Dr. Y. PATIL EDUCATIONAL FEDERATION VARALE

Dr. D. Y. Patil Educational Federation's

Dr. D. Y. Patil College of Engineering & Innovation, Varale, Talegaon, Pune

217531: Internet of Things Laboratory

Practical / Term Work Journal
Department of Artificial Intelligence and Data Science
Academic Year (2022-23)

Name of Student : ______________________________________

Branch : _______________Division:______________________

Roll No. :_______________ Batch :_______________________

Exam Seat No. :____

Dr. Y. PATIL EDUCATIONAL FEDERATION VARALE

Dr. D. Y. Patil Educational Federation's

Dr. D. Y. Patil College of Engineering & Innovation

At. Sr.27/A/1/2C Near Talegaon Railway Station, A/p Varale, Talegaon, Tal- Maval, Dist- Pune 410507

Certificate

This is to certify that Mr./Miss./Mrs.___________________ Roll No. _____________________ has completed the Practical/term work satisfactory in subject of Internet of Things Laboratory (217531) as prescribed by the Pune University in the academic year 2022 to 2023 in the Department of Artificial Intelligence and Data Science of the Institute.

Prof.Sarojini Naik	**Prof.(Dr.)Latika Desai**	**Prof.(Dr.) Suresh Mali**
Subject In charge	**Head, AI_DS Engg.**	**Principal**

Dr. D. Y. Patil Educational Federation’s

Dr. D. Y. Patil College of Engineering & Innovation

At. Sr.27/A/1/2C Near Talegaon Railway Station, A/p Varale, Talegaon, Tal- Maval, Dist- Pune 410507

217531: Internet of Things Laboratory

Laboratory Journal

INDEX

Expt. No.	Name of Experiment	Page No.	Date of Performance	Remark

Date **Subject In charge**

Dr. D. Y. Patil Educational Federation's

Dr. D. Y. Patil College of Engineering & Innovation

At. Sr.27/A/1/2C Near Talegaon Railway Station, A/p Varale, Talegaon, Tal- Maval, Dist- Pune 410507

217531: Internet of Things Laboratory

Laboratory Journal

INDEX

Expt. No.	Name of Experiment	Page No.	Date of Performance	Remark

Date **Subject In charge**

Dr. D.Y. PATIL EDUCATIONAL FEDERATION	*Dr. D. Y. Patil Educational Federation's* **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ____________________	**Marks** ______ / 10
Name : ____________________ **Subject:** ____________ **Class :** __________ **Roll No :** __________ **Expt. No.** ________ **Date :** __________ **Batch :** ____________ **Title :** ____________________		**Remarks & Sign**

Experiment Title

AIM: Study of Raspberry-Pi, Beagle board, Arduino.

Objective: To study IoT platforms such as Raspberry-Pi/Beagle board/Arduino.

Theory:

Internet of Things: - IoT is short for **Internet of Things**. **The Internet of Things** refers to the ever-growing 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 into a 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.

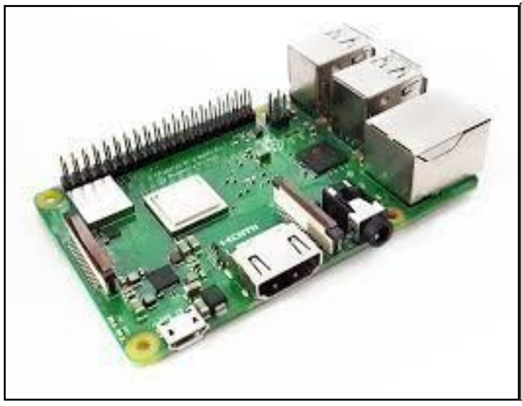
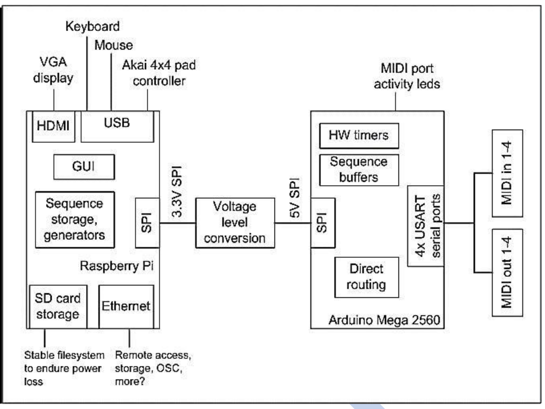



Fig.A.1: - **Raspberry-Pi Architecture**

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

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 Beagle Board 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.

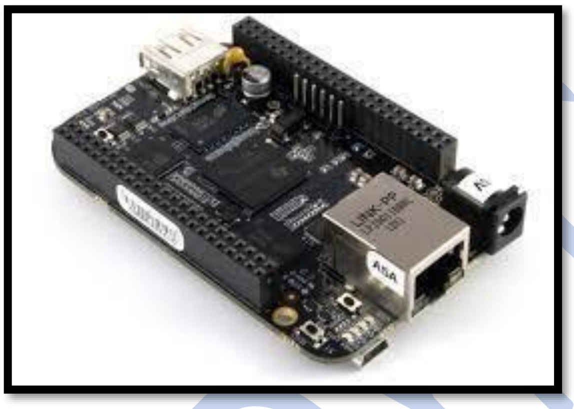


Fig.B.1: -Beagle Board Black architecture

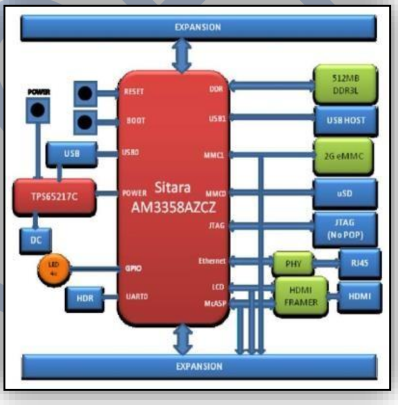


Fig.B.1: - Beagle Board Black

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.

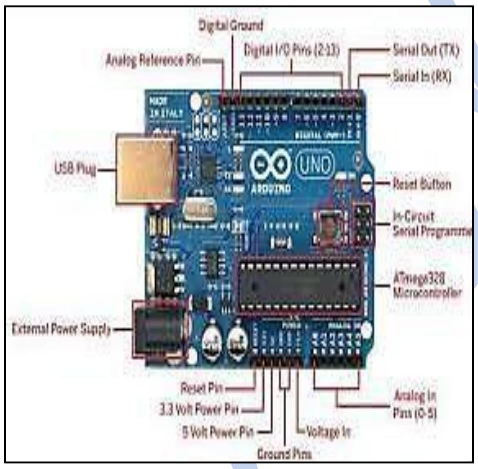


Fig.C.1: - Arduino Board

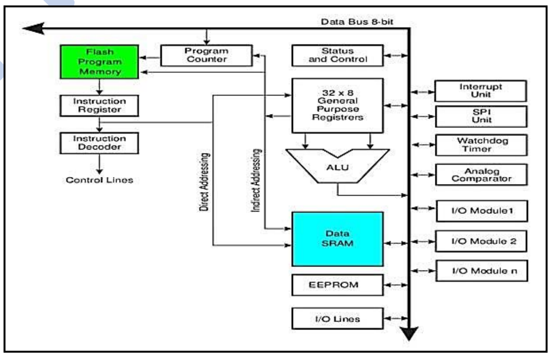


Fig.C.1: - Arduino Board Architecture.

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(), and digitalWrite() commands. Each pin has an internal pull-up resistor which can be turned on and 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 digital pins.

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 power source). 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.

Other Pins

- **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 the board and the computer; can be used to power the board) (yellow)

Conclusion: - Thus, we have studied of IoT platforms such as Raspberry-Pi/Beagle board/Arduino.

	Dr. D. Y. Patil Educational Federation's Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION Varale, Talegaon, Pune - 410507 **Department of First Year** ____________________	**Marks** ______ / 10
Name : ____________________ **Subject:**__________ **Class :**__________ **Roll No :**__________ **Expt. No.**______ **Date :**__________ **Batch :**__________ **Title :** ____________________		**Remarks & Sign**

Assignment No. 02 Date of Performance: ………………….……………

Assignment No:2

Title: Study of different operating systems for Raspberry Pi / Beagle board. Understanding the process of Os installation on Raspberry – Pi/ Beagle board

Objectives: To study various supporting OS platforms for Raspberry-Pi /Beagle board

Outcomes: Students will be able to understand the different supporting OS platforms of Raspberry- Pi/ Beagle board

Pre-requisites: Fundamentals of Operating Systems
Fundamentals of Computer Organization

Hardware Requirements: Raspberry Pi starter kit
Unit of Beagle board

Software requirement: Windows 7 64 bit or higher/ Ubuntu 16.04 or higher,
Raspberry Pi desktop RASBIAN

Operating systems for single board computers

• Single board computers support a wide range of operating system software. The purpose of the operating system is to allow control of and interaction with a single board computer and to provide a framework of system services (Disk I/O, Communications, memory management, scheduling, etc) on which to run applications.

• The major types of operating system software are:

1. RTOS-Real time Operating System
2. Embedded Linux
3. Desktop Linux
4. Embedded Windows
5. Desktop Windows
6. Roll your own or in-house
7. UNIX
8. Sun Solaris
9. BSD

Desktop Operating systems

• Desktop operating systems (Windows and Linux) are used in products such as Kiosks and point of sale (POS) terminals as well as for general purpose computing. Desktop Operating systems make no guarantees about speed or responsiveness to real world events. Mission critical systems (systems that can't be allowed to fail) are usually not built using desktop operating systems.

Soft Real Time or Non-Realtime Operating Systems

• Embedded operating systems such as Embedded Linux or Embedded Window are often used to power so-called "intelligent products" such as cell phones, home electronics and flat screen TV sets.

• These devices do not require hard real time response to computing deadlines, Response times are often dependent on system load and as such cannot be guaranteed. These operating systems support other embedded features such as instant ON/ Boot to make them more suitable for embedded devices.

Real- Time Operating Systems (RTOS)

• Real time simply means that a response must be correct and must meet a timing deadline every time or the systems has failed.

• Real time operating systems are used for the same types of embedded devices as Embedded LINUX and Embedded Windows but due to their ability to meet hard timing and response deadlines, can also be used for controlling things like industrial instruments, anti-lock braking system etc. Real-time operating systems will guarantee a response to an interrupt or the completion of a system call in all cases, regardless of the load on the system.

Roll your Own or in-house System Software

• Some Single Board Computer applications do not use an operating system. This may be because the system must be hand optimized to meet tight real-time requirements or because it does not require the services (and attendant overhead) that an operating system brings. In these cases, engineers will write all the code required to run their embedded application using an embedded compiler and assembler. These embedded systems are typically written in C, C++ and assembler.

Operating System for Raspberry-Pi

• The software offered are RASPIAN, PIDORA, OPENELEC, RASPBMC, RISC OS, ARCH LINUX. All this software can be downloaded easily and for free from the official forum under the NOOBS (new out of the box software) category.

• It provides support for functioning and coding in Python as the main programming language.It also provides support for BASIC C, C++, Java, Perl and Ruby.

• Booting Process:

• Since the board has been designed with curious school children in mind. It’s easy to use. The booting method involves the following steps:

• Downloading the NOOBS operating system install manager from the official forum of Raspberry Pi.

• Formatting a microSD Card.

• Burning the NOOBS image onto a microSD Card.

• Inserting the card into the microSD card slot on the RaspberryPi

• Plugging in keyboard, mouse and monitor cable onto the board and to the monitor

• Plugging in the USB power cable

The boot process has now begun an a configuration window appears to enable the camera module if present and setting the date and time.

• The command line interface loads up asking for the username and password, upon submitting successful information the board is fully operational.

• The graphical user interface can be chosen by typing startx.

• Default username and passwords for the first boot are: username:pi,
Password: raspberry.

After the booting process the board can be utilized for any project.

Conclusion: Study of different operating systems for Raspberry Pi / Beagle board. Understanding the process of Os installation on Raspberry – Pi/ Beagle board

Dr. D.Y. PATIL EDUCATIONAL FEDERATION	*Dr. D. Y. Patil Educational Federation's* **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ______________________	**Marks** ______ **/ 10**
Name : ______________________ **Subject:**__________ **Class :**__________ **Roll No :**__________ **Expt. No.**______ **Date :**__________ **Batch :**__________ **Title :** ______________________		**Remarks & Sign**

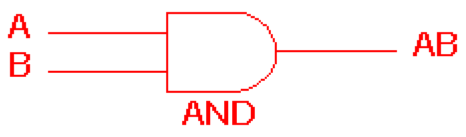
Assignment:3
Title: Study of different GATES(AND, OR, XOR), Sensors and basic Binary operations.
Objective: To study different GATES, sensor and basic operation such as addition, subtraction etc.
Hardware Requirement: Logical Gates, Sensor etc.
Software Requirement: Raspbian OS

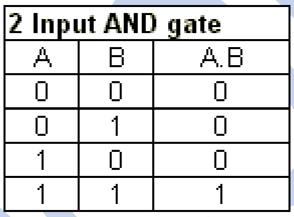
Theory:

1) All Basic logic GATES:

Digital systems are said to be constructed by using logic gates. These gates are the AND, OR, NOT, NAND, NOR, EXOR and EXNOR gates. The basic operations are described below with the aid of truth tables.

AND gate

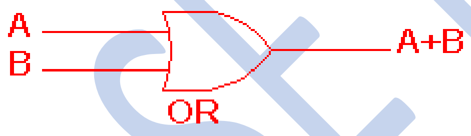


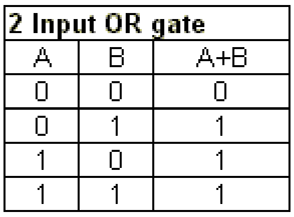


2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

The AND gate is an electronic circuit that gives a **high** output (1) only if **all** its inputs are high. A dot (.) is used to show the AND operation i.e. A.B. Bear in mind that this dot is sometimes omitted i.e. AB

OR gate

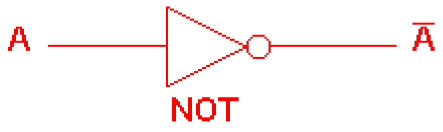


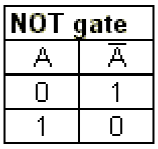


2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

The OR gate is an electronic circuit that gives a high output (1) if **one or more** of its inputs are high. A plus (+) is used to show the OR operation.

NOT gate





NOT gate	
A	$\bar{A}$
0	1
1	0

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an *inverter*. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs. The diagrams below show two ways that the NAND logic gate can be configured to produce a NOT gate. It can also be done using NOR logic gates in the same way.

- Circuit Diagram
- Truth Table
- Logic Symbol

2) Sensor:
IoT sensors are pieces of hardware that detect changes in an environment and collect data. They're the pieces of an IoT ecosystem that bridge the digital world to the physical world. IoT sensors may detect things like temperature, pressure, and motion, and if they are connected to a network, they share data with the network.

- Types of sensor

1. Temperature Sensors

Temperature sensors measure the amount of heat generated from an area or an object. They detect a temperature change and convert the findings to data. Temperature sensors are used in various industries, including manufacturing, healthcare, and agriculture. Some examples are thermistors, thermocouples, and resistor temperature detectors (RTD).

2. Proximity Sensors

Proximity sensors detect the presence or absence of objects near the sensor without physical contact. They often emit a beam of radiation like infrared or an electromagnetic field. They can be used for process monitoring and control, object counting, assembly lines, and determining available space. Proximity sensors are common in retail settings, industrial complexes, and parking lots. Some examples are photoelectric, magnetic, capacitive, inductive, and ultrasonic.

Inductive Proximity Sensors
Ultrasonic Proximity Sensor
Capacitive Proximity Sensor
Optical Proximity Sensor
https://chemicalengineeringworld.com

v

3. Pressure Sensors

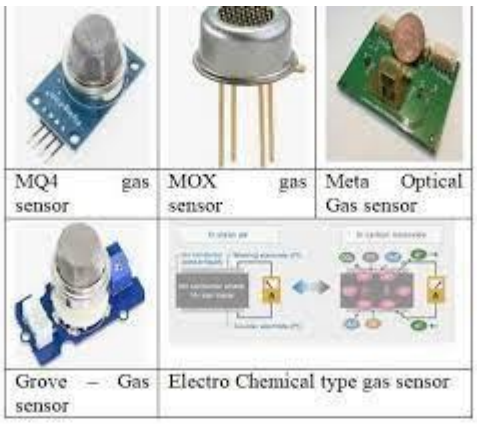
These sensors detect changes in a gas or liquid. When the pressure range is beyond a set threshold, pressure sensors alert to the problem. They are used for leak testing, water systems, vehicles, and aircraft. For example, the BMP180 is a digital pressure sensor found in cell phones and GPS navigation devices. And some vehicles use a tire pressure monitoring system (TPMS) to alert when tire pressure is low and potentially unsafe.

4. Water Quality Sensors

As you'd expect, water quality sensors monitor the quality of water. They are often used in water distribution systems, but they function in a variety of industries. There are different kinds of water sensors, including residual chlorine sensors, turbidity sensors, pH sensors, and total organic carbon sensors.

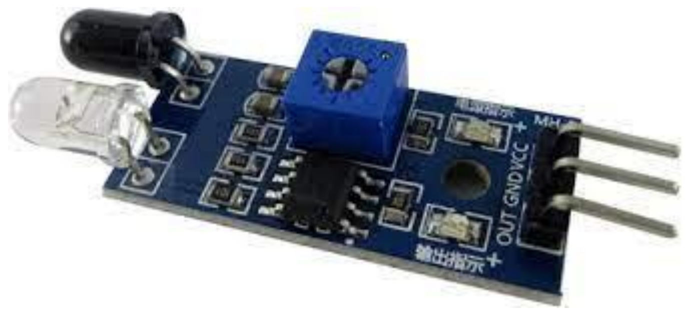
5. Chemical and Gas Sensors

These sensors monitor air quality for the presence of toxic or hazardous gas. They often use semiconductor, electrochemical, or photo-ionization technologies for detection. They are typically used in industrial and manufacturing settings, though they are also found in carbon dioxide detectors.



6. Infrared Sensors

Some sensors either detect or emit infrared radiation to sense characteristics and changes in the surrounding area. They're useful for measuring heat emissions from an object. Infrared sensors are used in remote controls, healthcare settings, and even by art historians authenticating artwork.



7. Smoke Sensors

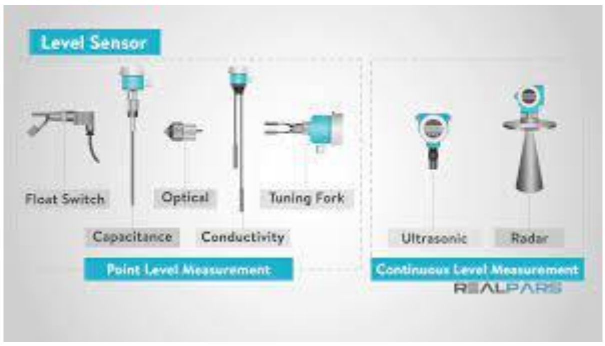
Most people are familiar with smoke detectors, as they have protected our homes and businesses for a long time. However, with improvements based on IoT, smoke detectors are now more user-friendly, convenient, and wire-free.

8. Motion Sensors

Motion sensors detect physical movement in an area. Of course, these sensors play a significant role in the security industry, but they are used in nearly every industry. Applications include automated sinks and toilet flushers, automatic door controls, energy management systems, and automated parking systems. Standard motion sensors include ultrasonic, microwave, and passive infrared (PIR).

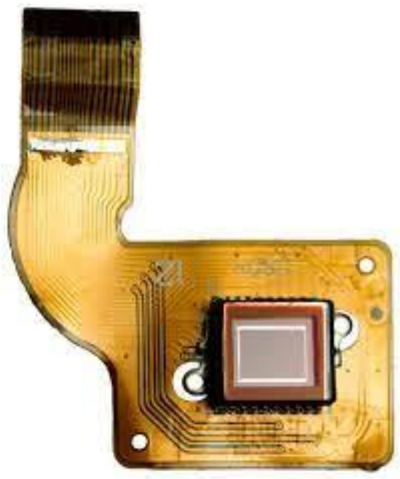
9. Level Sensors

Level sensors detect the level of various substances, including powder, granular material, and liquids. Industries that use them include water treatment, food and beverage manufacturing, oil manufacturing, and waste management. They can detect the level of liquid in a container and can even determine the amount of waste in a dumpster.



10. Image Sensors

These sensors convert optical images into signals and are generally used to display or store files electronically. They are found in radar and sonar, biometric devices, night vision equipment, medical imaging, digital cameras, and even some cars. Charge-coupled devices (CCD) and complementary metal-oxide semiconductors (CMOS) are most commonly used.



11. Humidity Sensors

These sensors measure the amount of water vapor in the air. Typical uses include heating and air conditioning systems (HVAC) and weather monitoring and prediction. When humidity must be tightly controlled, such as in museums, hospitals, and greenhouses, humidity sensors assist the process.

12. Accelerometer Sensors

Accelerometer sensors detect the orientation of an object and the rate of change, including tap, shake, tilt, and positioning. They are used in many industries for smart pedometers, anti-theft protection, and monitoring auto fleets. Some types are capacitive accelerometers and hall-effect accelerometers.

13. Gyroscope Sensors

A gyroscope sensor measures the angular rate or velocity, or the speed of rotation around an axis. They are generally used for navigation in the auto industry for navigation and anti-skid systems as well as in video games and drones. Some examples include optical gyroscopes, rotary gyroscopes, and vibrating structure gyroscopes.

14. Optical Sensors

Optical sensors measure light and convert it into electrical signals. Many industries make use of optical sensors, including auto, energy, healthcare, and aerospace. Sensors include fiber optics, photodetector, and pyrometer.

- Application
- **healthcare for monitoring the flow of blood, BP, etc**.
- **boiler, washing machines and dishwashers for heating systems, and other white products**. Many vehicles, including automobiles, trains, buses, etc., employ sensors to monitor petroleum temperature and pressure, jets and steering systems
- Advantages and Disadvantage

Sensor	**Advantages**	**Disadvantages**	**Applications**
Limit Switch Sensor	•High Current Capability •Low Cost •Familiar “Low- Tech” Sensing	Requires Physical Contact with Target Very Slow Response Contact Bounce	•Interlocking •Basic End-of- Travel Sensing
Photoelectric	•Senses all Kinds of Materials	•Lens Subject to	•Packaging

Sensor	•Long Life •Longest Sensing Range •Very Fast Response Time	Contamination •Sensing Range Affected by Color and Reflectivity of Target	•Material Handling •Parts Detection
Inductive **Sensor**	•Resistant to Harsh Environments •Very Predictable •Long Life •Easy to Install	•Distance Limitations	•Industrial & Machines •Machine Tool •Senses Metal- Only Targets
Capacitive **Sensor**	•Detects Through Some Containers •Can Detect Non-Metallic Targets	•Very Sensitive to Extreme Environmental Changes	•Level Sensing
Ultrasonic **Sensor**	•Senses all Materials	•Resolution •Repeatability	•Anti-Collision •Doors •Web Brake

		•Sensitive to Temperature Changes	•Level Control

-

3) Basic Binary

- All Laws of binary operations

Binary Operations are arithmetic operations such as addition, subtraction, division, and multiplication that are performed on two or more operands. Binary stands for two. In the case of more than two numbers, we first perform the given mathematical operation on two numbers and subsequently perform the other operations on its result. Thus, for any binary operation two inputs are required.

Closure Property

Operation * with non-empty set A has closure property if **a ∈ P, b ∈ P ⇒ a * b ∈ P.**

For example, addition is a binary operation closed on natural numbers, rational numbers, and integers.

Associative Property

For a non-empty set A with * as the binary operation, the following associative property holds true,

(a * b) *c = a*(b * c), where {a, b, c} ∈ A

Note: Subtraction denoted by '-' is not associative .

Commutative Property

The binary operation * is commutative for a non-empty set A,

a * b = b * a, for all (a, b) ∈ A

For example, 4 and 6 are elements of the natural number set. 4+6 = 6+4= 10.

Distributive Property

Let \$ and # be binary operation on a non-empty set A, then by **distributive property**

a\$ (b # c) = (a \$ b) # (a \$ c), for all {a, b, c} ∈ A

For example, 4(3+5) = 4x3 + 4x5= 32

Identity Element

For a binary operation # on an non-empty set A,

e ∈A, if e#a = a#e= a, ∀ a ∈ A

where e is the identity element.

Inverse Property

If \$ is a binary operation on a non-empty set A,

a \$ b = b \$ a = e, ∀ {a, b, e}∈A

where a is the **inverse** of and e is the identity element.

Conclusion: Studied of different GATES(AND, OR, XOR), Sensors and basic Binary operations.

Dr. D.Y. PATIL EDUCATIONAL FEDERATION	*Dr. D. Y. Patil Educational Federation's* **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ______________________________	**Marks** ______ **/ 10**
Name : ____ **Subject:** ________________ **Class :** ____________ **Roll No :** ____________ **Expt. No.** ________ **Date :** ______________ **Batch :** ________________ **Title :** ____		**Remarks & Sign**

Assignment -4

Title: Study of Connectivity and Configuration of Raspberry-Pi/ Beagle Board circuit with basic peripherals, LEDs, Understanding GPIO and its use in program.

Objectives:

• To study the fundamentals of connectivity schemes of Raspberry-pi/ Beagle board.
• To study the configuration of with basic peripherals, LEDs
• To understand the GPIO pins of Raspberry-Pi 3
• 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

Outcomes:

• Students will be able to use Raspberry Pi/ Beagle board circuit with external resources
• To program the GPIO pins of Raspberry Pi 3 using python.

Pre-requisites:

• Fundamentals of Operating Systems
• Fundamentals of Computer Organization

Hardware Requirement:

▪ Raspberry-Pi Starter kit
▪ Unit of Beagle Black board
▪ LEDs
▪ Breadboard
▪ 5V Power Supply

Software Requirement:

• Windows 7 64 bit or Higher/ Ubuntu 16.04 or higher
• Raspberry Pi Desktop
• RASPBAIN
• Beagle Board
• GCC 6.0 of Higher/ Python 3.0 or Higher 26

Introduction:

• Raspberry Pi 3 model is the latest version of raspberry pi board
• It is released on 29 February
• The above figure show the Raspberry Pi 3 model B and its gPIO pins
• General -purpose input/output (GPIO) is a generic pin on an integrated circuit or computer.
• Board whose behaviour 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 model 3
• The pins are arranged in a 2x20 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 this row have odd numbers i.e from 1 to 39.
• The 2nd (Top) row starts with number '2', so the pins in this row have even numbers i.e from 2 to 24.
• Out of 40 pins
26 pins are GPIO pins
8 pins are Ground (GND) pins.
2 pins are 5V power supply pins
2 pins are 3.3V power supply pins
2 pins are not used

Conclusion: Studied of Connectivity and Configuration of Raspberry-Pi/ Beagle Board circuit with basic peripherals, LEDs, Understanding GPIO and its use in program.

	Dr. D. Y. Patil Educational Federation's **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ______________________________	**Marks** ______ **/ 10**
Name : ____ **Subject:**________________ **Class :**____________ **Roll No :**___________ **Expt. No.**________ **Date :**______________ **Batch :**_________________ **Title :** ____		**Remarks & Sign**

Assignment: 5
Title: Write a program using Arduino to control LED(One or more ON/OFF) or Blinking.
Objective: Connectivity and configuration of Raspberry-pi/ Beagal board/ Arduino circuit 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.

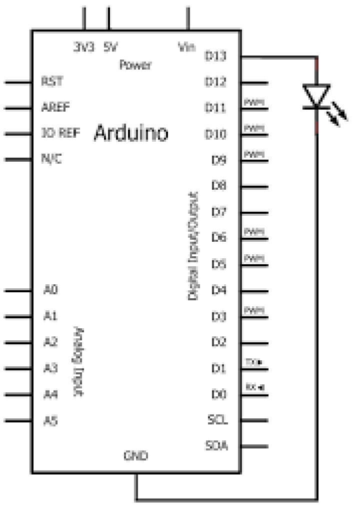
This example uses the built-in LED that most Arduino boards have. This LED is connected to a digital pin and its number may vary from board type to board type. To make your life easier, we have a constant that is specified in every board descriptor file. This constant is LED_BUILTIN and allows you to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

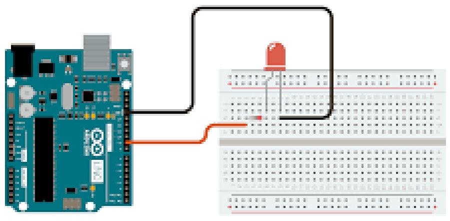
If you want to lit an external LED with this sketch, you need to build this circuit, where you connect one end of the resistor to the digital pin correspondent to the LED_BUILTIN constant. Connect the long leg of the LED (the positive leg, called the anode) to the other end of the resistor. Connect the 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.

Diagram:

Pin Diagram:



Schematic



Apparatus: Arduino Uno board, Micro-IoT sensor actuator board, Power adaptor. **Interface:**

LED	**Arduino Pin**
LED 1	0
LED 2	1
LED 3	2
LED 4	4

Procedure:

Step 1: Connect the Arduino board to the Micro-IoT Sensor board using the FRC cable provided with the board.

Step 2: Connect the Power supply adaptor and power on the circuit.

Step 3: Open Arduino IDE and create a new sketch (program) for LED blinking using the above pins.

Step 4: In the Arduino IDE go to tools□Port and select the appropriate COM port.

Step 5: In the Arduino IDE click on the upload button ( ) to compile and download the code into the Arduino UNO. When successfully downloaded the code will start running and you can observe the LED's blinking on the board.

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.

```
void setup() {

  // set the mode of the pins a s GPIO and direction as OUTPUT

  pinMode(0, OUTPUT);

  pinMode(1, OUTPUT);

  pinMode(2, OUTPUT);
  pinMode(4, OUTPUT);
}

void loop()
{
  // put your main code here, to run repeatedly:
  //Make LED ON
  digitalWrite(0, HIGH);
  delay(100);
  digitalWrite(1, HIGH);
  delay(100);
  digitalWrite(2, HIGH);
  delay(100);
  digitalWrite(4, HIGH);
  delay(200);
  //Make LED OFF
  digitalWrite(0, LOW);
  delay(100);
  digitalWrite(1, LOW);
  delay(100);
  digitalWrite(2, LOW);
  delay(100);
  digitalWrite(4, LOW);
  delay(200);
}
```

Conclusion: LED blinking is controlled by connecting to Arduino board

	Dr. D. Y. Patil Educational Federation's **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ______________________	**Marks** ______ **/ 10**
Name : ______________________________ **Subject:**____________ **Class :**__________ **Roll No :**__________ **Expt. No.**________ **Date :**____________ **Batch :**____________ **Title :** ______________________________		**Remarks & Sign**

Assignment: 6

Title: 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.

Objective: 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

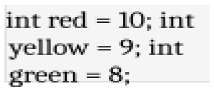
Basics

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.

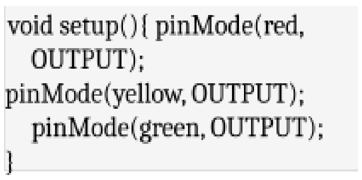
Code for the Arduino Traffic Light

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:



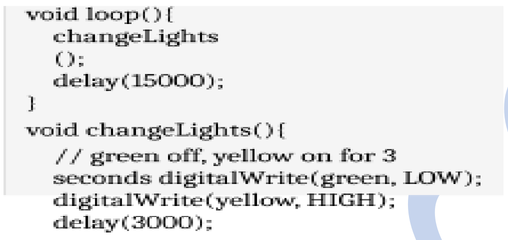
```
int red = 10; int
yellow = 9; int
green = 8;
```

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:

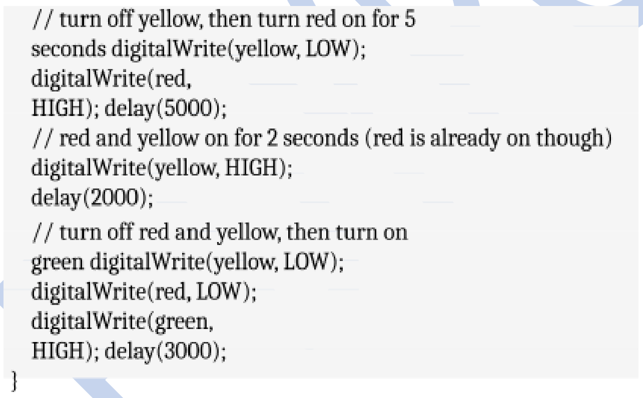


```
void setup(){ pinMode(red,
   OUTPUT);
pinMode(yellow, OUTPUT);
   pinMode(green, OUTPUT);
}
```

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 traffic light 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 is very 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 resistor attached 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 would be 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 never change, 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 to

have two traffic lights:

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

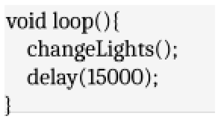
Code for the Arduino Traffic Light with Junction

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:

Apparatus: Arduino Uno board, Micro-IoT sensor actuator board, Power adaptor.
Interface:

LED	Arduino Pin
LED 1	0
LED 2	1
LED 3	2
LED 4	4

Procedure:

Step 1: Connect the Arduino board to the Micro-IoT Sensor board using the FRC cable provided with the board.

Step 2: Connect the Power supply adaptor and power on the circuit. **Step 3:** Open Arduino IDE and create a new sketch (program) using the above pins. **Step 4:** In the Arduino IDE go to tools□Port and select the appropriate COM port.

Step 5: In the Arduino IDE click on the upload button ( ) to compile and download the code into the Arduino UNO. When successfully downloaded the code will start running and you can observe the LED's getting illuminated after the count is changed on the board.

Observation:

You can observe the LED's turning ON after the specific counter has expired. Additionally you can print the value of the counter on the serial terminal using the ***serial.println*** function.

Code:

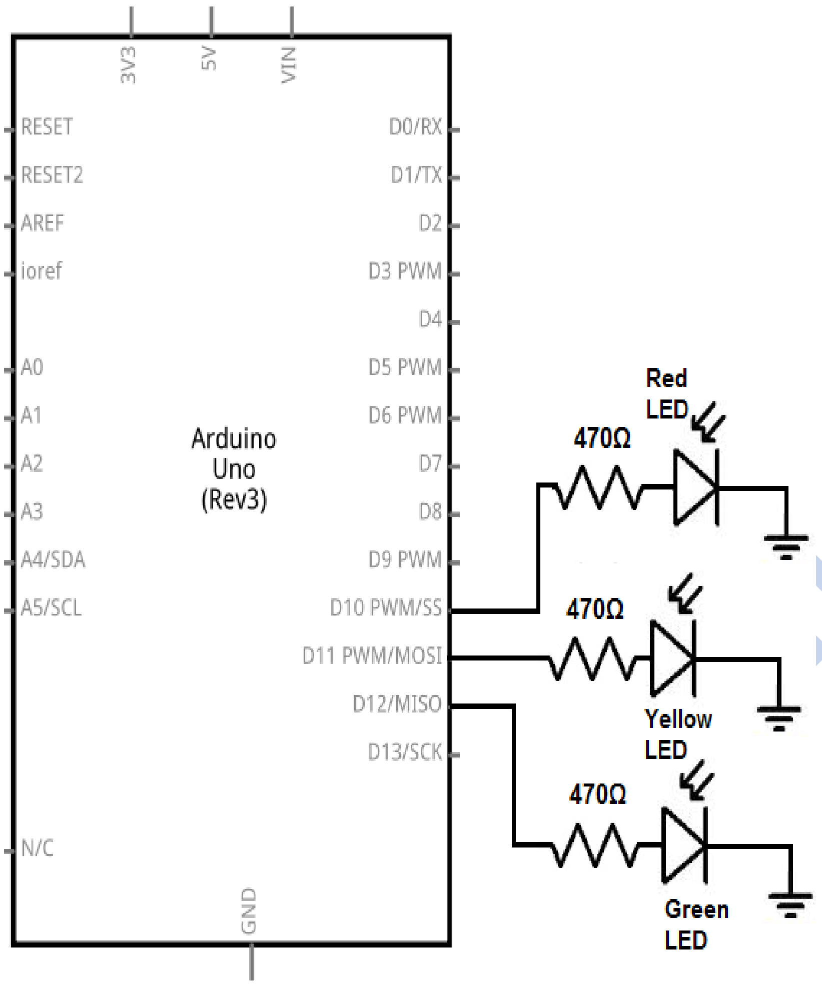
```
const int green_led = 0;
const int yellow_led = 1;
const int red_led = 2;
unsigned int counter;
void setup() {
 // put your setup code here, to run once:
 pinMode(green_led, OUTPUT);
 pinMode(red_led, OUTPUT);
 pinMode(yellow_led, OUTPUT);

 digitalWrite(yellow_led,HIGH);
 digitalWrite(green_led,HIGH);
 digitalWrite(red_led,HIGH);
 }
void loop() {
counter = counter+1;
delay(200);
if(counter <= 100)
 {
 digitalWrite(yellow_led,HIGH);
 digitalWrite(green_led,LOW);
 digitalWrite(red_led,HIGH);
 }
else if((counter > 100)&& (counter <= 200)) {
 digitalWrite(yellow_led,LOW);
 digitalWrite(green_led,HIGH);
 digitalWrite(red_led,HIGH);
 }
else if(counter > 200)
 {
 digitalWrite(yellow_led,HIGH);
 digitalWrite(green_led,HIGH);
 digitalWrite(red_led,LOW);
 }
if (counter > 210) counter =0;
 }
```

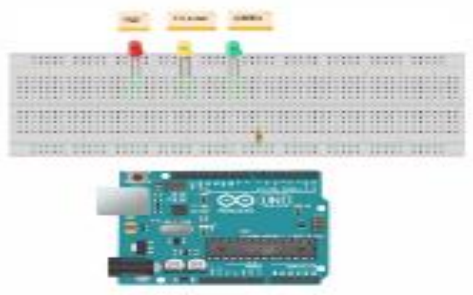
Conclusion: Executed the program that illuminates the green LED if the counter is less than 100, illuminates the yellow LED if the counter is between 101 and 200 and illuminates the red LED if the counter is greater than 200.

Dr. D.Y. PATIL EDUCATIONAL FEDERATION	*Dr. D. Y. Patil Educational Federation's* **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ________________	**Marks** ______ **/ 10**
	Name :____________________ **Subject:**__________ **Class :**__________ **Roll No :**__________ **Expt. No.**______ **Date :**__________ **Batch :**__________ **Title :**____________________	**Remarks & Sign**

Assignment: 7
Title: 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
Objective: 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.
Diagram: Pin Diagrams



Semantic



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 LEDsto 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 traffic light 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 is very 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 resistor attached 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 would be 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 never change, 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 to have 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);
}
```

Apparatus: Arduino Uno board, Micro-IoT sensor actuator board, Power adaptor.

Interface:

LED	Arduino Pin
LED 1	0
LED 2	1
LED 3	2
LED 4	4

Serial terminal of Arduino IDE.

Procedure:

Step 1: Connect the Arduino board to the Micro-IoT Sensor board using the FRC cable provided with the board.

Step 2: Connect the Power supply adaptor and power on the circuit. **Step 3:** Open Arduino IDE and create a new sketch (program) using the above pins. **Step 4:** In the Arduino IDE go to tools□Port and select the appropriate COM port.

Step 5: In the Arduino IDE click on the upload button ( ) to compile and download the code into the Arduino UNO. When successfully downloaded the code will start running.

Step 6: Open the serial monitor from **Tools** **Serial Monitor**. Set the baud rate to 9600 and type 'b', 'g' , 'y' or 'r' and check the color of the LED.

Observation:

You can observe that when the user enters 'b' the green LED blinks, when the user enters 'g' the green light is turned ON , when the user enters 'y' the yellow LED is ON and when the user enters 'r' the red LED in ON.

```
const int green_led = 0;

const int yellow_led = 1;

const int red_led = 2;

unsigned int counter;

void setup() {

 // put your setup code here, to run once: Serial.begin(9600);

 }

unsigned char key=0;

void loop() {

Serial.begin(9600);

while(1)

 {

 key = Serial.read();

 if((key == 'g')||(key == 'G')) break;  if((key == 'r')||(key ==
'R')) break;  if((key == 'y')||(key == 'Y')) break;  delay(100);

 }

//Serial.println((char)key);

Serial.end();

delay(200);

 pinMode(green_led, OUTPUT);

 pinMode(red_led, OUTPUT);
```

```
  pinMode(yellow_led, OUTPUT);

 if((key == 'g')||(key == 'G'))

 {

  digitalWrite(yellow_led,HIGH);

  digitalWrite(green_led,LOW);

  digitalWrite(red_led,HIGH);

  delay(5000);

  digitalWrite(yellow_led,HIGH);

  digitalWrite(green_led,HIGH);

  digitalWrite(red_led,HIGH);

  key = 0;

  }

 else if((key == 'y')||(key == 'Y')) {

  digitalWrite(yellow_led,LOW);

  digitalWrite(green_led,HIGH);

  digitalWrite(red_led,HIGH);

  delay(5000);

  digitalWrite(yellow_led,HIGH);

  digitalWrite(green_led,HIGH);

  digitalWrite(red_led,HIGH);

  key = 0;

  }

 else if((key == 'r')||(key == 'R')) {

  digitalWrite(yellow_led,HIGH);

  digitalWrite(green_led,HIGH);

  digitalWrite(red_led,LOW);

  delay(5000);

 digitalWrite(yellow_led,HIGH);
 digitalWrite(green_led,HIGH);
 digitalWrite(red_led,HIGH);  key = 0;

  }

 }
```

Conclusion: Executed 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

Dr. D.Y. PATIL EDUCATIONAL FEDERATION	*Dr. D. Y. Patil Educational Federation's* **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ________	**Marks** _______ / 10
	Name : ________ **Subject:** ________ **Class :** ________ **Roll No :** ________ **Expt. No.** ________ **Date :** ________ **Batch :** ________ **Title :** ________	**Remarks & Sign**

Assignment: 8

Title: Write a program that asks the user for a number and outputs the number squared that is entered

Objective: 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.

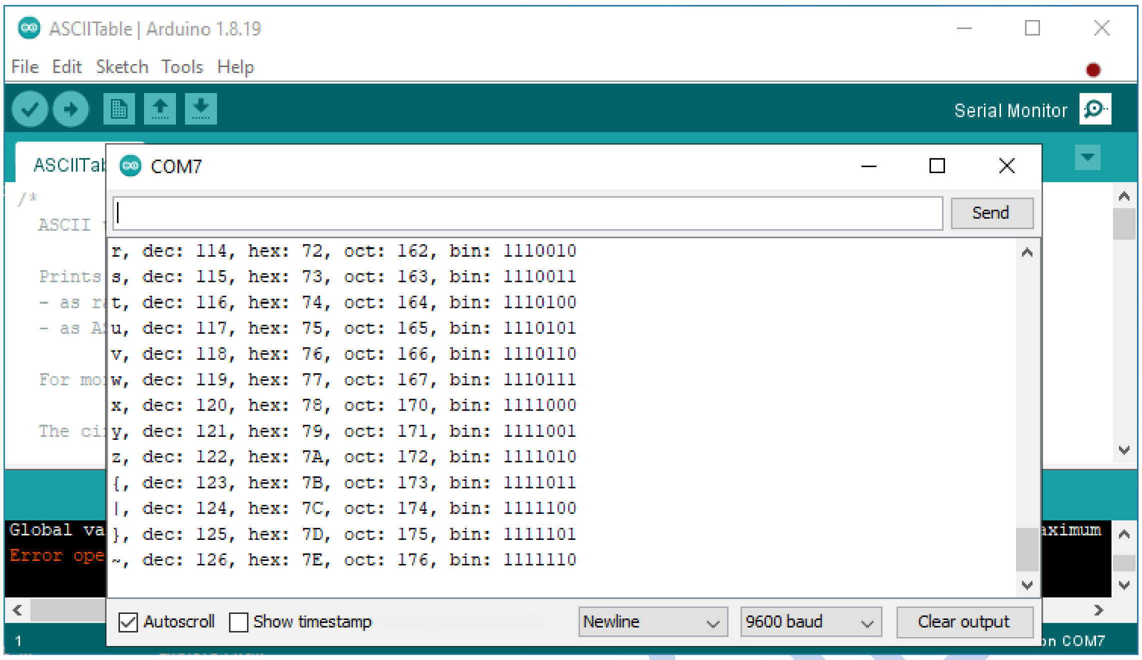
Load an Example Sketch that uses the Serial Monitor to an Arduino Board

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.



```
r, dec: 114, hex: 72, oct: 162, bin: 1110010
s, dec: 115, hex: 73, oct: 163, bin: 1110011
t, dec: 116, hex: 74, oct: 164, bin: 1110100
u, dec: 117, hex: 75, oct: 165, bin: 1110101
v, dec: 118, hex: 76, oct: 166, bin: 1110110
w, dec: 119, hex: 77, oct: 167, bin: 1110111
x, dec: 120, hex: 78, oct: 170, bin: 1111000
y, dec: 121, hex: 79, oct: 171, bin: 1111001
z, dec: 122, hex: 7A, oct: 172, bin: 1111010
{, dec: 123, hex: 7B, oct: 173, bin: 1111011
|, dec: 124, hex: 7C, oct: 174, bin: 1111100
}, dec: 125, hex: 7D, oct: 175, bin: 1111101
~, dec: 126, hex: 7E, oct: 176, bin: 1111110
```

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 to run 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.

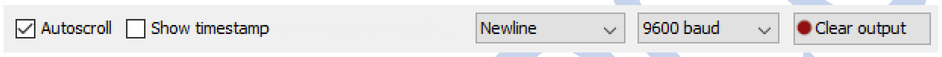
Reset the Arduino Board with the RESET Button

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 2560.

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.

Apparatus: Arduino Uno board, Micro-IoT sensor actuator board, Power adaptor.

Interface: Serial terminal of Arduino IDE.

Procedure:

Step 1: Connect the Arduino board to the Micro-IoT Sensor board using the FRC cable provided with the board.

Step 2: Connect the Power supply adaptor and power on the circuit. **Step 3:** Open Arduino

IDE and create a new sketch (program) using the above pins. **Step 4:** In the Arduino IDE go to

tools□Port and select the appropriate COM port.

Step 5: In the Arduino IDE click on the upload button (●) to compile and download the code into the Arduino UNO. When successfully downloaded the code will start running.

Step 6: Open the serial monitor from **Tools□Serial Monitor**. Set the baud rate to 9600 and give the number as input to the program and observe the output.

Observation:

In the Arduino IDE open the serial monitor and set the baud rate to 9600. When the program RUNs it prompts the user to input a number. Type (input) a number and observe that the program calculates and gives the square of the input number as output on the serial monitor.

Code:

```
void setup() {
 // put your setup code here, to run once:
Serial.begin(9600);
Serial.println("input the number");
}
void loop() {
 // put your main code here, to run repeatedly:
```

```
int input = Serial.parseInt(); // keep other operations outside the sq  function
int inputSquared = sq(input);
Serial.print(int(inputSquared));
delay(500);
}
```

Conclusion: Executed a program that asks the user for a number and outputs the number squared that is entered

Dr. D.Y. PATIL EDUCATIONAL FEDERATION	*Dr. D. Y. Patil Educational Federation's* **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ______________________________	**Marks** _______ **/ 10**
Name :____ **Subject:**________________ **Class :**____________ **Roll No :**___________ **Expt. No.**________ **Date :**______________ **Batch :**_________________ **Title :** ____		**Remarks & Sign**

Assignment: 9

Title: 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

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

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

Software Requirement: Arduino IDE

Apparatus: Arduino Uno board, Micro-IoT sensor actuator board, Power adaptor. Interface:

Peripheral	Arduino Pin
RGB_red	5
RGB_green	6
RGB_blue	3
POT_red	A0
POT_green	A2
POT_blue	A3

Procedure:

Step **1:** Connect the Arduino board to the Micro-IoT Sensor board using the FRC cable provided with the board.

Step 2: Connect the Power supply adaptor and power on the circuit. **Step 3:** Open Arduino IDE and create a new sketch (program) using the above pins. **Step 4:** In the Arduino IDE go to tools□Port and select the appropriate COM port.

Step 5: In the Arduino IDE click on the upload button ( ) to compile and download the code into the Arduino UNO. When successfully downloaded the code will start running.

Step 6: change the Potentiometers Pot_P1, Pot_p2 and Pot_P3 and observe the colour change in the RGB LED.

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.

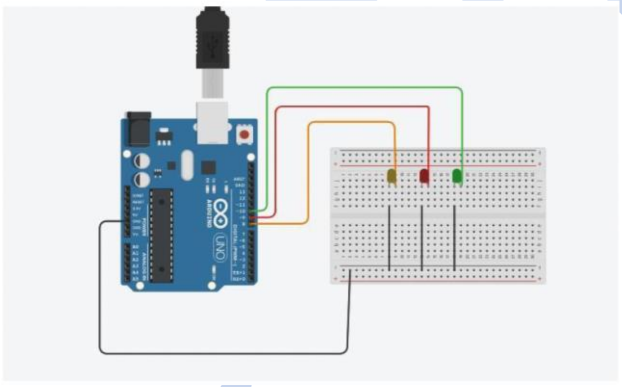
What You Need to Build an Arduino Traffic Light Controller

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

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:

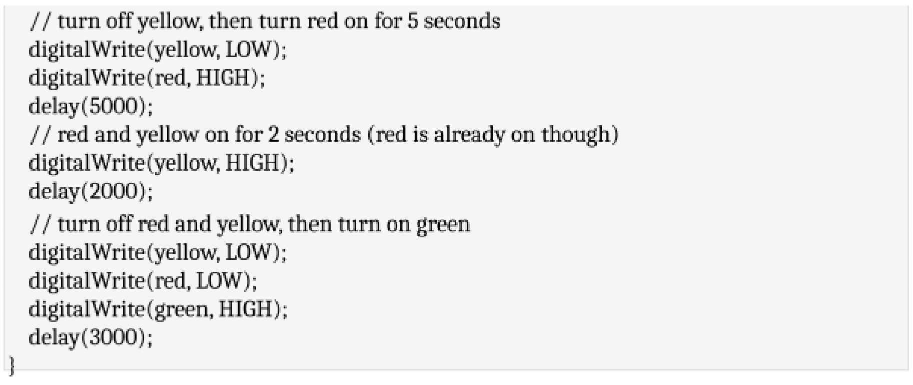
```
int red = 10; int
yellow = 9; int
green = 8;
```

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 traffic light 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 is very 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 resistor attached 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 would be 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 never change, 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 to

have 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:

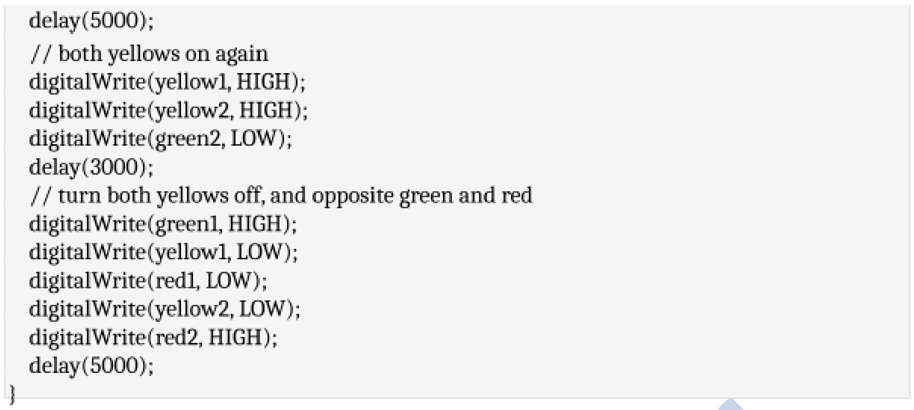
```
// light one int
red1 = 10;
int yellow1 = 9;
int green1 = 8;
// light two int
red2 = 13;
int yellow2 = 12;
int green2 = 11;
void setup(){
// light one pinMode(red1,
   OUTPUT);
pinMode(yellow1, OUTPUT);
   pinMode(green1, OUTPUT);
// light two pinMode(red2,
   OUTPUT);
pinMode(yellow2, OUTPUT);
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

Observation:

When you change the value of Potentiometers P1, P2 and P3 , we can observe that the corresponding color on the RGB led changes in intensity. Using all the 3 potentiometers we can see the whole spectrum of the color space

Code:

```
int red_light_pin= 5;

int green_light_pin = 6;

int blue_light_pin = 3;

unsigned int red,green,blue;

void setup() {

 pinMode(red_light_pin, OUTPUT);

 pinMode(green_light_pin, OUTPUT);

 pinMode(blue_light_pin, OUTPUT);

}

void loop() {

red = analogRead(A0);

red = (red/4);

green = analogRead(A2);

green = (green/4);

blue = analogRead(A3);
```

```
  blue = (blue/4);

   RGB_color(255-red, 255-green, 255-blue); // Red  delay(1000);
   }
  void RGB_color(int red_light_value, int green_light_value, int  blue_light_value)
     {
   analogWrite(red_light_pin, red_light_value);
 analogWrite(green_light_pin, green_light_value);  analogWrite(blue_light_pin, blue_light_
```

Conclusion: Executed the 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.

Dr. D.Y. PATIL EDUCATIONAL FEDERATION	*Dr. D. Y. Patil Educational Federation's* **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ______________________	**Marks** _______ **/ 10**
	Name :______________________ **Subject:**__________ **Class :**__________ **Roll No :**__________ **Expt. No.**______ **Date :**__________ **Batch :**__________ **Title :**______________________	**Remarks & Sign**

Assignment: 10

Title: 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

Apparatus: Arduino Uno board, Micro-IoT sensor actuator board, Power adaptor.

Interface:

Peripheral	**Arduino Pin**
DHT11	A1

Procedure:

Step 1: Connect the Arduino board to the Micro-IoT Sensor board using the FRC cable provided with the board.

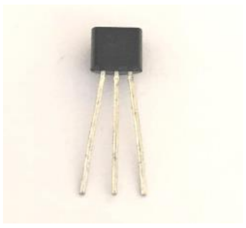
Step 2: Connect the Power supply adaptor and power on the circuit. **Step 3:** Open Arduino IDE and create a new sketch (program) using the above pins. **Step 4:** In the Arduino IDE go to tools□Port and select the appropriate COM port.

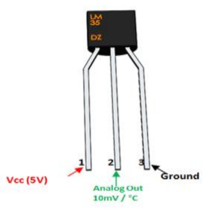
Step 5: In the Arduino IDE click on the upload button ( ) to compile and download the code into the Arduino UNO. When successfully downloaded the code will start running.

Step 6: Open the Serial Monitor in Arduino IDE Tools□Serial Monitor and observe the values of the temperature sensor.

Theory:

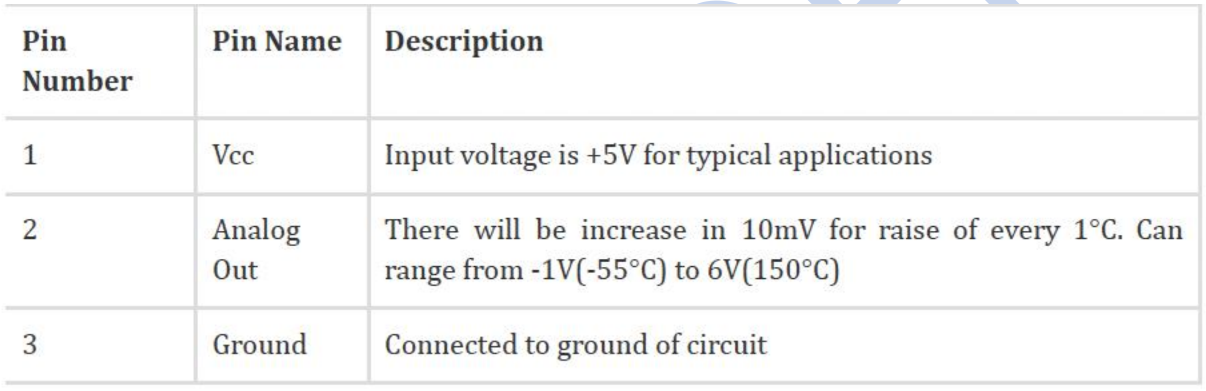
LM35 Temperature Sensor





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. Can range 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

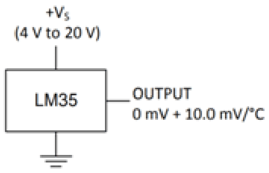
LM35 Temperature Sensor Equivalent

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 any development 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 × 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

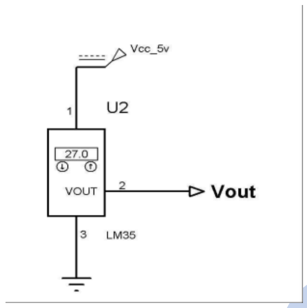
● Monitoring Battery Temperature

● 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.

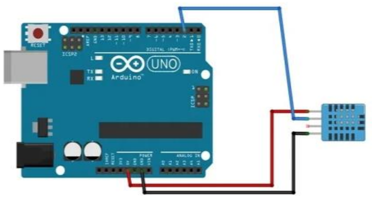
● The second pin of the DHT is Data pin.

● 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.



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 and
humidity sensor
#define DHTPIN 2 // Selecting the pin at which we have connected
DHT11
#define DHTTYPE DHT11 // Selecting the type of DHT sensors
DHT 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 it
is a very slow sensor.
float humidity = dht.readHumidity ( ) ; // Declaring h a variable and
storing the humidity in it.
float temp = dht.readTemperature ( ) ; // Declaring t a variable and
storing the temperature in it.
// Checking if the output is correct. If these are NaN, then there is
something in it.
if ( isnan ( t ) || isnan ( h ) ) {
Serial.println ( " Sensor not working " ) ;
}
else
{
Serial.print ( " Temp is " ) ;
Serial.print ( temp ) ; // Printing the temperature on
```

display.

Serial.println (" *C ") ; // Printing “ *C ” on display.

Serial.print (" Humidity in % is : ") ;

Serial.print (humidity) ; // Printing the humidity on display

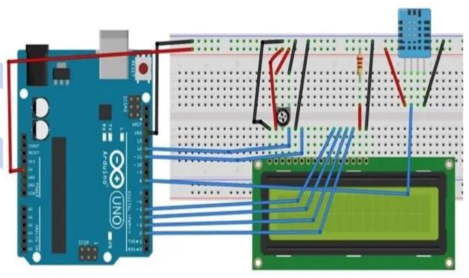
Serial.print (" % \t ") ; // Printing “%” on display

}

}

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.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.
}
```

Observation:

The DHT11 sensor is a digital sensor which measures the temperature and humidity. We can read the temperature and humidity using arduino and show the temperature in Celsius or Fahrenheit.

You can run the program and observe the output values on the Serial

Monitor Code:

```
#include <dht.h>

dht DHT;

#define DHT11_PIN A1

void setup()

 {

 Serial.begin(9600);

 Serial.println("Humidity (%),\tTemperature (C)"); }

void loop()

 {

 // READ DATA

 int chk = DHT.read11(DHT11_PIN);

 // DISPLAY DATA

 Serial.print(DHT.humidity, 1);

 Serial.print("\t");

 Serial.println(DHT.temperature, 1);

 delay(1000);

 }
```

Conclusion:

Executed the program to read the temperature sensor and send the values to the serial monitor on the computer

Assignment: 11

Title: Write a program read the temperature sensor and send the values to the serial monitor on the computer

Objective: Understanding working principle of DHT11, LM35 temperature sensor, Relationship between different temperature scales.

Hardware: Arduino, LED, LM35, DHT11, etc

Software: Arduino IDE

Apparatus: Arduino Uno board, Micro-IoT sensor actuator board, Power adaptor.

Interface:

Peripheral	Arduino Pin
DHT11	A1

Procedure:

Step 1: Connect the Arduino board to the Micro-IoT Sensor board using the FRC cable provided with the board.

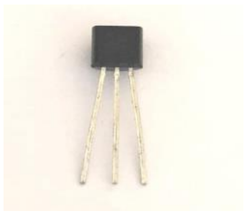
Step 2: Connect the Power supply adaptor and power on the circuit. **Step 3:** Open Arduino IDE and create a new sketch (program) using the above pins. **Step 4:** In the Arduino IDE go to tools□Port and select the appropriate COM port.

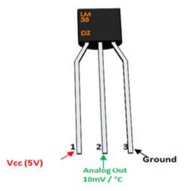
Step 5: In the Arduino IDE click on the upload button ( ) to compile and download the code into the Arduino UNO. When successfully downloaded the code will start running.

Step 6: Open the Serial Monitor in Arduino IDE Tools□Serial Monitor and observe the values of the temperature sensor.

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 application
2	Analog Out	There will be increase in 10mV for raise of every 1°C. Can range 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

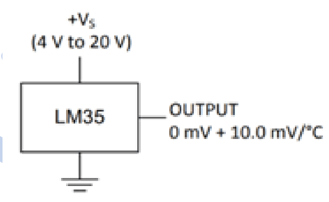
LM35 Temperature Sensor Equivalent

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 any development 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.

$$V_{OUT} = 10\ \text{mv/°C} \times T$$

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

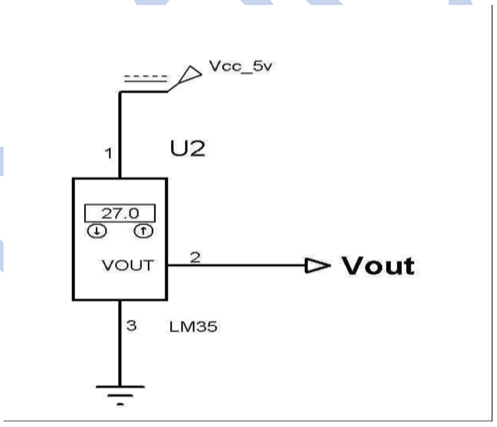
● Monitoring Battery Temperature

● 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.

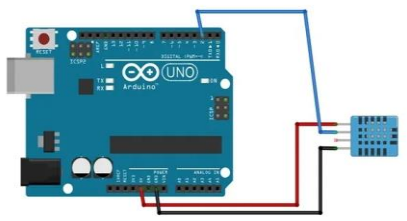
● LM35 Interfacing Circuit



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

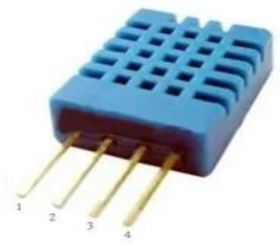
● It has temperature range from 0 – 50 C

● 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.

● 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.

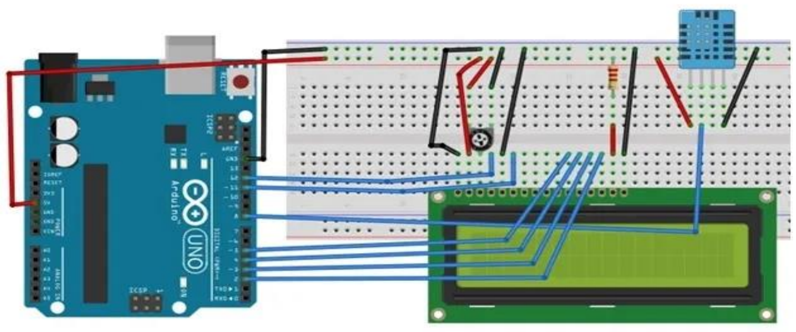
After copying the files, the Arduino library folder should have a new folder named DHT containing the dht.h and dht.cpp. After that copy the following code in the Arduino IDE and upload the code.

● Code of DHT11 interfacing with arduino

```
// Code for DHT11 Temperature and humidity sensor.
#include " DHT.h " // including the library of DHT11 temperature and
humidity sensor
#define DHTPIN 2 // Selecting the pin at which we have connected
DHT11
#define DHTTYPE DHT11 // Selecting the type of DHT sensors
DHT 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 it
is a very slow sensor.
float humidity = dht.readHumidity ( ) ; // Declaring h a variable and
storing the humidity in it.
float temp = dht.readTemperature ( ) ; // Declaring t a variable and
storing the temperature in it.
// Checking if the output is correct. If these are NaN, then there is
something in it.
if ( isnan ( t ) || isnan ( h ) ) {
Serial.println ( " Sensor not working " ) ;
}
else
{
Serial.print ( " Temp is " ) ;
Serial.print ( temp ) ; // Printing the temperature on
display.
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.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.
}
```

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	$T_{F^0} = \frac{9}{5}(T(K) - 273.15) + 32$

Observation:

The DHT11 sensor is a digital sensor which measures the temperature and humidity. We can read the temperature and humidity using arduino and show the temperature in Celsius or Fahrenheit.

You can run the program and observe the output values in Fahrenheit on the Serial Monitor.

Code:

```
#include <dht.h>

dht DHT;

#define DHT11_PIN A1

float min_t,max_t;

void setup()

{
 Serial.begin(9600);
 Serial.println("Humidity (%),\tTemperature (F)");  min_t = 0xffff;
 max_t=0x00;
 }
void loop()
 {
 // READ DATA
 int chk = DHT.read11(DHT11_PIN);
 // DISPLAY DATA
 Serial.print(DHT.humidity, 1);
 Serial.print("\t");
 Serial.println((DHT.temperature*1.8)+32, 1);


 delay(1000);
 }
```

Conclusion: Executed a program to read the temperature sensor and send the values to the serial monitor on the computer

	Dr. D. Y. Patil Educational Federation's **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ________________	**Marks** _______ **/ 10**
Name : ________________		**Remarks & Sign**
Subject: ________ **Class :** ________ **Roll No :** ________		
Expt. No. ______ **Date :** ________ **Batch :** ________		
Title : ________________		

Assignment: 12

Title: Write a program to show the temperature and shows a graph of the recent measurements

Objective: Understanding working principle of DHT11 temperature sensor, Blynk IOT Platform

Hardware: Arduino(Node MCU), LM35, DHT11, etc

Software: Arduino IDE

Apparatus: Arduino Uno board, Micro-IoT sensor actuator board, Power adaptor.

Interface:

Serial Plotter

Peripheral	**Arduino Pin**
DHT11	A1

Procedure:

Step 1: Connect the Arduino board to the Micro-IoT Sensor board using the FRC cable provided with the board.

Step 2: Connect the Power supply adaptor and power on the circuit. **Step 3:** Open Arduino IDE and create a new sketch (program) using the above pins. **Step 4:** In the Arduino IDE go to tools□Port and select the appropriate COM port.

Step 5: In the Arduino IDE click on the upload button ( ) to compile and download the code into the Arduino UNO. When successfully downloaded the code will start running.

Step 6: Open the Serial Monitor in Arduino IDE Tools□Serial Plotter and observe the graph of values from the temperature sensor.

Theory:

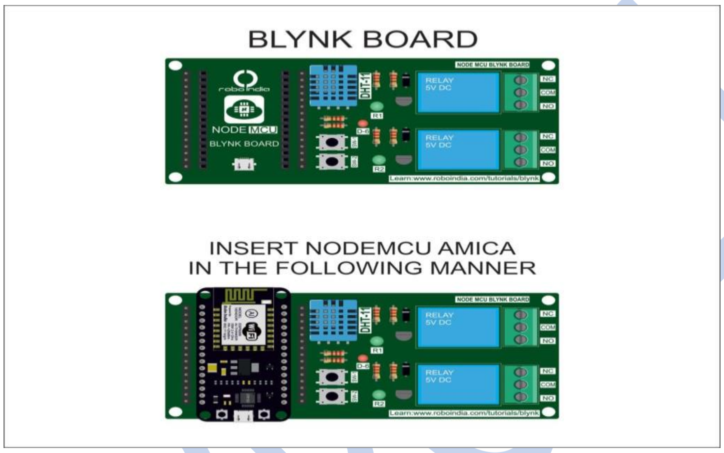
1. Introduction:

In this project using an esp8266, to show the temperature and humidity DHT11 sensor on your

Smartphone or tablet. The NodeMCU collects the temperature and humidity from DHT11 sensor and sends it to Blynk app every second.

1.2 Hardware required

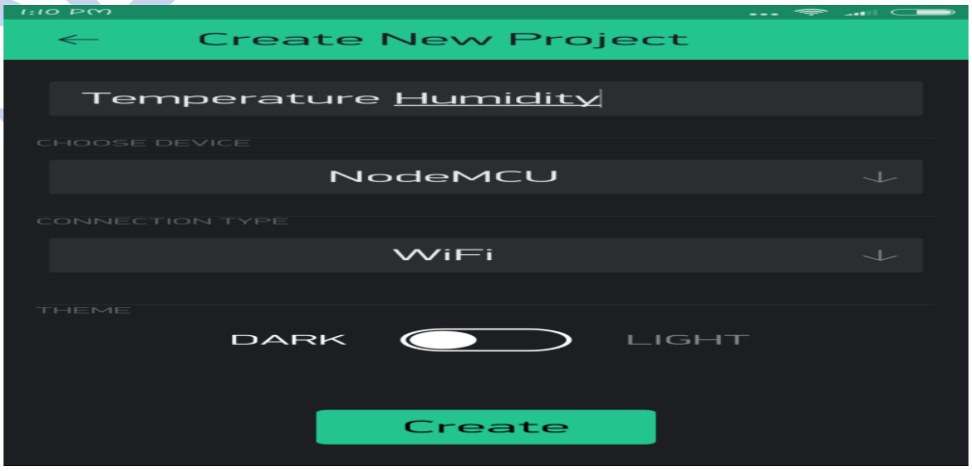
Blynk Board and NodeMCU is used in this example. Inset NodeMCU to the Blynk board as shown in the image ahead then connect NodeMCU to PC or Laptop through USB cable.



2. On Blynk App

You need to perform following steps on Blynk App.

2.1 Create a New Project in BLYNK app.Write Project name Temperature Humidity and Select NodeMCU from drop down.


1:10 PM

Create New Project

Temperature Humidity

CHOOSE DEVICE

NodeMCU

CONNECTION TYPE

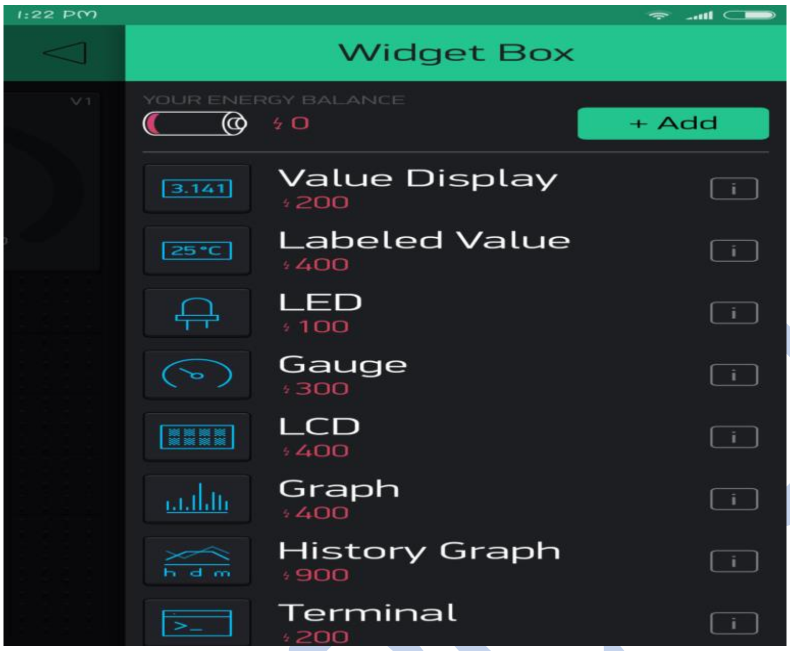
WiFi

THEME

DARK LIGHT

Create

2.2 An AUTH token will be sent to your registered email, note this down. Tap on the screen and add a 2 Gauges.

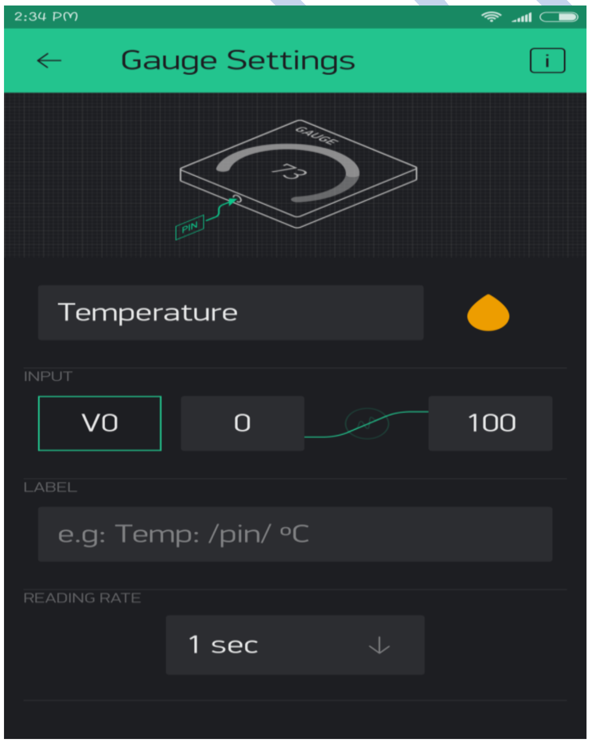

1:22 PM

Widget Box

YOUR ENERGY BALANCE ⚡0 + Add

Value Display	⚡200
Labeled Value	⚡400
LED	⚡100
Gauge	⚡300
LCD	⚡400
Graph	⚡400
History Graph	⚡900
Terminal	⚡200

2.3 Tap on the Widget and select the respective Virtual pins for temperature and humidity data (V0 for temperature and V1 for humidity).


2:34 PM

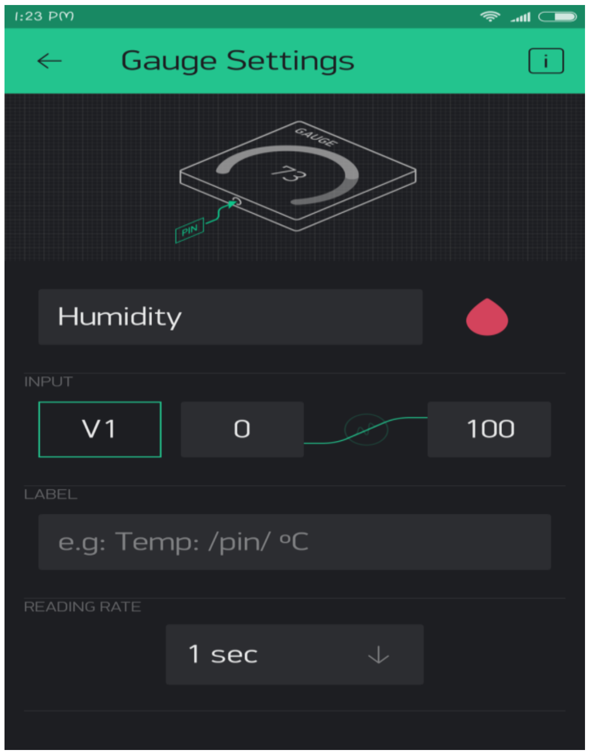
Gauge Settings

Temperature

INPUT: V0 0 100

LABEL: e.g: Temp: /pin/ ºC

READING RATE: 1 sec


1:23 PM

Gauge Settings

Humidity

INPUT: V1 0 100

LABEL: e.g: Temp: /pin/ ºC

READING RATE: 1 sec

Note: Make sure to setup Reading rate as ‘1’ second for all Widgets. And add gauges for both Humidity and Temperature.

3. Code the NodeMCU with the following code.

Before uploading, make sure to paste your authorization token into the auth [] variable. Also make sure to load your Wifi network settings into the Blynk.begin(auth, “ssid”, “pass”) function.

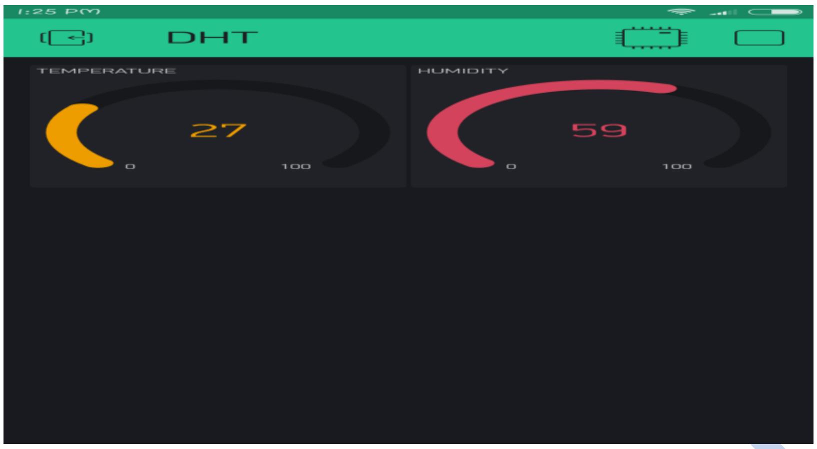
Following code may be downloaded from here.

```
// Robo India Tutorial
// Digital Output on LED
// Hardware: NodeMCU Blynk Board
#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
#include "DHT.h" // including the library of DHT11 temperature
and humidity sensor
#include <SimpleTimer.h> //including the library of SimpleTimer
#define DHTTYPE DHT11 // DHT 11
#define dht_dpin 14
DHT dht(dht_dpin, DHTTYPE);
SimpleTimer timer;
char auth[] = "Your Auth. Key"; // You should get Auth Token in
the Blynk App.
// Go to the Project Settings
(nut icon).
char ssid[] = "Your Wifi Network name"; // Your WiFi credentials.
char pass[] = "Password of your network"; // Set password to "" for open
networks.
```

```
float t; // Declare the variables
float h;
void setup()
{
Serial.begin(9600);// Debug console
Blynk.begin(auth, ssid, pass);
dht.begin();
timer.setInterval(2000, sendUptime);
}
void sendUptime()
{
float h = dht.readHumidity();
float t = dht.readTemperature();
Serial.println("Humidity and temperature\n\n");
Serial.print("Current humidity = ");
Serial.print(h);
Serial.print("% ");
Serial.print("temperature = ");
Serial.print(t);
Blynk.virtualWrite(V0, t);
Blynk.virtualWrite(V1, h);
}
void loop()
{
Blynk.run();
timer.run();
}
```

4. Output

After Uploading the Ardunio code IDE. Press the play button on blynk app to show the output.



Observation:

When you output the values using ***serial.println()*** function and use the serial plotter the plotter will plot the graph of the values.

You can run the program and observe the graph using the Serial Plotter.

Code:

```
#include <dht.h>

dht DHT;

#define DHT11_PIN A1

float min_t,max_t;

void setup()

{

 Serial.begin(9600);

 Serial.println("Humidity (%),\tTemperature (F)");  min_t = 0xffff;

 max_t=0x00;

}

void loop()

{

 // READ DATA

 int chk = DHT.read11(DHT11_PIN);

 // DISPLAY DATA
Serial.print(DHT.humidity, 1);
Serial.print("\t");
Serial.println((DHT.temperature*1.8)+32, 1);


delay(1000);
}
```

Conclusion:
Executed a program to show the temperature and shows a graph of the recent measurements

	Dr. D. Y. Patil Educational Federation's **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ______________	**Marks** ______ **/ 10**
Name : ______________ **Subject:**__________ **Class :**__________ **Roll No :**__________ **Expt. No.**________ **Date :**__________ **Batch :**__________ **Title :** ______________		**Remarks & Sign**

Assignment: 13
Title: Write a program using piezo element and use it to play a tune after someone knocks
Objective: Understanding working principle of Buttons & Buzzers
Hardware: Arduino, Button, Buzzer, etc
Software: Arduino IDE

Apparatus: Arduino Uno board, Micro-IoT sensor actuator board, Power adaptor.

Interface:

Peripheral	Arduino Pin
Piezo Element	A1

Procedure:

Step 1: Connect the Arduino board to the Micro-IoT Sensor board using the FRC cable provided with the board.

Step 2: Connect the Power supply adaptor and power on the circuit. **Step 3:** Open Arduino IDE and create a new sketch (program) using the above pins. **Step 4:** In the Arduino IDE go to tools□Port and select the appropriate COM port.

Step 5: In the Arduino IDE click on the upload button ( ) to compile and download the code into the Arduino UNO. When successfully downloaded the code will start running.

Step 6: Tap on the piezo element and hear the tune.

Theory:

In this Practical, we will make a circuit which will play a sound using a buzzer and an LED which sound and as example of a complex project, it will light up LED when a button is pressed. For this project, we need :

● Arduino UNO or similar board

● Piezo Buzzer – One

● LED – One

● 220 Ohm Resistor – One

● 10K Ohm Resistor – One

● Pushbutton – One

● Breadboard – One

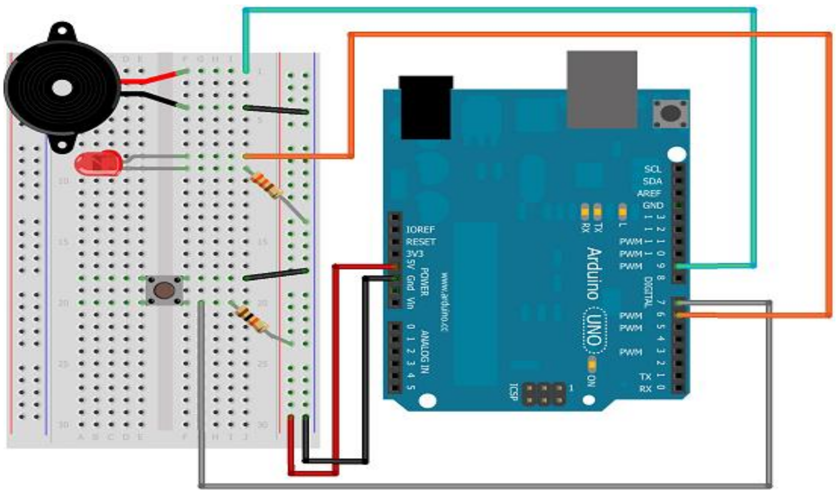
● Jumper Wires – Few

Push buttons are the basic on-off switching buttons most commonly used in less to highly complex electronic devices. Push buttons are able to connect two points whenever they are pressed. Push-button acts as a manually operated control device. When the button is pressed that means the short circuit is achieved between two points. On the other side when the button is not pressed then the open circuit or no connection developed between the points.

In this article, I will explain to you all the basics about the push button. A push-button is a very simple control device but we can operate it with pull-up and pull-down resistors. I will explain to you how an LED can be operated using a push button with a pull-up and a pull-down resistor. I will also explain to you how a buzzer can be operated with the help of a push-button. All these examples will help you to understand all the basics about the push button.

The push-button has four legs but we can use only two of them because of their internal structure. We can connect the push button directly with the VCC or 5-volt power supply. After that when we press it then the software will read it as HIGH until we change the state and perform the operation. But when the button is released after that system will get confused and does not able to recognize what is connected in the input. As result, the software will read HIGH or LOW value non-uniformly and assume some noise in this condition, due to which unexpected output will be observed. To avoid this situation it is necessary to connect pull-up or pull-down resistors with the push button.

The circuit itself quite simple circuit to set up. Be careful with the positive and negative poles of piezo buzzer:



Observation:

You can use the Piezo element for both input sensing and output tone generation**.**

In this experiment the user will tap the sensor and read the input values and once a certain threshold is passed can start generating a tone from the Piezo Element.

Code:

```
// these constants won't change:

const int buzzer = A1; // LED connected to digital pin 13 const int knockSensor = A1; // the piezo is connected
to analog pin 0 const int threshold = 400; // threshold value to decide when the detected  sound is a knock or
not


// these variables will change:

int sensorReading = 0; // variable to store the value read from the  sensor pin

void setup() {

 pinMode(buzzer,INPUT);

 }

void loop() {

 // read the sensor and store it in the variable sensorReading:  pinMode(buzzer,INPUT);

 sensorReading = analogRead(knockSensor);
 //analogWrite(buzzer,255);


 // if the sensor reading is greater than the threshold:  if (sensorReading >=
threshold) {
 pinMode(buzzer,OUTPUT);
 tone(buzzer,261);
 // Waits some time to turn off
 delay(200);
 //Turns the buzzer off
 noTone(buzzer);
 // Sounds the buzzer at the frequency relative to the note D in Hz   tone(buzzer,293);
 delay(200);
 noTone(buzzer);
 // Sounds the buzzer at the frequency relative to the note E in Hz  tone(buzzer,329);
 delay(200);
 noTone(buzzer);
 // Sounds the buzzer at the frequency relative to the note F in Hz  tone(buzzer,349);
 delay(200);
 noTone(buzzer);
 // Sounds the buzzer at the frequency relative to the note G in Hz  tone(buzzer,392);
 delay(200);
 noTone(buzzer);

 }
 delay(100);
 }
```

Conclusion: Executed a program using piezo element and use it to play a tune after someone knocks

Dr. D.Y. PATIL EDUCATIONAL FEDERATION	*Dr. D. Y. Patil Educational Federation's* **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ______	**Marks** ______ **/ 10**
	Name : ______ **Subject:** ______ **Class :** ______ **Roll No :** ______ **Expt. No.** ______ **Date :** ______ **Batch :** ______ **Title :** ______	**Remarks & Sign**

Assignment: 13
Title: Write a program using piezo element and use it to play a tune after someone knocks
Objective: Understanding working principle of Buttons & Buzzers
Hardware: Arduino, Button, Buzzer, etc
Software: Arduino IDE

Apparatus: Arduino Uno board, Micro-IoT sensor actuator board, Power adaptor.

Interface:

Peripheral	**Arduino Pin**
Piezo Element	A1

Procedure:

Step 1: Connect the Arduino board to the Micro-IoT Sensor board using the FRC cable provided with the board.

Step 2: Connect the Power supply adaptor and power on the circuit. **Step 3:** Open Arduino IDE and create a new sketch (program) using the above pins. **Step 4:** In the Arduino IDE go to tools□Port and select the appropriate COM port.

Step 5: In the Arduino IDE click on the upload button ( ) to compile and download the code into the Arduino UNO. When successfully downloaded the code will start running.

Step 6: Tap on the piezo element and hear the tune.

Theory:

In this Practical, we will make a circuit which will play a sound using a buzzer and an LED which sound and as example of a complex project, it will light up LED when a button is pressed. For this project, we need :

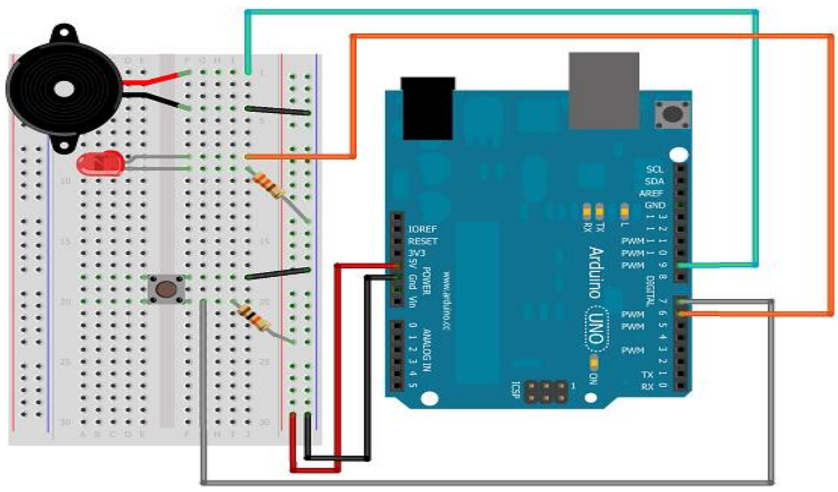
- Arduino UNO or similar board
- Piezo Buzzer – One
- LED – One
- 220 Ohm Resistor – One
- 10K Ohm Resistor – One
- Pushbutton – One
- Breadboard – One
- Jumper Wires – Few

Push buttons are the basic on-off switching buttons most commonly used in less to highly complex electronic devices. Push buttons are able to connect two points whenever they are pressed. Push-button acts as a manually operated control device. When the button is pressed that means the short circuit is achieved between two points. On the other side when the button is not pressed then the open circuit or no connection developed between the points.

In this article, I will explain to you all the basics about the push button. A push-button is a very simple control device but we can operate it with pull-up and pull-down resistors. I will explain to you how an LED can be operated using a push button with a pull-up and a pull-down resistor. I will also explain to you how a buzzer can be operated with the help of a push-button. All these examples will help you to understand all the basics about the push button.

The push-button has four legs but we can use only two of them because of their internal structure. We can connect the push button directly with the VCC or 5-volt power supply. After that when we press it then the software will read it as HIGH until we change the state and perform the operation. But when the button is released after that system will get confused and does not able to recognize what is connected in the input. As result, the software will read HIGH or LOW value non-uniformly and assume some noise in this condition, due to which unexpected output will be observed. To avoid this situation it is necessary to connect pull-up or pull-down resistors with the push button.

The circuit itself quite simple circuit to set up. Be careful with the positive and negative poles of piezo buzzer:



SCL
SDA
AREF
GND
IOREF
RESET
3V3
5V
Gnd
Vin
POWER
ANALOG IN
www.arduino.cc
Arduino
UNO
DIGITAL
PWM
TX
RX
ICSP
ON

Dr. D.Y. PATIL EDUCATIONAL FEDERATION	*Dr. D. Y. Patil Educational Federation's* **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ______________________________	**Marks** ______ **/ 10**
	Name :____ **Subject:**________________ **Class :**____________ **Roll No :**__________ **Expt. No.**________ **Date :**______________ **Batch :**________________ **Title :** ____	**Remarks & Sign**

Assignment: 15
Title: Study of ThingSpeak – an API and Web Service for the Internet of Things
Objective: Understanding Procedure to establish project with Thingspeak
Hardware: Arduino(NodeMCU), DHT11, LM35,, etc
Software: Arduino IDE

Theory:

In this Practical we are going to learn how to send LM35 sensor's temperature data to Thingspeak using NodeMCU. We will explore each method in detail and you may utilize any one of the method mentioned here.

There is a good chance that you are a beginner in IoT project and you have chosen LM35 and Arduino development environment, if so you landed on the right place. We will walk you through all the little things that you need to take care while constructing one of your first IoT project and we will also guide you where you may go wrong, in the end of this article you will be able to upload temperature data to thingspeak successfully.

We will see:

● Setting up Thingspeak account for receiving LM35 data.

● Installing ESP8266 package to Arduino IDE.

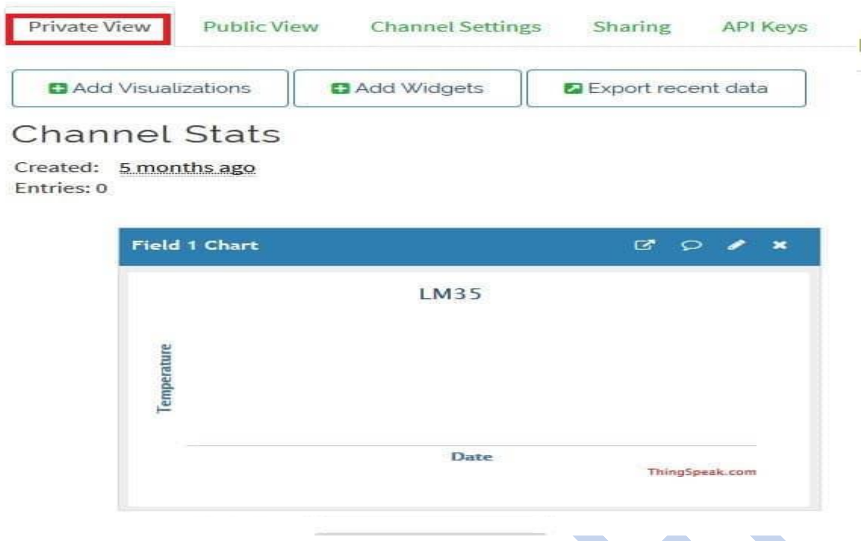
● Sending LM35 data to thingspeak using NodeMCU.

Setting up Thingspeak account for LM35 sensor:

Before we proceed towards construction details we need to setup our thingspeak account correctly to receive LM35 temperature data, this procedure need to be done for all three methods mentioned here.

You can sign up for thingspeak account here, if you haven't signed up yet.

● You need to create a new channel by clicking "New Channel" button and in the channel you should do the changes as shown:



LM35 Data to Thingspeak

1) Go to Channel settings tab and edit the name.

2) Enable Field 1 by checking the box and write the label as "Temperature".

3) Scroll down and click save.

● Now click on API keys tab:

API keys are the access keys to your channel using which you can write and read values. In this project we are using only write API key which is already generated for your channel. You need to take note of it and this write API key need to be inserted to the given program codes.

● Now by clicking "Private View" tab you will see a blank channel. You will see some data once we send to it.

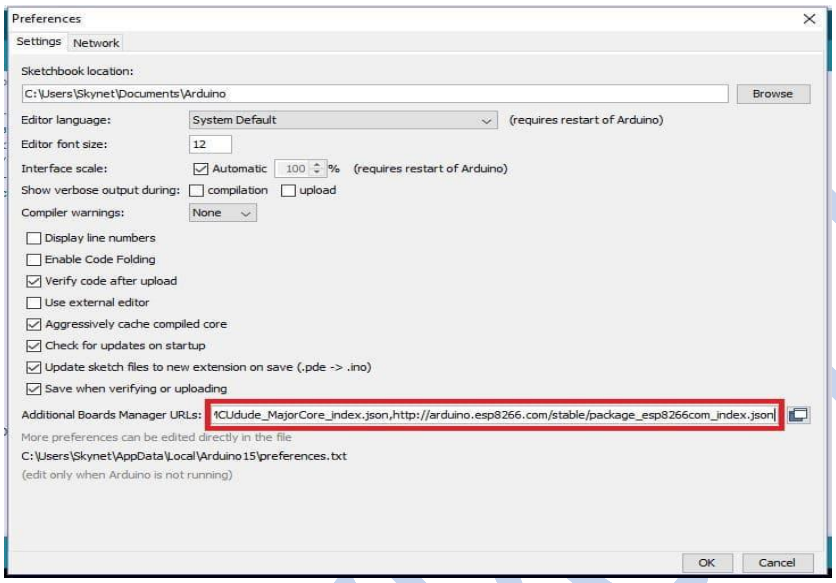
LM35 Data to Thingspeak

Installing ESP8266 package to Arduino IDE:

This process needs to be done if you are using NodeMCU or generic ESP8266 to connect to thingspeak. If you are going to use GSM modem to send LM35 data, this step is irrelevant for you, but for NodeMCU and ESP8266 boards this step is mandatory.

● In this step you need internet; we are going to install core files to Arduino IDE for IoT based boards.

- Copy this link: http://arduino.esp8266.com/stable/package_esp8266com_index.json
- Now open Arduino IDE and click on “File” >> “Preferences”.
- A window will open like this:



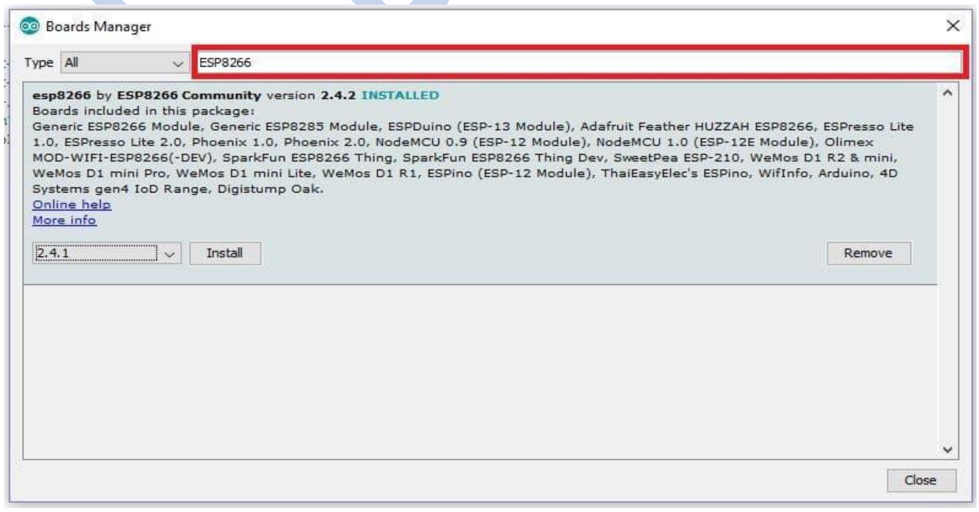
Preferences

Settings Network

Sketchbook location:
C:\Users\Skynet\Documents\Arduino
Editor language: System Default (requires restart of Arduino)
Editor font size: 12
Interface scale: ☑ Automatic 100 % (requires restart of Arduino)
Show verbose output during: ☐ compilation ☐ upload
Compiler warnings: None
☐ Display line numbers
☐ Enable Code Folding
☑ Verify code after upload
☐ Use external editor
☑ Aggressively cache compiled core
☑ Check for updates on startup
☑ Update sketch files to new extension on save (.pde -> .ino)
☑ Save when verifying or uploading
Additional Boards Manager URLs: ACUdude_MajorCore_index.json,http://arduino.esp8266.com/stable/package_esp8266com_index.json
More preferences can be edited directly in the file
C:\Users\Skynet\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

Preferences

- Paste the URL on the box and click “OK”.
- Now go to tools > Board > Boards Manager.
- Now a window will popup:



Boards Manager

Type All ESP8266

esp8266 by **ESP8266 Community** version **2.4.2** **INSTALLED**
Boards included in this package:
Generic ESP8266 Module, Generic ESP8285 Module, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SweetPea ESP-210, WeMos D1 R2 & mini, WeMos D1 mini Pro, WeMos D1 mini Lite, WeMos D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WifInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak.
Online help
More info

2.4.1

boards manager

● Type ESP8266 on the box as shown and you will get an installation option, select the latest version and click install.

Downloading files may take several minutes depending on your internet connection.

Downloading Thingspeak library:

You need to download Thingspeak library for projects using NodeMCU and generic ESP8266, this library is irreverent if you are going to use GSM module in this project.

Download Thingspeak library here.

Installing CH340 driver:

This step is relevant only if you are going to use NodeMCU to send LM35 data to thingspeak. You may skip for other two mentioned methods.

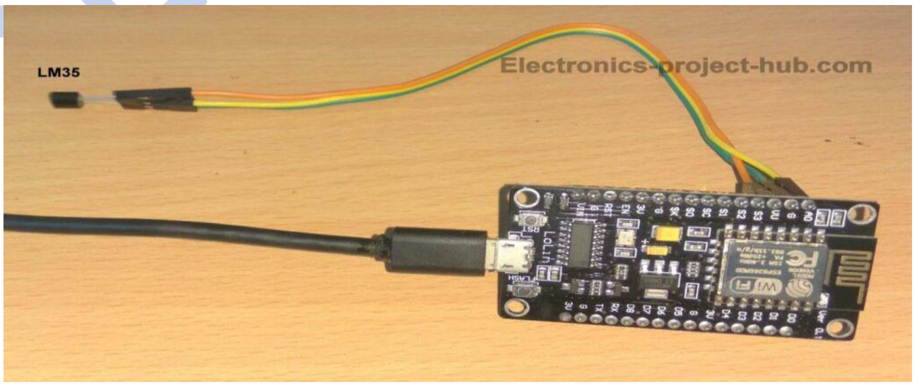
Installing CH340 driver is mandatory not only for NodeMCU but also for all Arduino clones, Uno R3, nano etc. without it you cannot upload any program.

We have explained the steps how to install CH340 driver here.

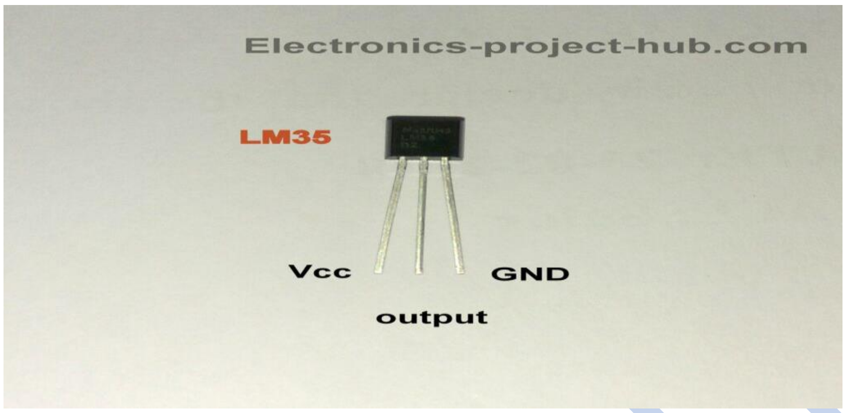
Once you completed the mentioned relevant steps, we are good to go with construction details.

Send LM35 Data to Thingspeak using NodeMCU

Circuit Diagram:



LM35 with NodeMCU



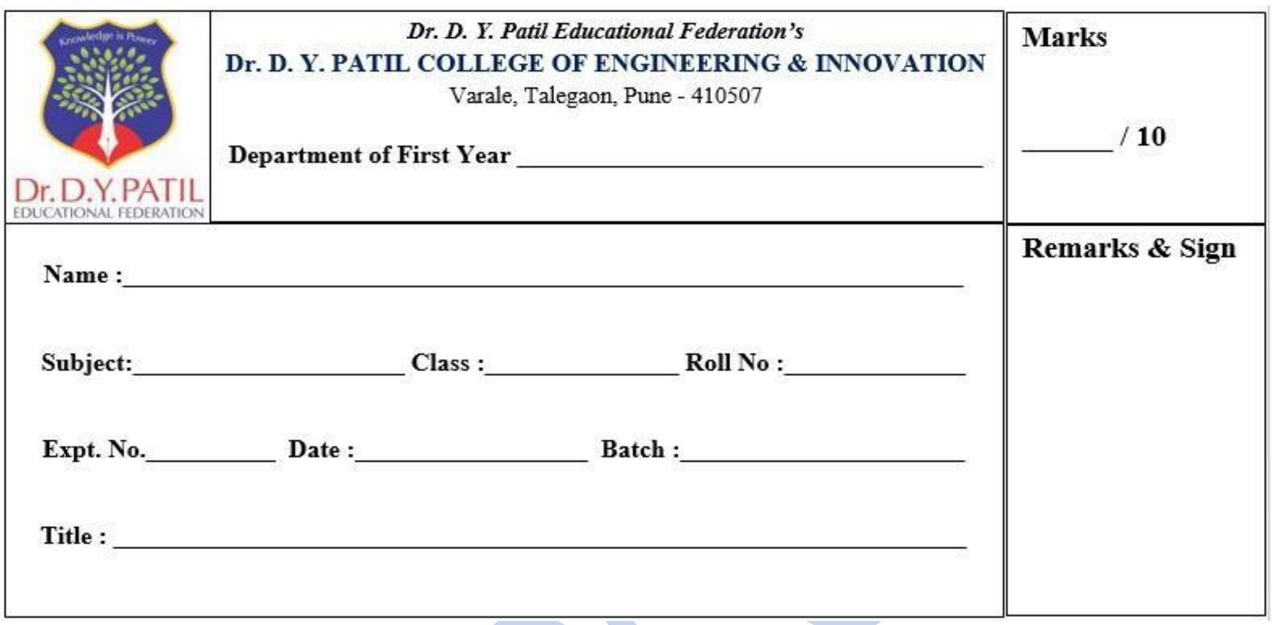
LM35 Pin Diagram

The circuit is so simple that we didn't consider drawing one. Connections are mentioned below:

- Vcc of LM35 connects to 'VU' pin of NodeMCU. VU pin provides 5V output.
- The output terminal of LM35 connects to 'A0' of NodeMCU.
- GND of LM35 connects to GND of NodeMCU.

Conclusion: Studied of ThingSpeak – an API and Web Service for the Internet of Things

Note: Choose any one practical from Group C



Dr. D. Y. Patil Educational Federation's

Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION

Varale, Talegaon, Pune - 410507

Department of First Year ______________________

Marks

______ / 10

Remarks & Sign

Name : ______________________

Subject:______________ **Class :**__________ **Roll No :**__________

Expt. No.________ **Date :**____________ **Batch :**______________

Title : ______________________

Experiment No 16: Write an application to control the operation of hardware simulated traffic signals.

Aim: To implement the simulation of a Traffic Signal using Raspberry Pi hardware. **Apparatus:** Raspberry Pi, Micro-IoT Traffic Simulation Board, Power adaptor. **Interface:**

Device Details	Pins Rpi (Arduino)
North Green	GPIO6 (A2)
East Green	GPIO7 (0)
South Green	GPIO8 (10)
West Green	GPIO9 (12)
North Amber	GPIO10 (11)
East Amber	GPIO12 (5)
South Amber	GPIO11 (13)
West Amber	GPIO16 (9)
North Red	GPIO17 (A1)
East Red	GPIO18 (6)
South Red	GPIO5 (A3)
West Red	GPIO4 (A0)

Procedure:

Step 1: Connect the RaspberryPi board to the Micro-IoT Sensor board using the

FRC cable provided with the board.

Step 2: Connect the Power supply adaptor and power on the circuit. **Step 3:**

Open a terminal and create a new python program using the above pins. **Step**

4: Run the program python program .

Step 5: check the traffic simulation.

Observation:

The program simulates the working of a traffic signal.

Code:

```
import time

import RPi.GPIO as GPIO

TRUE = 1

#north led definition NRED
= 17

NAMBER = 10

NGREEN = 6

#east led definition

SRED = 5

SAMBER = 11

SGREEN = 8

#south led definition ERED
= 18

EAMBER = 12

EGREEN = 7

#west led definition

WRED = 4

WAMBER = 16

WGREEN = 9

GPIO.setmode(GPIO.BCM)
GPIO.setup(NRED,GPIO.OUT)
GPIO.setup(NAMBER,GPIO.OUT)
GPIO.setup(NGREEN,GPIO.OUT)
GPIO.setup(SRED,GPIO.OUT)
GPIO.setup(SAMBER,GPIO.OUT)
GPIO.setup(SGREEN,GPIO.OUT)
GPIO.setup(WRED,GPIO.OUT)
GPIO.setup(WAMBER,GPIO.OUT)
GPIO.setup(WGREEN,GPIO.OUT)
GPIO.setup(ERED,GPIO.OUT)
GPIO.setup(EAMBER,GPIO.OUT)
GPIO.setup(EGREEN,GPIO.OUT)



def NState(r,a,g):

        GPIO.output(NRED,r)

        GPIO.output(NAMBER,a)
        GPIO.output(NGREEN,g)

def SState(r,a,g):

        GPIO.output(SRED,r)

        GPIO.output(SAMBER,a)
        GPIO.output(SGREEN,g)

def EState(r,a,g):

        GPIO.output(ERED,r)

        GPIO.output(EAMBER,a)
        GPIO.output(EGREEN,g)
        return

def WState(r,a,g):

        GPIO.output(WRED,r)

        GPIO.output(WAMBER,a)

        GPIO.output(WGREEN,g)

        return

try:

        while TRUE:

                NState(1,1,1)

                SState(1,1,1)

                EState(1,1,1)

                WState(1,1,1)

        #S on

                NState(1,0,0)

                SState(0,0,1)

                EState(1,0,0)

                WState(1,0,0)

                time.sleep(4)

                SState(0,1,0)

                time.sleep(0.5)

        #E on
```

```
            NState(1,0,0)
            SState(1,0,0)
            EState(0,0,1)
            WState(1,0,0)
            time.sleep(4)
            EState(0,1,0)
            time.sleep(0.5)
            #W on
            NState(1,0,0)
            SState(1,0,0)
            EState(1,0,0)
            WState(0,0,1)
            time.sleep(4)
            WState(0,1,0)
            time.sleep(0.5)
            #N on
            NState(0,0,1)
            SState(1,0,0)
            EState(1,0,0)
            WState(1,0,0)
            time.sleep(4)
            NState(0,1,0)
            time.sleep(0.5)
# If CTRL+C is pressed the main loop is broken except
KeyboardInterrupt:
 RUNNING = False
 print "\Quitting"
# Actions under 'finally' will always be called finally:
 # Stop and finish cleanly so the pins  # are available to
be used again  GPIO.cleanup()
```

	Dr. D. Y. Patil Educational Federation's **Dr. D. Y. PATIL COLLEGE OF ENGINEERING & INNOVATION** Varale, Talegaon, Pune - 410507 **Department of First Year** ____________________	**Marks** _______ **/ 10**
Name : ______________________________ **Subject:**____________ **Class :**__________ **Roll No :**__________ **Expt. No.**________ **Date :**__________ **Batch :**____________ **Title :** ______________________________		**Remarks & Sign**

Experiment 17: IoT based Web Controlled Home Automation using Raspberry Pi.

Aim: To control the Home Appliances using a webpage.

Apparatus: RaspberryPi, relays, power supply.

Procedure:

Step 1: Connect the RaspberryPi board to the Micro-IoT Sensor board.

Step 2: Understand the connections for the Micro-IoT Sensor board and RaspberryPi board.

Step 3: Connect the Raspberry Pi setup and write the program for interfacing relays. Step 4: Send the files to the **/var/www/html/HomeAutomation** directory.

Step 5: open the browser on any other PC and connect to the IP address of Raspberry Pi : **e.g. //http:192.168.1.100/ HomeAutomation.**

The webpage will take instructions from the user and pass it on to the PHP page which will in-turn run shell commands to switch the Home Appliances.

Interfacing Details:

Peripherals	**RaspberryPi Pin**
Buzzer (Alarm)	GPIO 6
Relay 1- Door Latch	GPIO 7
Relay 2- Light	GPIO 12
Relay 3- Fan	GPIO 16

Program for Reference: **index.php** (for stepper)

```
<!DOCTYPE html>
<?php
  $op = $_GET['op'];
```

```
shell_exec("/usr/local/bin/gpio -g mode 7 out"); #Latch
shell_exec("/usr/local/bin/gpio -g mode 12 out"); #Light
shell_exec("/usr/local/bin/gpio -g mode 16 out"); #FAN1
shell_exec("/usr/local/bin/gpio -g mode 6 out"); #Buzzer

switch($op)
{
case 1: shell_exec("/usr/local/bin/gpio -g write 7 0"); #Door Latch OFF  break;
case 2: shell_exec("/usr/local/bin/gpio -g write 7 1"); #Door Latch ON  break;
case 3: shell_exec("/usr/local/bin/gpio -g write 12 0"); #Light 1  break;
case 4: shell_exec("/usr/local/bin/gpio -g write 12 1");
 break;
case 5: shell_exec("/usr/local/bin/gpio -g write 16 0"); #FAN1  break;
case 6: shell_exec("/usr/local/bin/gpio -g write 16 1");
 break;
case 7: shell_exec("/usr/local/bin/gpio -g write 6 1"); #Buzzer  break;
case 8: shell_exec("/usr/local/bin/gpio -g write 6 0");
 break;
case 9: shell_exec("/usr/local/bin/gpio -g write 7 0");
 shell_exec("/usr/local/bin/gpio -g write 12 0");
 shell_exec("/usr/local/bin/gpio -g write 16 0");
        shell_exec("/usr/local/bin/gpio -g write 6 1");
        break;
}


 include("page.html");
?>
```

<u>Program for Reference</u>**<u>: page.html</u>** <u>(webpage for stepper)</u>

```
<html>
 <title>Home Automation Demo</title>
 <body bgcolor=White>
 <center>
 <h1 style="color:Blue">Internet of Things</h1>
 <h2 style="color:Black">Home Appliance Control</h2>
 <form action="index.php">
```

```
&nbsp<button type="submit" name="op" value="1">LATCH OFF</button> &nbsp  &nbsp <button type="submit" name="op" value="2">LATCH ON</button> <br><br> &nbsp<button type="submit" name="op" value="3">LIGHT OFF</button> &nbsp  &nbsp <button type="submit" name="op" value="4">LIGHT ON</button> <br><br> &nbsp<button type="submit" name="op" value="5">FAN OFF</button> &nbsp  &nbsp <button type="submit" name="op" value="6"> FAN ON</button> <br><br> &nbsp<button type="submit" name="op" value="7">BUZZER OFF</button> &nbsp &nbsp <button type="submit" name="op" value="8">BUZZER ON</button> <br><br> &nbsp<button type="submit" name="op" value="9">STOP</button> <br><br>  <!--input type="submit" value="Submit"-->

 </form>

 </center>

 </body>

</html>
```