Module 4 : Embedded Applications for IoT

Presenter : Assoc. Prof. Ts. Dr. Ahmad Shukri Bin Mohd Noor

MODULE OUTLINE

4.1

MODULE 4.1

Introduction to Embedded System

4.2

MODULE 4.2

Basic Applications of Sensors and Actuation Technology

4.3

MODULE 4.3

Fundamental of Embedded Programming for IoT

4.4

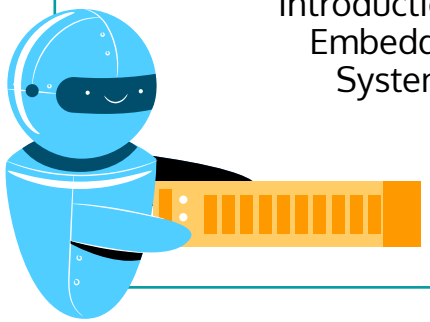
MODULE 4.4

Connecting IoT Devices to Network

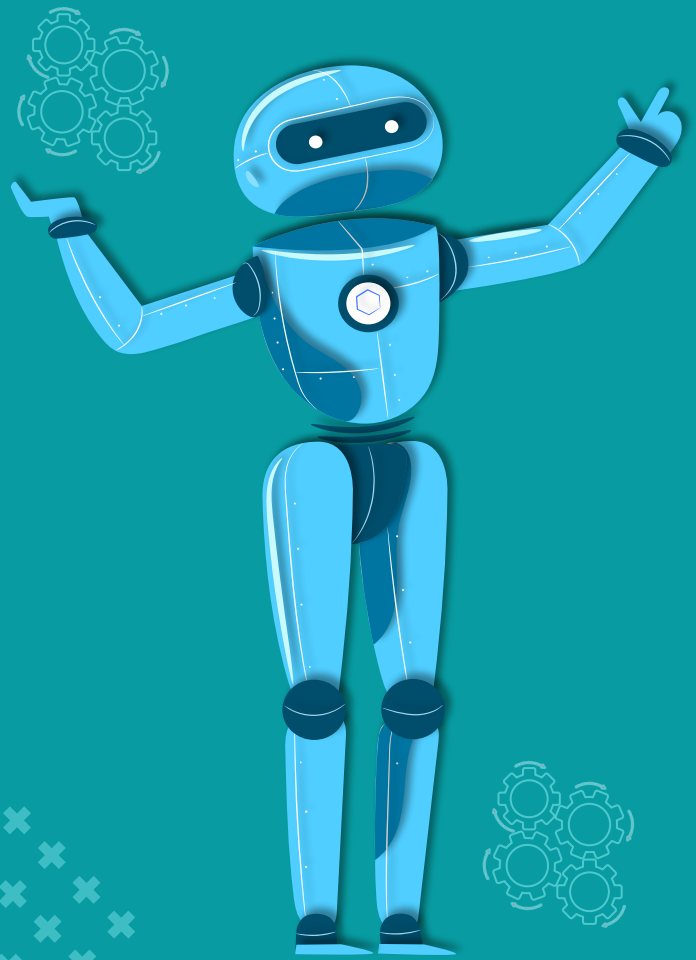
4.5

MODULE 4.5

Data Transport Protocol for IoT



4.1



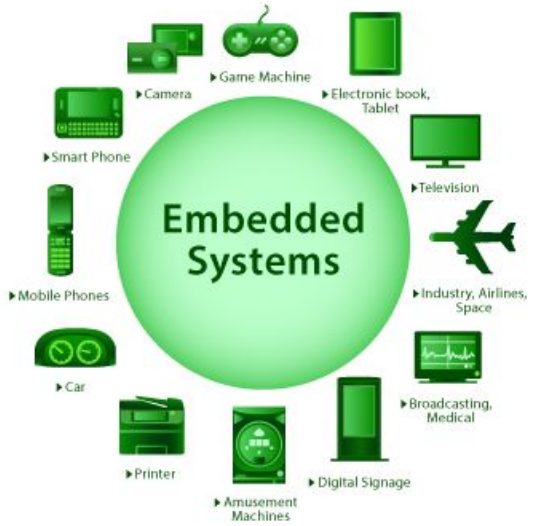
Introduction to Embedded System

- Embedded System
- Microcontroller
- Microprocessor
- Development board

4.1 Introduction to Embedded System

4.1.1 Embedded System

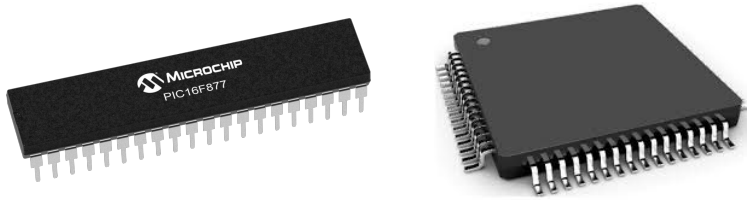
- A mixture of hardware and software for different functions.
- Can be a controller of large design.
- These include planes, home appliances, medical devices, cameras, televisions, etc.
- Modern embedded system mostly relies on microcontroller, microprocessor, or chip system (SoC).

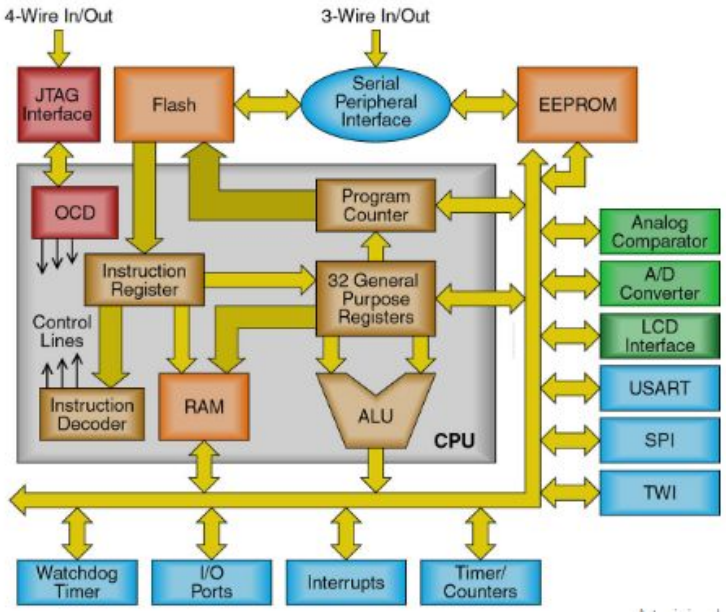


4.1 Introduction to Embedded System

4.1.2 Microcontroller

- It's like a small computer on a single IC.
- It contains a processor core, ROM, RAM and I/O pins dedicated to perform various tasks.
- Microcontrollers are generally used in projects and applications that require direct control of user.
- As it has all the components needed in its single chip, it does not need any external circuits to do its task so microcontrollers are heavily used in embedded systems and major microcontroller manufacturing companies are making them to be used in embedded market.
- A microcontroller can be called the heart of embedded system.
- Some examples of popular microcontrollers are 8051, AVR, PIC series of microcontrollers.





4.1 Introduction to Embedded System

4.1.2 Microcontroller

- Applications using microcontroller are digital camera and washing machine.
- Used for specific task.
- Based on inputs, it does some processing and generates result as an output.
- The task performed is predefined

Internal Structure

- Amount of memory and I/O ports are limited
- Memory elements and I/O ports are integrated along with the CPU inside a single chip
- Size of an overall system is much smaller

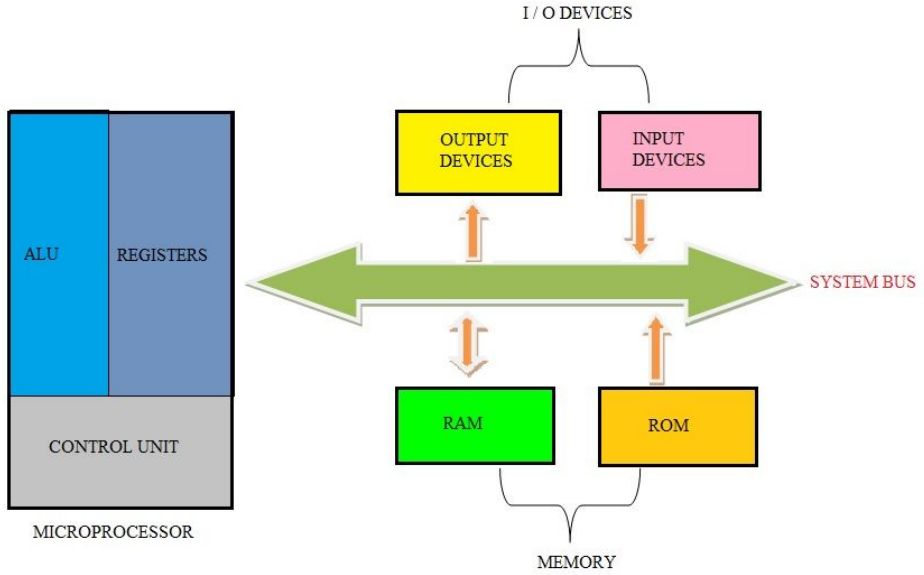
Memory & Speed

- CPU : single-/dual-core 32-bit
- ROM : 448 KB
- SRAM : 520 KB
- Clock speed : 160MHz or 240MHz

4.1 Introduction to Embedded System

4.1.3 Microprocessor

- Microprocessor has only a CPU inside them in one or few Integrated Circuits. unlike microcontrollers it does not have RAM, ROM and other peripherals.
- They are dependent on external circuits of peripherals to work.
- But microprocessors are not made for specific task but they are required where tasks are complex and tricky like development of software's, games and other applications that require high memory and where input and output are not defined.
- It may be called heart of a computer system. Some examples of microprocessor are Pentium, i3, and i5 etc.



4.1 Introduction to Embedded System

4.1.3 Microprocessor

- Classic application is computer or laptop
  - Gaming
  - Web browsing
  - Photo editing
  - Creating documents
  - Simulations
- Basically used in an application where the task are not predefined
- Used in application where intensive processing is required

Internal Structure

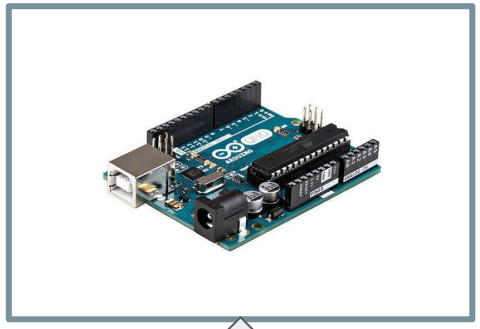
- Only content of CPU
- Memory elements and the I/O interfaces are connected to it externally

Memory & Speed

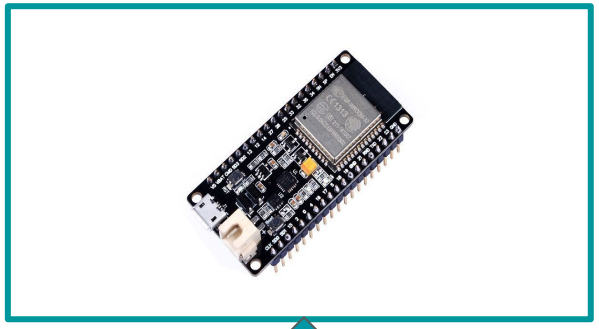
- CPU : octa-core 64-bit
- ROM : 15 TB
- RAM : 128 GB
- Clock speed : 2.9GHz – 4.8GHz

4.1 Introduction to Embedded System

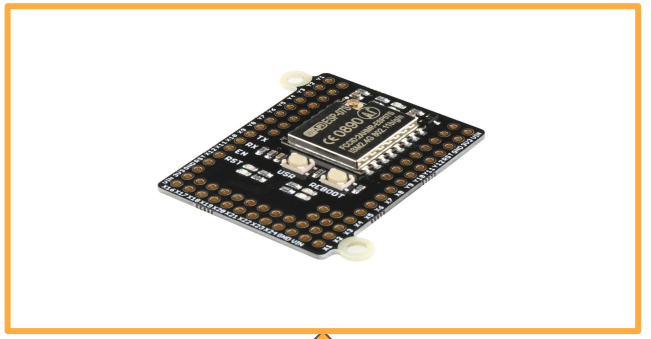
4.1.4 Development Board



Arduino
Language: C



ESP8266 & ESP32
Language: LUA, C
Supports micropython firmware



PyBoard and Pycom
Language: Micropython

4.1 Introduction to Embedded System

4.1.4 Development Board - ESP8266

- Create a web server
- Send http requests
- Control outputs
- Read inputs and interrupts
- Send emails
- Post tweets

4.1 Introduction to Embedded System

4.1.4 Development Board - ESP8266

- Processor: Tensilica Xtensa Dual-Core 32-bit LX6 microprocessor, running at 160 or 240 MHz
- ROM: 448 KB
- SRAM: 512 KB
- Low Power: ensures that you can still use ADC conversions, for example, during deep sleep.

Peripheral Input/Output:

- capacitive touch
- ADCs (Analog-to-Digital Converter)
- DACs (Digital-to-Analog Converter)
- I²C (Inter-Integrated Circuit)
- UART (Universal Asynchronous Receiver/Transmitter)
- CAN 2.0 (control area network)
- SPI (Serial Peripheral Interface)
- I²S (Integrated Interchip Sound)
- RMII (Reduced Media-Independent Interface)
- PWM (Pulse-Width Modulation).

Memory & Speed

- Security: hardware accelerators for AES and SSL/TLS
- Arduino IDE compatible:
  - you can program the ESP32 with the Arduino IDE
- Compatible with MicroPython:
  - you can program the ESP32 with MicroPython firmware

4.1.4 Development Board - ESP32

- If you're familiar with the ESP8266, the ESP32 is its successor.
- The ESP32 is loaded with lots of new features.
- The most relevant:
  - it combines WiFi and Bluetooth wireless capabilities and it's dual core.

4.1 Introduction to Embedded System

4.1.4 Development Board - ESP32

- Processor: Tensilica Xtensa Dual-Core 32-bit LX6 microprocessor, running at 160 or 240 MHz
- ROM: 448 KB
- SRAM: 512 KB
- Low Power: ensures that you can still use ADC conversions, for example, during deep sleep.

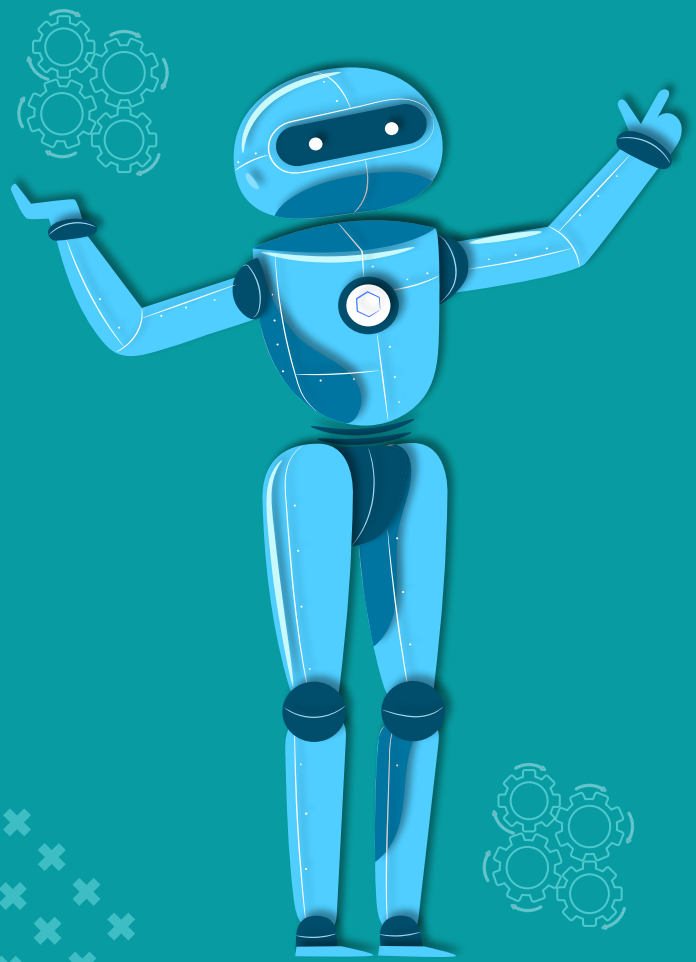
Peripheral Input/Output:

- capacitive touch
- ADCs (Analog-to-Digital Converter)
- DACs (Digital-to-Analog Converter)
- I²C (Inter-Integrated Circuit)
- UART (Universal Asynchronous Receiver/Transmitter)
- CAN 2.0 (control area network)
- SPI (Serial Peripheral Interface)
- I²S (Integrated Interchip Sound)
- RMII (Reduced Media-Independent Interface)
- PWM (Pulse-Width Modulation).

Memory & Speed

- Security: hardware accelerators for AES and SSL/TLS
- Arduino IDE compatible:
  - you can program the ESP32 with the Arduino IDE
- Compatible with MicroPython:
  - you can program the ESP32 with MicroPython firmware

4.2



Basic Applications of Sensors and Actuators Technology

- Digital and Analog Signal Overview
- Analog Signal
- Digital Signal
- Analog and Digital Combined
- Pulse-Width Modulation
- Sensors
- Actuators

4.2 Basic Application of Sensors and Actuators Technology

4.2.1 Digital and Analog Signal Overview

- We're living in an analog universe.
  - Infinity of colors.
  - We can hear endless tones.
  - Infinite odors we can detect.
- The common feature of these analog signals is their limitless possibilities.

4.2 Basic Application of Sensors and Actuators Technology

4.2.1 Digital and Analog Signal Overview

- Digital signals and objects deal with the realm of discrete or finite, meaning there is a limited set of values that they can be.
- That might mean only two potential total values, 255, 4,294,967,296, or something as long as it's not ∞(infinity).
- Operating with electronics involves working with analog and digital signals, inputs and outputs.
- Our electronics projects have to communicate in some way with the actual analog world, but most of our microprocessors, computers and logic units are solely digital components.

4.2 Basic Application of Sensors and Actuators Technology

4.2.1 Digital and Analog Signal Overview

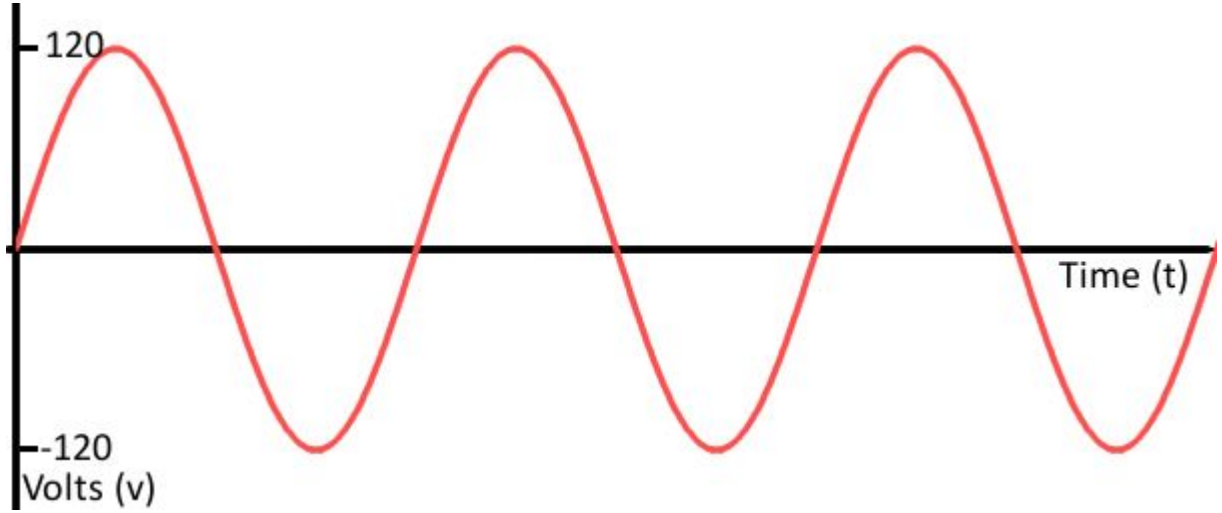
Signals

- What a signal is, actually electronic signals (as opposed to traffic signals, ultimate power trio records, or a general means of communication).
- The signals are time-varying "quantities" that convey some detail.
- For electrical engineering, the time-varying quantity is generally voltage (if not, then normally current).
- So when we talk about signals, think of them as a voltage that varies over time.
- Signals are exchanged between devices to send and receive information that may be video, audio, or other encoded data.
- Signals are typically transmitted via wires, but they can also move via air through radio frequency (RF) waves.
- For example, audio signals may be transmitted between your computer's audio card and speakers, while data signals can move through the air between a tablet and a WiFi router.

4.2 Basic Application of Sensors and Actuators Technology

4.2.2 Analog Signal

- Since a signal varies over time, plotting it on a graph is helpful.
- Looking at a signal graph is typically the best way to determine whether it is analog or digital
  - a time-versus-voltage graph should be smooth and continuous.

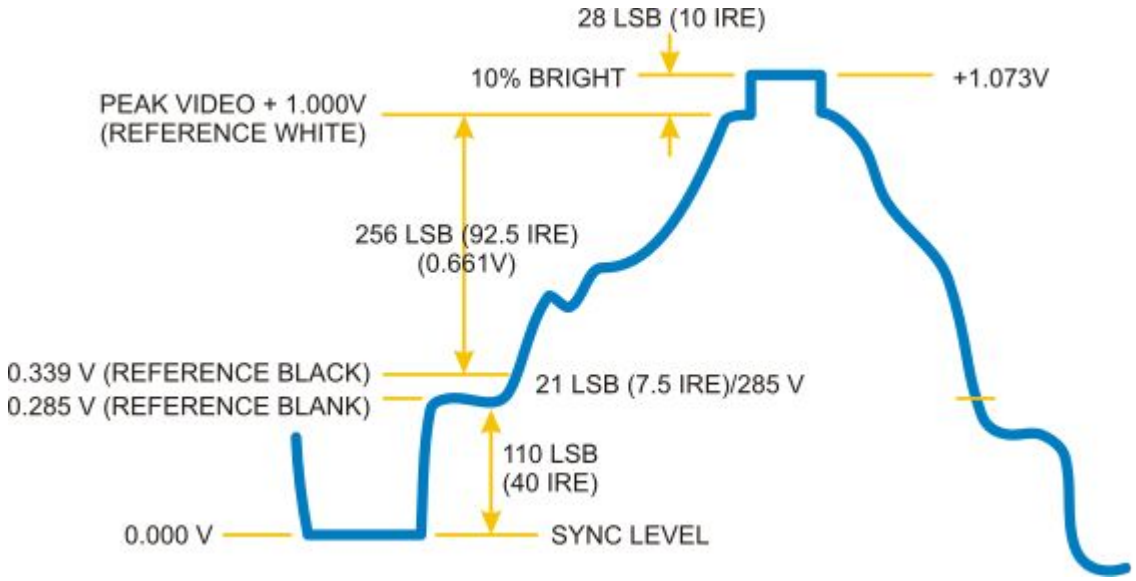


Although such signals can be limited to a range of maximum and minimum values, there are still countless potential values within that range. For example, the analog voltage from your wall socket can be clamped between -120V and + 120V, but as you increase the range more and more, you find an infinite number of values that the signal may actually be (such as 64.4V, 64.42V, 64.424V, and infinite, increasingly precise values).

4.2 Basic Application of Sensors and Actuators Technology

4.2.2 Analog Signal - Example

- Analog signals often transmit or record video and audio transmissions.
- For example, the composite video from an old RCA jack is a coded analog signal usually between 0 and 1.073V. Tiny signal changes have a major impact on video color or location.

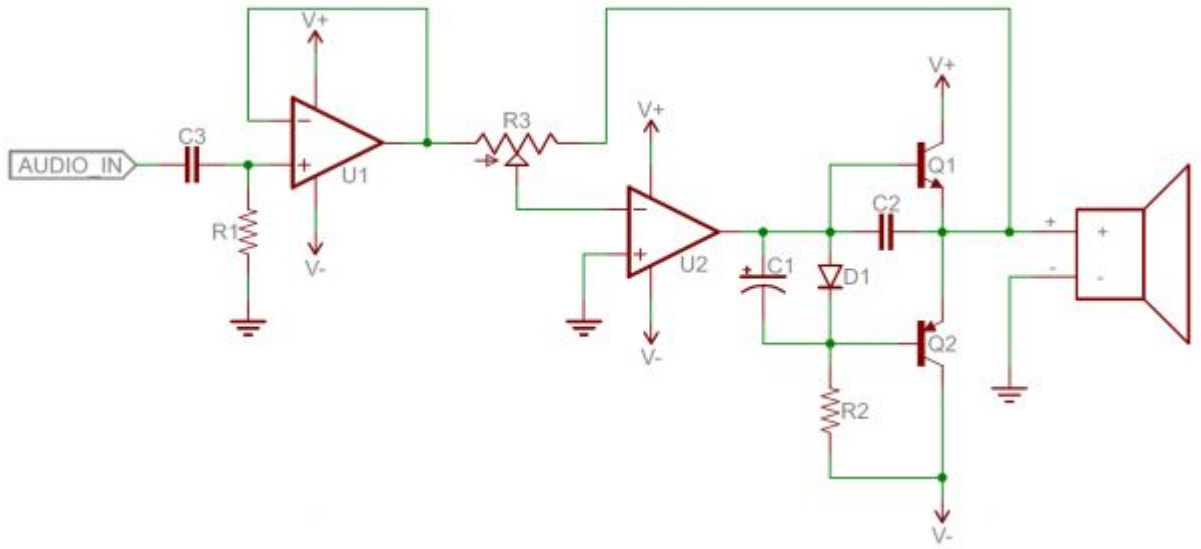


- Pure audio signals are also analog.
- A microphone's signal is full of analog frequencies and harmonics that combine to create beautiful music.

4.2 Basic Application of Sensors and Actuators Technology

4.2.2 Analog Signal - Electronics

- Most of the basic electronic components like resistors, capacitors, inductors, diodes, transistors, and amplifiers are all essentially analogous.
- Circuits constructed with a combination of those components alone are usually analog.
- Analog circuits with many components can be very complex designs, or they can be very plain, like two resistors combining to create a voltage divider.
- However, analog circuits are generally much harder to design than those which do the same job digitally.

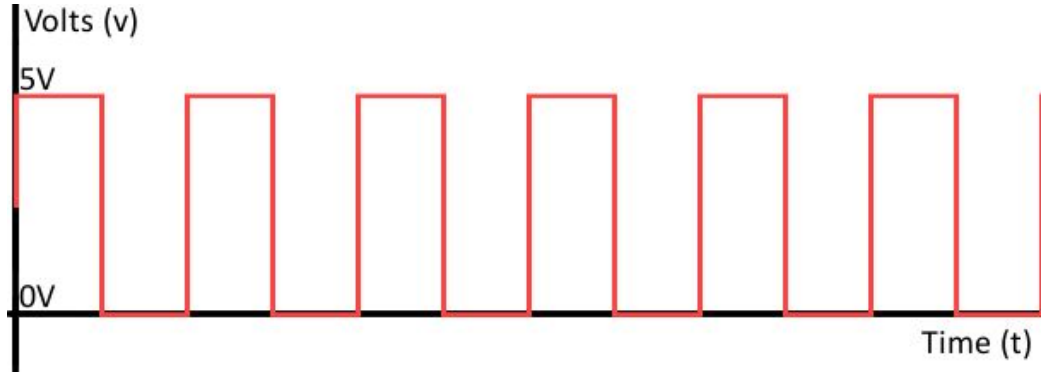


- Analog circuits are usually much more susceptible to noise (small, undesired variations in voltage). Small changes in the voltage level of an analog signal may produce significant errors when being processed.

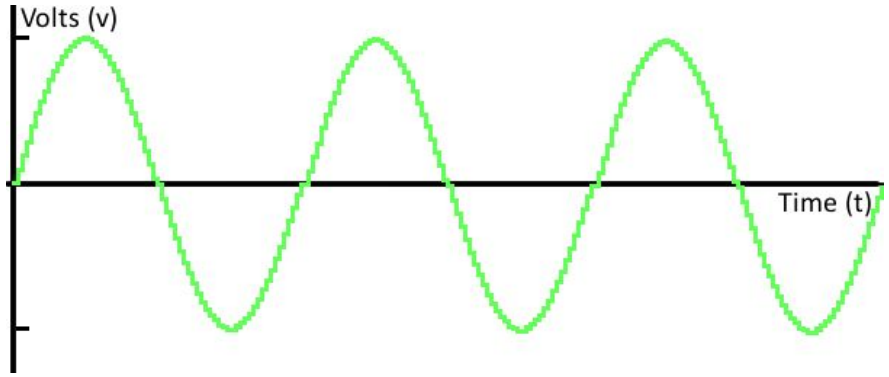
4.2 Basic Application of Sensors and Actuators Technology

4.2.3 Digital Signal

- Digital signals must contain a finite set of potential values.
- The fixed number of values can be between two and a-very-large-number-that's-not-infinit y wherever it is.
- Either of two values — as either 0V or 5V-would be most usually optical signals. Square waves appear like timing graphs of these signals.
- Or a digital signal may be a distinct depiction of an analog waveform.
- Viewed from afar, the wave function below may seem smooth and similar, but there are tiny discrete steps when you look closer as the signal attempts to approximate values:

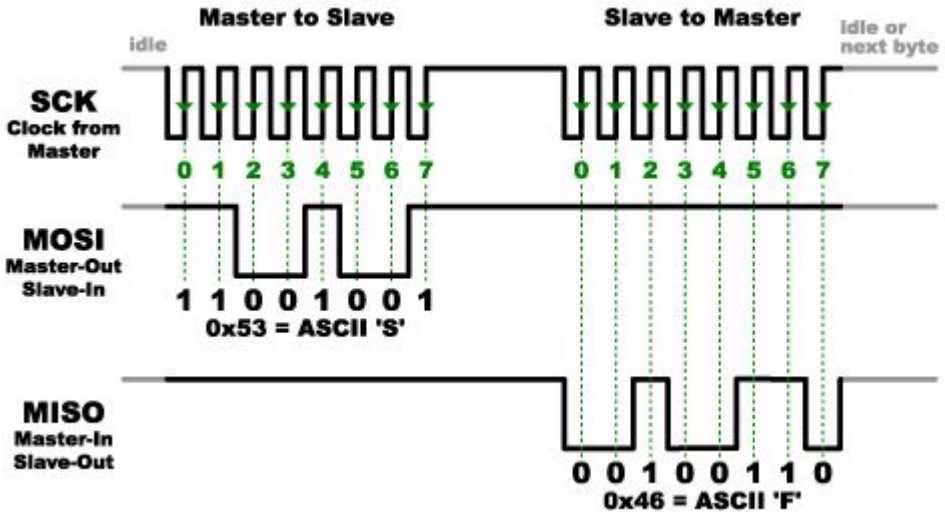


- That's the big difference between analog and digital waves. Analog waves are smooth and continuous, digital waves are stepping, square, and discrete.



4.2.3 Digital Signal - Example

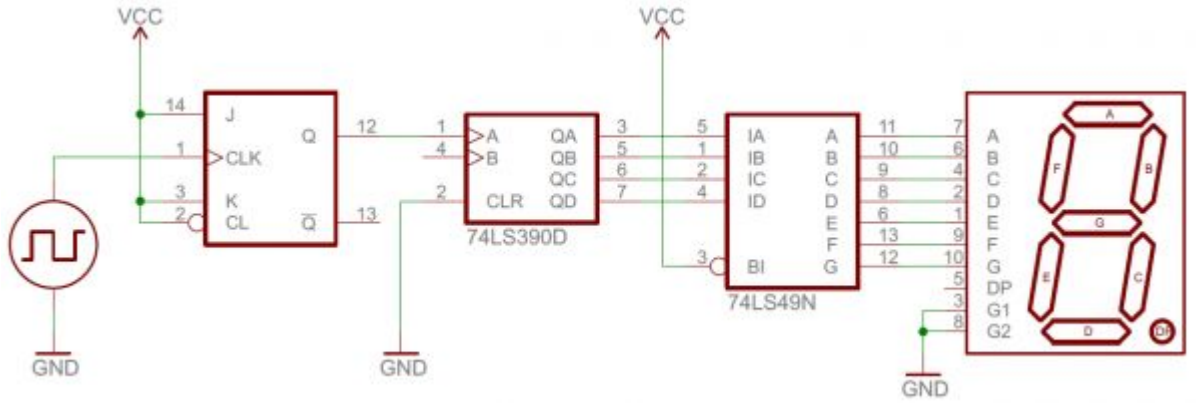
- Not all audio and video signals are analog. Standardized signals like HDMI for video (and audio) and MIDI, I2S, or AC'97 for audio are all digitally transmitted.
- Most communication between integrated circuits is digital. Interfaces like serial, I2C, and SPI all transmit data via a coded sequence of square waves.



4.2 Basic Application of Sensors and Actuators Technology

4.2.3 Digital Signal - Electronics

- Digital circuits operate using digital, discrete signals.
- These circuits are usually made of a combination of transistors and logic gates and, at higher levels, microcontrollers or other computing chips.
- Digital circuits usually use a binary scheme for digital signaling.
- These systems assign two different voltages as two different logic levels
  - a high voltage (usually 5V, 3.3V, or 1.8V) represents one value and a low voltage (usually 0V) represents the other.



- Although digital circuits are generally easier to design, they do tend to be a bit more expensive than an equally tasked analog circuit.

4.2 Basic Application of Sensors and Actuators Technology

4.2.4 Analog and Digital Combined

- It's not rare to see a mixture of analog and digital components in a circuit.
- Although microcontrollers are usually digital beasts, they often have internal circuitry which enables them to interface with analog circuitry (analog-to-digital converters, pulse-width modulation, and digital-to-analog converters.
- An analog-to-digital converter (ADC) allows a microcontroller to connect to an analog sensor (like photocells or temperature sensors), to read in an analog voltage.
- The less common digital-to-analog converter allows a microcontroller to produce analog voltages, which is handy when it needs to make sound.

4.2 Basic Application of Sensors and Actuators Technology

4.2.4 Analog and Digital Combined - ADC

- Microcontrollers can detect binary signals: is the button pressed, or is it not? Such are digital signals. When a five-volt microcontroller is powered, it understands zero volts (0V) as binary 0 and five volts (5V) as binary 1. However, what if the 2.72V signal is in? Is it a 0, or a 1? We often need to measure signals that vary; these are called analog signals. A 5V analog sensor may be 0.01V or 4.99V output, or something in between. Fortunately, almost all microcontrollers have a device built into them that allows us to convert those voltages into values that we can use to make a decision in a program.

4.2.4 Analog and Digital Combined - ADC

- An Analog to Digital Converter (ADC) is a feature that converts an analog voltage on a pin to a digital number.
- Not every pin on a microcontroller has the ability to do analog to digital conversions.
- ADCs can vary greatly between microcontroller.
- The ADC on the Arduino is a 10-bit ADC meaning it has the ability to detect 1,024 (2^10) discrete analog levels. Some microcontrollers have 8-bit ADCs (2^8 = 256 discrete levels) and some have 16-bit ADCs (2^16 = 65,536 discrete levels).

Relating ADC Value to Voltage

The ADC reports a *ratiometric value*. This means that the ADC assumes 5V is 1023 and anything less than 5V will be a ratio between 5V and 1023.

$$\frac{Resolution\ of\ the\ ADC}{System\ Voltage} = \frac{ADC\ Reading}{Analog\ Voltage\ Measured}$$

Analog to digital conversions are dependant on the system voltage. Because we predominantly use the 10-bit ADC of the Arduino on a 5V system, we can simplify this equation slightly:

$$\frac{1023}{5} = \frac{ADC\ Reading}{Analog\ Voltage\ Measured}$$

If your system is 3.3V, you simply change 5V out with 3.3V in the equation. If your system is 3.3V and your ADC is reporting 512, what is the voltage measured? It is approximately 1.65V.

If the analog voltage is 2.12V what will the ADC report as a value?

$$\frac{1023}{5.00V} = \frac{x}{2.12V}$$

Rearrange things a bit and we get:

$$\frac{1023}{5.00V} * 2.12V = x$$

$$x = 434$$

Ahah! The ADC should report 434.

4.2 Basic Application of Sensors and Actuators Technology

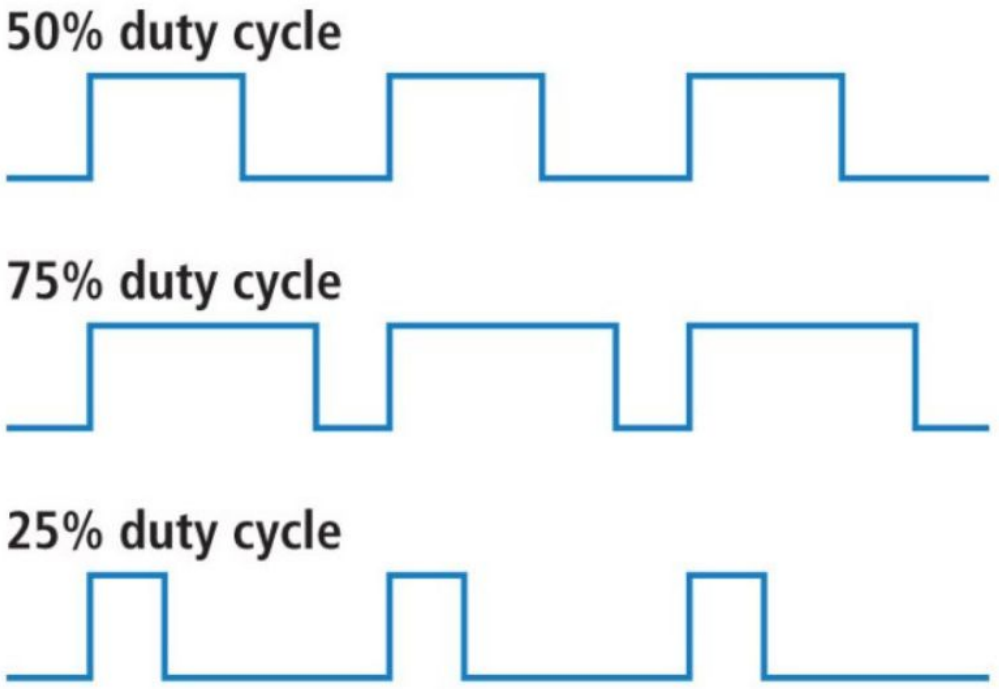
4.2.5 Pulse-Width Modulation

- Pulse Width Modulation (PWM) is a fancy term for describing a type of digital signal.
- Pulse width modulation is used in numerous applications including complex control circuitry.
- A common way we use them in learning is to control dimming of RGB LEDs or to control the direction of a servo.
- We can accomplish a range of results in both applications because pulse width modulation allows us to vary how much time the signal is high in an analog fashion.
- While the signal can only be high (usually 5V) or low (ground) at any time, we can change the proportion of time the signal is high compared to when it is low over a consistent time interval.

4.2 Basic Application of Sensors and Actuators Technology

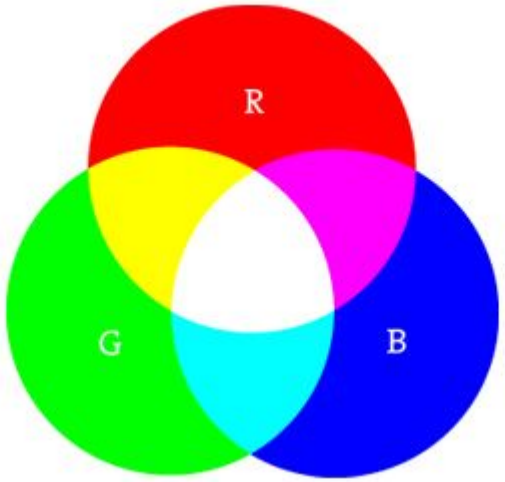
4.2.5 Pulse-Width Modulation - Duty Cycle

- Is a kind of digital signal. Pulse width modulation allows us to vary how much time the signal is high in an analog fashion.
- While the signal can only be high (usually 5V) or low (ground) at any time, we can change the proportion of time the signal is high compared to when it is low over a consistent time interval.



4.2.5 Pulse-Width Modulation - Example

- You can control the brightness of an LED by adjusting the duty cycle.
- With an RGB (red green blue) LED, you can control how much of each of the three colors you want in the mix of color by dimming them with various amounts.
- If all three are on in equal amounts, the result will be white light of varying brightness. Blue equally mixed with green will get teal. As slightly more complex example, try turning red fully on, and green 50% duty cycle and blue fully off to get an orange color.



4.2 Basic Application of Sensors and Actuators Technology

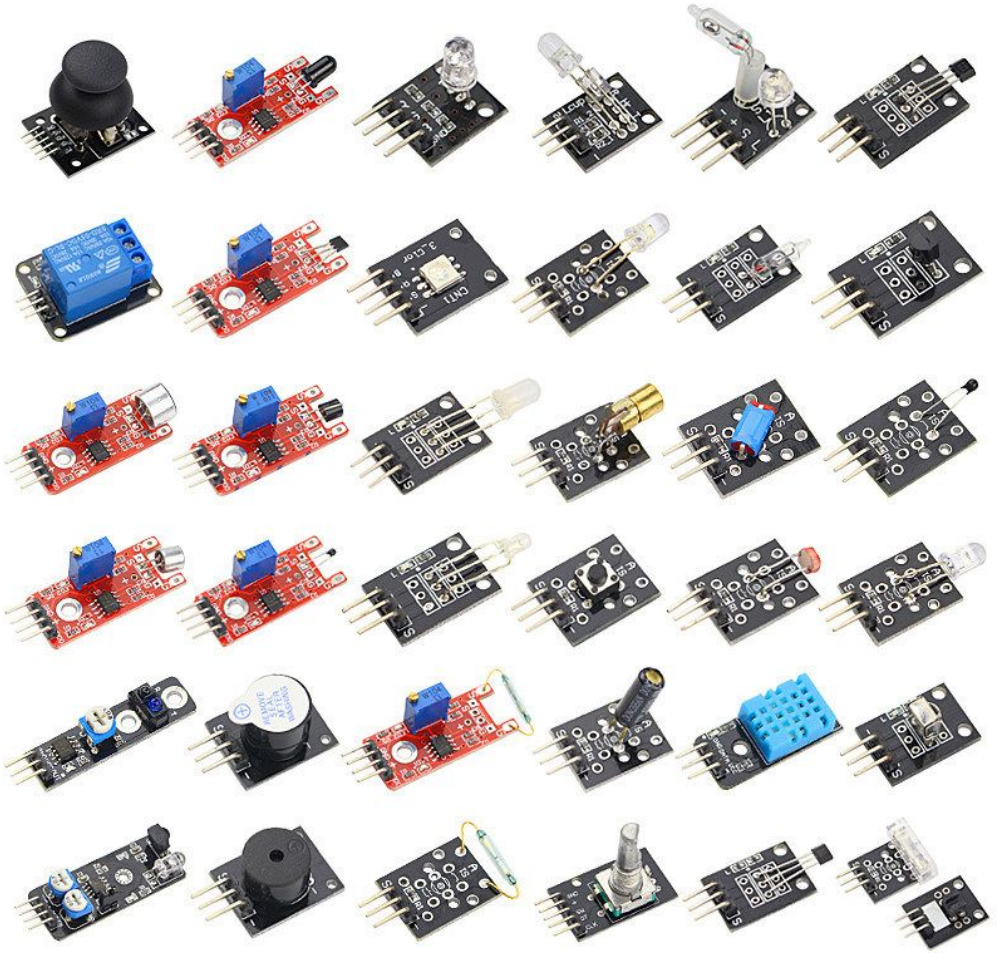
4.2.5 Pulse-Width Modulation - Example

- You can also use PWM to control the angle of a servo motor attached to something mechanical like a robot arm.
- Servos have a shaft that turns to specific position based on its control line. Our servo motors have a range of about 180 degrees.
- Frequency/period are specific to controlling a specific servo. A typical servo motor expects to be updated every 20 ms with a pulse between 1 ms and 2 ms, or in other words, between a 5 and 10% duty cycle on a 50 Hz waveform.
- With a 1.5 ms pulse, the servo motor will be at the natural 90 degree position.
- With a 1 ms pulse, the servo will be at the 0 degree position, and with a 2 ms pulse, the servo will be at 180 degrees.
- You can obtain the full range of motion by updating the servo with an value in between.

4.2 Basic Application of Sensors and Actuators Technology

4.2.6 Sensors

- A sensor is a device that detects and responds to some type of input from the physical environment.
- The specific input could be light, heat, motion, moisture, pressure, or any one of a great number of other environmental phenomena.
- The output is generally a signal that is converted to human-readable display at the sensor location or transmitted electronically over a network for reading or further processing.



4.2 Basic Application of Sensors and Actuators Technology

https://youtu.be/72DBEkGgu-w

4.2.6 Sensors - Flame Sensor

- These sensors are used for short range fire detection. They can be used for safety or to monitor projects.
- This flame sensor with grove interface can be used to detect fire source or other light sources of the wavelength in the range of 760nm - 1100 nm.
- It is based on the YG1006 sensor which is a high speed and high sensitive NPN silicon phototransistor.
- Due to its black epoxy, the sensor is sensitive to infrared radiation.

4.2 Basic Application of Sensors and Actuators Technology

https://youtu.be/s5_2QVIT-oE

4.2.6 Sensors - Magnetic Contact Switch Sensor

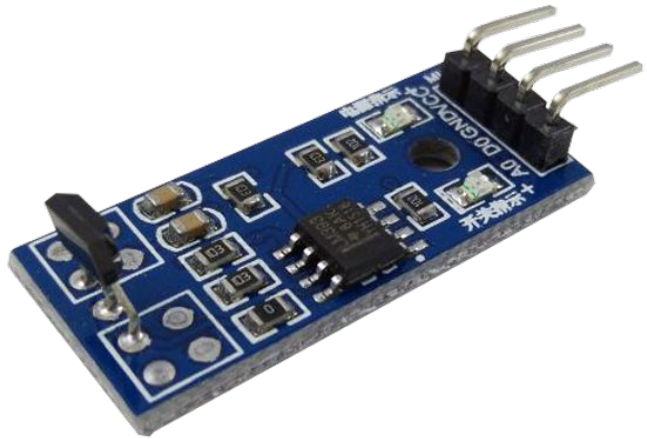
- MC-38 Magnetic Contact Switch Sensor can be used as a door or window security system.
- It produces the signal when moved away from each other which can be fed to the microcontroller (e.g Arduino) to perform the desired action as per requirement.
- This sensor suitable to use for trigger alarm or ON/OFF light inside a cupboard sliding door.
- This wired sensor is trigger by the magnet. When the magnet is closed by, the circuit is closed or open if the magnet far from the sensor.

4.2 Basic Application of Sensors and Actuators Technology

https://youtu.be/jyx6VWk-5Ok

4.2.6 Sensors - Hall Magnetic Sensor

- The Hall Magnetic Sensor is a magnetic switch.
- If no magnetic field is present, the signal line of the sensor is HIGH.
- If a magnetic field is presented to the sensor, the signal line goes LOW, at the same time the LED on the sensor lights up.
- The polarity of the magnetic field is of influence to the switching action.
- The front side of the sensor needs the opposite polarity as the back of the sensor to switch on.

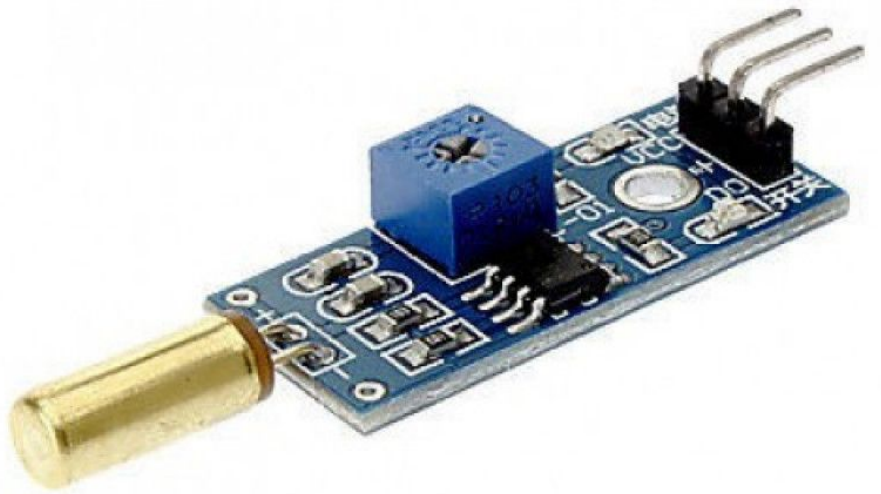


4.2 Basic Application of Sensors and Actuators Technology

https://youtu.be/YghN9MvCRQQ

4.2.6 Sensors - Tilt Switch

- The Tilt Switch is a mercury tilt switch that allows you to detect the tilt of your object.
- It provides a digital output.
- The tilt sensor module come with the basic components for operation.
- Attach it to object and it will detect whether the object is tilted.
- Simple usage as it is digital output, so you will know the object is tilted or not by reading the output.
- It uses SW-460D or SW-520D tilt sensor.
- The tilt sensor is ball rolling type, NOT Mercury type.



4.2 Basic Application of Sensors and Actuators Technology

https://youtu.be/53hGjRBFj40

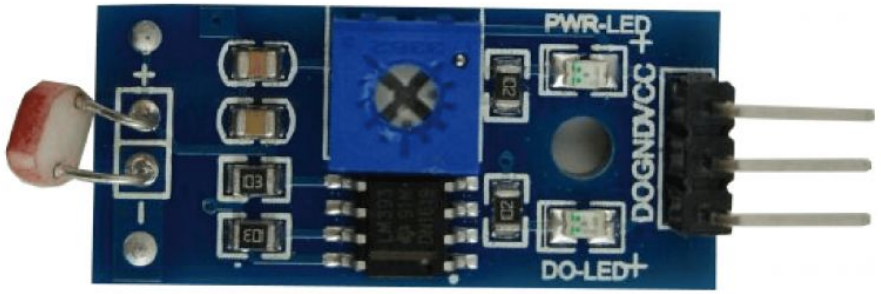
4.2.6 Sensors - Ball Switch

- This Water Level Sensor Float Switch is a sensor used to sense the level of the liquid within a tank, it may further trigger a pump, an indicator, an alarm, or other device.
- This water level float sensor is very easy to use.
- Act as a mechanical switch or dry contact, extended out into 2 wires.
- There is a float-able cylinder component surrounding the sensor.
- When the cylinder at the bottom, the switch is open. When water level rises, it push the cylinder upwards and the switch closes.

https://youtu.be/_75rU5aNt_I

4.2.6 Sensors - Photoresistor

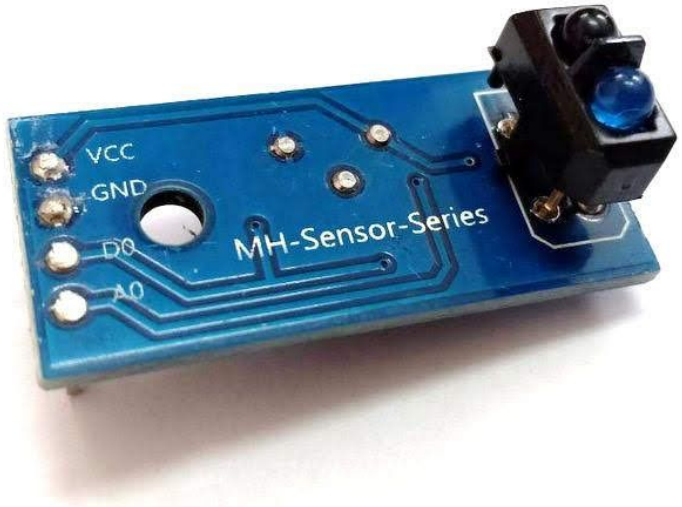
- This is a variable resistor.
- The resistance value depends on the amount of light coming onto the LDR (Light Dependent Resistor).
- This could be used to switch on a light when it becomes dark.



https://youtu.be/X6qFEwy2X2w

4.2.6 Sensors - Tracking Sensor

- This sensor can detect lines in black and white.
- You could for example make a robot/car follow a line.

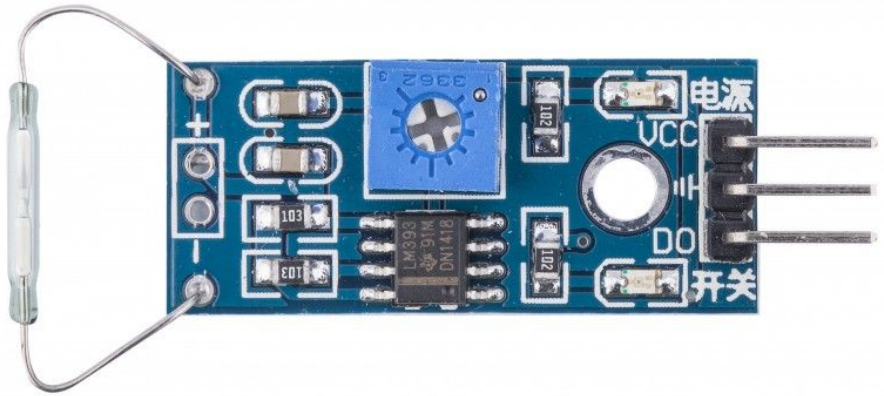


4.2 Basic Application of Sensors and Actuators Technology

https://youtu.be/2RBFKqoaual

4.2.6 Sensors - Reed Switch

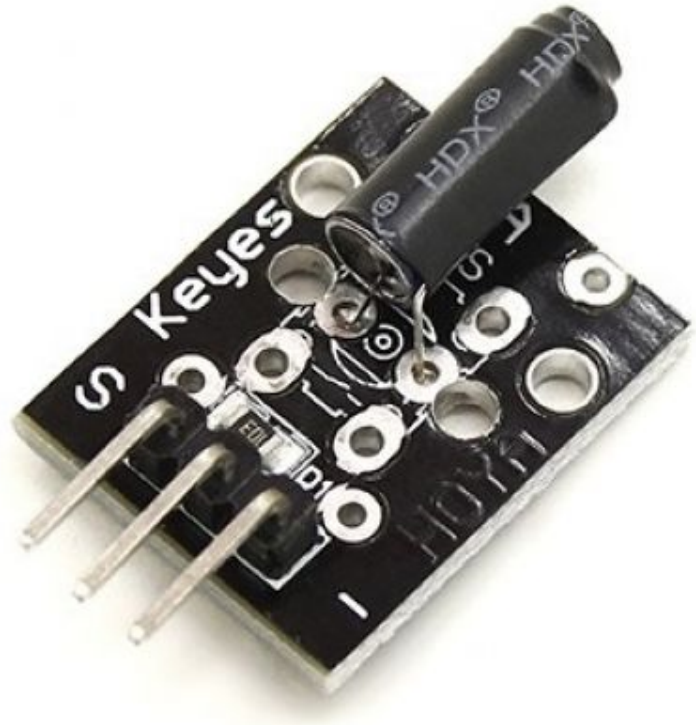
- A reed switch is a switch that needs a magnet in front of it to switch on or off.
- Reed switch is a kind of passive electronic switching component with contacts with a simple structure, small size and easy to use.
- It consists of a sealed glass envelope where there are two ferrous elastic reeds and is filled with inert gas called rhodium.



https://youtu.be/aKt1Fo00GIA

4.2.6 Sensors - Shock Sensor

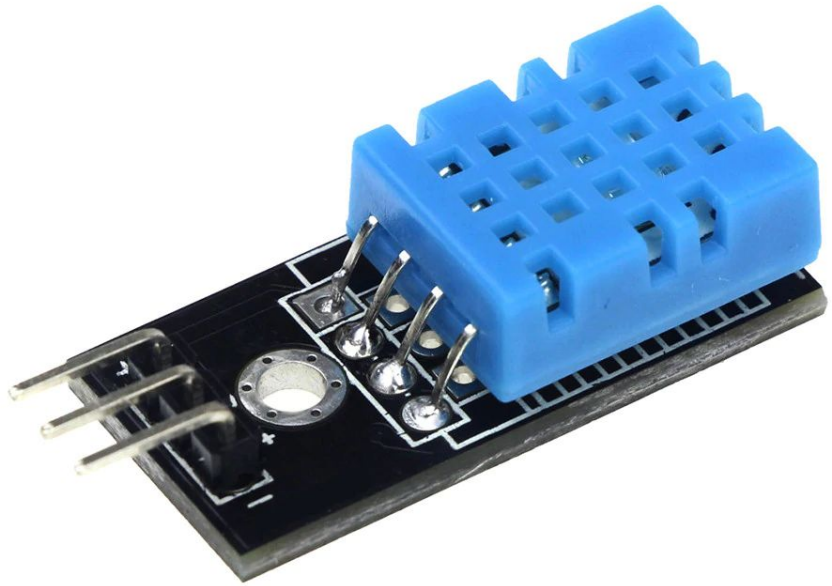
- This sensor module can detect vibrations/shocks.
- It's switched off in the neutral state and will switch on when it catches a vibration.



https://youtu.be/OogldLc9uYc

4.2.6 Sensors - Temperature and Humidity Sensor

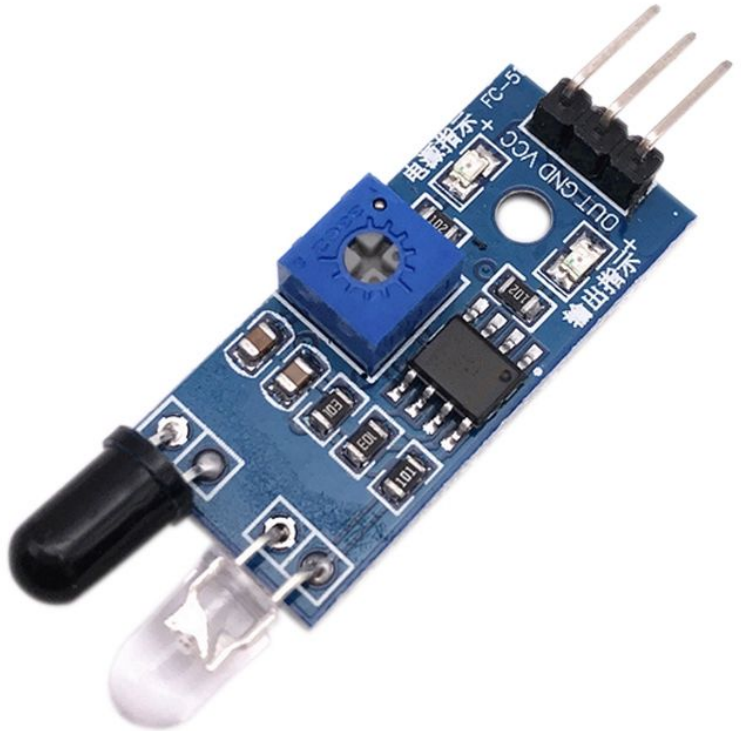
- This sensor can monitor the temperature and humidity.
- Since it's so small it's not very effective in larger area's, but it works fine in for example a reptile sanctuary.



https://youtu.be/QWCdcwxF3RE

4.2.6 Sensors - Avoidance Sensor

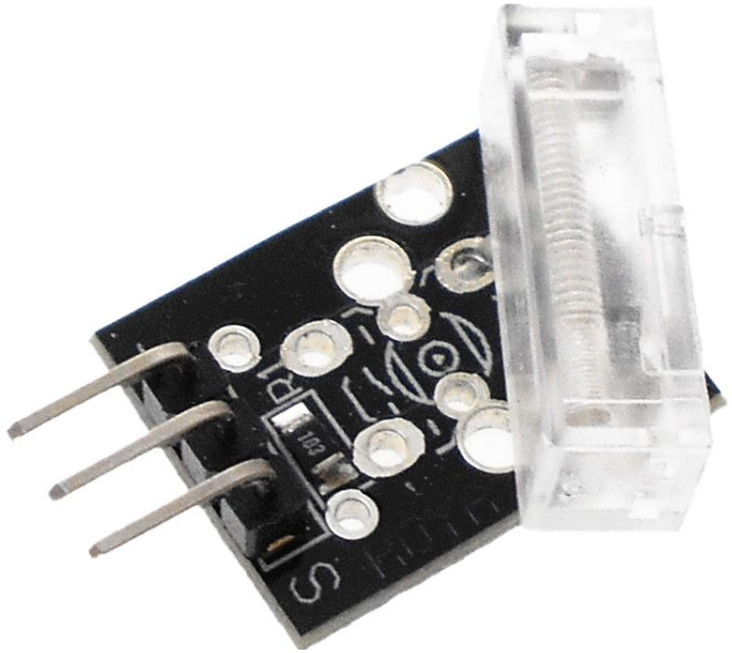
- This sensor uses a infrared emitter and receiver to check whether there are any obstacles in front of it.
- This can be useful for a robot.



https://youtu.be/eyO89JlGysk

4.2.6 Sensors - Tap Module

- The knock sensor, detects the knocks and the taps.
- It can work like a switch.
- The sensor sends data momentarily to the board.

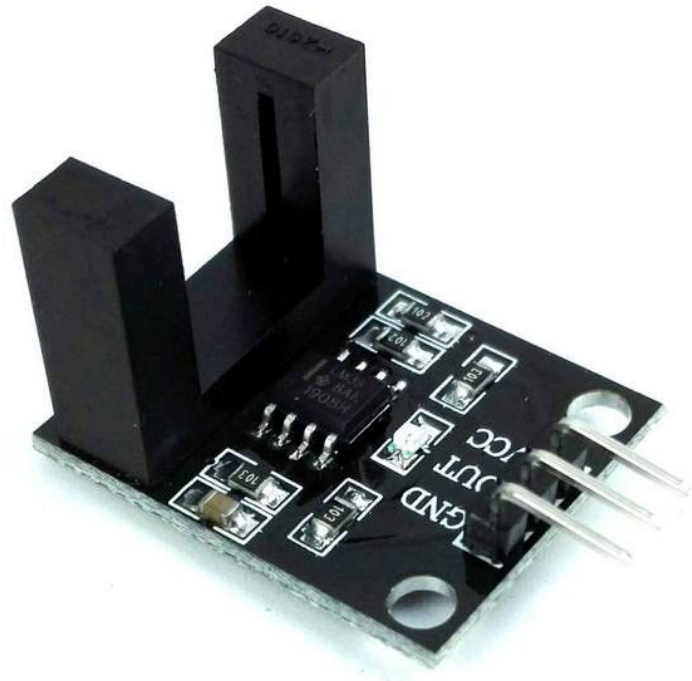


4.2 Basic Application of Sensors and Actuators Technology

https://youtu.be/9deOgUOJvQE

4.2.6 Sensors - Light Blocking Sensor

- This is a light block sensor module, where there is an object in the middle of the U shape. The sensor will output a high level signal.
- This Encoder Sensor Module Digital Output can use to check the rate of an electric motor.
- This module can be used for motor speed detection, pulse count and position limit.
- In principle, any rate meter simply measures the rate at which some event occurs. Usually, this is done by counting the events for a given period of time (integration interval) and then simply dividing the number of events by the time to get the rate.



4.2 Basic Application of Sensors and Actuators Technology

https://youtu.be/O6W_dbIkkmE

4.2.6 Sensors - Moisture Sensor

- This sensor comes in 2 parts, sensor probes and module board.
- The sensor is basically two probes to be inserted into soil.
- This sensor uses the two probes to pass current through the soil, and then it reads that resistance to get the moisture level.
- More water makes the soil conduct electricity more easily (less resistance), while dry soil conducts electricity poorly (more resistance).
- The module come with a comparator and adjustable potentiometer for user to adjust the threshold to toggle digital output.

4.2 Basic Application of Sensors and Actuators Technology

https://youtu.be/kA_pbMR6jVs

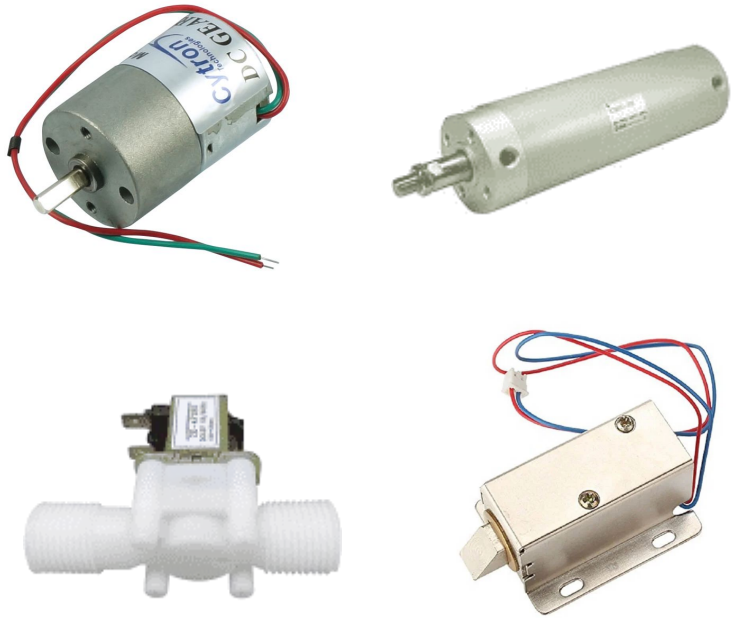
4.2.6 Sensors - Joystick

- The JoyStick is a analog sensor that can be used to control your board.
- Analog JoySticks are basically potentiometers, so they return values which can be made visible using the provided code in this step.
- This is 2-Axis Joystick & a digital button (center push).
- The 2-Axis Joystick contains two independent potentiometers (one per axis) for reporting the joystick's position, with wiring options for voltage output.
- The 2-Axis Joystick includes spring auto return to center and a comfortable cup-type knob.

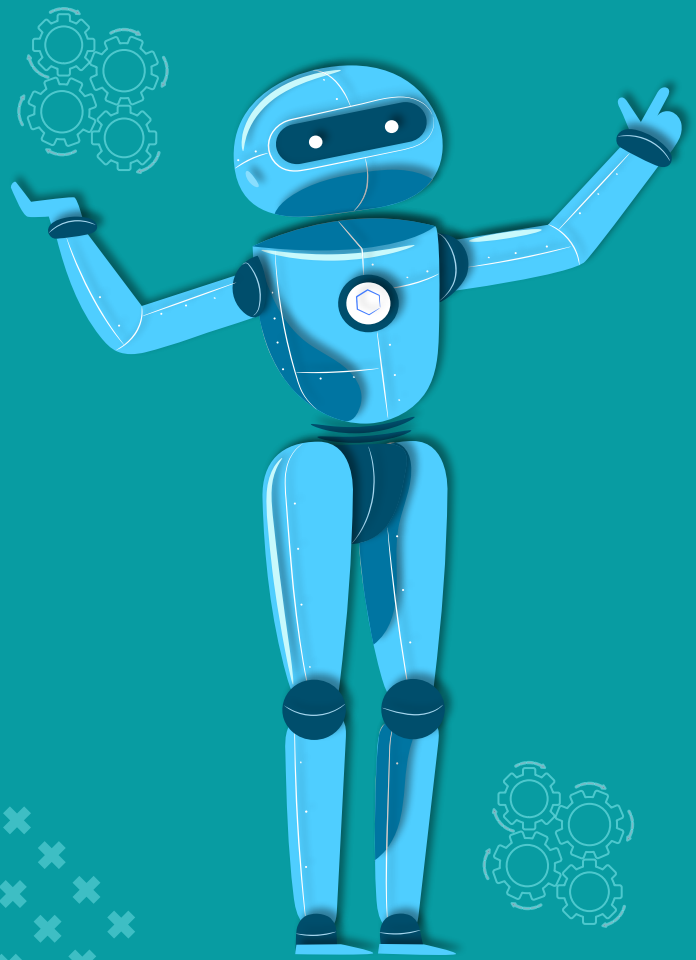
4.2 Basic Application of Sensors and Actuators Technology

4.2.7 Actuators

- An actuator is a component of a machine that is responsible for moving and controlling a mechanism or system, for example by opening a valve. In simple terms, it is a "mover".
- An actuator requires a control signal and a source of energy. The control signal is relatively low energy and may be electric voltage or current, pneumatic or hydraulic pressure, or even human power. Its main energy source may be an electric current, hydraulic fluid pressure, or pneumatic pressure. When it receives a control signal, an actuator responds by converting the source's energy into mechanical motion.
- Some types of actuators includes;
  - Electric motors
  - Comb drive
  - Hydraulic cylinder
  - Pneumatic cylinder



4.3



Fundamental of Embedded Programming for IoT

- Micropython
- Micropython - uPyCraft IDE
- Micropython - ESP32 GPIO
- Micropython - Let's code
- Micropython - Basic programming using ESP32

4.3 Fundamental of Embedded Programming for IoT

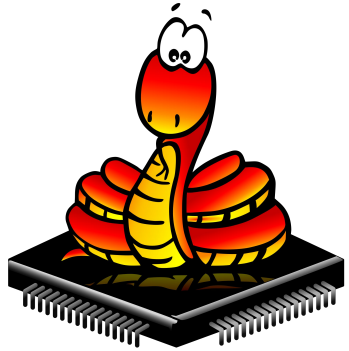
4.3.1 Micropython

- MicroPython is a re-implementation of Python 3 targeted for microcontrollers and embedded systems.
- MicroPython is very similar with regular Python.
- So, if you already know how to program in Python, you also know how to program in MicroPython.
- MicroPython is a lean and efficient implementation of the Python 3 programming language.
- It includes a small subset of the Python standard library and is optimised to run on microcontrollers and in constrained environments.
- Python programming language is an interpreted language,
    - user do not compile to machine code, rather, python compiler will compile the source codes on the fly into machine code which is executable on the underlying hardware architecture.

4.3 Fundamental of Embedded Programming for IoT

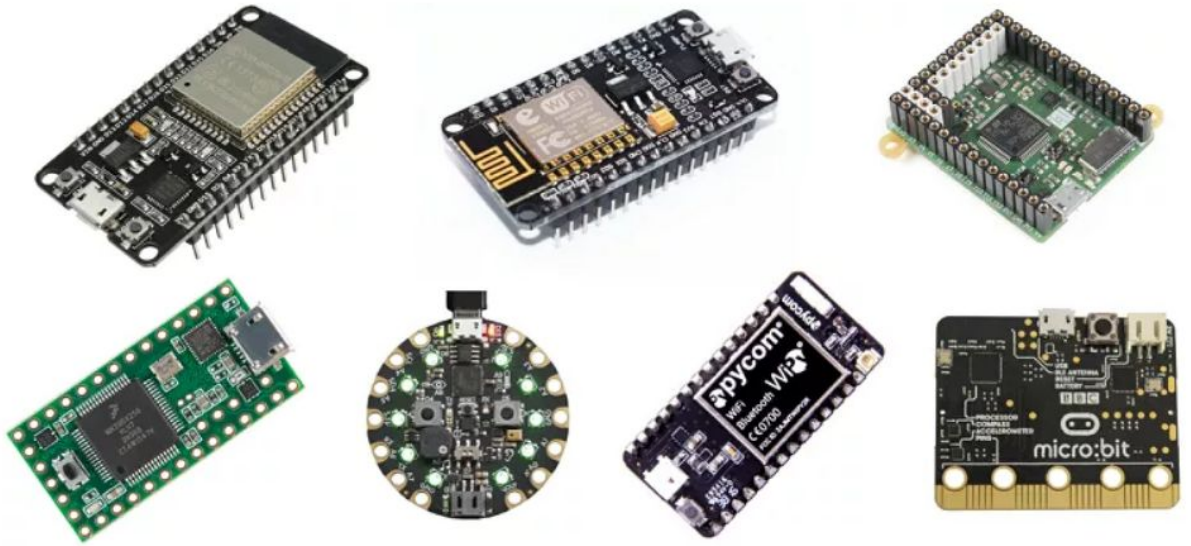
4.3.1 Micropython

- Python is one of the most widely used, simple and easy-to-learn programming languages around.
- So, the emergence of MicroPython makes it extremely easy and simple to program digital electronics.
- If you've never programmed digital electronics before, MicroPython is a good starting point.
- MicroPython's goal is to make programming digital electronics as simple as possible, so it can be used by anyone.
- Currently, MicroPython is used by hobbyists, researchers, teachers, educators, and even in commercial products.



4.3.1 Micropython - Boards Support

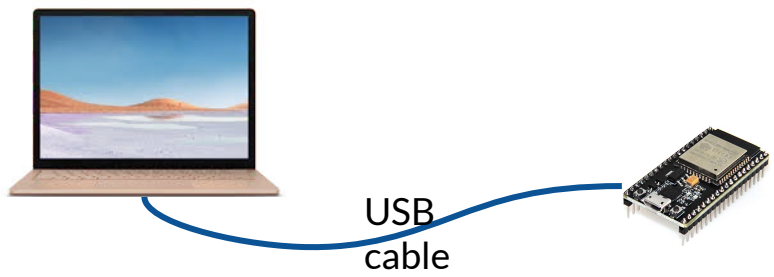
- MicroPython runs on many different devices and boards, such as:
  - ESP32
  - ESP8266
  - PyBoard
  - Micro:Bit
  - Teensy 3.X
  - WiPy – Pycom
  - Adafruit Circuit Playground Express
  - Other ESP32/ESP8266 based boards



4.3.2 Micropython - uPyCraft IDE

- Step to install
  - Install uPyCraft IDE
  - Download micropython firmware
  - Install USB-Serial chipset driver
  - Connect microcontroller to computer USB port
  - Identify the communication port name
  - Select communication port in IDE
  - Select the microcontroller model / version
  - Execute burn-firmware
  - Select the firmware file
  - Flash the firmware to the microcontroller

- Requirements
  - A development board
  - Micropython firmware suitable for the board
  - Micro USB cable + driver
  - IDE



4.3.2 Micropython - uPyCraft IDE

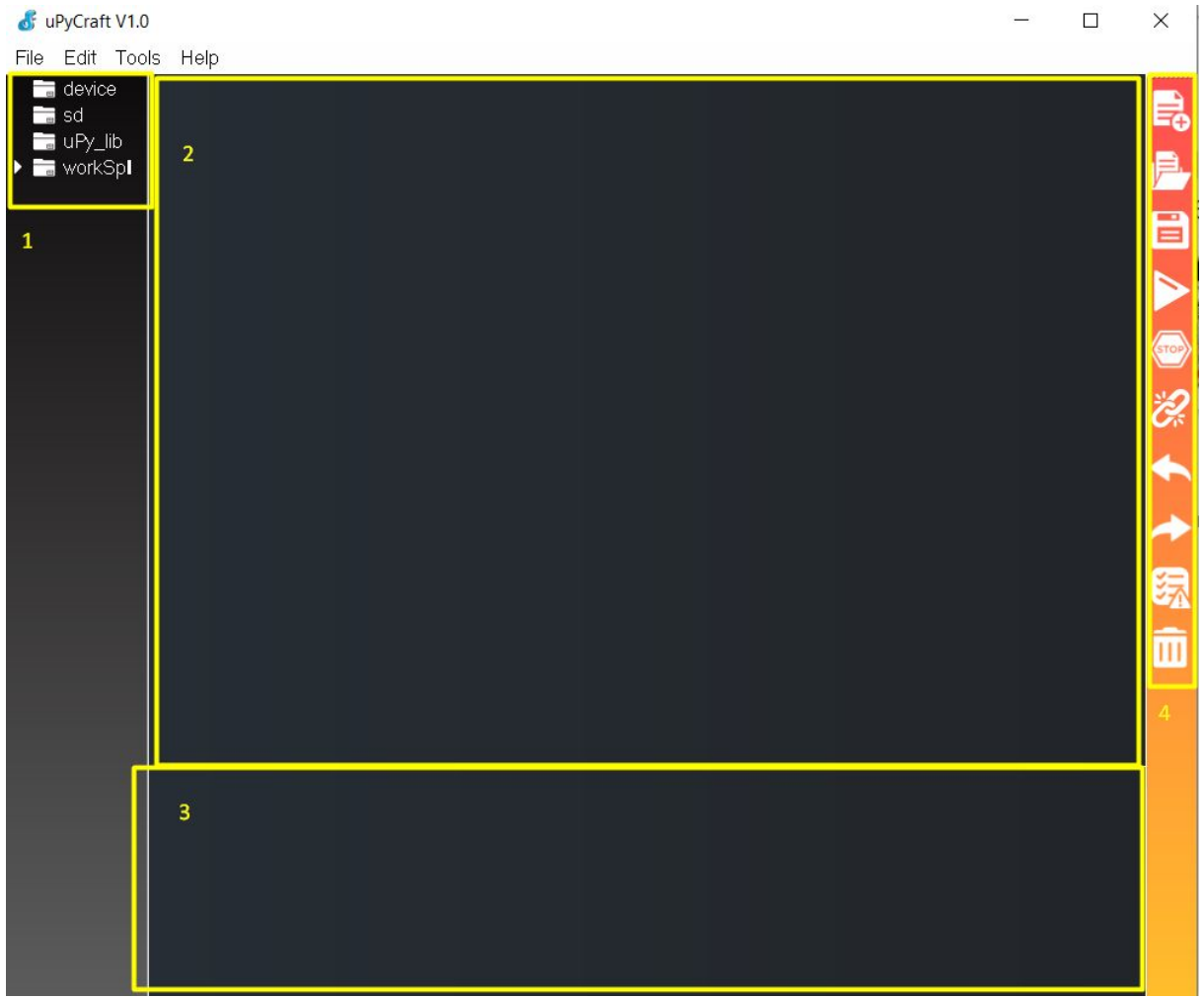
- In this section we'll give you an overview of the uPyCraft IDE software, so that you can start programming the ESP32/ESP8266 with MicroPython.
- The IDE is a software that contains tools to make the process of development, debugging and upload code easier. There are many ways to program your ESP board with MicroPython. We've chosen uPyCraft IDE because it is simple and intuitive to use and works great with the ESP boards.

- At this point, we assumed that you have:

  - uPyCraft IDE installed on your computer
  - ESP32/ESP8266 flashed with MicroPython firmware

4.3.2 Micropython - uPyCraft IDE

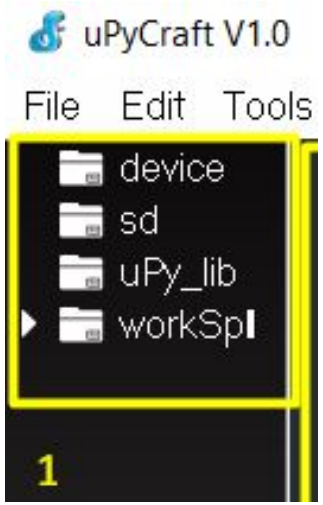
- MicroPython runs on many different devices and boards, such as:
  1. Folder and files
  2. Editor
  3. MicroPython Shell/Terminal
  4. Tools



4.3.2 Micropython - uPyCraft IDE

1. Folder and files

- This section shows several folders and files.
- The device folder shows the files that are currently stored on your ESP board.
  - If you have your ESP32 or ESP8266 connected via serial to uPyCraft IDE, when you expand the device folder, all files stored should load.
  - By default, you should only have a boot.py file.
  - To run your main code, it is recommended to create a main.py file.

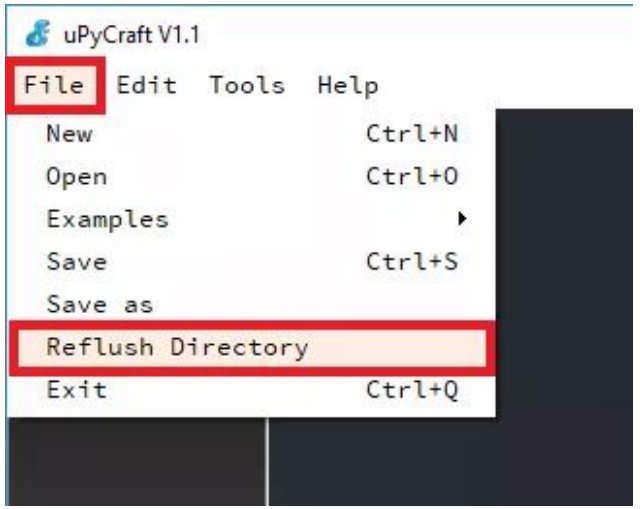


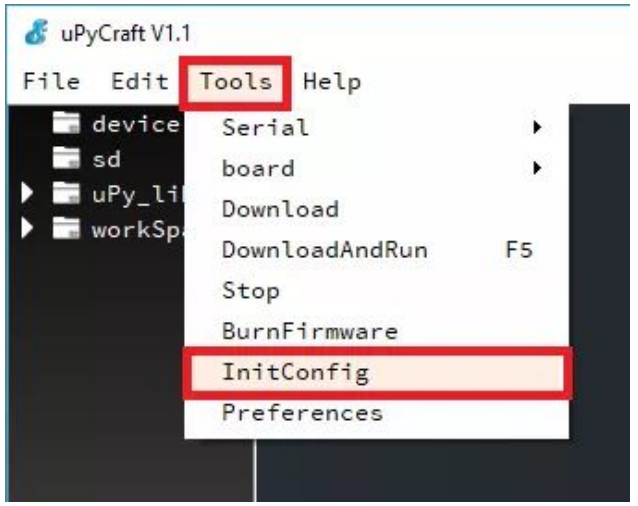
- **boot.py**: runs when the device starts and sets up several configuration options;
- **main.py**: this is the main script that contains your code. It is executed immediately after the boot.py.
- The sd folder is meant to access files stored on SD cards – this is only works with boards like the PyBoard that come with an SD card slot.
- The uPy_lib shows the built-in IDE library files.

4.3.2 Micropython - uPyCraft IDE

1. Folder and files

- Finally, the workspace is a directory to save your files. These files are saved in your computer in a directory defined by you. This is a specially useful to keep all your files organized at hand.
- When using uPycraft for the first time, to select your working directory, click the workspace folder. A new window pops up for you to chose your workspace path. Create a new folder or select an existing folder to be your working directory.
- **Then, go to File > Reflush Directory to update the directory.**



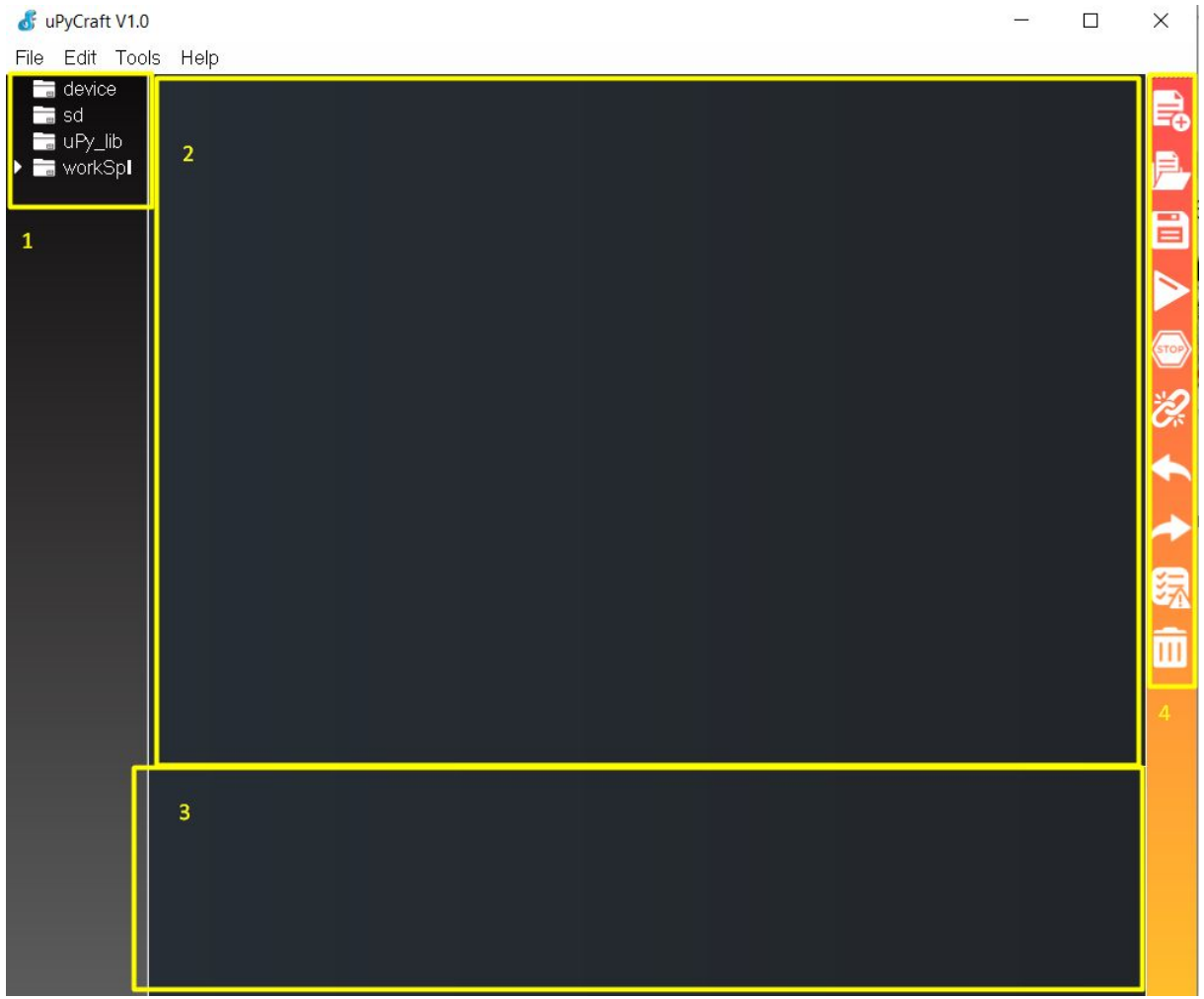


Note: to change your user directory, **simply go to Tools >InitConfig** and click the workspace directory folder to chose a different path.

4.3.2 Micropython - uPyCraft IDE

2. Editor

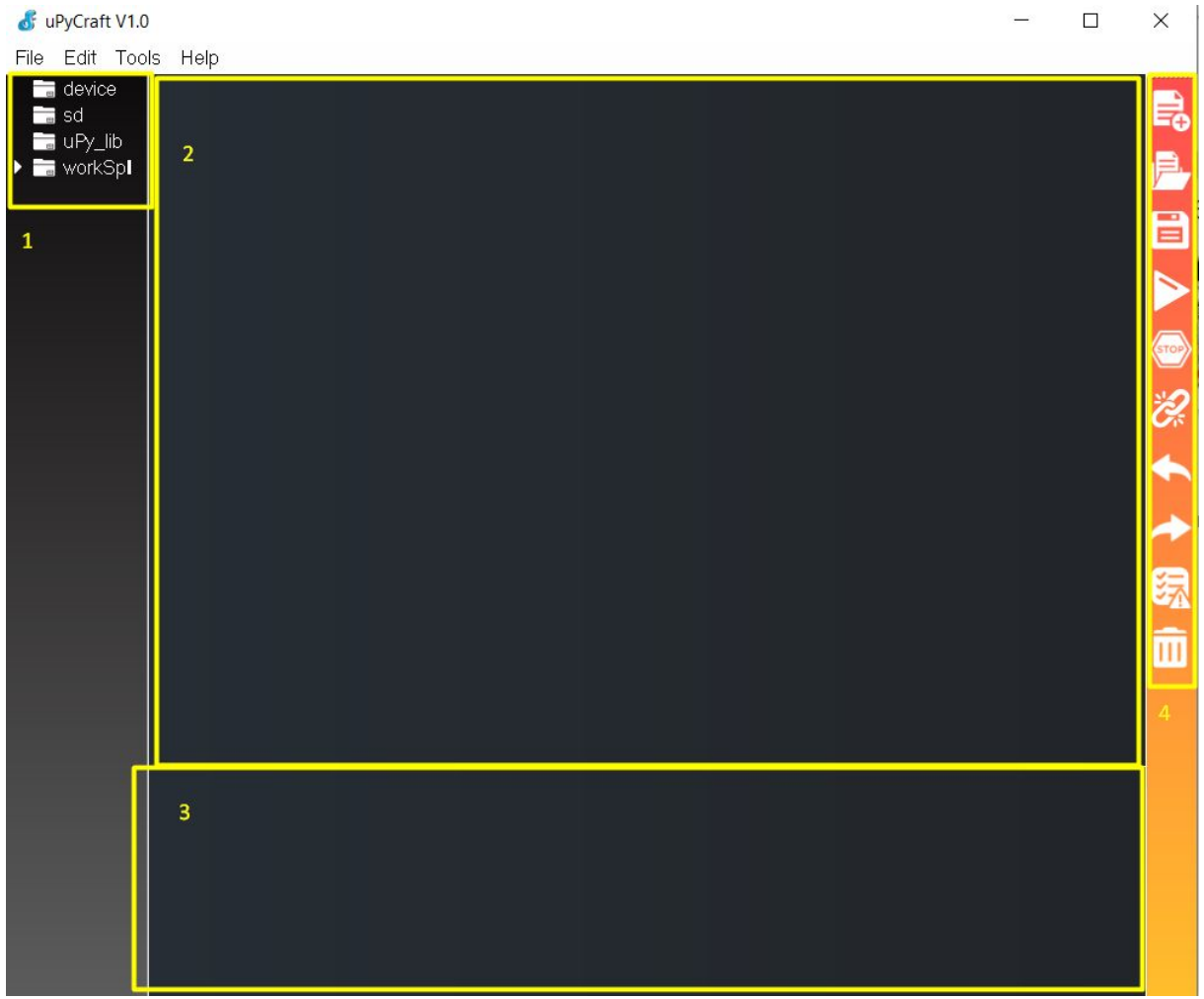
- The Editor section is where you write your code and edit your .py files. You can open more than one file, and the Editor will open a new tab for each file.



4.3.2 Micropython - uPyCraft IDE

3. MicroPython Shell/terminal

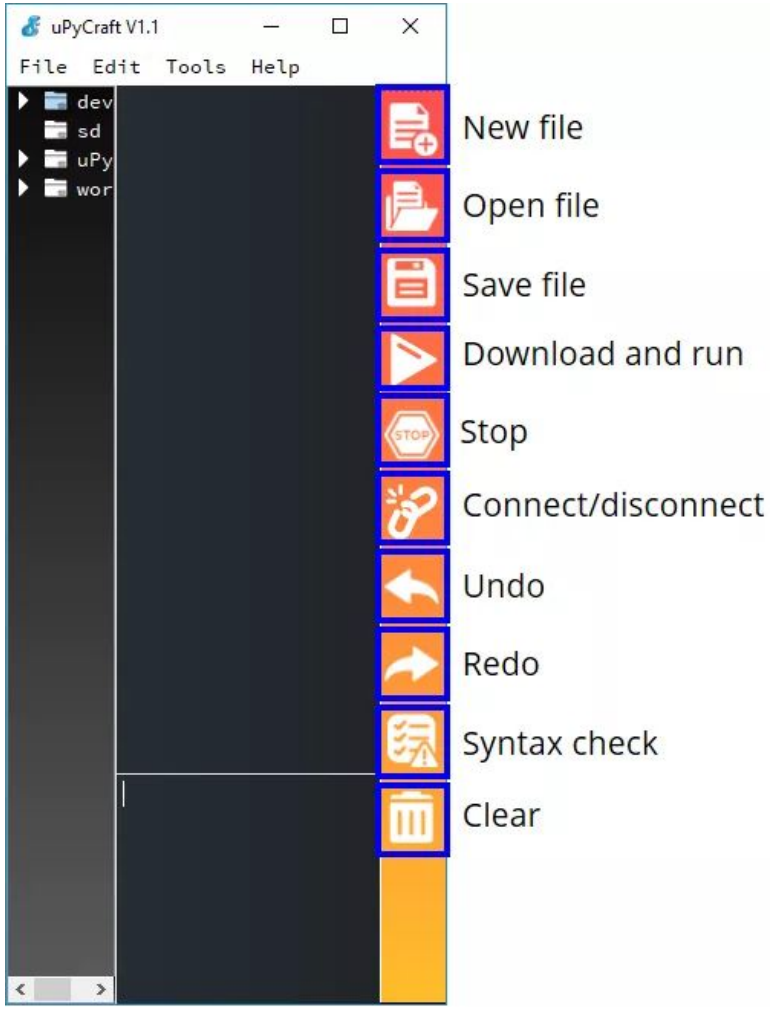
- On the MicroPython Shell you can type commands to be executed immediately by your ESP board without the need to upload new files. The terminal also provides information about the state of an executing program, shows errors related with upload, syntax errors, prints messages, etc...



4.3.2 Micropython - uPyCraft IDE

4. Tools

- New file: creates a new file on the Editor;
- Open file: open a file from your computer;
- Save file: saves a file;
- Download and run: upload the code to your board and execute the code;
- Stop: stop the execution of the code – it's the same as entering CRTL+C on the Shell to stop all scripts from running;
- Connect/Disconnect: connect or disconnect to your board via Serial. You must select the serial port first in Tools > Serial;
- Undo: undo last change in the code Editor;
- Redo: redo last change in the code Editor;
- Syntax check: checks the syntax of your code;
- Clear: clear the Shell/terminal window messages.



4.3.3 Micropython - ESP32 GPIO

Digital I/O

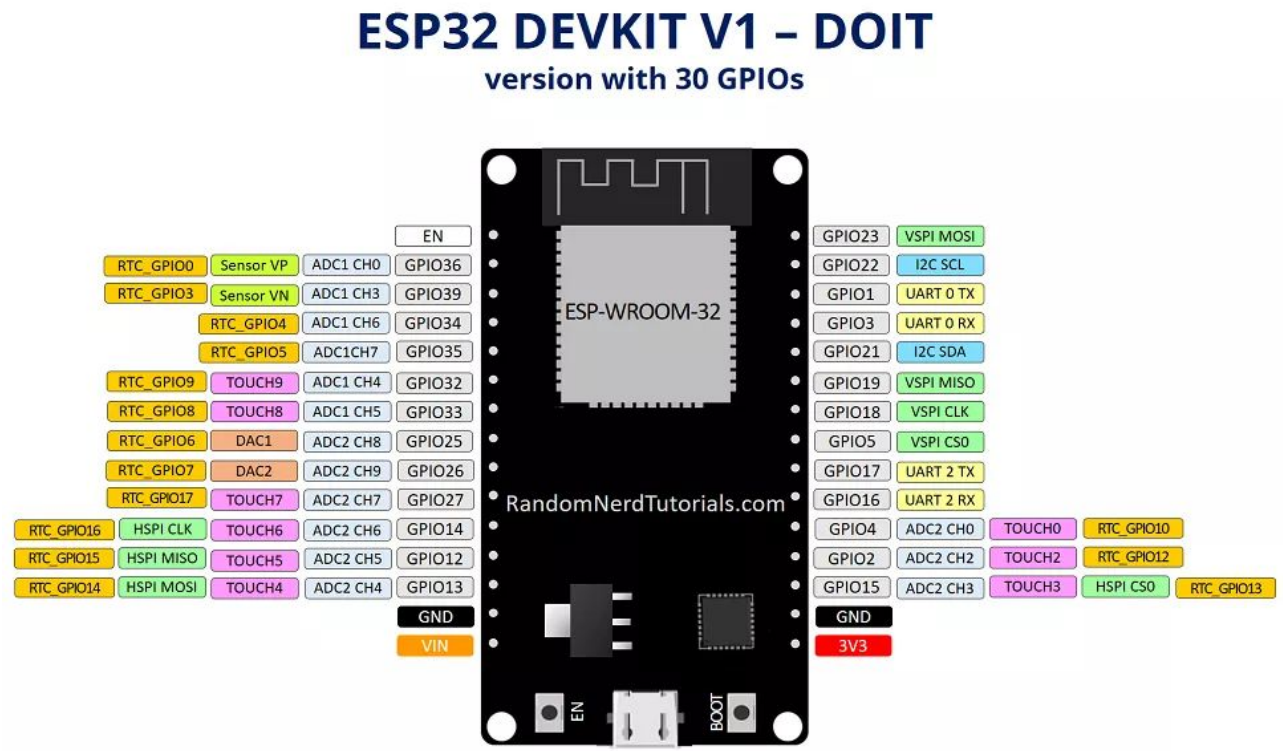
- Digital I/O stands for Digital Input and Output. Digital Inputs allow a microcontroller to detect logic states. High or Low, True or False, 1 or 0.
- Digital Outputs allow a microcontroller to output logic states. High or Low, True or False, 1 or 0.
- Each digital I/O on a pin can be individually configured to one of 3 states: input, output-high, or output-low.

ESP32 - I/O Pin Capability

- Voltage and current restrictions:
- The ESP32 is a 3.3V microcontroller, so its I/O operates at 3.3V as well.
- The pins are not 5V tolerant, applying more than 3.6V on any pin will kill the chip.
- The maximum current that can be drawn from a single GPIO pin is 40mA.

4.3.3 Micropython - ESP32 GPIO

- 18 Analog-to-Digital Converter (ADC) channels
- 3 SPI interfaces
- 3 UART interfaces
- 2 I2C interfaces
- 16 PWM output channels
- 2 Digital-to-Analog Converters (DAC)
- 2 I2S interfaces
- 10 Capacitive sensing GPIOs



4.3.3 Micropython - ESP32 GPIO

- There are pins with specific features that make them suitable or not for a specific project
- Green are OK to use.
- Yellow are OK to use, but you need to pay attention because they may have unexpected behavior mainly at boot.
- Red are not recommended to use as inputs or outputs.

GPIO	Input	Output	Notes
0	pulled up	OK	outputs PWM signal at boot
1	TX pin	OK	debug output at boot
2	OK	OK	connected to on-board LED
3	OK	RX pin	HIGH at boot
4	OK	OK	
5	OK	OK	outputs PWM signal at boot
6	x	x	connected to the integrated SPI flash
7	x	x	connected to the integrated SPI flash
8	x	x	connected to the integrated SPI flash
9	x	x	connected to the integrated SPI flash
10	x	x	connected to the integrated SPI flash
11	x	x	connected to the integrated SPI flash
12	OK	OK	boot fail if pulled high

4.3.3 Micropython - ESP32 GPIO

- There are pins with specific features that make them suitable or not for a specific project
- Green are OK to use.
- Yellow are OK to use, but you need to pay attention because they may have unexpected behavior mainly at boot.
- Red are not recommended to use as inputs or outputs.

13	OK	OK	
14	OK	OK	outputs PWM signal at boot
15	OK	OK	outputs PWM signal at boot
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	

4.3.3 Micropython - ESP32 GPIO

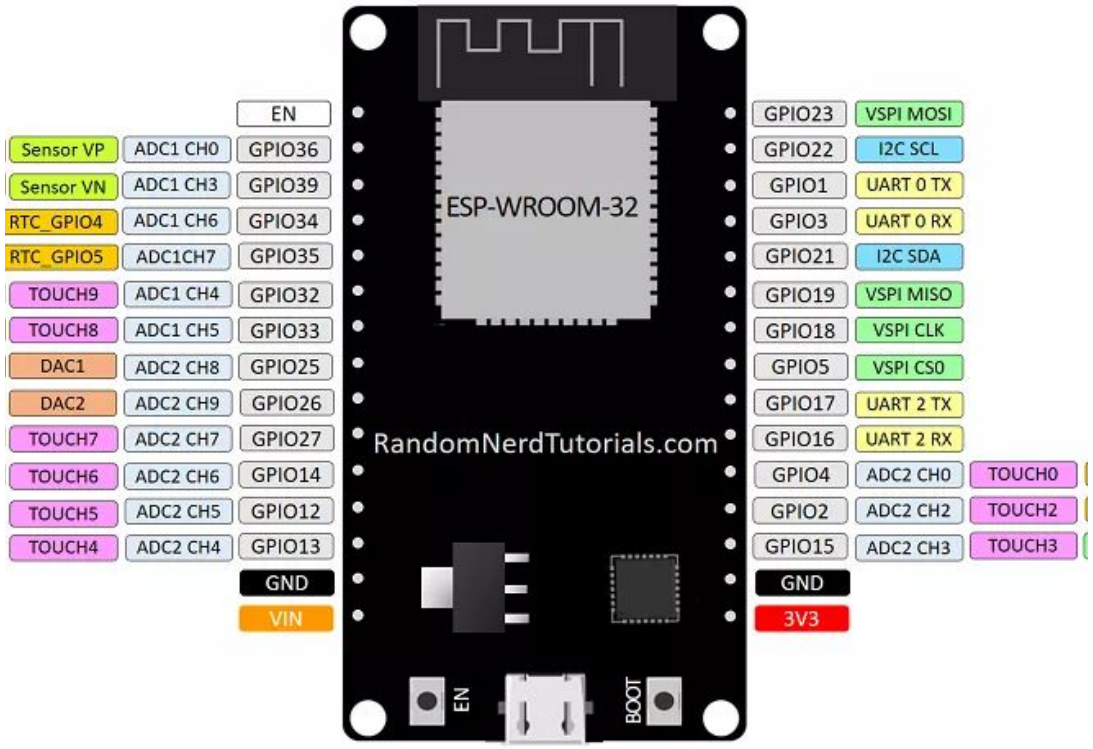
- There are pins with specific features that make them suitable or not for a specific project
- Green are OK to use.
- Yellow are OK to use, but you need to pay attention because they may have unexpected behavior mainly at boot.
- Red are not recommended to use as inputs or outputs.

Notes: These pins don't have internal pull-ups or pull-down resistors. They can't be used as outputs, so use these pins only as inputs

34	OK		input only
35	OK		input only
36	OK		input only
39	OK		input only

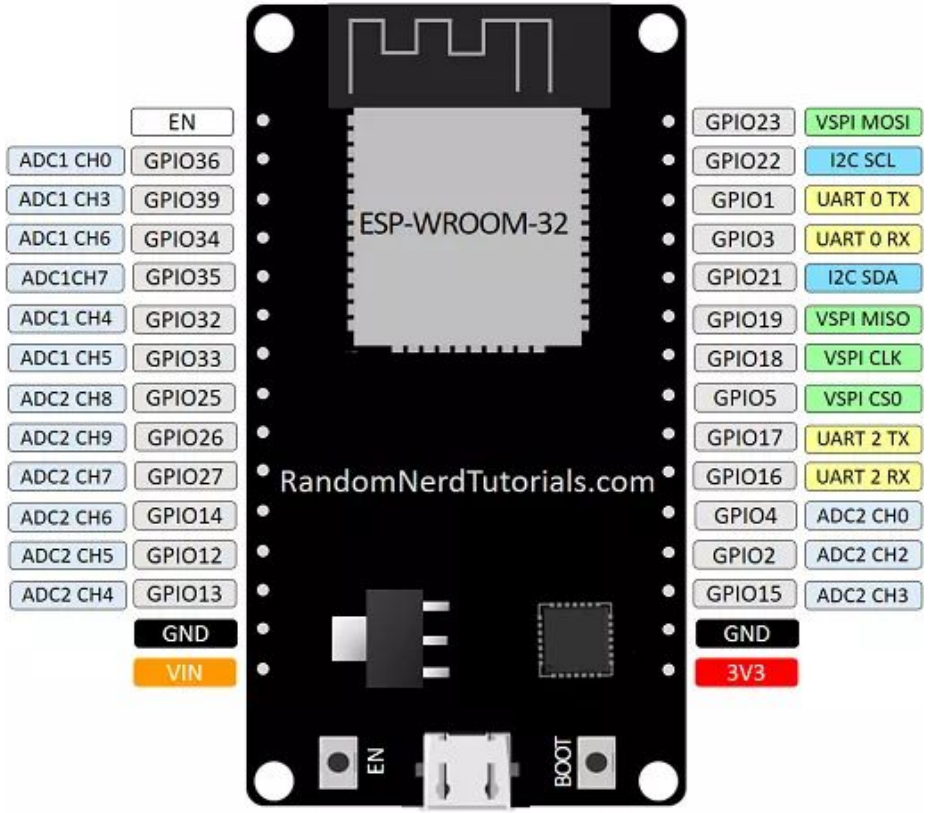
4.3.3 Micropython - ESP32 GPIO

- The ESP32 has 10 internal capacitive touch sensors.
- These can sense variations in anything that holds an electrical charge, like the human skin.
- Can detect variations induced when touching the GPIOs with a finger.
- Can be easily integrated into capacitive pads, and replace mechanical buttons.
- Can also be used to wake up the ESP32 from deep sleep.



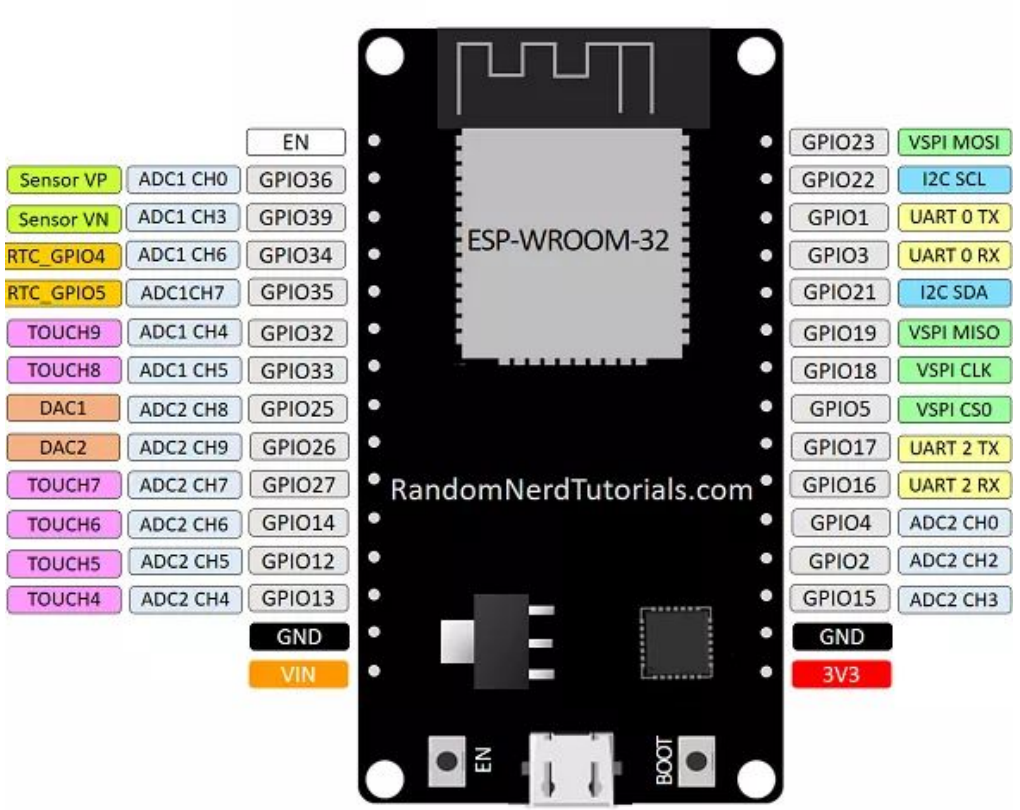
4.3.3 Micropython - ESP32 GPIO

- The ESP32 has 18 x 12 bits ADC input channels (while the ESP8266 only has 1x 10 bits ADC).
- These are the GPIOs that can be used as ADC.
- The ADC input channels have a 12 bit resolution.
- Can get analog readings ranging from 0 to 4095, in which 0 corresponds to 0V and 4095 to 3.3V.



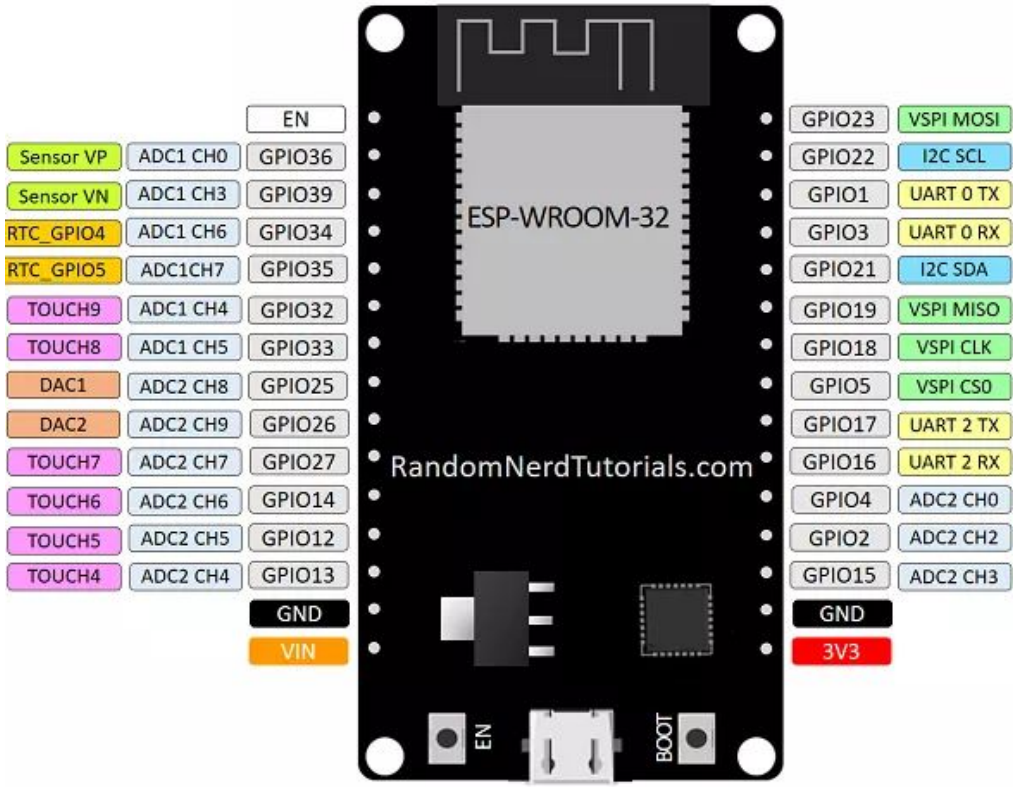
4.3.3 Micropython - ESP32 GPIO

- There are 2 x 8 bits DAC channels on the ESP32 to convert digital signals into analog voltage signal outputs.



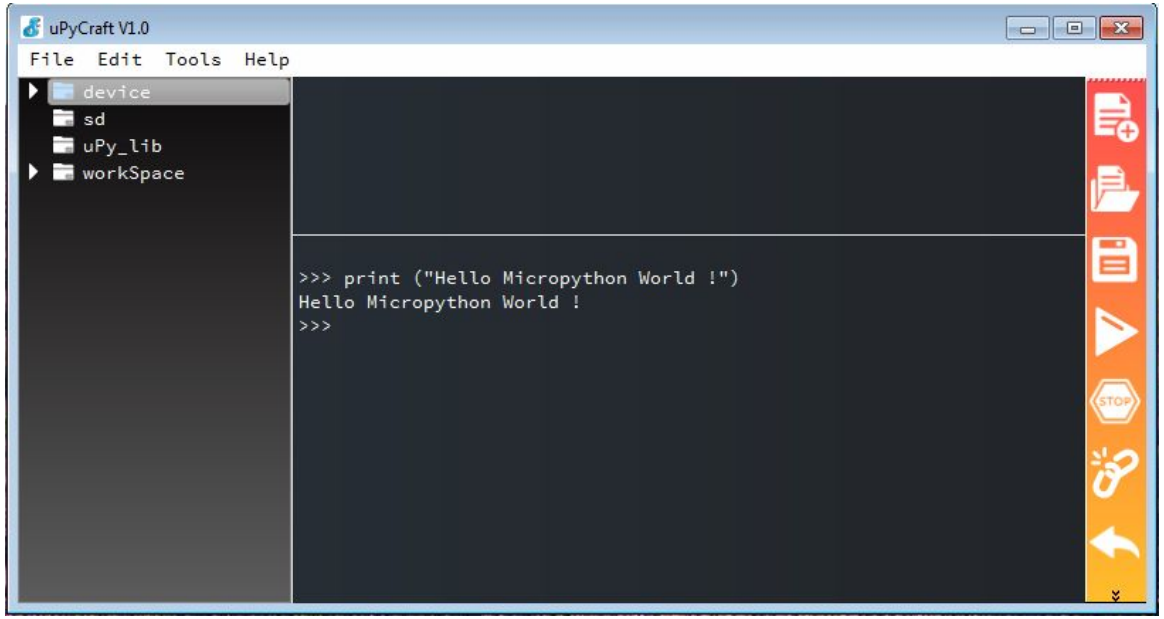
4.3.3 Micropython - ESP32 GPIO

- The ESP32 LED PWM controller has 16 independent channels that can be configured to generate PWM signals with different properties.
- All outputs can be used as PWM pins (GPIOs 34 to 39 can't generate PWM).
- To set a PWM signal, you need to define these parameters in the code:
  - Signal's frequency
  - Duty cycle
  - PWM channel
  - GPIO where you want to output the signal.



4.3.4 Micropython - Let's code

- Let's execute an embedded program in microcontroller,
- We start by using python REPL (read, evaluate, print loop)



4.3.4 Micropython - Let's code

Establishing a communication with the board

- After having the MicroPython firmware installed on your board and having the board connected to your computer through an USB cable, follow the next steps:

1. Go to Tools > Board and select the board you're using.
2. Go to Tools > Port and select the com port your ESP is connected to.
3. Press the Connect button to establish a serial communication with your board.

4. The >>> should appear in the Shell window after a successful connection with your board. You can type the print command to test if it's working:

```
>>> print('Hello')
Hello
>>>
```

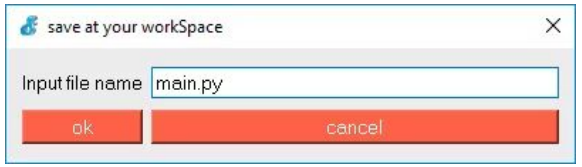
4.3.4 Micropython - Let's code

Creating the main.py file on your board

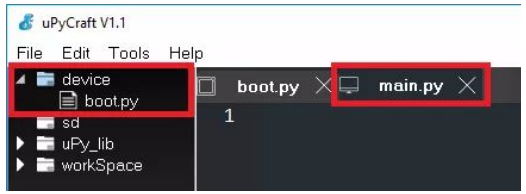
1. Press the "New file" button to create a new file. 

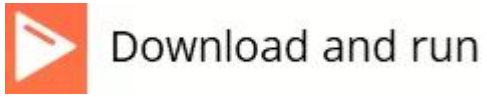
2. Press the "Save file" button to save the file in your computer. 

3. A new window opens, name your file main.py and save it in your computer:



4. After that, you should see the following in your uPyCraft IDE (the boot.py file in your device and a new tab with the main.py file):



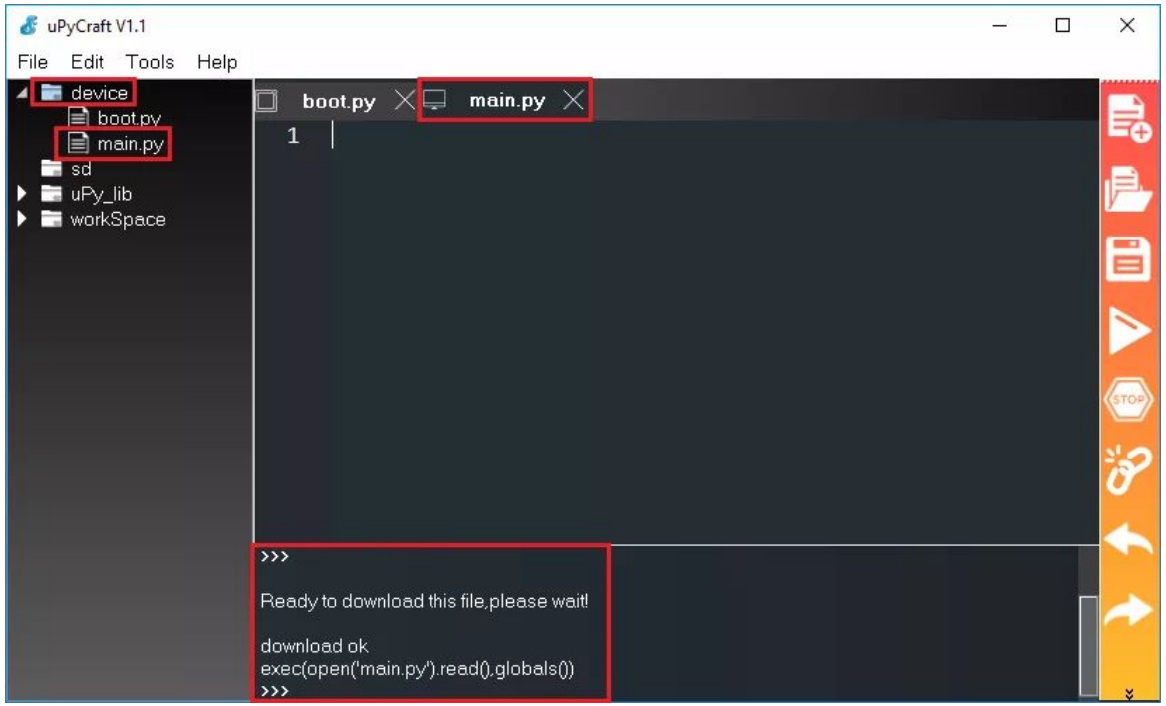
5. Click the "Download and run" button to upload the file to your ESP board: 

6. The device directory should now load the main.py file. Your ESP has the file main.py stored.

4.3.4 Micropython - Let's code

Creating the main.py file on your board - continue

6. The device directory should now load the main.py file. Your ESP has the file main.py stored.



4.3.4 Micropython - Let's code

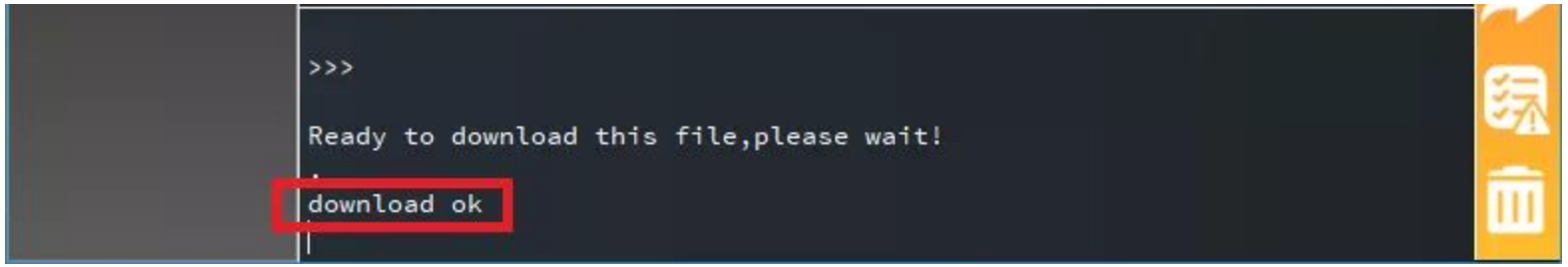
Uploading the blink LED script

1. Code to the Editor on the main.py file.

2. Press the "Stop" button to stop any script from running in your board: Stop

3. Click the "Download and Run button" to upload the script to the ESP32 or ESP8266: Download and run

4. You should see a message saying "download ok" in the Shell window.

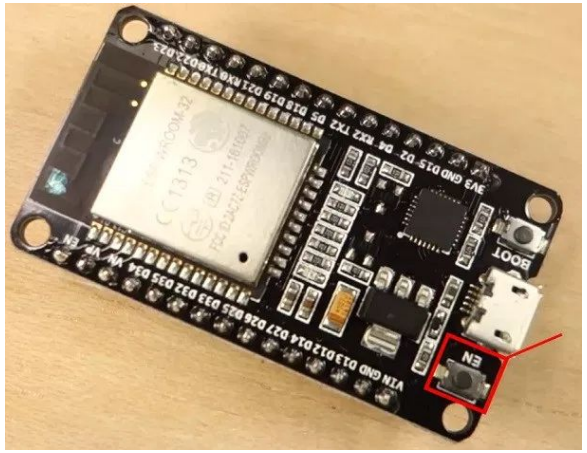


4.3.4 Micropython - Let's code

Testing the script

1. Press the "Stop" button 

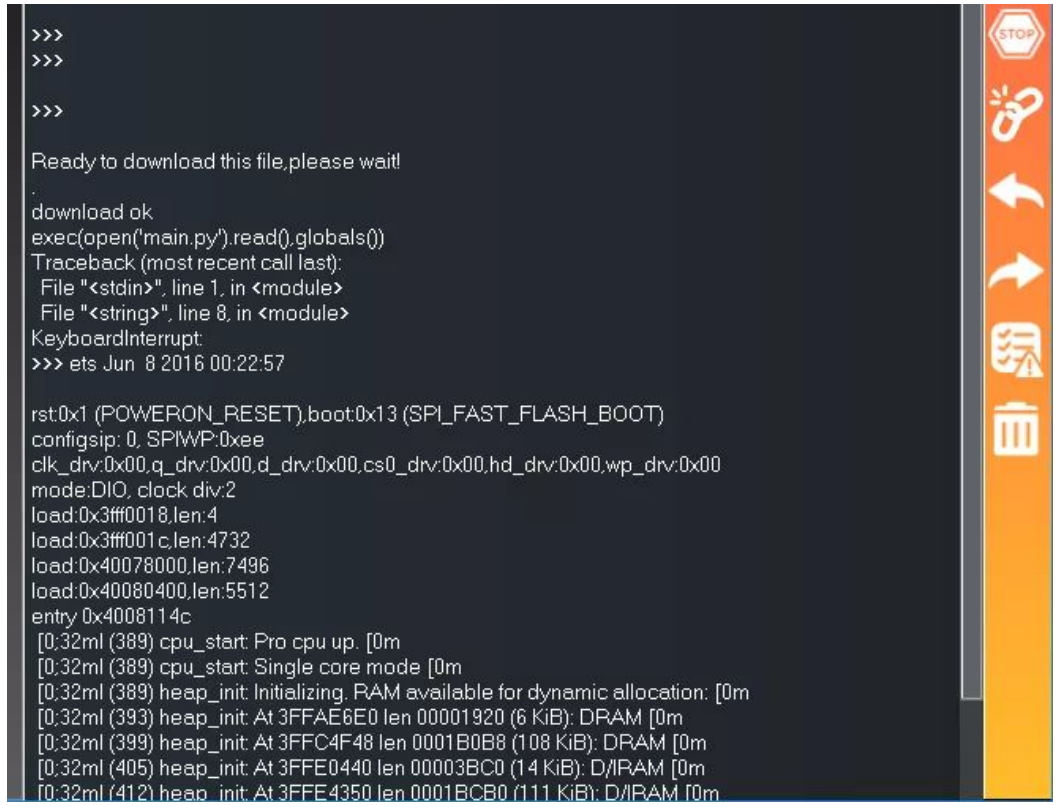
2. Press the on-board ESP32/ESP8266 EN (ENABLE) or RST (RESET) button to restart your board and run the script from the start:



4.3.4 Micropython - Let's code

Testing the script- continue

If you're using an ESP32, your Terminal messages should look something as shown in the following figure after a EN/RST button press:



4.3.5 Micropython - Basic programming using ESP32

- In this article we're going to take a look on how to interact with the ESP32 and ESP8266 GPIOs using MicroPython.
- We'll show you how to read digital and analog inputs, how to control digital outputs and how to generate PWM signals.

Prerequisites

- Install uPyCraft IDE: Windows PC, MacOS X, or Linux Ubuntu
- Flash/Upload MicroPython Firmware to ESP32 and ESP8266

4.3.5 Micropython - Basic programming using ESP32

print() and sleep()

- Import the **sleep** class from the **time** module.

```
from time import sleep
```

- To delay, use sleep(time_in_second). For example:

```
sleep(1)
```

- To print any value use print(val). For example:

```
print("Hello Micropython")
```

4.3.5 Micropython - Basic programming using ESP32

Digital Outputs

- You start by importing the **Pin** class from the **machine** module.

```
from machine import Pin
```

- To set a GPIO on or off, first you need to set it as an output. For example:

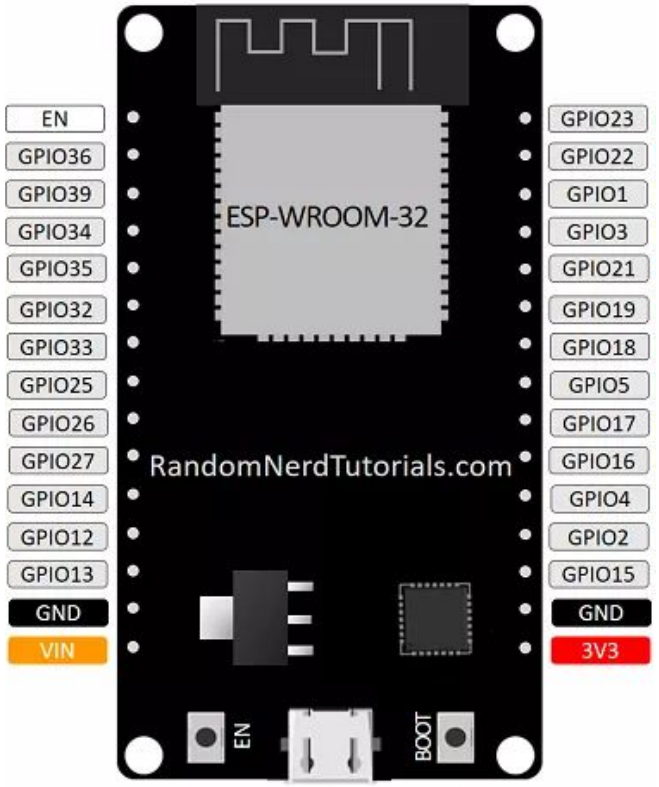
```
led = Pin(5, Pin.OUT)
```

- To control the GPIO, use the value() method on the Pin object and pass 1 or 0 as argument. For example, the following command sets a Pin object (led) to HIGH:

```
led.value(1)
```

- To set the GPIO to LOW, pass 0 as argument:

```
led.value(0)
```

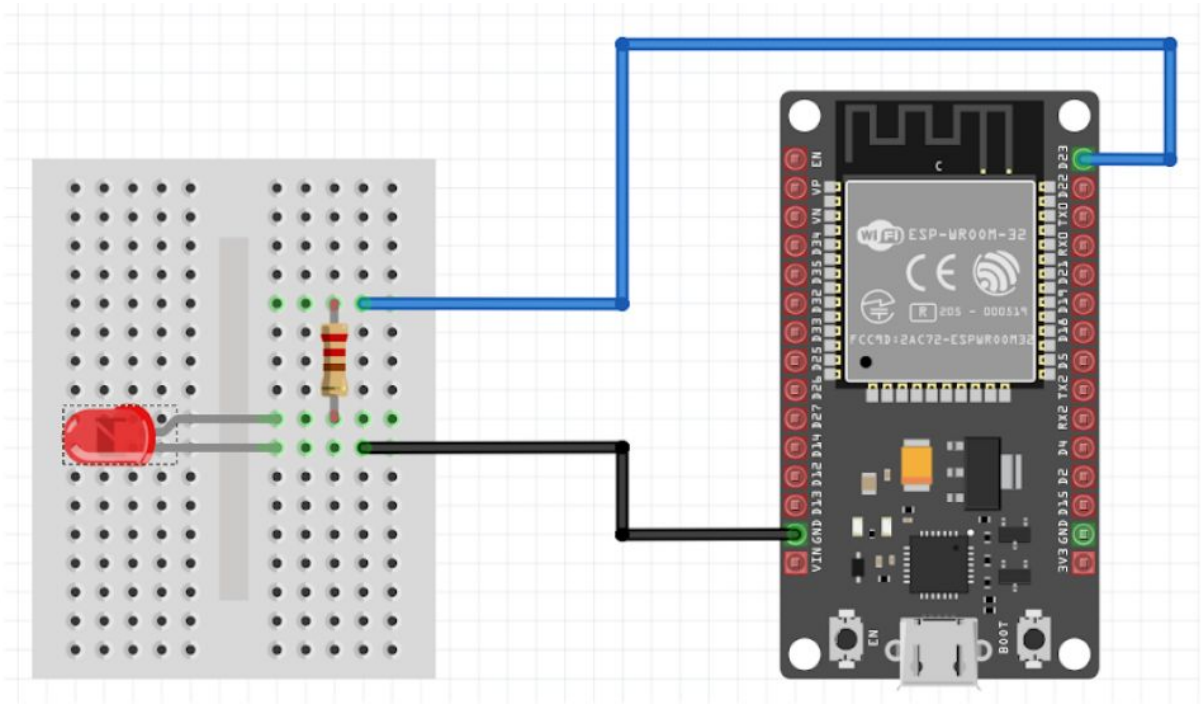


4.3.5 Micropython - Basic programming using ESP32

Blinking LED

What you'll need:

1) LED in any color (1 unit)
2) Resistor 330 Ohm (1 unit)
3) Male to Female jumper wire (2 unit)
4) ESP32 (1 unit)
5) Breadboard (1 unit)
6) USB Micro B cable (1 unit)



Instructions:

The LED will light up for 2 seconds and turn off for 1 second.

4.3.5 Micropython - Basic programming using ESP32

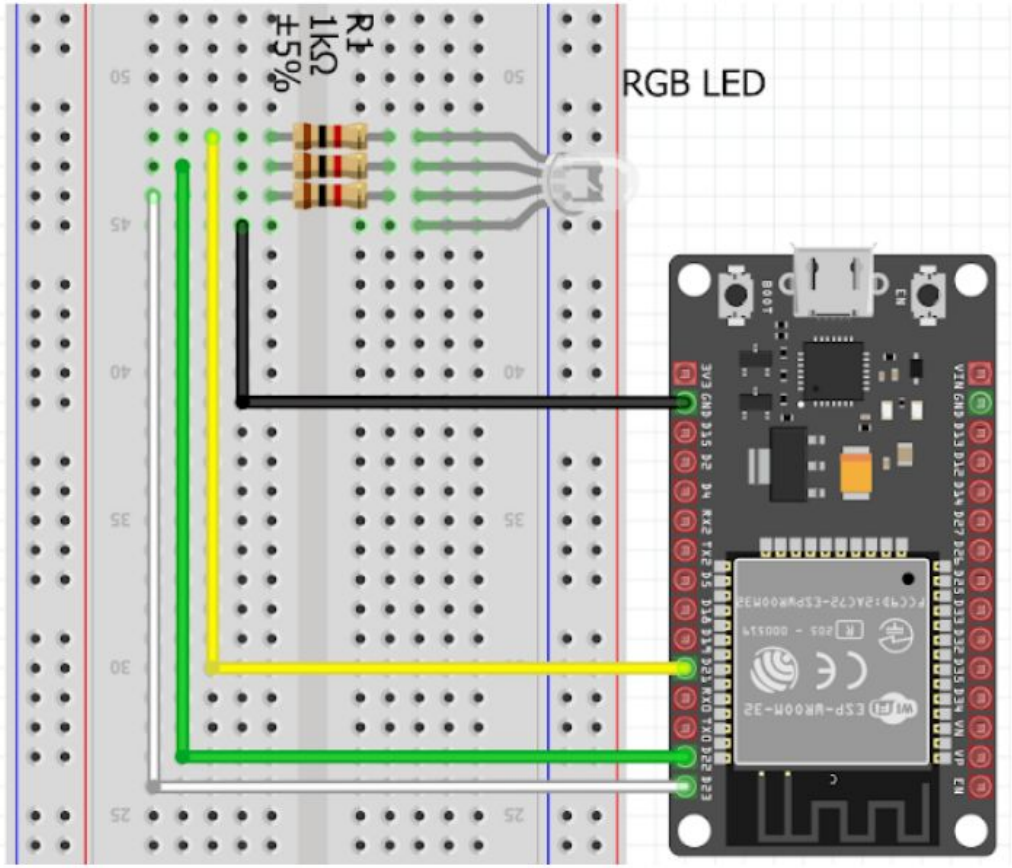
RGB LED Runnning Light

What you'll need:

1) RGB LED (1 unit)
2) Resistor 1K ohm (3 unit)
3) Male to Female jumper wire (5 unit)
4) ESP32 (1 unit)
5) Breadboard (1 unit)
6) USB Micro B cable (1 unit)

Instructions:

The LED will light up in GREEN for 1 second follows by RED and lastly with BLUE



4.3.5 Micropython - Basic programming using ESP32

Digital Inputs

- You start by importing the **Pin** class from the **machine** module.

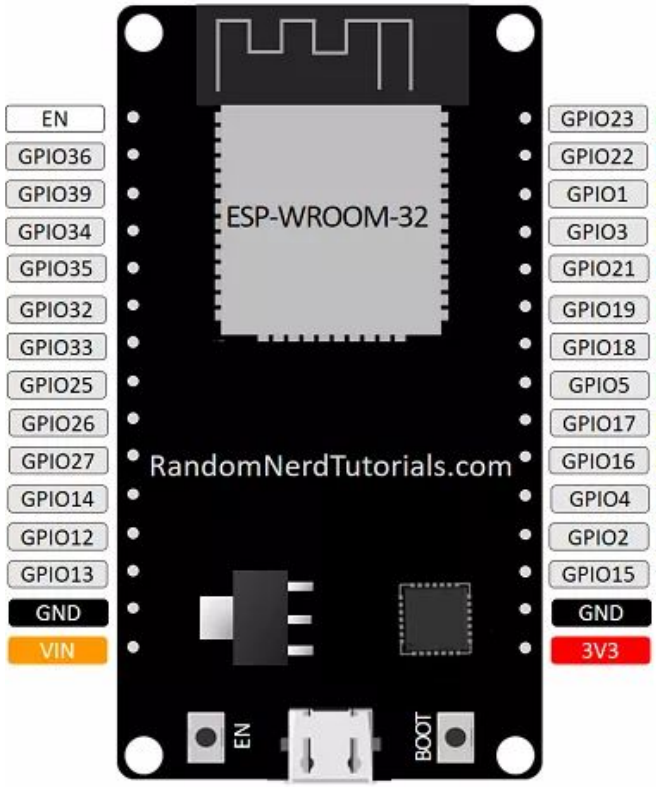
```
from machine import Pin
```

- To get the value of a GPIO, first you need to create a Pin object and set it as an input. For example:

```
button = Pin(4, Pin.IN)
```

- Then, to get is value, you need to use the value() method on the Pin object without passing any argument. For example, to get the state of a Pin object called button, use the following expression:

```
button.value()
```



4.3.5 Micropython - Basic programming using ESP32

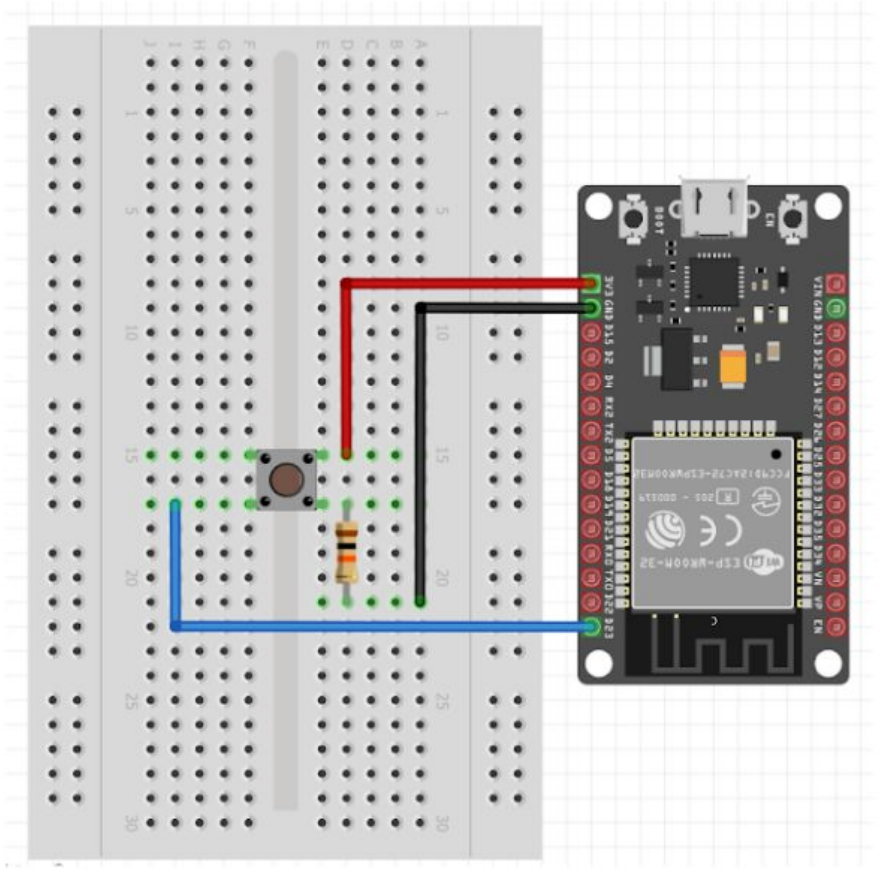
Button - Digital Input

What you'll need:

1) Contact switch or button (1 unit each)
2) Resistor 10K ohm (1 unit)
3) Male to Female jumper wire (3 unit)
4) NodeMCU (1 unit)
5) Breadboard (1 unit)
6) USB Micro B cable (1 unit)

Instructions:

The button will write a value of 1 (ON) when pushed and 0 (OFF) when released.
Show your result at the terminal

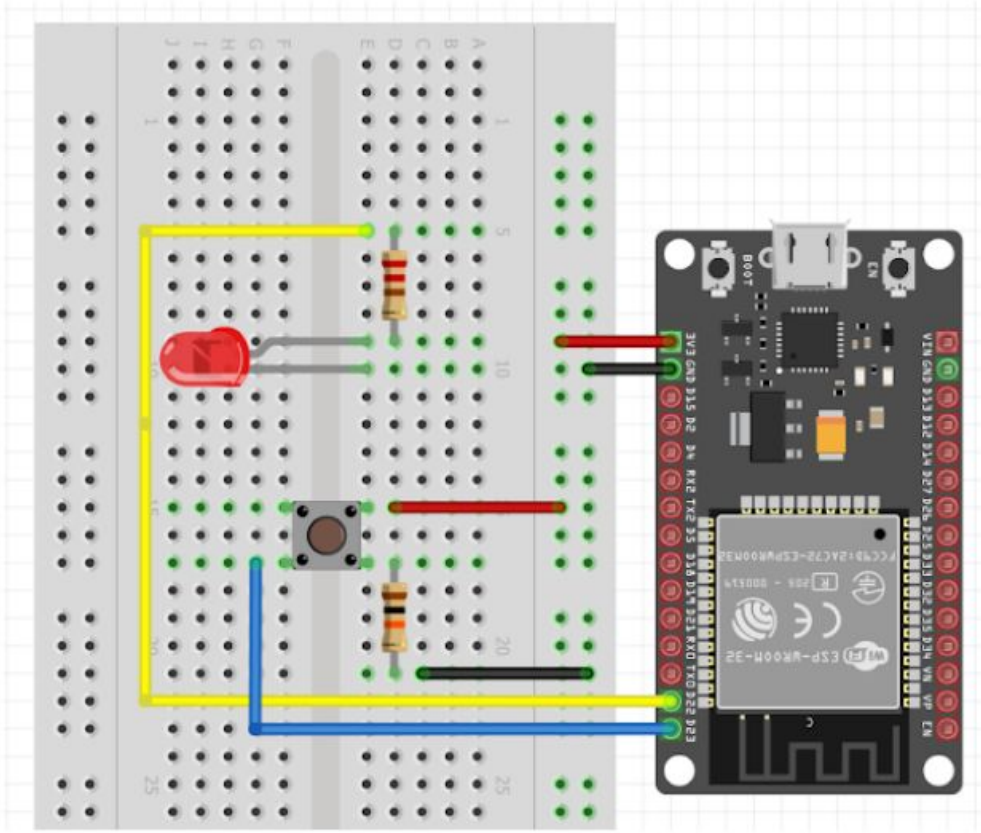


4.3.5 Micropython - Basic programming using ESP32

Digital Input & Digital Output

What you'll need:

1) Contact switch or button (1 unit)
2) LED in any color (1 unit)
3) Resistor 10K ohm (1 unit)
4) Resistor 220 ohm (1 unit)
5) Male to Female jumper wire (3 unit)
6) ESP32 (1 unit)
7) Breadboard (1 unit)
8) USB Micro B cable (1 unit)



Instructions:

The button will write a value of 1 (ON) when pushed and will also lights up the LED

4.3.5 Micropython - Basic programming using ESP32

DHT Sensor

- Import the **DHT22** class from the **dht** module.

```
from dht import DHT11
```

- To get the value of a DHT sensor, first you need to create a DHT11 object and set at which pin it connected. For example:

```
sensor = DHT11(Pin(2))
```

- Then, to get is value, you need to use the temperature() or humidity() method on the DHT11 object without passing any argument. For example, to get the reading of a DHT11 object called sensor, use the following expression:

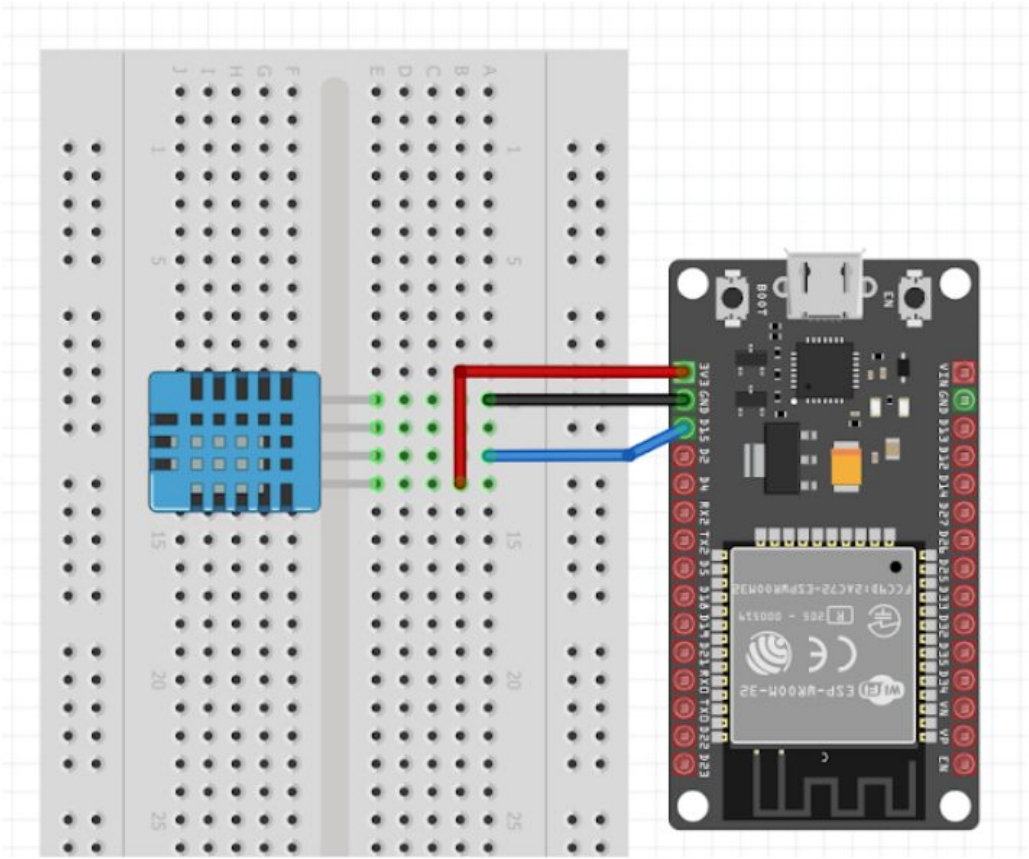
```
sensor.temperature() or sensor.humidity()
```

4.3.5 Micropython - Basic programming using ESP32

DHT Sensor

What you'll need:

1) DHT11 (1 unit)
2) Male to Female jumper wire (3 unit)
3) ESP32 (1 unit)
4) Breadboard (1 unit)
5) USB Micro B cable (1 unit)



Instructions:

DHT11 will read room temperature and result can be seen on terminal

4.3.5 Micropython - Basic programming using ESP32

Analog Readings – ESP32

- There are several pins on the ESP32 that can act as analog pins – these are called ADC pins. All the following GPIOs can act as ADC pins: 0, 2, 4, 12, 13, 14, 15, 25, 26, 27, 32, 33, 34, 35, 36, and 39.
- ESP32 ADC pins have 12-bit resolution by default. These pins read voltage between 0 and 3.3V and then return a value between 0 and 4095. The resolution can be changed on the code.

4.3.5 Micropython - Basic programming using ESP32

Analog Readings – ESP32

- To read analog inputs, import the ADC class in addition to the Pin class from the machine module.

```
from machine import Pin, ADC
```

- Then, create an ADC object called pot on GPIO 34.

```
pot = ADC(Pin(34))
```

- The following line defines that we want to be able to read voltage in full range. This means we want to read voltage from 0 to 3.3V.

```
pot.atten(ADC.ATTN_11DB)
```

- Read the pot value and save it in the pot_value variable. To read the value from the pot, simply use the read() method on the pot object.

```
pot_value = pot.read()
```

4.3.5 Micropython - Basic programming using ESP32

Analog Readings – ESP32

- The atten() method can take the following arguments:
  - ADC.ATTN_0DB — the full range voltage: 1.2V
  - ADC.ATTN_2_5DB — the full range voltage: 1.5V
  - ADC.ATTN_6DB — the full range voltage: 2.0V
  - ADC.ATTN_11DB — the full range voltage: 3.3V

Analog Readings – ESP32

- When you rotate the potentiometer, you get values from 0 to 4095 – that's because the ADC pins have a 12-bit resolution by default. You may want to get values in other ranges. You can change the resolution using the width() method as follows:
- The bit argument can be one of the following parameters:
  - ADC.WIDTH_9BIT: range 0 to 511
  - ADC.WIDTH_10BIT: range 0 to 1023
  - ADC.WIDTH_11BIT: range 0 to 2047
  - ADC.WIDTH_12BIT: range 0 to 4095

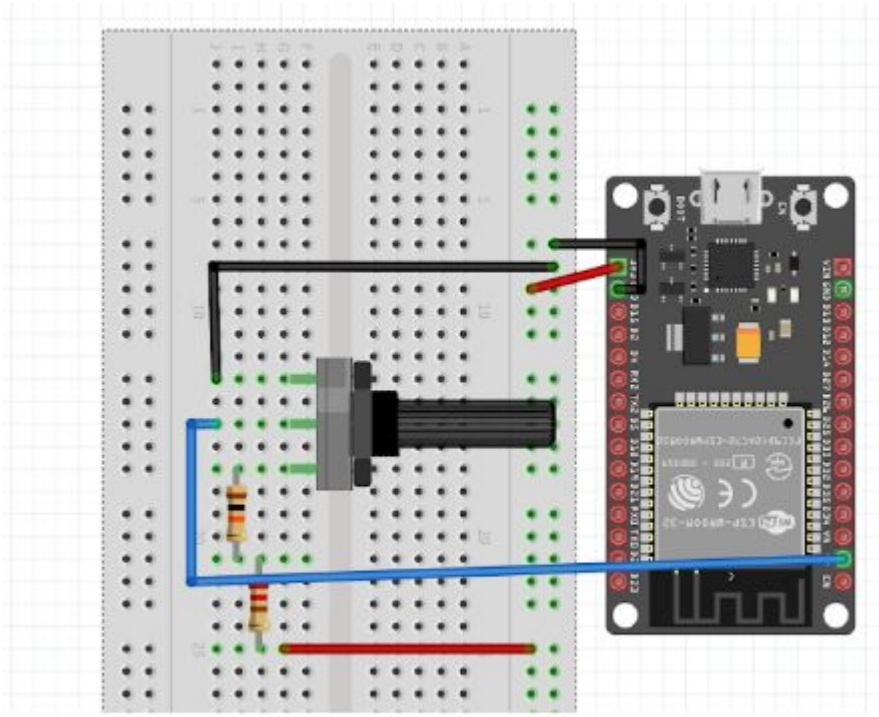
```
ADC.width(ADC.WIDTH_12BIT)
```

4.3.5 Micropython - Basic programming using ESP32

Analog Input

What you'll need:

1) Potentiometer/Variable Resistor (1 unit)
2) Male to Female jumper wire (3 unit)
3) ESP32 (1 unit)
4) Breadboard (1 unit)
5) USB Micro B cable (1 unit)



Instructions:

Analog reading can be seen on terminal

4.3.5 Micropython - Basic programming using ESP32

PWM – ESP32

- To create a PWM pin, import the PWM class in addition to the Pin class from the machine module.

```
from machine import Pin, ADC, PWM
```

- Then, create a PWM object called led.

```
led = PWM(Pin(5), frequency)
```

- To create a PWM object, you need to pass as parameters, the pin it is connected to, the signal frequency and the duty cycle.
  - Frequency: The frequency can be a value between 0 and 78125. A frequency of 5000 Hz can be used to control the LED brightness.
  - Duty cycle: The duty cycle can be a value between 0 and 1023. In which 1023 corresponds to 100% duty cycle (full brightness), and 0 corresponds to 0% duty cycle (unlit LED).

4.3.5 Micropython - Basic programming using ESP32

PWM – ESP32

- We'll just set the duty cycle on the while loop, so we don't need to pass the duty cycle parameter. If you don't set the duty cycle when instantiating the PWM object, it will be 0 by default.
- To set the duty cycle use the duty() method on the PWM object and pass the duty cycle as an argument:

```
led.duty(duty_cycle)
```

- Inside the while loop, we create a for loop that increases the duty cycle by 1 in each loop with an interval of 5 ms between each change.

```
for duty_cycle in range(0, 1024):
led.duty(duty_cycle)
sleep(0.005)
```

4.3.5 Micropython - Basic programming using ESP32

PWM – ESP32

- The range() function has the following syntax:

```
range(start, stop, step)
```

  - Start: a number that specifies at which position to start. We want to start with 0 duty cycle;
  - Stop: a number that specifies at which position we want to stop, excluding that value. The maximum duty cycle is 1023, because we are incrementing 1 in each loop, the last value should be 1023+1. So, we'll use 1024.
  - Step: an integer number that specifies the incrementation. By default, incrementation is 1.

- In each for loop, we set the LED's duty cycle to the current duty_cycle value:

```
led.duty(duty_cycle)
```

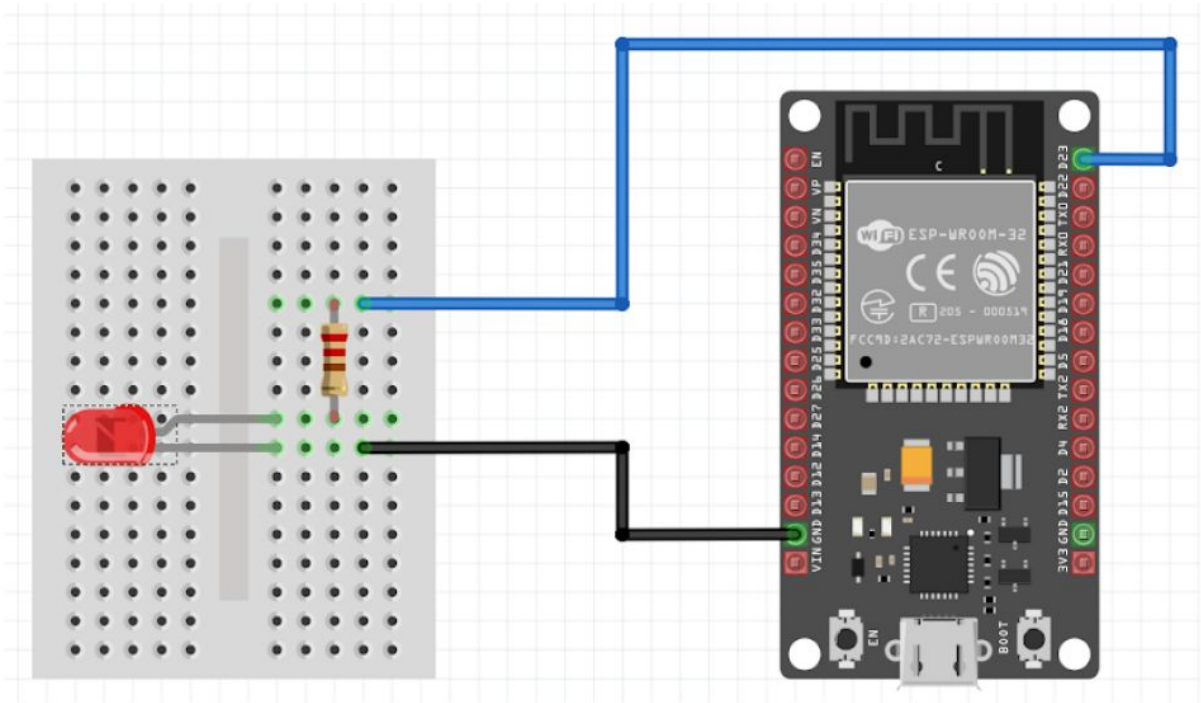
- After that, the duty_cycle variable is incremented by 1.

4.3.5 Micropython - Basic programming using ESP32

PWM

What you'll need:

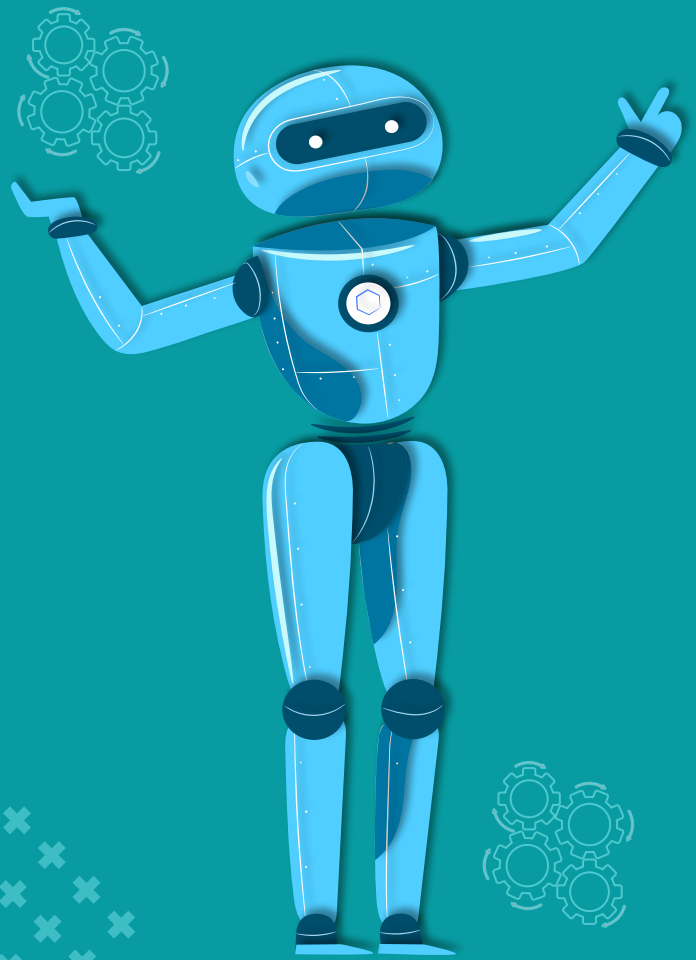
1) LED (1 unit)
2) Male to Female jumper wire (3 unit)
3) ESP32 (1 unit)
4) Breadboard (1 unit)
5) potentiometer
6) USB Micro B cable (1 unit)



Instructions:

Dim the brightness of an LED by changing the duty cycle over time.

4.4



Connecting IoT Devices to Network

- MicroPython - Connecting to a WiFi Network
- MicroPython - Summary - ESP as Client
- MicroPython - Summary - ESP as Access Point

4.4 Connecting IoT Devices to Network

4.4.1 MicroPython - Connecting to a WiFi Network

- There are module in MicroPython used to connect ESP32 to network based on MicroPython 1.12 Libraries.
- Modules that are related is network module and usocket module.
- Network module provides network drivers and routing configuration.
- To use this module, a MicroPython variant/build with network capabilities must be installed.
- Network drivers for specific hardware are available within this module and are used to configure hardware network interface(s).
- Network services provided by configured interfaces are then available for use via the usocket module.

4.4 Connecting IoT Devices to Network

4.4.1 MicroPython - Connecting to a WiFi Network

- First of all, we will need to import the network module, in order to access all the functions needed for establishing the connection to the WiFi Network.

```
import network
```

- Since we are going to connect to a WiFi network, our device will operate in station/client mode. So, we need to create an instance of the station WiFi interface.
- To do it, we just need to call the constructor of the WLAN class and pass as its input the identifier of the interface we want. In this case, we will use the network.STA_IF interface.

```
station = network.WLAN(network.STA_IF)
```

- Now we will activate the network interface by calling the active method on our station object and passing True as input, since it accepts Boolean values.

```
station.active(True)
```

4.4.1 MicroPython - Connecting to a WiFi Network

- Again, after executing this command, you should get an output on the command line, indicating that we are in station mode and that the interface was started.
- Finally we will use the connect method to connect to the WiFi network. This method receives as input both the SSID (network name) and the password.

```
station.connect("YourNetworkName", "YourNetworkPassword")
```

- It will again print some information to the console. Take in consideration that the connection may take a while. Also note that the ">>>>" is not printed once the connection is established, so it may seem that the device is still processing after the last message is printed, as shown in figure 3. You can hit enter to continue normally.

- Finally, we will confirm the connection by calling the isconnected method, which returns true if the device is connected to a WiFi network [2]. We are also going to call the ifconfig method, which returns the IP address, subnet mask, gateway and DNS as output parameters [2].

```
station.isconnected()
station.ifconfig()
```

4.4.2 MicroPython - Summary - ESP as Client

Summary - ESP as Client.

- Create the interface.

```
wlan = network.WLAN(network.STA_IF)
```

- Activate the interface.

```
wlan.active(True)
```

- Scan for access points.

```
wlan.scan()
```

- Check if the station is connected to an AP.

```
wlan.isconnected()
```

- Connect to an AP.

```
wlan.connect('essid', 'password')
```

- Get the interface's MAC address.

```
wlan.config('mac')
```

- Get the interface's IP/netmask/gw/DNS addresses.

```
wlan.ifconfig()
```

- Retrieve the Receive Signal Strength Indicator(RSSI) of the AP signal.

```
wlan.status('rssi')
```

4.4.3 MicroPython - Summary - ESP as Access Point

Summary - ESP as Access Point.

- Create the interface.

```
ap = network.WLAN(network.AP_IF)
```

- Set the ESSID of the access point.

```
ap.config(essid='AP',authmode=network.
AUTH_WPA_WPA2_PSK, password =’12345678’)
```

- Activate the interface.

```
ap.active(True)
```

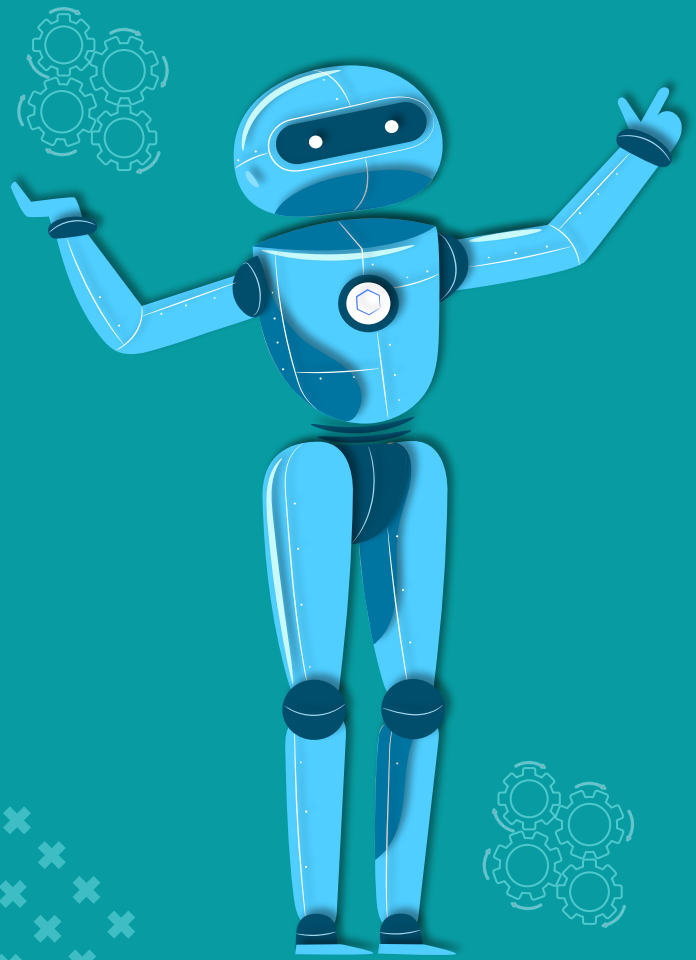
- Set how many clients can connect to the network.

```
ap.config(max_clients=10)
```

- Retrieve a list of all STAs connected to the AP. The list contains tuples of the form (MAC,RSSI).

```
ap.status(‘stations’)
```

4.5



Data Transport Protocol for IoT

- MQTT
- HTTP-REST

4.5 Data Transport Protocol for IoT

4.5.1 MQTT

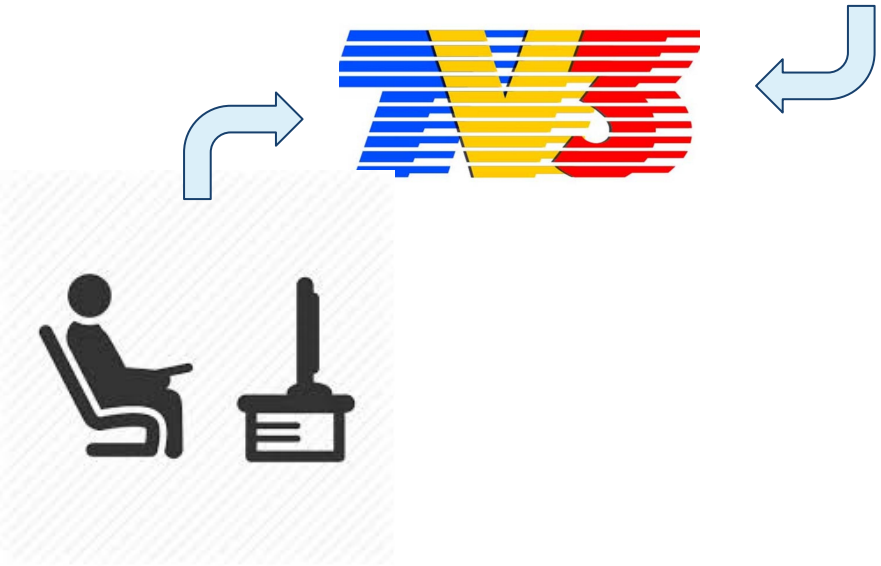
- MQTT stands for **Message Queuing Telemetry Transport** and it is a machine-to-machine (M2M) or Internet of Things connectivity protocol.
- It was designed as an extremely lightweight publish/subscribe messaging transport.
- It is useful for connections with remote locations where a small code footprint is requires and/or network bandwidth is at a premium.

4.5 Data Transport Protocol for IoT

4.5.1 MQTT

A useful analogy is TV or radio.

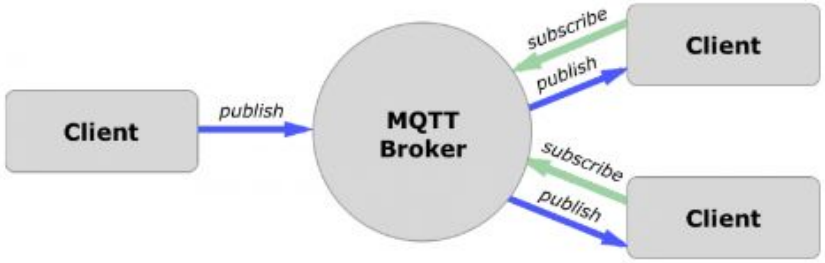
- A TV broadcaster broadcasts a TV program using a specific channel and a viewer tunes into this channel to view the broadcast.
- There is no direct connection between the broadcaster and the viewer.
- In MQTT a publisher publishes messages on a topic and a subscriber must subscribe to that topic to view the message.

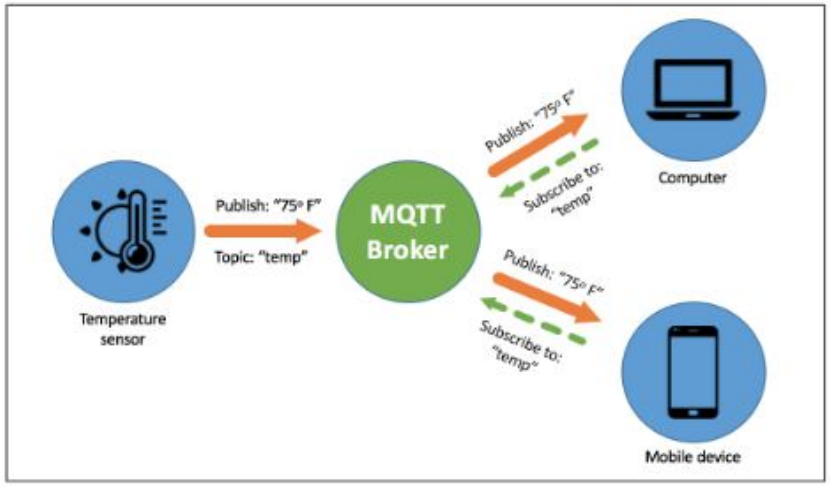


4.5 Data Transport Protocol for IoT

4.5.1 MQTT

- MQTT is a messaging protocol i.e it was designed for transferring messages, and uses a publish and subscribe model.
- This model makes it possible to send messages to 0,1 or multiple clients.
- MQTT requires the use of a central Broker.





Something to remember

➔ When a device (client) wants to send data to the broker, we call this operations as “publish”.

➔ When a device (client) wants to receive data from the broker, we call this operations as “subscribe”

4.5 Data Transport Protocol for IoT

4.5.1 MQTT

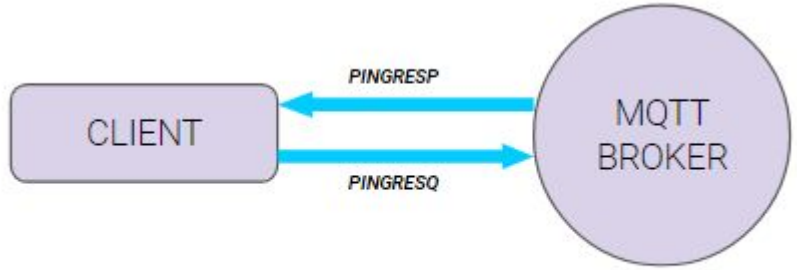
Important Points to Note

- Clients do not have addresses like in email systems, and messages are not sent to clients.
- Messages are published to a broker on a topic.
- The job of an MQTT broker is to filter messages based on topic, and then distribute them to subscribers.
- A client can receive these messages by subscribing to that topic on the same broker
- There is no direct connection between a publisher and subscriber.
- All clients can publish (broadcast) and subscribe (receive).
- MQTT brokers do not normally store messages.

4.5 Data Transport Protocol for IoT

4.5.1 MQTT - Client-Broker Connection

- MQTT uses TCP/IP to connect to the broker.
- Most MQTT clients will connect to the broker and remain connected even if they aren't sending data.
- Connections are acknowledged by the broker using a connection acknowledgement message.
- MQTT clients publish a keepalive message at regular intervals (usually 60 seconds) which tells the broker that the client is still connected.



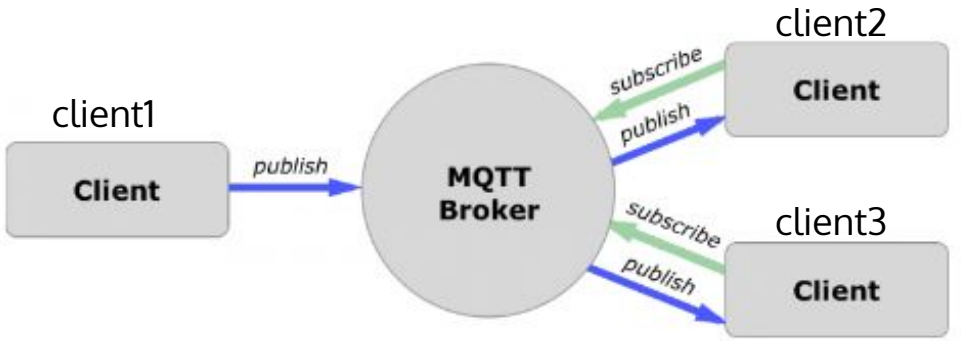
MQTT Keep Alive Interval

4.5 Data Transport Protocol for IoT

4.5.1 MQTT - Client-Broker Connection

Client

- All clients are required to have a client name or ID.
- The client name is used by the MQTT broker to track subscriptions etc.
- Client names must also be unique.
- If you attempt to connect to an MQTT broker with the same name as an existing client then the existing client connection is dropped.



4.5.1 MQTT - Client-Broker Connection

Last Will Messages

- The idea of the last will message is to notify a subscriber that the publisher is unavailable due to network outage.
- The last will message is set by the publishing client, and is set on a per topic basis which means that each topic can have its own last will message.
- This means that each topic cab have its own last will message associated with it.
- The message is stored on the broker and sent to any subscribing client (to that topic) if the connection to the publisher fails.
- If the publisher disconnects normally the last Will Message is not sent.
- The actual will messages is including with the connect request message.

4.5 Data Transport Protocol for IoT

4.5.1 MQTT - Topic

Understanding MQTT Topic

- MQTT topics are a form of addressing that allows MQTT clients to share information.
- MQTT Topics are structured in a hierarchy similar to folders and files in a file system using the forward slash (/)as a delimiter.
- Using this system you can create a user friendly and self descriptive naming structures of you own choosing.
- Topic names are:
    - Case sensitive
    - use UTF-8 strings.
    - Must consist of at least one character to be valid.

4.5 Data Transport Protocol for IoT

4.5.1 MQTT - Topic

Subscribing to Topic

- A client can subscribe to individual or multiple topics.
- When subscribing to multiple topics two wildcard characters can be used. They are:
  - # (hash character) – multi level wildcard
  - + (plus character) -single level wildcard
- Wildcards can only be used to denote a level or multi-levels i.e /house/# and not as part of the name to denote multiple characters e.g. hou# is not valid

4.5 Data Transport Protocol for IoT

4.5.1 MQTT - Topic naming Examples

Valid Topic Subscription

Single topic subscriptions

- ➔ /
- ➔ /house
- ➔ house/room/main-light
- ➔ house/room/side-light

Using Wildcards

Subscribing to topic house/#, cover

- ➔ house/room1/main-light
- ➔ house/room1/alarm
- ➔ house/garage/main-light
- ➔ house/main-door

Using Wildcards

Subscribing to topic house/+/main-light, covers

- ➔ house/room1/main-light
- ➔ house/room2/main-light
- ➔ house/garage/main-light

but doesn't cover

- ➔ house/room1/side-light
- ➔ house/room2/side-light

4.5 Data Transport Protocol for IoT

4.5.1 MQTT - Topic

Publishing to Topic

- A client can only publish to an individual topic. That is, using wildcards when publishing is not allowed.
- E.G- To publish a message to two topics you need to publish the message twice

When are Topics Created

- Someone subscribes to a topic
- Someone publishes a message to a topic with the retained message set to True.

When are Topics Removed from a Broker

- When the last client that is subscribing to that broker disconnects, and clean session is true.
- When a client connects with clean session set to True.

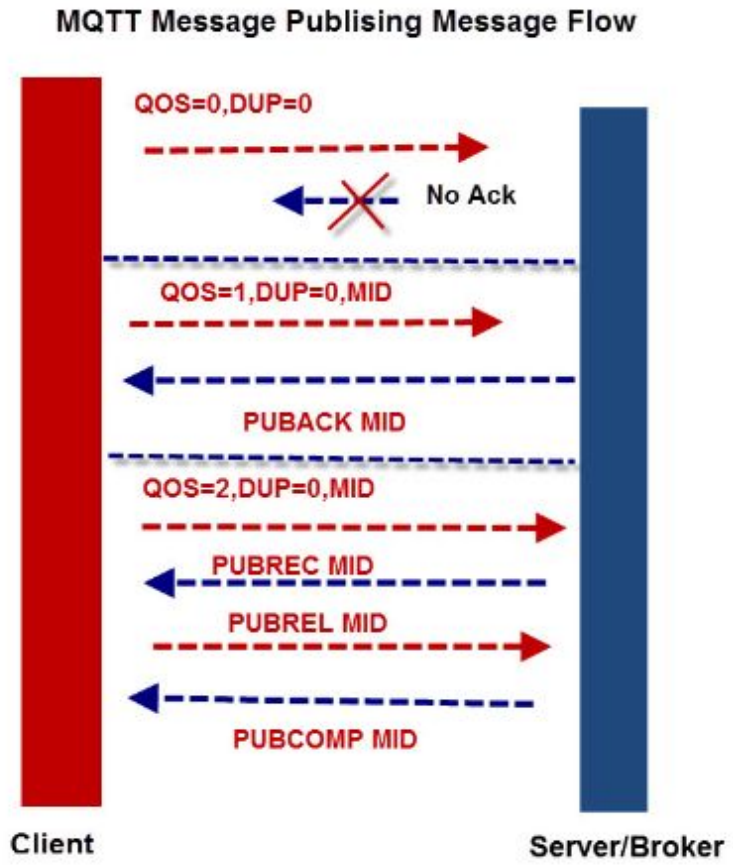
4.5 Data Transport Protocol for IoT

4.5.1 MQTT - Message Flow and QOS on Published Messages

MQTT supports 3 QOS levels 0,1,2.

- QOS -0 – Default and doesn't guarantee message delivery.
- QOS -1 – Guarantees message delivery but could get duplicates.
- QOS -2 -Guarantees message delivery with no duplicates.

A message is published using one of these levels with QOS level 0 being the default.



4.5 Data Transport Protocol for IoT

4.5.1 MQTT - Public Broker

Established MQTT Broker

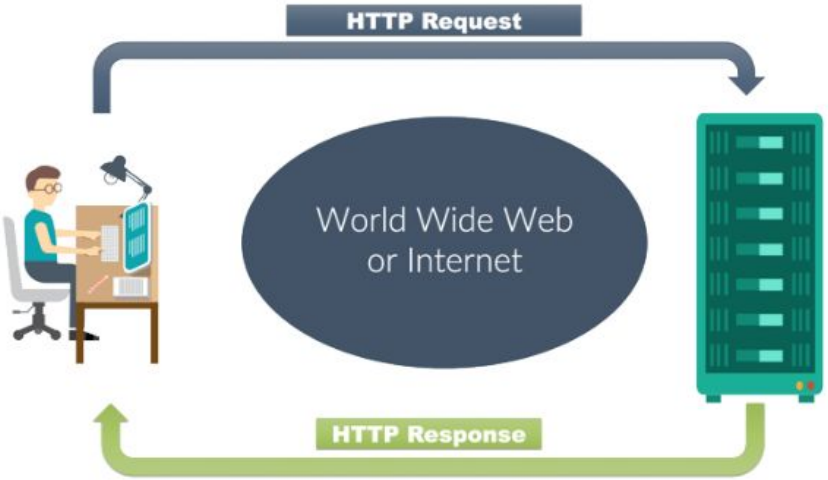
- Adafruit (https://io.adafruit.com/)
- ThingsBoard (https://thingsboard.io/)
- ThingSpeak (https://thingspeak.com/channels/public)
- CloudMQTT (https://www.cloudmqtt.com/)

4.5 Data Transport Protocol for IoT

4.5.2 HTTP - REST

HTTP Request & HTTP Response

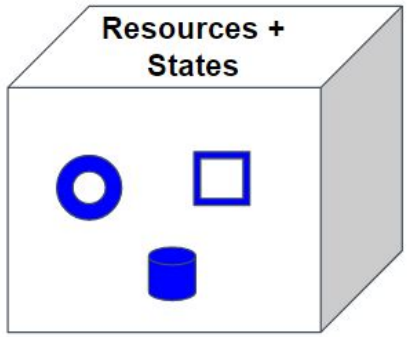
- HTTP (Hypertext Transfer Protocol) is perhaps the most popular application protocol used in the Internet (or The WEB).
- HTTP is an asymmetric request-response client-server protocol as illustrated. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message. In other words, HTTP is a pull protocol, the client pulls information from the server (instead of server pushes information down to the client).



4.5 Data Transport Protocol for IoT

4.5.2 HTTP - REST

REST stands for REpresentational State Transfer



It means when a RESTful API is called, the server will transfer to the client a representation of the state of the requested resource.

HTTP : REQUEST / RESPONSE
HTTP: METHODS (VERBS)
RESOURCE'S IDENTIFIER:
ENDPOINTS

API

4.5 Data Transport Protocol for IoT

4.5.2 HTTP - REST

HTTP Header

- **HTTP Request header fields** are components of the header section of request and response messages in the Hypertext Transfer Protocol (HTTP). They define the operating parameters of an HTTP transaction.

Common parameters for http request header are:

```
Accept: application/json
Authorization: Basic 34i3j4iom2323==
Cache-Control: no-cache
Content-Type:application/json
```

4.5 Data Transport Protocol for IoT

4.5.2 HTTP - REST Verb

GET

- The HTTP GET method is used to **read** (or retrieve) a representation of a resource. In the “happy” (or non-error) path, GET returns a representation in XML or JSON and an HTTP response code of 200 (OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).

According to the design of the HTTP specification, GET (along with HEAD) requests are used only to read data and not change it. Therefore, when used this way, they are considered safe. That is, they can be called without risk of data modification or corruption—calling it once has the same effect as calling it 10 times, or none at all. Additionally, GET (and HEAD) is idempotent, which means that making multiple identical requests ends up having the same result as a single request.

4.5.2 HTTP - REST Verb

POST

- The POST verb is most-often utilized to **create** new resources. In particular, it's used to create subordinate resources. That is, subordinate to some other (e.g. parent) resource. In other words, when creating a new resource, POST to the parent and the service takes care of associating the new resource with the parent, assigning an ID (new resource URI), etc.

On successful creation, return HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status.POST is neither safe nor idempotent. It is therefore recommended for non-idempotent resource requests. Making two identical POST requests will most-likely result in two resources containing the same information.

4.5.2 HTTP - REST Verb

PUT

- PUT is most-often utilized for **update** capabilities, PUT-ing to a known resource URI with the request body containing the newly-updated representation of the original resource.
- However, PUT can also be used to create a resource in the case where the resource ID is chosen by the client instead of by the server. In other words, if the PUT is to a URI that contains the value of a non-existent resource ID.

PUT for create, return HTTP status 201 on successful creation. A body in the response is optional—providing one consumes more bandwidth. It is not necessary to return a link via a Location header in the creation case since the client already set the resource ID.

PUT is not a safe operation, in that it modifies (or creates) state on the server, but it is idempotent. In other words, if you create or update a resource using PUT and then make that same call again, the resource is still there and still has the same state as it did with the first call.

4.5.2 HTTP - REST Verb

HTTP PATCH

- HTTP PATCH requests are to make partial update on a resource. If you see PUT requests also modify a resource entity so to make more clear – PATCH method is the correct choice for partially updating an existing resource and PUT should only be used if you're replacing a resource in its entirety.

4.5.2 HTTP - REST Verb

DELETE

- As the name applies, DELETE APIs are used to delete resources (identified by the Request-URI).
- A successful response of DELETE requests SHOULD be HTTP response code 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has been queued, or 204 (No Content) if the action has been performed but the response does not include an entity.
- DELETE operations are idempotent. If you DELETE a resource, it's removed from the collection of resource. Repeatedly calling DELETE API on that resource will not change the outcome – however calling DELETE on a resource a second time will return a 404 (NOT FOUND) since it was already removed. Some may argue that it makes DELETE method non-idempotent. It's a matter of discussion and personal opinion.

4.5 Data Transport Protocol for IoT

4.5.2 HTTP - REST Endpoint

Endpoint Example

- REST endpoint are like other standard web URL , URI

http://myinfluxdb.net:8086/write

http://myserver.net/mydata

REST service usually responses with data in ***JSON*** format.

Example JSON is:

```
{ "name":"Afif", "age":30, "car":"Produa" }
```

JavaScript example using JSON data

```
> af = { "name":"Afif", "age":30, "car":"Produa"
  };
<· ▶{name: "Afif", age: 30, car: "Produa"}
> af.name
<· "Afif"
> af.car
<· "Produa"
```

Trivia

JSON stands for JavaScript Object Notation a data type JavaScript compiler treats as object.

4.5 Data Transport Protocol for IoT

4.5.2 HTTP - REST Verb

Custom Server Processing Using REST Library or Framework

- There are many REST library available to create RESTful service that can process IoT data. They are available in many programming language such as JAVA, PHP, Go, Ruby, Python etc...
- Using these library can make your application more flexible, robust and tailored made to exactly what you need it to do.

```
rom flask import Flask
from flask import jsonify
from flask import request
from influxdb import InfluxDBClient
client = InfluxDBClient(host='192.168.150.219',
port=8086)
client.switch_database('timedemo')
class fields(object):
    @property
    def info(self):
        return self.__dict__

class tags(object):
    @property
    def info(self):
        return self.__dict__
class point(object):
    def __init__(self):
        self.measurement = "measure"
        self._fields = fields()
        self._tags = tags()

    def info(self):
        p = {}
        p["measurement"] = self.measurement
        p["fields"] = self._fields.__dict__
        p["tags"] = self._tags.__dict__
        return p
app = Flask(__name__)
```

```
@app.route('/write/<device>/<location>/<me
asurement>/<param>/<unit>',
methods=['POST'])
def
writepoint(device,location,measurement,para
m,unit):
    data = float(request.data)
    p = point()
    p.measurement = measurement
    p._fields.__setattr__(param, data)
    p._fields.__setattr__('unit',unit)
    p._tags.__setattr__('location',location)
    p._tags.__setattr__('device',device)
    client.write_points([p.info()])
    return f"saved data:{str(p.info())}"

if __name__ == '__main__':
   app.run(host="0.0.0.0",debug=False)
```

Example REST server code in python using Flask framework.