Arduino IDE

• Theory:

The problem statement is like Arduino traffic light, a fun little project that you can build in under an hour. Here's how to build your own using an Arduino, and how to change the circuit for an advanced variation.

What You Need to Build an Arduino Traffic Light Controller

Apart from the basic Arduino, you'll need:

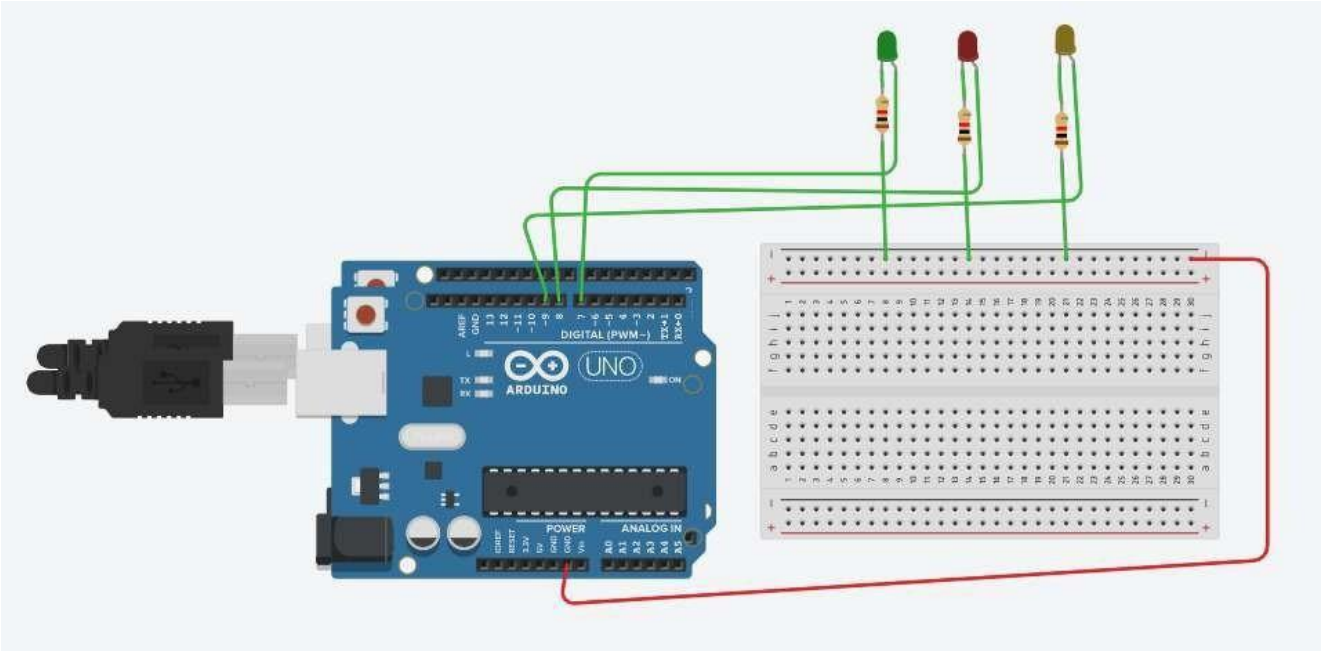
- 1 x 10k-ohm resistor
- 1 x pushbutton switch
- 6 x 220-ohm resistors
- A breadboard
- Connecting wires
- Red, yellow and green

LEDs Arduino Traffic Light: The

Basics

Let's start small. A basic, single traffic light is a good place to start. Here's the circuit:

Connect the anode (long leg) of each LED to digital pins eight, nine, and ten (via a 220-ohm resistor). Connect the cathodes (short leg) to the Arduino's ground.




```
// C++ code
//
int mychar=0;
void setup()
{
  Serial.begin(9600);
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);

}

void loop()
{
  if(Serial.available()>0)
  {
    //read the incoming byte
    mychar=Serial.read();
    Serial.print("I received 1");
    Serial.print(mychar);
  }

    if(mychar==114)
    {
      digitalWrite(7,HIGH);
      digitalWrite(8,LOW);
      digitalWrite(9,LOW);

     }
    if(mychar==103)
    {
      digitalWrite(7,LOW);
      digitalWrite(8,HIGH);
      digitalWrite(9,LOW);

     }
     if(mychar==98)
    {
      digitalWrite(7,LOW);
      digitalWrite(8,LOW);
      digitalWrite(9,HIGH);

     }
```

Code for the Arduino Traffic Light

Start by defining variables so that you can address the lights by name rather than a number. Start a new Arduino project, and begin with these lines:

```
int red = 10; int
yellow = 9; int
green = 8;
```

Next, let's add the setup function, where you'll configure the red, yellow and green LEDs to be outputs. Since you have created variables to represent the pin numbers, you can now refer to the pins by name instead:

```
void setup(){ pinMode(red,
   OUTPUT);
pinMode(yellow, OUTPUT);
   pinMode(green, OUTPUT);
}
```

The **pinMode** function configures the Arduino to use a given pin as an output. You have to do this for your LEDs to work at all. Now for the actual logic of the traffic light. Here's the code you need. Add this below your variable definitions and setup function:

```
void loop(){
   changeLights();
   delay(15000);
}
void changeLights(){
   // green off, yellow on for 3 seconds
   digitalWrite(green, LOW);
   digitalWrite(yellow, HIGH);
   delay(3000);
```

```
  // turn off yellow, then turn red on for 5 seconds
  digitalWrite(yellow, LOW);
  digitalWrite(red, HIGH);
  delay(5000);
  // red and yellow on for 2 seconds (red is already on though)
  digitalWrite(yellow, HIGH);
  delay(2000);
  // turn off red and yellow, then turn on green
  digitalWrite(yellow, LOW);
  digitalWrite(red, LOW);
  digitalWrite(green, HIGH);
  delay(3000);
}
```

Upload this code to your Arduino, and run (make sure to select the correct board and port from the **Tools** > Board and **Tools** > **Port** menus). You should have a working trafficlight that changes every 15 seconds, like this (sped up):

Let's break down this code. The **changeLights** function performs all the hard work. This rotates the traffic light through yellow and red, then back to green. As this gets called inside the **loop** function, the Arduino will run this code forever, with a 15-second pause every time.

The **changeLights** function consists of four distinct steps:

- Green on, yellow off
- Yellow off, red on
- Yellow on, red on
- Green on, red off, yellow off

These four steps replicate the process used in real traffic lights. For each step, the code isvery similar. The appropriate LED gets turned on or off using **digitalWrite**. This is an Arduino function used to set output pins to HIGH (for on), or LOW (for off).

After enabling or disabling the required LEDs, the **delay** makes the Arduino wait for a given amount of time. Three seconds in this case.

Going Deeper: Arduino Pedestrian Crossing

Now that you know the basics, let's improve it. Add in a pushbutton for pedestrians to change the light whenever they like:

Notice how the traffic light is exactly the same as the previous example. Connect the button to digital pin 12. You'll notice that the switch has a high-impedance 10k-ohm resistorattached to it, and you may be wondering why. This is a pull-down resistor.

A switch either lets the current flow or doesn't. This seems simple enough, but in a logic circuit, the current should be always flowing in either a high or low state (remember, 1 or 0, HIGH or LOW). You might assume that a pushbutton switch that isn't actually pressed wouldbe in a LOW state, but in fact, it's said to be 'floating', because no current gets drawn at all.

In this floating state, it's possible that a false reading will occur as it fluctuates with electrical interference. In other words, a floating switch is giving neither a reliable HIGH nor LOW reading. A pull-down resistor keeps a small amount of current flowing when the switch gets closed, thereby ensuring an accurate low state reading.

In other logic circuits, you may find a pull-up resistor instead, and this works on the same principle, but in reverse, making sure that particular logic gate defaults to high.

Now, in the loop part of the code, instead of changing the lights every 15 seconds, you're going to read the state of the pushbutton switch instead, and only change the lights when it's activated.

Code for the Arduino Pedestrian Crossing

Start by adding a new variable to store your button pin:

```
int button = 12; // switch is on pin 12
```

Now, in the setup function, add a new line to declare the switch as an input. Add a line to set the traffic lights to the green stage. Without this initial setting, they would off until the first time **changeLights** runs.

```
pinMode(button, INPUT);
digitalWrite(green, HIGH);
```

Change the entire loop function to the following instead:

```
void loop() {
   if (digitalRead(button) == HIGH){
      delay(15); // software debounce if
      (digitalRead(button) == HIGH) {
// if the switch is HIGH, ie. pushed down - change the lights!
         changeLights();
delay(15000); // wait for 15 seconds
}
}
}
```

That should do it. You may be wondering why the button checking happens twice (**digitalRead(button)**), separated by a small delay. This is debouncing. Much like the pull-down resistor for the button, this simple check stops the code detecting minor interference as a button press.

By waiting inside the **if** statement for 15 seconds, the traffic lights can't change for at least that duration. Once 15 seconds is over the loop restarts. Each restart of the loop, it reads the state of the button again, but if it isn't pressed, the **if** statement never activates,the lights neverchange, and the program restarts again.

Here's how this looks (sped up):

Arduino Traffic Light with Junction

Let's try a more advanced model. Instead of a pedestrian crossing, change your circuit tohave two traffic lights:

Connect the second traffic light to digital pins 11, 12, and 13.

Code for the Arduino Traffic Light with Junction

First, assign your new traffic light pins to variables, and configure them as outputs, like in the first example:

```
// light one int
red1 = 10;
int yellow1 = 9;
int green1 = 8;
// light two int
red2 = 13;
int yellow2 = 12;
int green2 = 11;
void setup(){
// light one pinMode(red1,
   OUTPUT);
pinMode(yellow1, OUTPUT);
   pinMode(green1, OUTPUT);
// light two pinMode(red2,
   OUTPUT);
pinMode(yellow2, OUTPUT);
   pinMode(green2, OUTPUT);
}
```

Now, update your loop to use the code from the first example (instead of the pedestrian crossing):

```
void loop(){
   changeLights();
   delay(15000);
}
```

Once again, all the work is carried out in the **changeLights** function. Rather than going**red** > **red** & **yellow** > **green**, this code will alternate the traffic lights. When one is on green, the other is on red. Here's the code:

```
void changeLights(){
   // turn both yellows on
   digitalWrite(green1, LOW);
   digitalWrite(yellow1, HIGH);
   digitalWrite(yellow2, HIGH);
   delay(5000);
   // turn both yellows off, and opposite green and red
   digitalWrite(yellow1, LOW);
   digitalWrite(red1, HIGH);
   digitalWrite(yellow2, LOW);
   digitalWrite(red2, LOW);
   digitalWrite(green2, HIGH);
```

```
  delay(5000);
  // both yellows on again
  digitalWrite(yellow1, HIGH);
  digitalWrite(yellow2, HIGH);
  digitalWrite(green2, LOW);
  delay(3000);
  // turn both yellows off, and opposite green and red
  digitalWrite(green1, HIGH);
  digitalWrite(yellow1, LOW);
  digitalWrite(red1, LOW);
  digitalWrite(yellow2, LOW);
  digitalWrite(red2, HIGH);
  delay(5000);
}
```

Conclusion: -

EXPERIMENT NO. 8 (Group B)

- **Aim:** Write a program that asks the user for a number and outputs the number squared that is entered

- **Outcome:** Connectivity, configuration and serial communication with Arduino.

- **Hardware Requirement:** Arduino, USB cable etc.

- **Software Requirement**: Arduino IDE

- **Theory:**

- **Arduino Serial Monitor for Beginners**

Arduino serial monitor for beginners in electronics. Send and receive data between the serial monitor window on a computer and an Arduino. The serial monitor is a utility that is part of the Arduino IDE. Send text from an Arduino board to the serial monitor window on a computer. In addition, send text from the serial monitor window to an Arduino board. Communications between the serial monitor and Arduino board takes place over the USB connection between the computer and Arduino.

- **Demonstration of the Arduino Serial Monitor for Beginners**

Part 2 of this Arduino tutorial for beginners shows how to install the Arduino IDE. In addition, it shows how to load an example sketch to an Arduino. It is necessary to know how to load a sketch to an Arduino board in this part of the tutorial. Therefore, first finish the previous parts of this tutorial before continuing with this part. A sketch loaded to an Arduino board demonstrates how the serial monitor works in the sub-sections that follow.

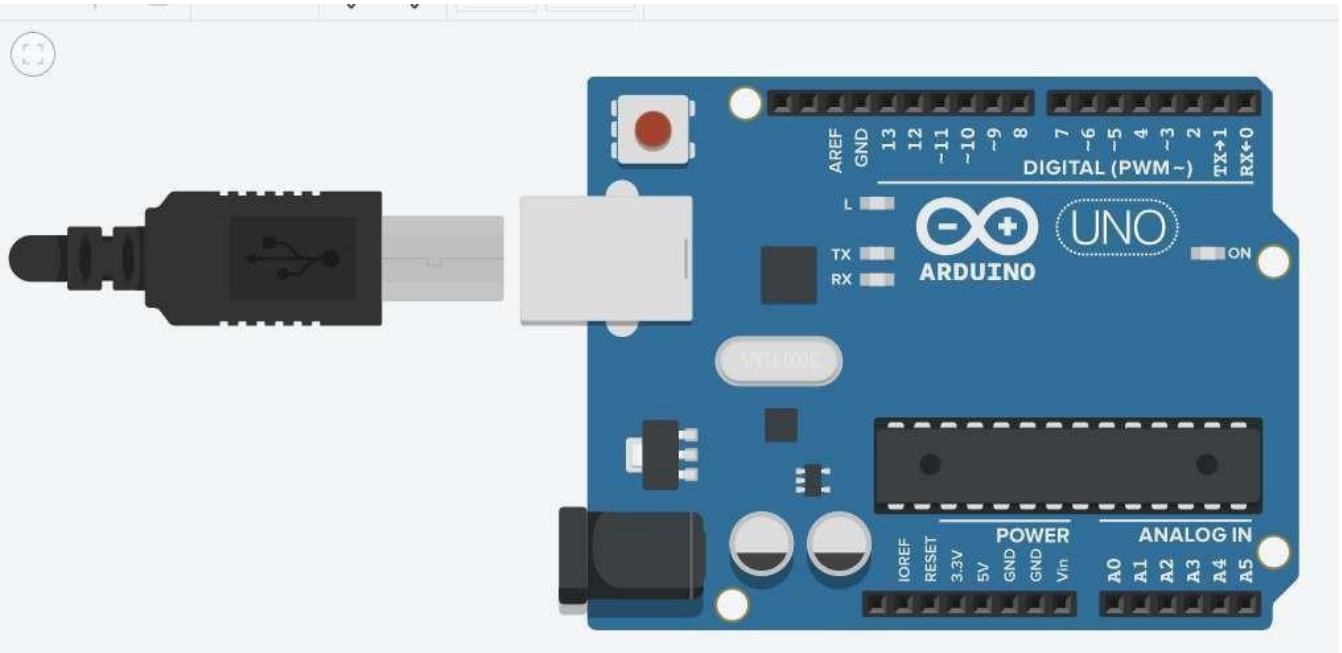
- **Load an Example Sketch that uses the Serial Monitor to an Arduino Board**

Start the Arduino IDE application. Select File → Examples → 04.Communication → ASCIITable from the top Arduino IDE menu bar. As a result, the ASCIITable example sketch opens in a new Arduino IDE window. Upload the ASCIITable example sketch to the Arduino Uno or MEGA 2560 board.

After the ASCIITable sketch is uploaded, nothing is seen to happen. This is because this example sketch sends text out of the USB port of the Arduino board. Because there is nothing running on the computer to receive this text, nothing is seen.

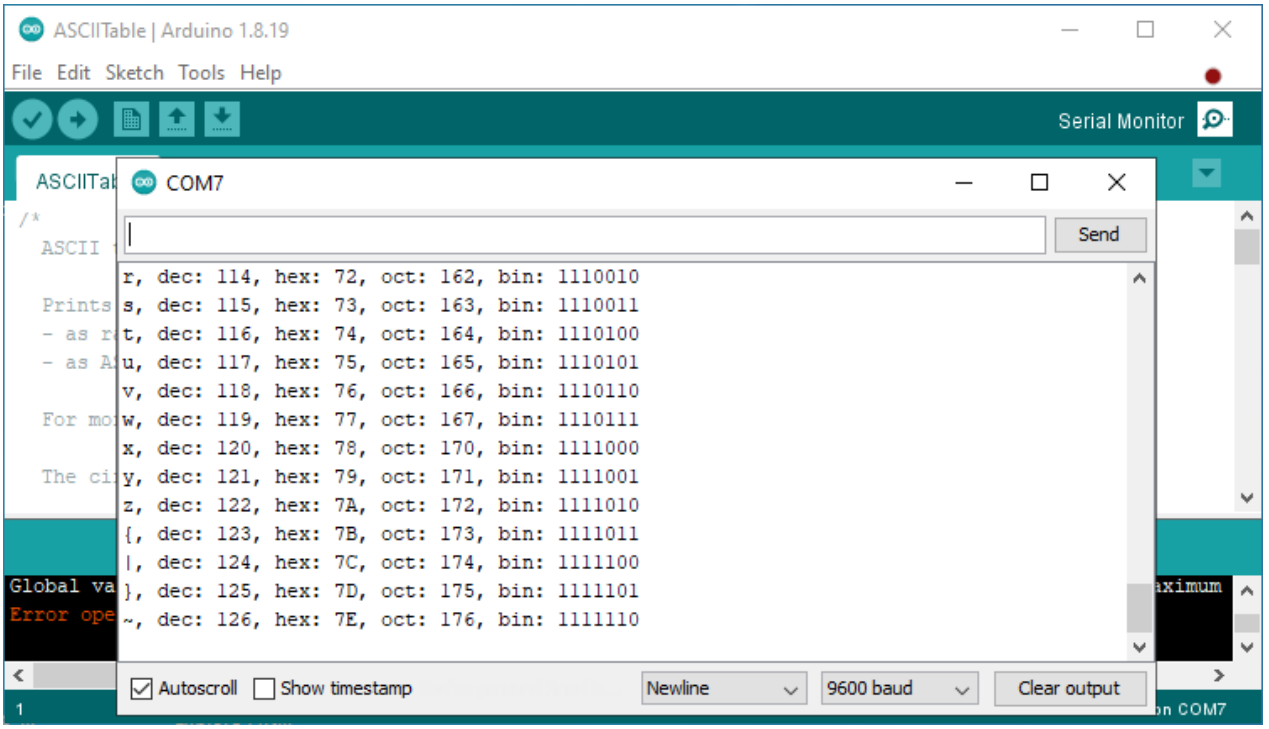
- **How to Open the Arduino Serial Monitor Window for Beginners**

The following image shows the location of the serial monitor window icon on the Arduino IDE toolbar. A red dot near the top right of the image shows the serial monitor toolbar icon location.




```
// C++ code
//
int mychar=0;//for incoming serial data
int power=2;
int out=0;
void setup()
{
  Serial.begin(9600);  // open serial port,set data rate to 9600
}
void loop()
{
  //send data only when you receive it
  if(Serial.available()>0)
  {
    //read the incoming byte
    mychar=Serial.read();
    out=pow(mychar,power);
    Serial.print("I received. .. ");
    Serial.print(mychar);
    Serial.print("Square of no....");
    Serial.print(out);
```

}



Click the Serial Monitor icon near the top right of the Arduino IDE to open the serial monitor window. The above image shows the serial monitor window opened, and on top of the Arduino IDE window. Because the ASCIITable example is loaded on the Arduino board, when the serial monitor window opens, the Arduino sends text to the serial monitor window. This is also because opening the serial monitor window resets the Arduino board, causing the ASCIITable sketch torun from the beginning again.

The ASCIITable sketch sends text out of the USB port of the Arduino. Because the serial monitor is connected to the USB port, it receives the text and displays it in the big receive area of the window. As a result, text scrolls on the serial monitor window for a while. The text then stops because the Arduino has finished sending text. Use the right scrollbar in the serial monitor window to scroll up. Scrolling up reveals all of the text that the Arduino sent.

- **What to do When Junk Characters are Displayed**

When junk, or garbage characters, or even nothing is displayed in the serial monitor, it is usually because of an incorrect baud rate setting. Look at the bottom of the serial monitor in the above image. Notice the value 9600 baud in a box. This is the baud setting of communications between the Arduino and serial monitor. The ASCIITable, and most other built-in example sketches, set the Arduino to communicate at 9600 baud. If your serial monitor window shows a different baud rate, change it to 9600 baud. Do this by clicking the baud drop-down list. Select 9600 baud on the list that drops down.

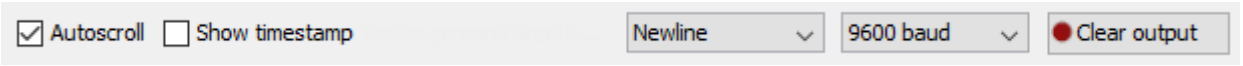
- **Reset the Arduino Board with the RESET Button**

Press and release the RESET button on the Arduino board and the ASCIITable sketch runs from the beginning again. As a result of the reset, the same text scrolls down the serial monitor window and then stops again. The RESET button is the only push button on the Arduino Uno or MEGA

Pushing the RESET button in holds the board in reset. This means that the sketch currently loaded on the board stops running. Releasing the RESET button takes the board out of reset. As a result, the sketch currently loaded on the Arduino starts running from the beginning again.

- **Clear the Serial Monitor Window Receive Area**

The red dot in the image below shows the location of the Clear output button at the bottom of the serial monitor window. Click the Clear output button and text is cleared from the receive area of the serial monitor window. Reset the Arduino, and the receive area fills with text from the ASCIITable sketch again.



Serial Monitor Window Clear Output Button

- **What the ASCIITable Sketch Does**

ASCII stands for American Standard Code for Information Interchange. ASCII is a standard way that uses numbers to represent various characters. For example, the decimal number 65 represents the letter A. Another example is the decimal number 125 represents a closing brace: }.This allows computers to send and receive text by sending and receiving numbers. For example when a computer receives the number 65, it knows to display the letter A.

The ASCIITable sketch sends the numbers 33 through to 126 out of the USB port. This results in the printable text characters from the ASCII table displayed in the serial monitor window. In addition to the ASCII characters, the number that represents each character is displayed. Each number is shown in four different numbering systems. These are the decimal, hexadecimal, octal and binary number systems. In the serial monitor window, these number systems are abbreviated to dec, hex, oct and bin.

Conclusion: -

EXPERIMENT NO. 9 (Group B)

- **Aim:** Write a program to control the color of the LED by turning 3 different potentiometers. One will be read for the value of Red, one for the value of Green, and one for the value of Blue

- **Outcome:** Connectivity, configuration and control of LED using Arduino circuit under different conditions.

- **Hardware Requirement:** Arduino, LED, 220 ohm resistor etc.

- **Software Requirement**: Arduino IDE

• Theory:

The problem statement is like Arduino traffic light, a fun little project that you can build in under an hour. Here's how to build your own using an Arduino, and how to change the circuit for an advanced variation.

What You Need to Build an Arduino Traffic Light Controller

Apart from the basic Arduino, you'll need:

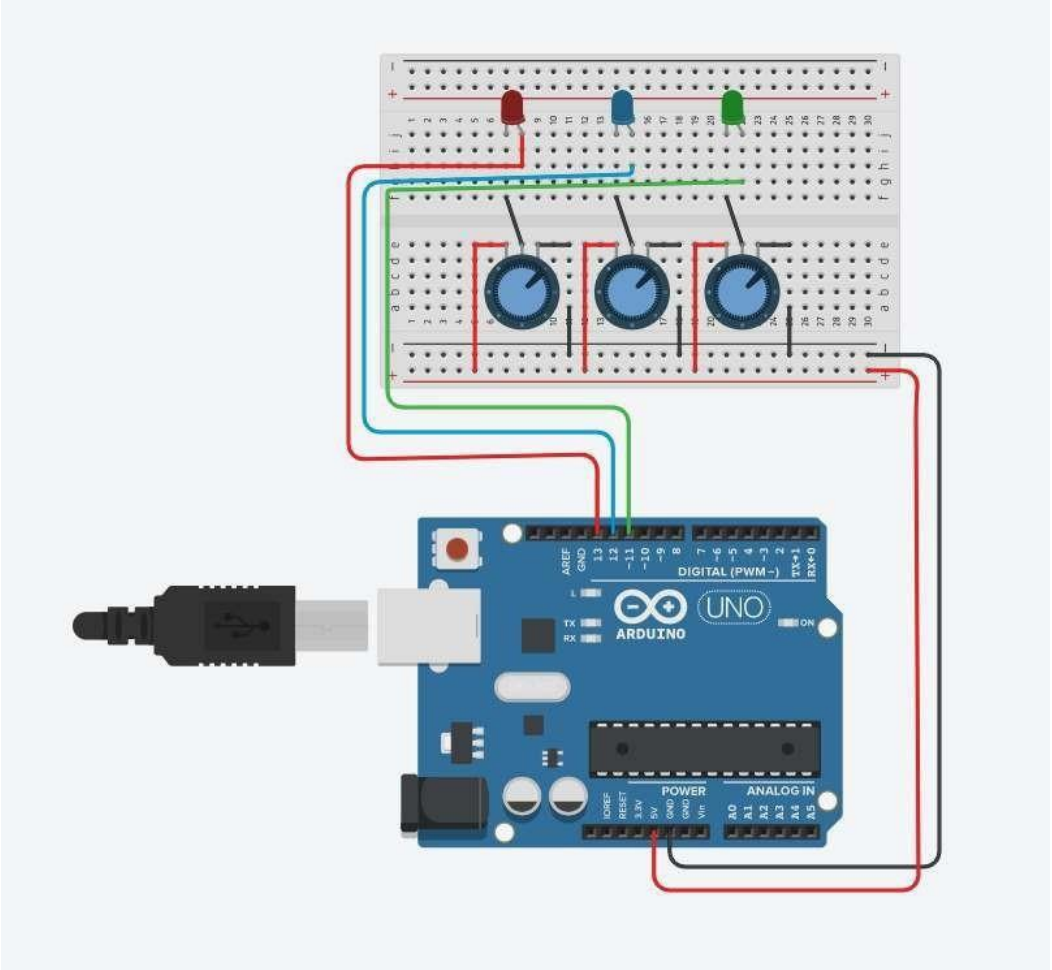
- 1 x 10k-ohm resistor
- 1 x pushbutton switch
- 6 x 220-ohm resistors
- A breadboard
- Connecting wires
- Red, yellow and green

LEDs Arduino Traffic Light: The

Basics

Let's start small. A basic, single traffic light is a good place to start. Here's the circuit:

Connect the anode (long leg) of each LED to digital pins eight, nine, and ten (via a 220-ohm resistor). Connect the cathodes (short leg) to the Arduino's ground.



Code for the Arduino Traffic Light

```
// C++ code//
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);

}

void loop()
{
  digitalWrite(13, HIGH);
  digitalWrite(12, HIGH);
  digitalWrite(11, HIGH);
}
```

Start by defining variables so that you can address the lights by name rather than a number. Start a new Arduino project, and begin with these lines:

Next, let’s add the setup function, where you'll configure the red, yellow and green LEDs to be outputs. Since you have created variables to represent the pin numbers, you can now refer to the pins by name instead:

```
void setup(){ pinMode(red,
   OUTPUT);
pinMode(yellow, OUTPUT);
   pinMode(green, OUTPUT);
}
```

The **pinMode** function configures the Arduino to use a given pin as an output. You have to do this for your LEDs to work at all. Now for the actual logic of the traffic light. Here's the code you need. Add this below your variable definitions and setup function:

```
void loop(){
   changeLights();
   delay(15000);
}
void changeLights(){
   // green off, yellow on for 3 seconds
   digitalWrite(green, LOW);
   digitalWrite(yellow, HIGH);
   delay(3000);
```

```
  // turn off yellow, then turn red on for 5 seconds
  digitalWrite(yellow, LOW);
  digitalWrite(red, HIGH);
  delay(5000);
  // red and yellow on for 2 seconds (red is already on though)
  digitalWrite(yellow, HIGH);
  delay(2000);
  // turn off red and yellow, then turn on green
  digitalWrite(yellow, LOW);
  digitalWrite(red, LOW);
  digitalWrite(green, HIGH);
  delay(3000);
}
```

Upload this code to your Arduino, and run (make sure to select the correct board and port from the **Tools** > Board and **Tools** > **Port** menus). You should have a working trafficlight that changes every 15 seconds, like this (sped up):

Let's break down this code. The **changeLights** function performs all the hard work. This rotates the traffic light through yellow and red, then back to green. As this gets called inside the **loop** function, the Arduino will run this code forever, with a 15-second pause every time.

The **changeLights** function consists of four distinct steps:

- Green on, yellow off
- Yellow off, red on
- Yellow on, red on
- Green on, red off, yellow off

These four steps replicate the process used in real traffic lights. For each step, the code isvery similar. The appropriate LED gets turned on or off using **digitalWrite**. This is an Arduino function used to set output pins to HIGH (for on), or LOW (for off).

After enabling or disabling the required LEDs, the **delay** makes the Arduino wait for a given amount of time. Three seconds in this case.

Going Deeper: Arduino Pedestrian Crossing

Now that you know the basics, let's improve it. Add in a pushbutton for pedestrians to change the light whenever they like:

Notice how the traffic light is exactly the same as the previous example. Connect the button to digital pin 12. You'll notice that the switch has a high-impedance 10k-ohm resistorattached to it, and you may be wondering why. This is a pull-down resistor.

A switch either lets the current flow or doesn't. This seems simple enough, but in a logic circuit, the current should be always flowing in either a high or low state (remember, 1 or 0, HIGH or LOW). You might assume that a pushbutton switch that isn't actually pressed wouldbe in a LOW state, but in fact, it's said to be 'floating', because no current gets drawn at all.

In this floating state, it's possible that a false reading will occur as it fluctuates with electrical interference. In other words, a floating switch is giving neither a reliable HIGH nor LOW reading. A pull-down resistor keeps a small amount of current flowing when the switch gets closed, thereby ensuring an accurate low state reading.

In other logic circuits, you may find a pull-up resistor instead, and this works on the same principle, but in reverse, making sure that particular logic gate defaults to high.

Now, in the loop part of the code, instead of changing the lights every 15 seconds, you're going to read the state of the pushbutton switch instead, and only change the lights when it's activated.

Code for the Arduino Pedestrian Crossing

Start by adding a new variable to store your button pin:

```
int button = 12; // switch is on pin 12
```

Now, in the setup function, add a new line to declare the switch as an input. Add a line to set the traffic lights to the green stage. Without this initial setting, they would off until the first time **changeLights** runs.

```
pinMode(button, INPUT);
digitalWrite(green, HIGH);
```

Change the entire loop function to the following instead:

```
void loop() {
   if (digitalRead(button) == HIGH){
      delay(15); // software debounce if
      (digitalRead(button) == HIGH) {
// if the switch is HIGH, ie. pushed down - change the lights!
         changeLights();
delay(15000); // wait for 15 seconds
}
}
}
```

That should do it. You may be wondering why the button checking happens twice (**digitalRead(button)**), separated by a small delay. This is debouncing. Much like the pull-down resistor for the button, this simple check stops the code detecting minor interference as a button press.

By waiting inside the **if** statement for 15 seconds, the traffic lights can't change for at least that duration. Once 15 seconds is over the loop restarts. Each restart of the loop, it reads the state of the button again, but if it isn't pressed, the **if** statement never activates,the lights neverchange, and the program restarts again.

Here's how this looks (sped up):

Arduino Traffic Light with Junction

Let's try a more advanced model. Instead of a pedestrian crossing, change your circuit tohave two traffic lights:

Connect the second traffic light to digital pins 11, 12, and 13.

Code for the Arduino Traffic Light with Junction

First, assign your new traffic light pins to variables, and configure them as outputs, like in the first example:

```
// light one int
red1 = 10;
int yellow1 = 9;
int green1 = 8;
// light two int
red2 = 13;
int yellow2 = 12;
int green2 = 11;
void setup(){
// light one pinMode(red1,
   OUTPUT);
pinMode(yellow1, OUTPUT);
   pinMode(green1, OUTPUT);
// light two pinMode(red2,
   OUTPUT);
pinMode(yellow2, OUTPUT);
   pinMode(green2, OUTPUT);
}
```

Now, update your loop to use the code from the first example (instead of the pedestrian crossing):

```
void loop(){
   changeLights();
   delay(15000);
}
```

Once again, all the work is carried out in the **changeLights** function. Rather than going**red** > **red** & **yellow** > **green**, this code will alternate the traffic lights. When one is on green, the other is on red. Here's the code:

```
void changeLights(){
   // turn both yellows on
   digitalWrite(green1, LOW);
   digitalWrite(yellow1, HIGH);
   digitalWrite(yellow2, HIGH);
   delay(5000);
   // turn both yellows off, and opposite green and red
   digitalWrite(yellow1, LOW);
   digitalWrite(red1, HIGH);
   digitalWrite(yellow2, LOW);
   digitalWrite(red2, LOW);
   digitalWrite(green2, HIGH);
```

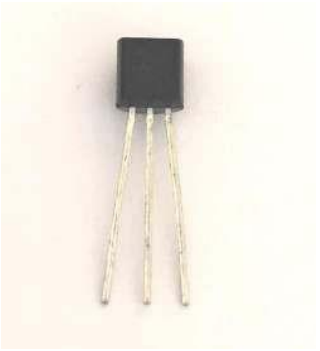
```
  delay(5000);
  // both yellows on again
  digitalWrite(yellow1, HIGH);
  digitalWrite(yellow2, HIGH);
  digitalWrite(green2, LOW);
  delay(3000);
  // turn both yellows off, and opposite green and red
  digitalWrite(green1, HIGH);
  digitalWrite(yellow1, LOW);
  digitalWrite(red1, LOW);
  digitalWrite(yellow2, LOW);
  digitalWrite(red2, HIGH);
  delay(5000);
}
```

Conclusion:

EXPERIMENT NO. 10 (Group B)

- **Aim:** Write a program read the temperature sensor and send the values to the serial monitor on the computer
- **Outcome:** Understanding working principle of DHT11, LM35 temperature sensor.
- **Hardware Requirement:** Arduino, LED, LM35, DHT11, etc
- **Software Requirement**: Arduino IDE
- **Theory:**

LM35 Temperature Sensor



LM35 Temperature Sensor Pinout
LM35 Sensor Pinout Configuration

Pin Number	Pin Name	Description
1	Vcc	Input voltage is +5V for typical applications
2	Analog Out	There will be increase in 10mV for raise of every 1°C. Canrange from -1V(-55°C) to 6V(150°C)
3	Ground	Connected to ground of circuit

- **LM35 Sensor Features**
  - Minimum and Maximum Input Voltage is 35V and -2V respectively. Typically 5V.
  - Can measure temperature ranging from -55°C to 150°C
  - Output voltage is directly proportional (Linear) to temperature (i.e.) there will be a rise of 10mV (0.01V) for every 1°C rise in temperature.
  - ±0.5°C Accuracy
  - Drain current is less than 60uA
  - Low cost temperature sensor
  - Small and hence suitable for remote applications
  - Available in TO-92, TO-220, TO-CAN and SOIC package

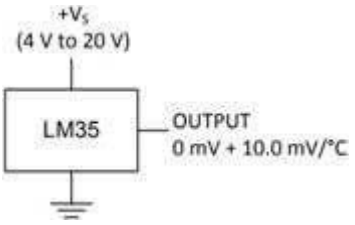
- **LM35 Temperature Sensor Equivalent**

LM34, DS18B20, DS1620, LM94022

How to use LM35 Temperature Sensor:

LM35 is a precession Integrated circuit Temperature sensor, whose output voltage varies, based on the temperature around it. It is a small and cheap IC which can be used to measure temperature anywhere between -55°C to 150°C. It can easily be interfaced with any Microcontroller that has ADC function or anydevelopment platform like Arduino.

Power the IC by applying a regulated voltage like +5V (VS) to the input pin and connected the ground pin to the ground of the circuit. Now, you can measure the temperature in form of voltage as shown



below.

If the temperature is 0°C, then the output voltage will also be 0V. There will be rise of 0.01V (10mV) for every degree Celsius rise in temperature. The voltage can converted into temperature using the below formulae.

$$V_{OUT} = 10\ mv/°C \times T$$

where

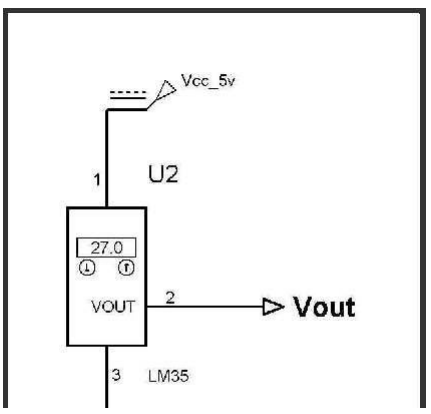
- V_{OUT} is the LM35 output voltage
- T is the temperature in °C

- **LM35 Temperature Sensor Applications**
  - Measuring temperature of a particular environment
  - Providing thermal shutdown for a circuit/component
  - Monitoring Battery Temperature
  - Measuring Temperatures for HVAC applications.

- **How Does LM35 Sensor Work?**

Main advantage of LM35 is that it is linear i.e. 10mv/°C which means for every degree rise in temperature the output of LM35 will rise by 10mv. So if the output of LM35 is 220mv/0.22V the temperature will be 22°C. So if room temperature is 32°C then the output of LM35 will be 320mv i.e. 0.32V.

- **LM35 Interfacing Circuit**



As such no extra components required to interface LM35 to ADC as the output of LM35 is linear with 10mv/degree scale. It can be directly interfaced to any 10 or 12 bit ADC. But if you are using an 8-bit ADC like ADC0808 or ADC0804 an amplifier section will be needed if you require to measure 1°C change.

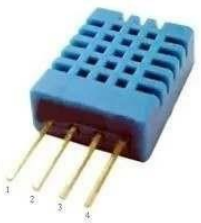
LM35 can also be directly connected to Arduino. The output of LM35 temperature can also be given to comparator circuit and can be used for over temperature indication or by using a simple relay can be used as a temperature controller.

- **DHT11 interfacing with arduino and weather station**

DHT11 sensor is used to measure the **temperature** and **humidity**. It has a resistive humidity sensing component and a negative temperature coefficient (NTC). An 8 bit MCU is also connected in it which is responsible for its fast response. It is very inexpensive but it gives values of both temperature and humidity at a time.

- **Specification of DHT11**
  - It has humidity range from 20 to 90% RH
  - It has temperature range from 0 – 50 C
  - It has signal transmission range of 20 m
  - It is inexpensive
  - It has fast response and it is also durable

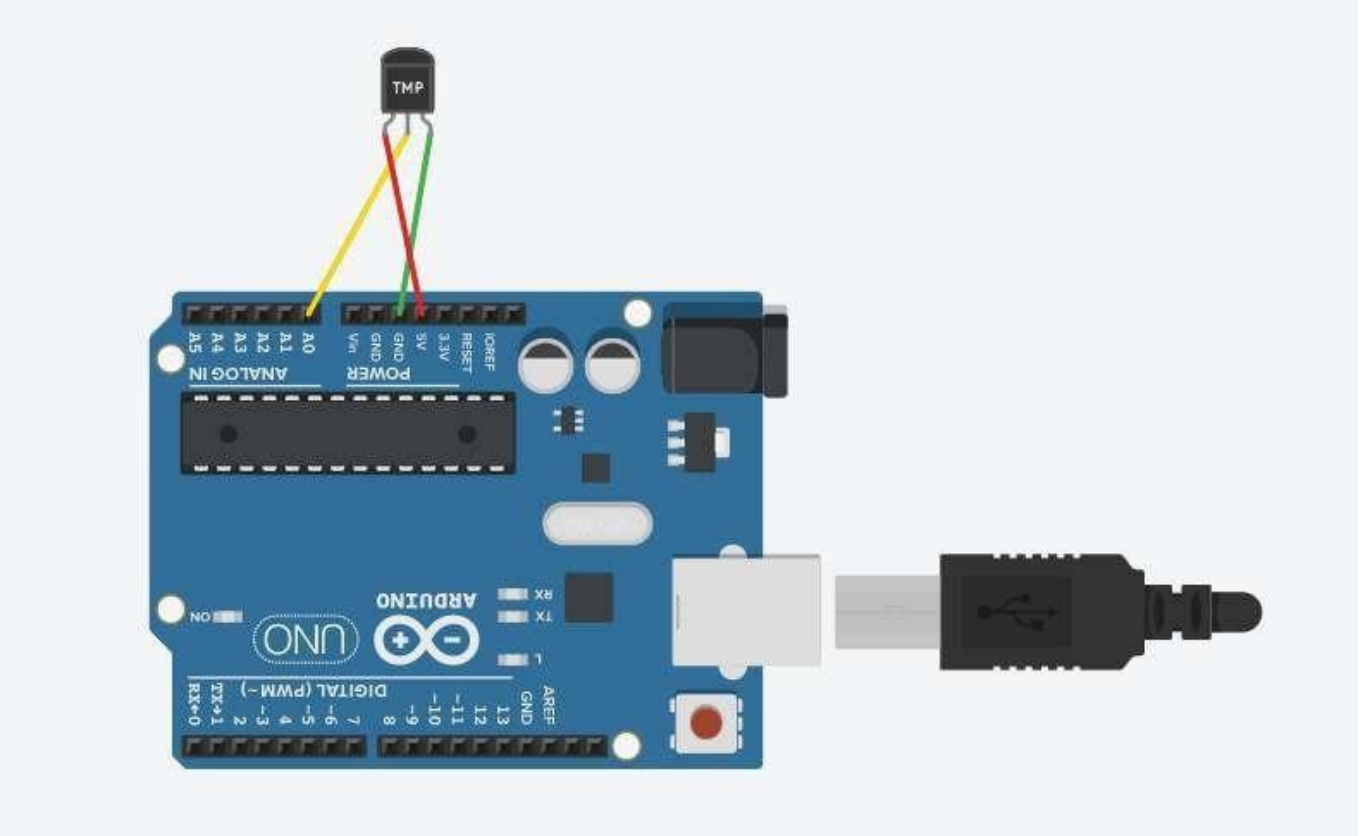
- **DHT11 Pin out**



  - The first pin of the DHT11 is vcc pin.
  - The second pin of the DHT is Data pin.
  - The third pin is not used.
  - The fourth pin of the DHT sensor is ground.

- **DHT11 interfacing with arduino**

First of all connect the ground and the VCC of the DHT11 temperature and humidity sensor to the ground and 5v of the **Arduino**. Then connect the data pin of the DHT11 sensor to the pin 2 of the Arduino.




```
// C++ code
//
int temp=0;
void setup()
{
  pinMode(A0,INPUT);
  Serial.begin(9600);
}

void loop()
{
 temp = map(((analogRead(A0) - 20) * 3.04), 0, 1023, -40, 125);
  Serial.println(temp);
  delay(1011); // Delay a little bit to improve simulation performance

}
```

- **Installing the DHT11 Library**

To run the following code in Arduino IDE you will first have to install the DHT library in you Arduino directory.

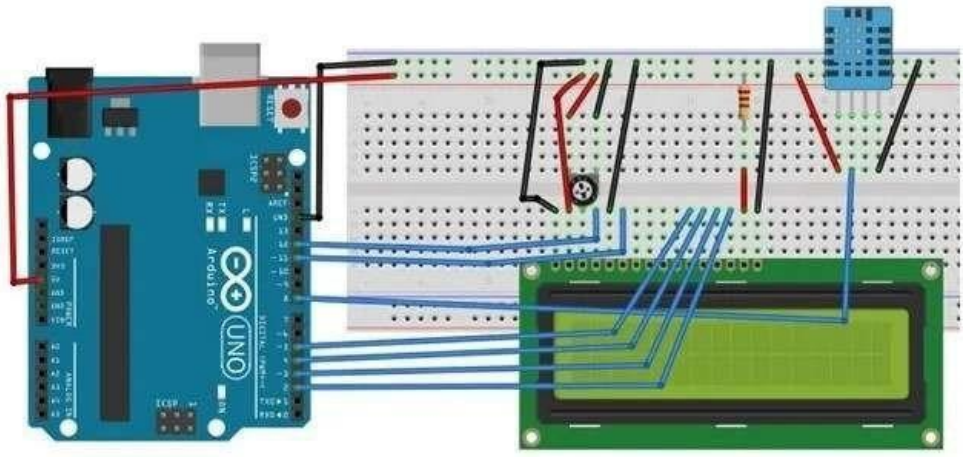
Download the zip file from here and place it in your Arduino library folder. The path to Arduino library folder for my computer is
Documents/ Arduino/ Libraries
Unzip the downloaded file and place it in this folder.
After copying the files, the Arduino library folder should have a new folder named DHT containing the dht.h and dht.cpp. After that copy the following code in the Arduino IDE and upload the code.

- **Code of DHT11 interfacing with arduino**

```
// Code for DHT11 Temperature and humidity sensor.
 #include " DHT.h " // including the library of DHT11 temperature
andhumidity sensor
 #define DHTPIN 2    // Selecting the pin at which we have
connectedDHT11
 #define DHTTYPE DHT11 // Selecting the type of DHT
 sensorsDHT dht ( DHTPIN, DHTTYPE ) ;
 void   setup   (   )   {
   Serial.begin ( 9600 ) ;
   dht.begin (  ) ;   // The sensor will start working
 }
 void loop ( ) {
   // Reading temperature or humidity may take about 2 seconds
because itis a very slow sensor.

float humidity = dht.readHumidity ( ) ; // Declaring h a variable
andstoring the humidity in it.
  float temp = dht.readTemperature ( ) ; // Declaring t a variable
andstoring the temperature in it.
  // Checking if the output is correct. If these are NaN, then there
issomething in it.
   if ( isnan ( t ) || isnan ( h ) ) {
     Serial.println ( " Sensor not working " )
     ;
   }
 else
{

Serial.print ( " Temp is " ) ;
     Serial.print ( temp ) ; // Printing the temperature
ondisplay.
     Serial.println ( " *C " ) ; // Printing " *C " on
     display.Serial.print ( " Humidity in % is : " ) ;
     Serial.print ( humidity ) ;  // Printing the humidity on display
     Serial.print ( " % \t " ) ;              // Printing "%" on
     display

   }
 }
```

- **Weather Station using DHT11 and arduino**

In this example we will make a weather station that will sense the humidity and temperature and will show it on the lcd attached to the Arduino. Make the circuit as shown in the diagram. The resistor in the circuit will make the black light darker. We have used the 220 ohm resistor but you can use any resistor having value near to that. The potentiometer we used in the circuit is used to set the screen contrast. We have used the 10 K ohm value but you can choose any value relative to that one.



- **Components Required**
  - Arduino Uno (you can use any)
  - 16 x 2 LCD
  - DHT11 Temperature and humidity sensor
  - 10 K ohm potentiometer
  - 220 ohm resistor

- **Code of weather station using arduino and DHT11**

```
// This code is for the weather station using the DHT11 humidity and temperature
sensor.

// Install the library of the DHT before uploading the code in the Arduino IDE
#include < dht.h >      // including the DHT library

#include < LiquidCrystal.h >   // including the LCD library
LiquidCrystal lcd ( 12, 11, 5, 4, 3, 2 ) ;  // initializing the lcd pins
dht DHT ;                      // declaring dht a variable

#define DHT11_PIN 8             // initializing pin 8 for dht
void setup ( ) {

  lcd.begin ( 16, 2 ) ;       // starting the 16 x 2 lcd

}

void loop ( )

{

  int chk = DHT.read11(DHT11_PIN ) ;          // Checking that either the dht is
working or not

  lcd.setCursor ( 0, 0 ) ;                 //  starting the cursor from  top left


  lcd.print ( " Temperature is : " ) ;        // printing the " Temperature is : " on
```

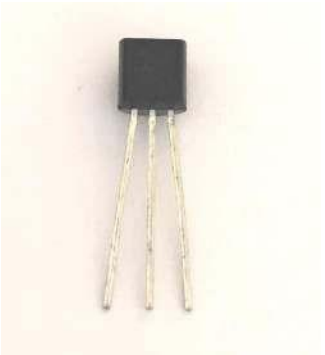
```
  lcd.print ( DHT.temperature ) ;            // printing the temperature on the lcd
  lcd.print ( ( char ) 223 ) ;

  lcd.print ( " C " ) ;                      // Printing " C " on the display
  lcd.setCursor ( 0 , 1 );

  lcd.print ( " Humidity is : " ) ;          // printing " humidity is : " on the
display

  lcd.print ( DHT.humidity ) ;               // printing humidity on the display
  lcd.print ( " % " ) ;                      // printing " % " on display

  delay ( 1000 ) ;                           // Giving delay of 1 second.


}
```

Conclusion:

EXPERIMENT NO. 11 (Group B)

- **Aim:** Write a program read the temperature sensor and send the values to the serial monitor on the computer
- **Outcome:** Understanding working principle of DHT11, LM35 temperature sensor, Relationship between different temperature scales
- **Hardware Requirement:** Arduino, LED, LM35, DHT11, etc
- **Software Requirement**: Arduino IDE
- **Theory:**

LM35 Temperature Sensor



LM35 Temperature Sensor Pinout

LM35 Sensor Pinout Configuration

Pin Number	Pin Name	Description
1	Vcc	Input voltage is +5V for typical applications
2	Analog Out	There will be increase in 10mV for raise of every 1°C. Canrange from -1V(-55°C) to 6V(150°C)
3	Ground	Connected to ground of circuit

- **LM35 Sensor Features**
  - Minimum and Maximum Input Voltage is 35V and -2V respectively. Typically 5V.
  - Can measure temperature ranging from -55°C to 150°C
  - Output voltage is directly proportional (Linear) to temperature (i.e.) there will be a rise of 10mV (0.01V) for every 1°C rise in temperature.
  - ±0.5°C Accuracy
  - Drain current is less than 60uA
  - Low cost temperature sensor
  - Small and hence suitable for remote applications

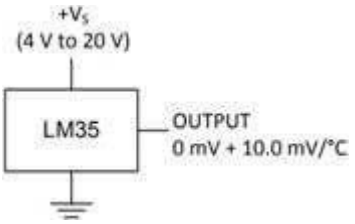
- **LM35 Temperature Sensor Equivalent**

LM34, DS18B20, DS1620, LM94022

How to use LM35 Temperature Sensor:

LM35 is a precession Integrated circuit Temperature sensor, whose output voltage varies, based on the temperature around it. It is a small and cheap IC which can be used to measure temperature anywhere between -55°C to 150°C. It can easily be interfaced with any Microcontroller that has ADC function or anydevelopment platform like Arduino.

Power the IC by applying a regulated voltage like +5V (VS) to the input pin and connected the ground pin to the ground of the circuit. Now, you can measure the temperature in form of voltage as shown



below.

If the temperature is 0°C, then the output voltage will also be 0V. There will be rise of 0.01V (10mV) for every degree Celsius rise in temperature. The voltage can converted into temperature using the below formulae.

$$V_{OUT} = 10\ mv/°C \times T$$

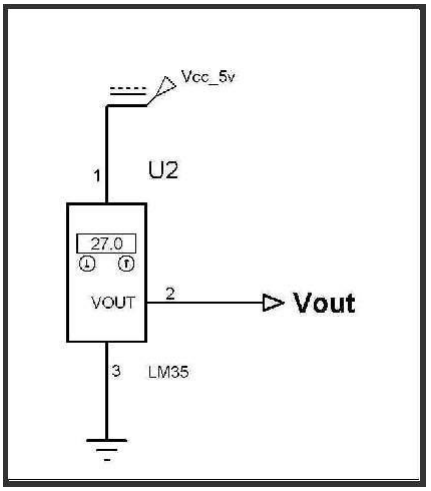
where

- V_{OUT} is the LM35 output voltage
- T is the temperature in °C

- **LM35 Temperature Sensor Applications**
  - Measuring temperature of a particular environment
  - Providing thermal shutdown for a circuit/component
  - Monitoring Battery Temperature
  - Measuring Temperatures for HVAC applications.

- **How Does LM35 Sensor Work?**

Main advantage of LM35 is that it is linear i.e. 10mv/°C which means for every degree rise in temperature the output of LM35 will rise by 10mv. So if the output of LM35 is 220mv/0.22V the temperature will be 22°C. So if room temperature is 32°C then the output of LM35 will be 320mv i.e. 0.32V.



- **LM35 Interfacing Circuit**

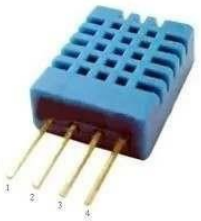
As such no extra components required to interface LM35 to ADC as the output of LM35 is linear with 10mv/degree scale. It can be directly interfaced to any 10 or 12 bit ADC. But if you are using an 8-bit ADC like ADC0808 or ADC0804 an amplifier section will be needed if you require to measure 1°C change.
LM35 can also be directly connected to Arduino. The output of LM35 temperature can also be given to comparator circuit and can be used for over temperature indication or by using a simple relay can be used as a temperature controller.

- **DHT11 interfacing with arduino and weather station**

DHT11 sensor is used to measure the **temperature** and **humidity**. It has a resistive humidity sensing component and a negative temperature coefficient (NTC). An 8 bit MCU is also connected in it which is responsible for its fast response. It is very inexpensive but it gives values of both temperature and humidity at a time.

- **Specification of DHT11**
  - It has humidity range from 20 to 90% RH
  - It has temperature range from 0 – 50 C
  - It has signal transmission range of 20 m
  - It is inexpensive
  - It has fast response and it is also durable

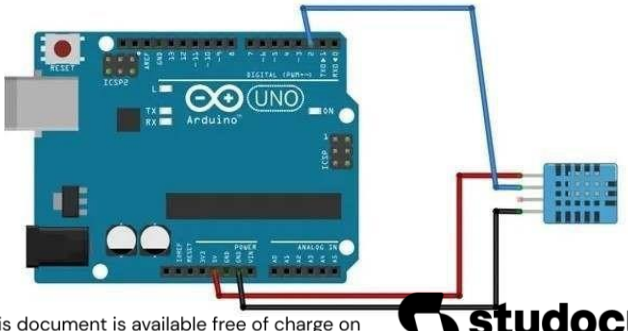
- **DHT11 Pin out**



  - The first pin of the DHT11 is vcc pin.
  - The second pin of the DHT is Data pin.
  - The third pin is not used.
  - The fourth pin of the DHT sensor is ground.

- **DHT11 interfacing with arduino**

First of all connect the ground and the VCC of the DHT11 temperature and humidity sensor to the ground and 5v of the **Arduino**. Then connect the data pin of the DHT11 sensor to the pin 2 of the Arduino.



To run the following code in Arduino IDE you will first have to install the DHT library in you Arduino directory.

Download the zip file from here and place it in your Arduino library folder. The path to Arduino library folder for my computer is
Documents/ Arduino/ Libraries
Unzip the downloaded file and place it in this folder.
After copying the files, the Arduino library folder should have a new folder named DHT containing the dht.h and dht.cpp. After that copy the following code in the Arduino IDE and upload the code.

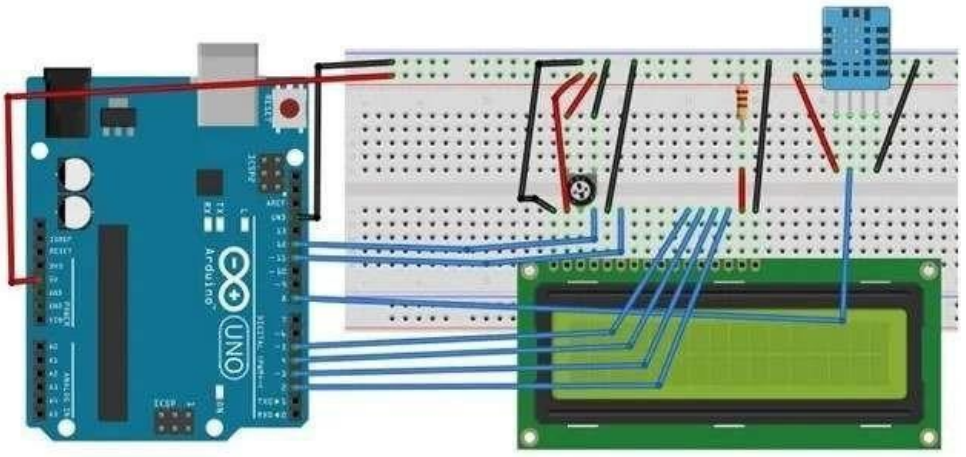
- **Code of DHT11 interfacing with arduino**

```
// Code for DHT11 Temperature and humidity sensor.
 #include " DHT.h " // including the library of DHT11 temperature
andhumidity sensor
 #define DHTPIN 2    // Selecting the pin at which we have
connectedDHT11
 #define DHTTYPE DHT11 // Selecting the type of DHT
 sensorsDHT dht ( DHTPIN, DHTTYPE ) ;
 void   setup   (   )   {
   Serial.begin ( 9600 ) ;
   dht.begin (  ) ;   // The sensor will start working
 }
 void loop ( ) {
   // Reading temperature or humidity may take about 2 seconds
because itis a very slow sensor.

float humidity = dht.readHumidity ( ) ; // Declaring h a variable
andstoring the humidity in it.
  float temp = dht.readTemperature ( ) ; // Declaring t a variable
andstoring the temperature in it.
  // Checking if the output is correct. If these are NaN, then there
issomething in it.
   if ( isnan ( t ) || isnan ( h ) ) {
     Serial.println ( " Sensor not working " )
     ;
   }
 else
{

Serial.print ( " Temp is " ) ;
     Serial.print ( temp ) ; // Printing the temperature
ondisplay.
     Serial.println ( " *C " ) ; // Printing " *C " on
     display.Serial.print ( " Humidity in % is : " ) ;
     Serial.print ( humidity ) ;  // Printing the humidity on display
     Serial.print ( " % \t " ) ;              // Printing "%" on
     display

   }
 }
```

- **Weather Station using DHT11 and arduino**

In this example we will make a weather station that will sense the humidity and temperature and will show it on the lcd attached to the Arduino. Make the circuit as shown in the diagram. The resistor in the circuit will make the black light darker. We have used the 220 ohm resistor but you can use any resistor having value near to that. The potentiometer we used in the circuit is used to set the screen contrast. We

have used the 10 K ohm value but you can choose any value relative to that one.



- **Components Required**
  - Arduino Uno (you can use any)
  - 16 x 2 LCD
  - DHT11 Temperature and humidity sensor
  - 10 K ohm potentiometer
  - 220 ohm resistor

- **Code of weather station using arduino and DHT11**

```
// This code is for the weather station using the DHT11 humidity and temperature
sensor.

// Install the library of the DHT before uploading the code in the Arduino IDE
#include < dht.h >      // including the DHT library

#include < LiquidCrystal.h >   // including the LCD library
LiquidCrystal lcd ( 12, 11, 5, 4, 3, 2 ) ;  // initializing the lcd pins
dht DHT ;                       // declaring dht a variable

#define DHT11_PIN 8              // initializing pin 8 for dht
void setup ( ) {

  lcd.begin ( 16, 2 ) ;        // starting the 16 x 2 lcd

}

void loop ( )

{

  int chk = DHT.read11(DHT11_PIN ) ;          // Checking that either the dht is
working or not

  lcd.setCursor ( 0, 0 ) ;                  //  starting the cursor from  top left


  lcd.print ( " Temperature is : " ) ;        // printing the " Temperature is : " on
the lcd

  lcd.print ( DHT.temperature ) ;             // printing the temperature on the lcd
  lcd.print ( ( char ) 223 ) ;

  lcd.print ( " C " ) ;                      // Printing " C " on the display
```

```
  lcd.print ( " Humidity is : " ) ;          // printing " humidity is : " on the
display

  lcd.print ( DHT.humidity ) ;               // printing humidity on the display
  lcd.print ( " % " ) ;                      // printing " % " on display

  delay ( 1000 ) ;                           // Giving delay of 1 second.


}
```

Temperature Scales

Thermometers measure temperature according to well-defined scales of measurement. The three most common temperature scales are the Fahrenheit, Celsius, and Kelvin scales.

- **Celsius Scale & Fahrenheit Scale**

The Celsius scale has a freezing point of water as 0ºC and the boiling point of water as 100ºC. On the Fahrenheit scale, the freezing point of water is at 32ºF and the boiling point is at 212ºF. The temperature difference of one degree Celsius is greater than a temperature difference of one degree Fahrenheit. One degree on the Celsius scale is 1.8 times larger than one degree on the Fahrenheit scale 180/100=9/5.

- **Kelvin Scale**

Kelvin scale is the most commonly used temperature scale in science. It is an absolute temperature scale defined to have 0 K at the lowest possible temperature, called absolute zero. The freezing and boiling points of water on this scale are 273.15 K and 373.15 K, respectively. Unlike other temperature scales, the Kelvin scale is an absolute scale. It is extensively used in scientific work. The Kelvin temperature scale possesses a true zero with no negative temperatures. It is the lowest temperature theoretically achievable and is the temperature at which the particles in a perfect crystal would become motionless.

- **Relationship Between Different Temperature Scales**

The relationship between three temperature scales is given in the table below:

Relationship between different Temperature Scales

Conversion	Equation
Celsius to Fahrenheit	$T_{F^o} = \frac{9}{5}T_{C^o} + 32$
Fahrenheit to Celsius	$T_{C^o} = \frac{5}{9}T_{F^o} - 32$
Celsius to Kelvin	$T_K = T_{C^o} + 273.15$
Kelvin to Celsius	$T_{C^o} = T_K - 273.15$
Fahrenheit to Kelvin	$T_K = \frac{5}{9}(T(F^0) - 32) + 273.15$
Kelvin to Fahrenheit	$T_{F^0} = \frac{9}{5}(T(K) - 273.15) + 32$

Conclusion: -

EXPERIMENT NO.

- **Aim:** Write a program using piezo element and use it to play a tune after someone knocks

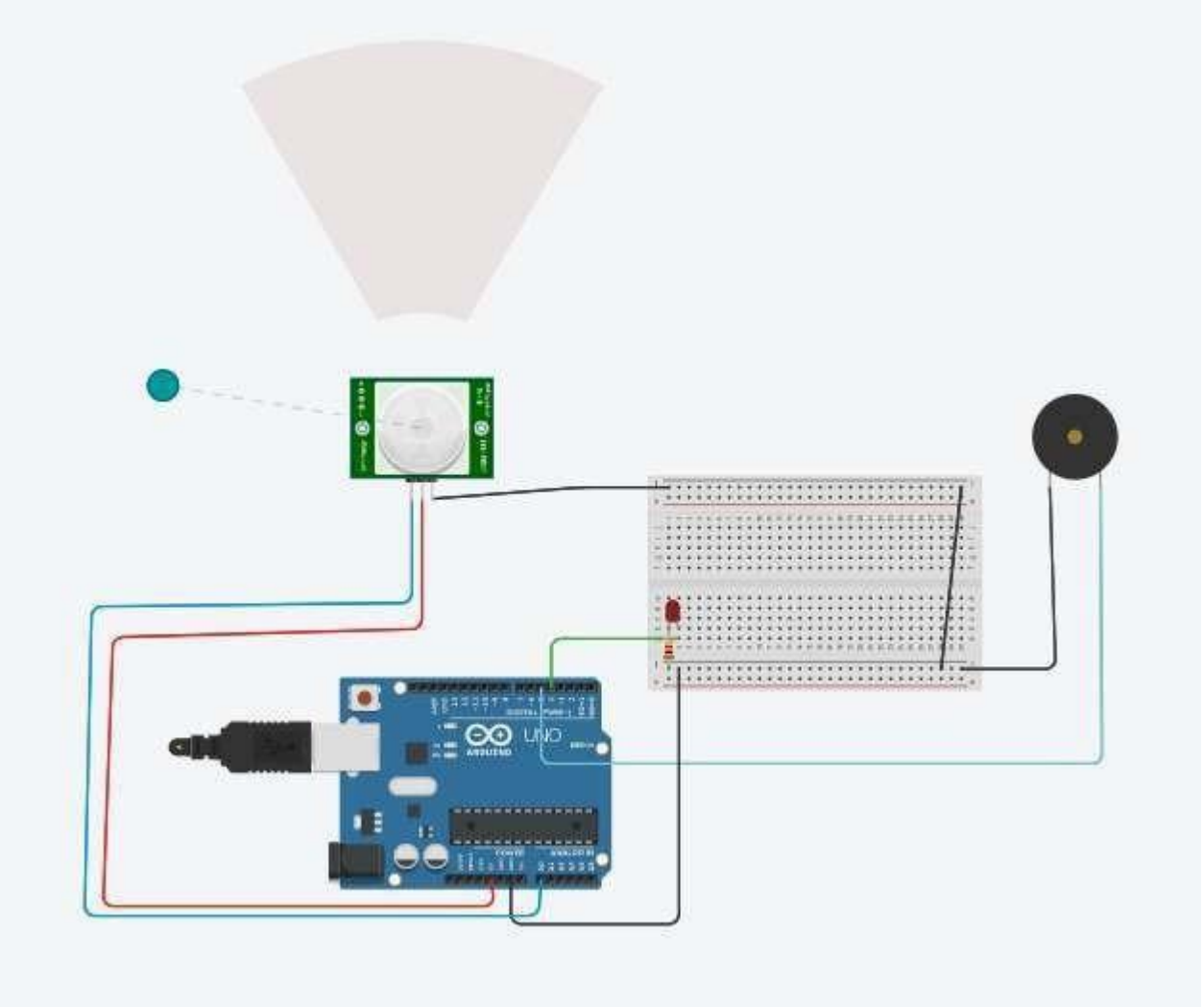
- **Outcome:** if any obstacle come in between…buzzer should knock
- **Hardware Requirement:** Arduino, LED, LM35, DHT11, etc

- **Software Requirement**: Arduino IDE

- **Theory:**

```
#define LED 4      //led at pin 4
#define buzzer 5 //buzzer at pin 5
#define sensor A0 //ir sensor at pin 6
int sound=250;     //set buzzer sound
void setup()
{
  Serial.begin(9600);
  pinMode(sensor,INPUT);
  pinMode(LED,OUTPUT);
  pinMode(buzzer,OUTPUT);
}
void loop()
{
  int detect=digitalRead(sensor);//read status of sensor
  if(detect==HIGH)            //if sensor detcts obstacle
  {
    digitalWrite(LED,HIGH);      //led on
    tone(buzzer,sound);         //buzzer sounds
  }
```

```
else
{
  digitalWrite(LED,LOW);
  noTone(buzzer);
}
delay(300);
```



EXPERIMENT NO. 13 (Group C)

- **Aim:** Study of ThingSpeak – an API and Web Service for the Internet of Things
  Connectivity, configuration and serial communication with Arduino.

- **Theory: write it by your own with the help of link provided to you.**