IoT Workshop

with ESP 8266 and Lua

Chathura De Silva
Kutila Gunasekera
Randika Pathirana
Department of Computer Science and Engineering, University of Moratuwa

Internet of Things

The Internet

Things

- Computers
- Mobile phones
- Refrigerators
- TVs
- Vehicles
- Sensors and actuators
- Clothes
- Food
- Medicine
- Books
- People
- Animals
- ...

Internet of Things (IoT) is…

A computing concept where all things, including every **physical objects**, can be **connected**, making those objects intelligent, **programmable**, and capable of interacting with humans.

(Source: IEEE-IoT portal)

Internet of Things (IoT) is…

A computing concept where all things, including every **physical objects**, can be **connected**, making those objects intelligent, **programmable**, and capable of interacting with humans. (Source: IEEE-IoT portal)

The network of physical objects that contain **embedded technology** to **communicate** and **sense** or **interact** with their internal states or the external **environment**. (Source: Gartner)

A network of connected devices which communicate over the Internet, and they do so autonomously, **machine to machine**, without the need for human intervention. (Source: Prof. Mischa Dohler, KCL)

Internet of Things (IoT) is…

A computing concept where all things, including every **physical objects**, can be **connected**, making those objects intelligent, **programmable**, and capable of interacting with humans. (Source: IEEE-IoT portal)

The network of physical objects that contain **embedded technology** to **communicate** and **sense** or **interact** with their internal states or the external **environment**. (Source: Gartner)

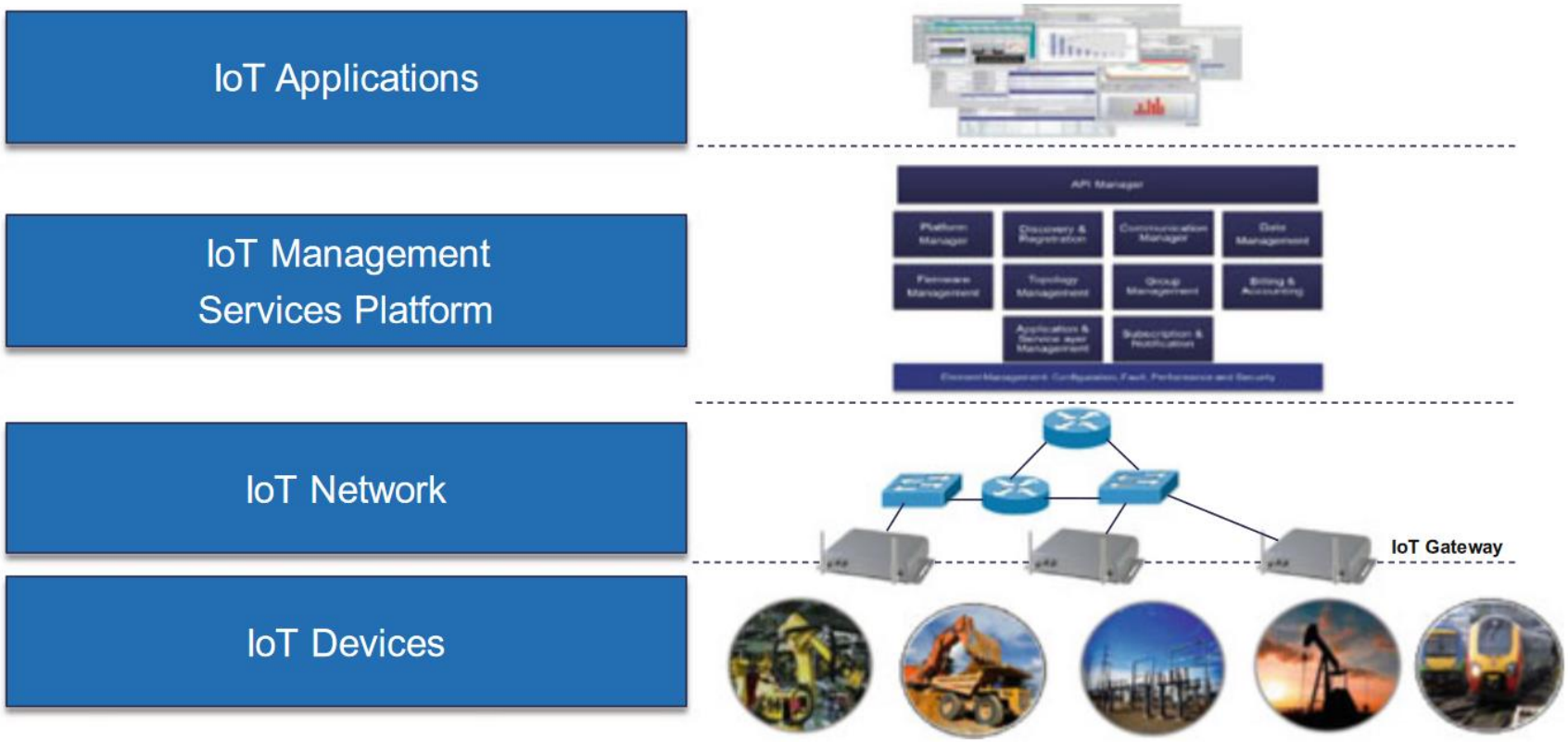
A network of connected devices which communicate over the Internet, and they do so autonomously, **machine to machine**, without the need for human intervention. (Source: Prof. Mischa Dohler, KCL)

Where is IoT used?

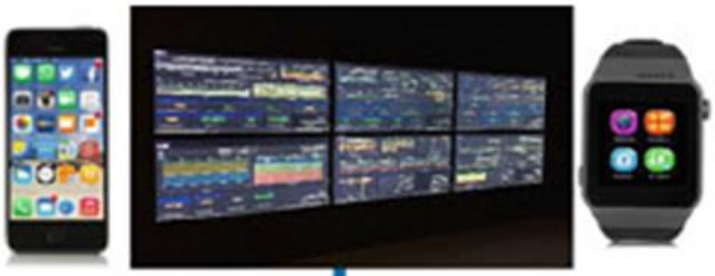
An example of IoT

Video: Life simplified with connected devices

IoT Reference Framework



IoT Reference Framework



4. Notification & Control

3. Ability to Communicate

IoT Network

2. Sensing & actuating

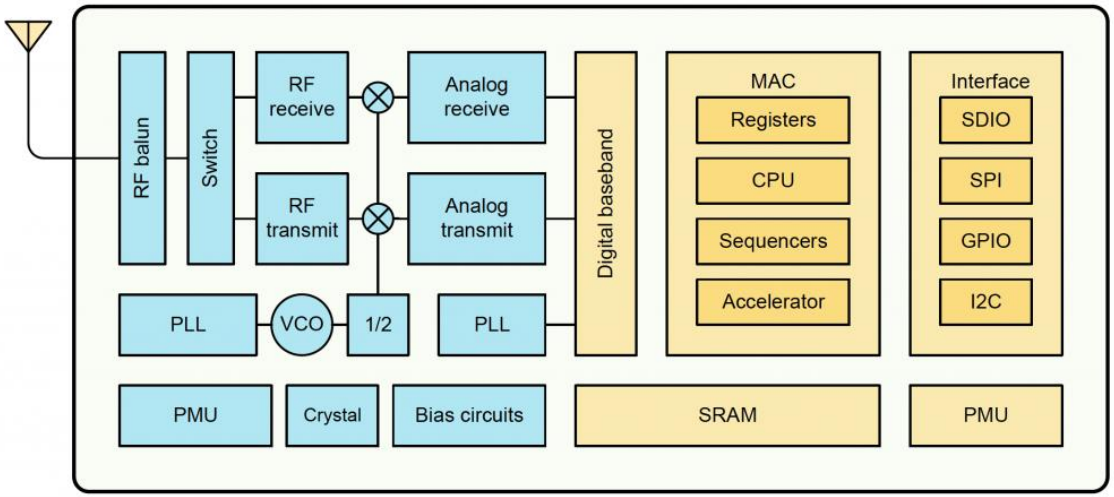
1. Unique Address

Things

Lua Programming Language

- A simple and easy to learn scripting language
- Powerful, efficient and lightweight
- Small and embeddable
- Fast

ESP 8266 WIFI Microcontroller Modules



- **Soft-microcontroller with WIFI Module**
- **32 bit microcontroller unit**
- **Several integrated peripheral units**
- **SRAM and FLASH storage**
- **Integrated power management**

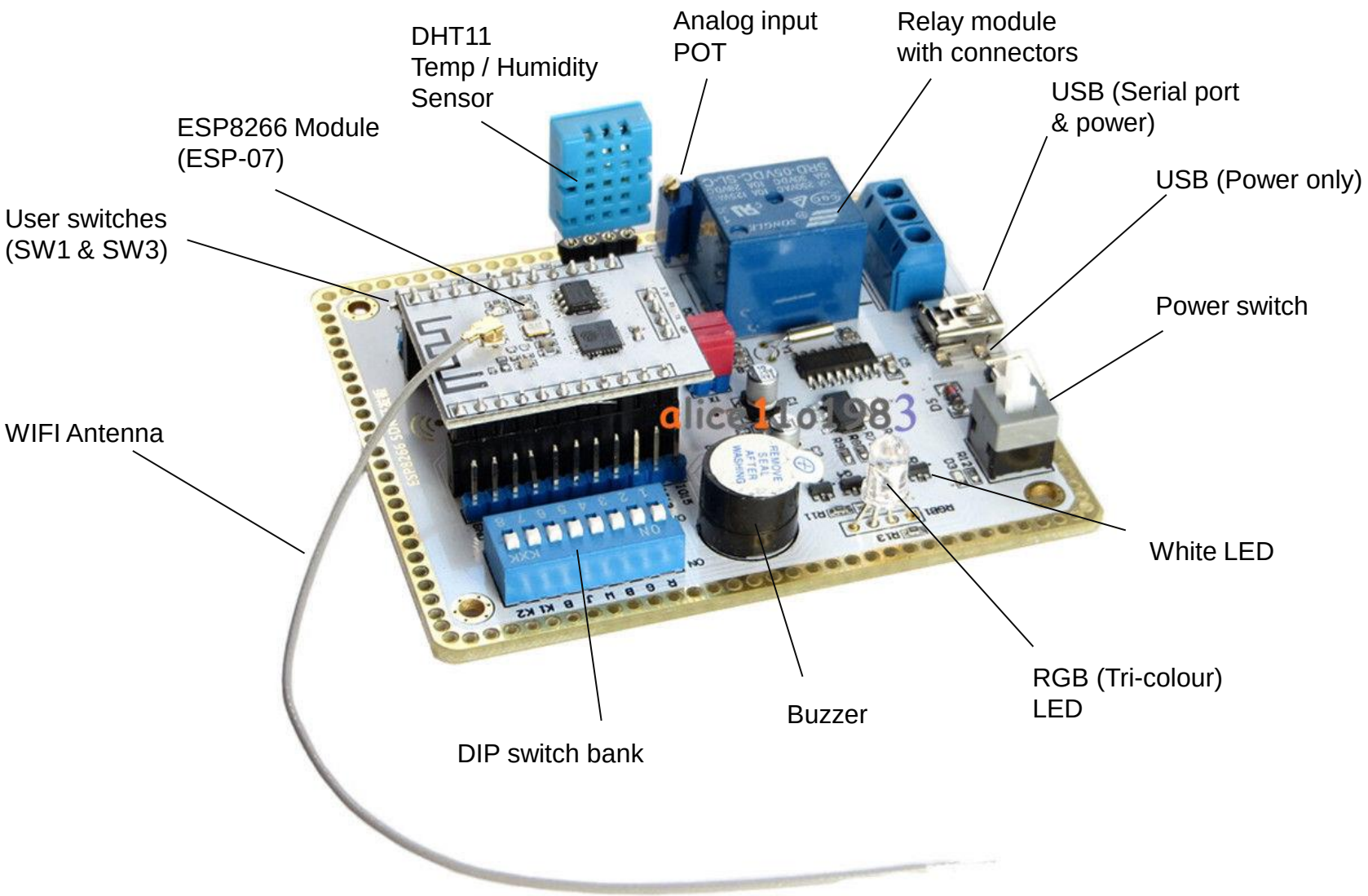
ESP8266 Software

- **Natively support a set of AT commands that control various functions of the device**
- **Can be flashed with other firmware versions to support different platforms**
  - Arduino platform (JAVA and C based)
  - NodeMCU platform (LUA based)
- **LUA platform is more popular among the IoT community**
  - We will be using a NodeMCU with Lua support

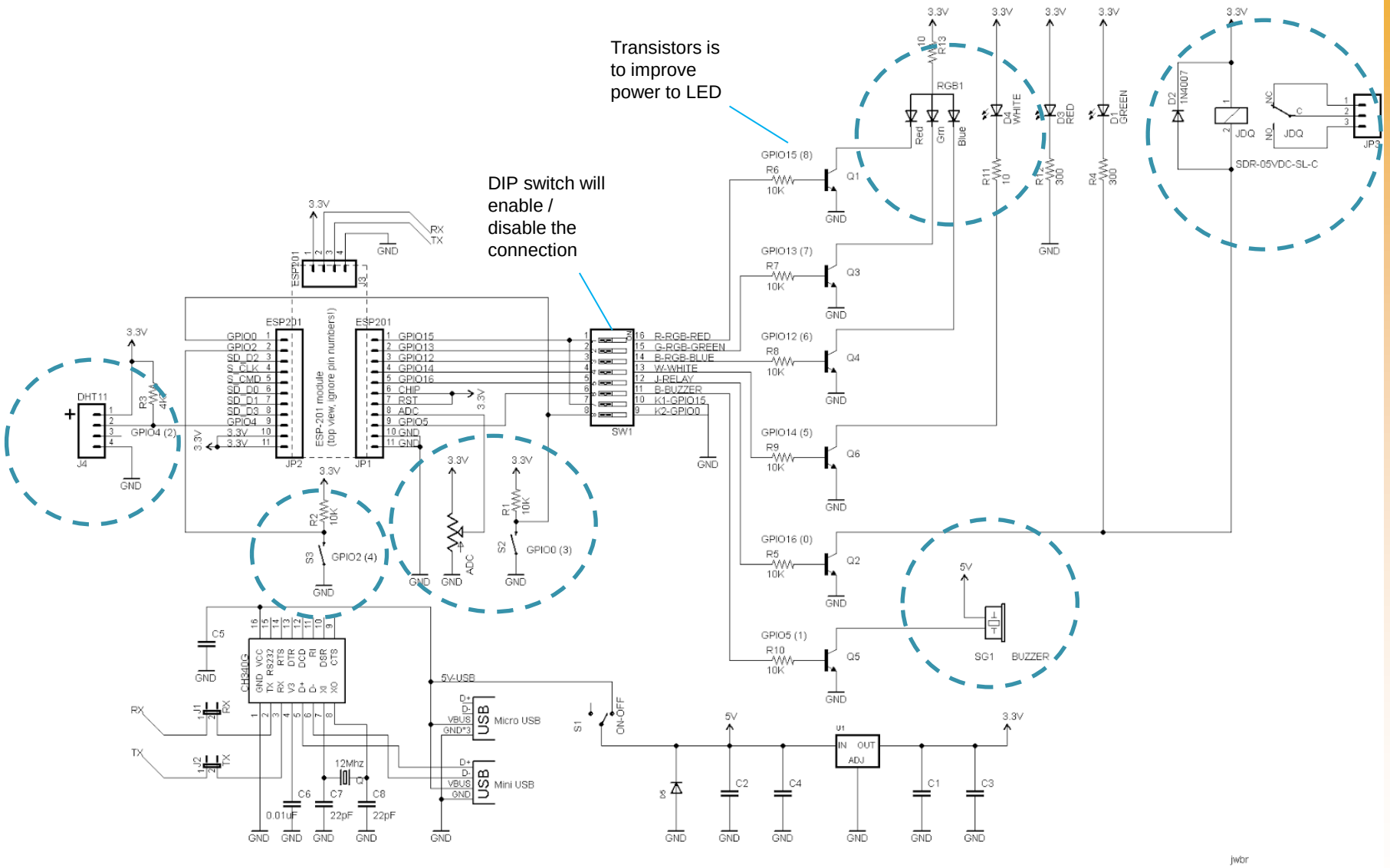
NodeMCU Platform

- **An open source platform based on ESP8266 module**
  - Especially suited for IoT applications
- **Arduino like hardware environment**
  - Simple interface – just need power to operate,
  - Serial port based console for programming and control
- **Based on the LUA language and environment**
  - Dynamic programming language
  - Interpreted, Scripting language
  - Simple, loosely typed

The development board



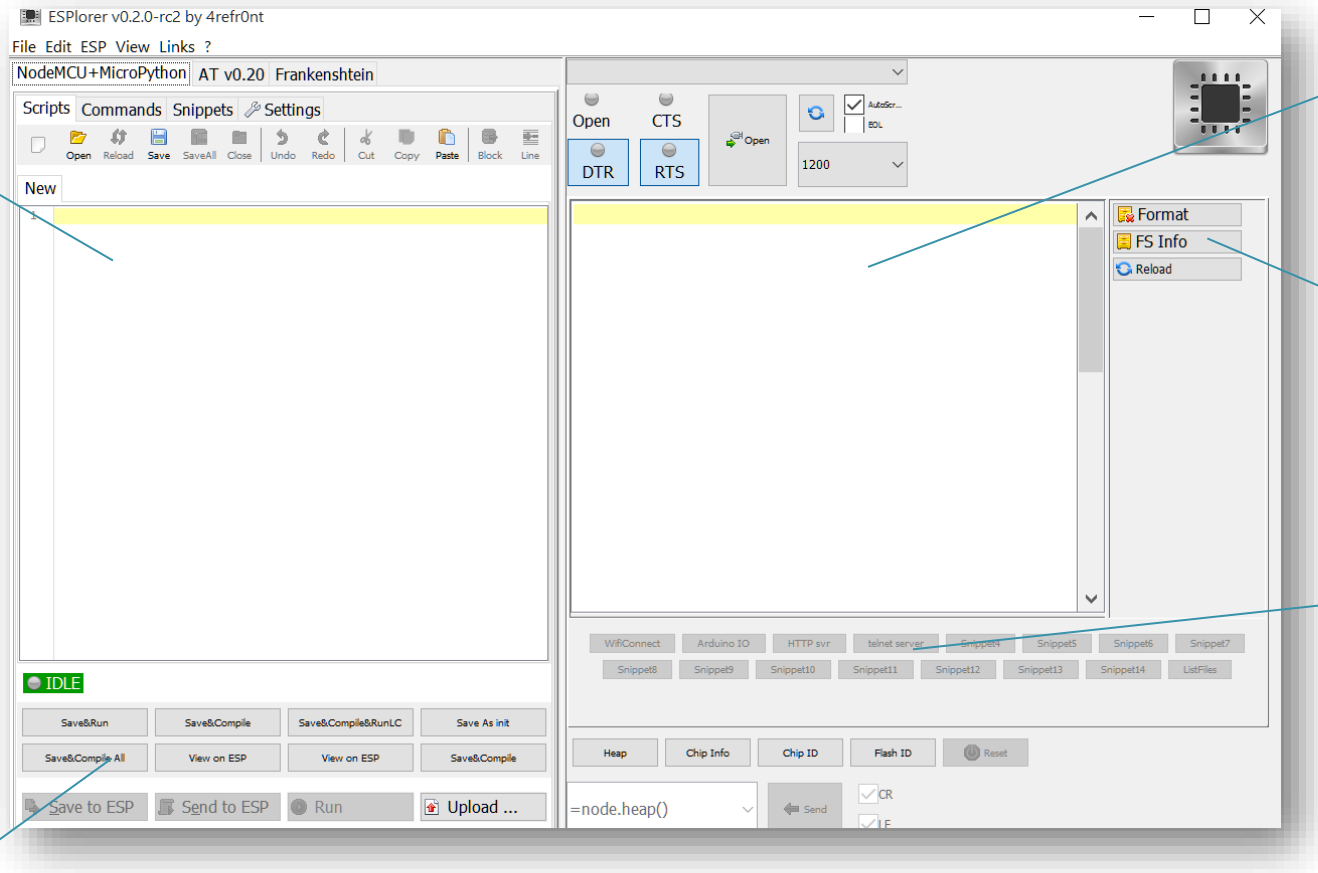
Schematic of the development board



Host interface to ESP8266 module

- **Can be used with any console / terminal application**
  - Just need plain text-based communication through serial (USB) port
  - Type & Send -> Commands and script lines to device
  - Receive & Print -> Output from the device
- **We will be using the ESPlorer tool**
  - An Open source tool with added functionality to a simple terminal program
  - Provide some specific features that are useful when learning

ESPlorer Interface



Script editor

Console output

Device storage info

Quick access scripts

Script management

Basic Lua Statements

Printing on console

```
Print("Hello World !")
```

```
print (10+5*5)
print ("This is a" .. " long string")
print ("10 x 5 is " .. 10*5)
```

String value

Concatenate two values

Result of expression

There are no explicit end-of-statement markers. All text up to carriage return is considered one line Statement is executed immediately after the carriage return is received

Comments

```
-- This is a comment line
```

Comment may start at any point

Data types and variables

Lua supports 5 basic data types

string	Represents an array of ASCII charactors **“This is a string”**
number	Represents all numerical values (*NodeMCU support only integer values*)
boolean	Represents values of Boolean logic true, false,
table	Represent arrays (one or multi-dimensions arrays)
nil	Special type that represent **null** or **un-assigned** value

Data types and variables

Lua is a loosely typed language

- Variables need not be pre-declared with a specific data type

```
a = 10
print(a)

a = "a is now a string"
print(a)

print(b)

b=a
print(b)

a=nil
b=nil
print(a)
print(b)
```

Assign value 10 to variable a
('a' becomes "number type")

Assign a string to the same
('a' change to "string type")

'b' has not been assigned a value
(the default type is "nil")

'b' is assigned with type and value of 'a'
('b' becomes "string type")

Clear and release memory

Lua operators

Arithmetic operators

- **Addition**
  - 10+4 → 14
- **Subtraction**
  - 10 - 4 → 6
- **Negation**
  - -10 -> Minus 10
- **Multiplication**
  - 10 * 4 → 40
- **Division**
  - 10 / 4 → 2

  (NodeMCU only support integer division)

Logical operators

- **Equal**
  - 10==15 → false, 10==10 → true
- **Not equal**
  - 10~=15 → true, 10~=10 → false
- **Greater than**
  - 10>15 → false, 15>10 → true
- **Greater than or equal**
  - 10>=15 → false, 11>=10 → true
- **Less than**
  - 10<15 → true, 10<10 → false
- **Less than or equal**
  - 10<15 → true, 10<10 → true

Objects in NodeMCU platform

- Application programming Interface (API) functions of the NodeMCU platform is categorised into several objects
- We will be using the following object types

gpio	• Functions to control general purpose digital IO pins
pwm	• Functions that support Pulse-Width-Modulated outputs on digital IO pins
tmr	• Functions that supports up to 6 timers that can generate periodic executions
wifi	• Has two sub modules (wifi.sta & wifi.ap) • Functions that support communication over wifi interface

gpio module – simple Digital output

Set IO mode of a pin

```
gpio.mode(5, gpio.OUTPUT)
```

Pin index *mode*

Supported modes

gpio.OUTPUT → output
gpio.INPUT → input
gpio.INT →interrupt

Pin index 5 is connected White colour LED on demo board

Control an output pin

```
gpio.write(5, gpio.HIGH)
```

Pin index *Output logic level*

Supported logic levels

gpio.HIGH → “on state”
gpio.LOW → “off” state

Digital outputs – blink RGB LED

Try the following script

```
gpio.mode(8,gpio.OUTPUT)
gpio.mode(7,gpio.OUTPUT)
gpio.mode(6,gpio.OUTPUT)
gpio.mode(1,gpio.OUTPUT)

gpio.write(8,gpio.HIGH)
tmr.delay(1000000)
gpio.write(8,gpio.LOW)

gpio.write(7,gpio.HIGH)
tmr.delay(1000000)
gpio.write(7,gpio.LOW)

gpio.write(6,gpio.HIGH)
tmr.delay(1000000)
gpio.write(6,gpio.LOW)

gpio.write(1,gpio.HIGH)
tmr.delay(1000000)
gpio.write(1,gpio.LOW)
```

Set RGB LED pins and Buzzer pin to output mode
8 →Red, 7 →Green, 6 →Blue
1 →Buzzer

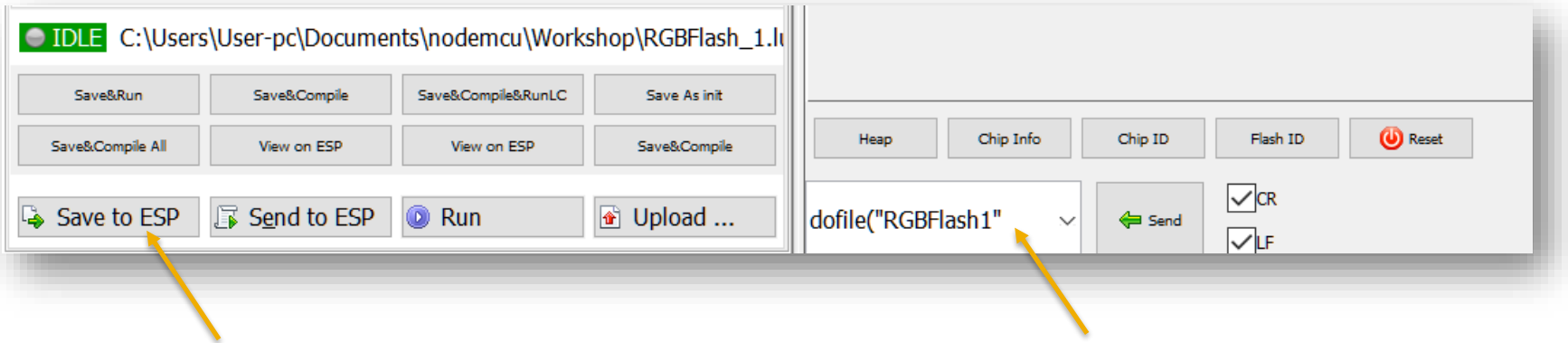
Turn each RGB LED on / off in turn

Tmr.delay(1000000) causes 1 second delay between on and off

This is a blocking delay – no other statement executes while waiting

Turn on Buzzer for 1 second (1000000 micro-seconds)

Saving a script in MCU storage space



Click on "Save to ESP" button transfer script file to MCU flash storage

Send the command **dofile("***<filename.lua>***")** *execute the file from storage*

- *Once saved, the file remains in the flash storage of the device, even after a power cycle*
- *Execution from flash storage is faster since there are no delays in serial transfer of commands from the host computer*

gpio module – Simple digital Input

Read from a digital IO pin

```
gpio.mode(5, gpio.INPUT)
pinState = gpio.read(3)
print(pinState)
```

Pin index

Pin index 3 is connected s2 push button switch

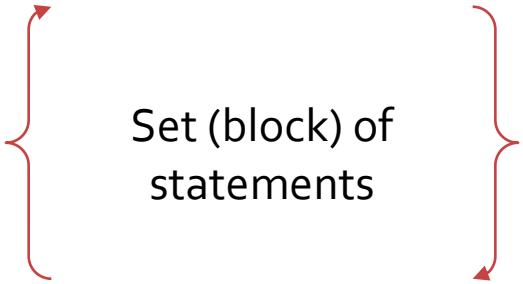
Due to the pull-up resister in the demo-board the pin would read as logic '1' (HIGH , true) when the switch is open

You may need to hold down S2 and send code to ESP chip to read a logic '0' state

Lua control structures – "Repeat .. Until" block

- **repeat** .. **until** <*boolean_value*>
  - Repeats a block of statements until the <Boolean_value> becomes true

repeat



until <logic expression>

Block of statements is repeated until the Boolean expression becomes true

Since the check is carried only at end of each iteration the Block of Statements is executed at least once

The repeat .. until is a blocking structure. All other CPU activity threads (including console interactions) are put on hold until the iterations are completed

Continuously reading an input pin

How is this script expected to behave?

```
gpio.setmode(3,gpio.INPUT)

repeat
   gpio.write(5,gpio.HIGH)
   tmr.delay(100000)
   gpio.write(5,gpio.LOW)
   tmr.delay(100000)
until false
```

Pin index 5 is connected White colour LED on demo board

Continuously reading an input pin

How is this script expected to behave?

```
gpio.setmode(3,gpio.INPUT)

repeat
   gpio.write(5,gpio.HIGH)
   tmr.delay(100000)
   gpio.write(5,gpio.LOW)
   tmr.delay(100000)
until false
```

Since terminating condition is a constant "**false**" we can expect this code repeat forever

ESP8266 has a safety mechanism to prevent processor hang-up – even if such is due to your program code

An internal counter (called the "Watch Dog Timer" is incremented each time a blocking statement is executed. This timer is automatically cleared when in a non-blocking state
When the timer reaches its maximum value, processor is re-started as a safety measure

Continuously reading an input pin

How is this script expected to behave?

```
gpio.setmode(3,gpio.INPUT)

repeat
   gpio.write(5,gpio.HIGH)
   tmr.delay(100000)
   gpio.write(5,gpio.LOW)
   tmr.delay(100000)
   tmr.wdclr()
until false
```

We can manually clear the Watch Dog Timer counter to prevent safety re-start (not advisable)

tmr.wdclr()

ESP8266 has a safety mechanism to prevent processor hang-up – even if such is due to your program code

An internal counter (called the "Watch Dog Timer" is incremented each time a blocking statement is executed. This timer is automatically cleared when in a non-blocking state
When the timer reaches its maximum value, processor is re-started as a safety measure

Blink white LED until S2 is pressed

Combine the flowing two scripts to create a program that will blink the White LED until S2 is pressed

```
gpio.setmode(3,gpio.INPUT)

repeat
   gpio.write(5,gpio.HIGH)
   tmr.delay(100000)
   gpio.write(5,gpio.LOW)
   tmr.delay(100000)
until false
```

```
gpio.mode(5, gpio.INPUT)
pinState = gpio.read(3)
print(pinState)
```

Blink white LED until S2 is pressed

```
gpio.mode(5,gpio.OUTPUT)
gpio.mode(3,gpio.INPUT)


repeat
 gpio.write(5,gpio.HIGH)
 tmr.delay(100000)
 gpio.write(5,gpio.LOW)
 tmr.delay(100000)
 tmr.wdclr()

 pinState = gpio.read(3)
until pinState==0
```

Exit condition of the repeat .. until block (pinState == 0) becomes true when gpio module reads a logic LOW from pin 3 during the last iteration

IF .. THEN .. ELSE statement

if <*boolean_expr*> **then**

Statement Block which is executed when the <*boolean_expr*> *is* **true**

else

Statement Block which is executed when the <*boolean_expr*> *is* **false**

end

Keyword else and the Statement Block following that is optional

if .. then ..end

```
if kittenCount > 0 then
   print("You have kitten(s)")
   feedKitten = true
end
```

if .. then ..end

```
if kittenCount > 0 then
   print("You have kitten(s)")
   feedKitten = true
else
   print("You don't have kitten(s)")
   feedKitten = else
end
```

Switch on White LED when S3 is pressed

```
gpio.mode(5,gpio.OUTPUT)
gpio.mode(4,gpio.INPUT)

repeat
  s3Pin = gpio.read(4)

if s3Pin==0 then
    gpio.write(5,gpio.HIGH)
   else
    gpio.write(5,gpio.LOW)
  end

  tmr.wdclr()
  s2Pin = gpio.read(3)
until (s2Pin==0) and (s3Pin==0)
```

Try this code

Some housekeeping code to disable Watch Dog restart and exit loop when both S2 & S3 are pressed

Write code to control Red, Green LEDs via S2 & S3

- **Write code that will:**
  - Switch on Red LED when S2 is pressed
  - Switch on Green LED when S3 is pressed
  - Exit from programme when both S2 and S3 are pressed

Device	Pin index
Red LED	8
Green LED	7
S2	3
S3	4

Step 1	Set pin 8 (Red LED) & pin 7 (Green LED) to output mode Set pin 3 (S2) & pin 4(S3) to input mode
Step 2	Read S2 pin logic state
Step 3	Drive Red LED based on S2 pin state
Step 4	Read S3 pin logic state
Step 5	Drive Green LED based on S3 pin state
	Put all code from Step 2 to Step 5 inside **a repeat .. until** block with exit condition to be true when both S2 and S3 are pressed Clear Watch Dog timer inside the repeat..until block

Write code to control Red, Green LEDs via S2 & S3

```
gpio.mode(8,gpio.OUTPUT)
gpio.mode(7,gpio.OUTPUT)
gpio.mode(4,gpio.INPUT)
gpio.mode(3,gpio.INPUT)

repeat
  s3Pin = gpio.read(4)
  s2Pin = gpio.read(3)
if s3Pin==0 then
    gpio.write(8,gpio.HIGH)
   else
    gpio.write(8,gpio.LOW)
  end
if s2Pin==0 then
    gpio.write(7,gpio.HIGH)
   else
    gpio.write(7,gpio.LOW)
  end
  tmr.wdclr()
until (s2Pin==0) and (s3Pin==0) )
```

Some housekeeping code to disable Watch Dog restart and exit loop when both S2 & S3 are pressed

Maintaining state & switch bounce

```
gpio.mode(5,gpio.OUTPUT)
gpio.mode(4,gpio.INPUT)
gpio.mode(3,gpio.INPUT)

WLedState = false

repeat
  s3Pin = gpio.read(4)
  s2Pin = gpio.read(3)

 if s3Pin==0 then
    WLedState = not WLedState
  end
 if WLedState then
    gpio.write(5,gpio.HIGH)
   else
    gpio.write(5,gpio.LOW)
  end

  tmr.wdclr()
until (s2Pin==0) and (s3Pin==0)
```

Make S3 a toggle switch for White LED

- Each time the switch is pressed the LED should change between ON / OFF

State variable *WLedState* is changed each time S3 is pressed

Output pin is driven based on the state variable

Maintaining state & switch bounce

```
gpio.mode(5,gpio.OUTPUT)
gpio.mode(4,gpio.INPUT)
gpio.mode(3,gpio.INPUT)

WLedState = false

repeat
  s3Pin = gpio.read(4)
  s2Pin = gpio.read(3)

 if s3Pin==0 then
    WLedState = not WLedState
  end
 if WLedState then
    gpio.write(5,gpio.HIGH)
   else
    gpio.write(5,gpio.LOW)
  end
 tmr.delay(500000)
  tmr.wdclr()
until (s2Pin==0) and (s3Pin==0)
```

Make S3 a toggle switch for White LED

- Each time the switch is pressed the LED should change between ON / OFF

State variable *WLedState* is changed each time S3 is pressed

Output pin is driven based on the state variable

Half a second delay can solve the issue with multiple read / bouncing

User defined functions

function *<function_name>* **(***<parameter list>***)**

Block of statements implementing the function

end

Parameter list is optional

Function can be defined anywhere within the script

```
function soundBuzzer (timeInMS)
    gpio.write(1,gpio.HIGH)
    tmr.delay(timeInMS * 1000)
    gpio.write(1,gpio.LOW)
end

gpio.mode(1,gpio.OUTPUT)

print("A short Beep")
soundBuzzer(1000)
tmr.delay(2000000)
print("A longer Beep")
soundBuzzer(2000)
```

Define the user function

Note: tmr.delay is in micro-seconds so we need to multiply time

Call function with parameters

Event driven systems

Procedural

- **Program follows a pre-defined sequence of steps (functions) from start to end**
  - Only one response can be given at each step
  - Other responses remain blocked
  - Not suitable for IoT environments where many activities can happen at the same time as well as in out-of-sequence

Event Driven

- **Program consist of a collection of functions (steps) , not in a specific sequence**
  - Each function is attached to an “event” (such as change in IO signal)
  - Function provide response to event with a sort sequence of actions
  - Event handlers do not block each other
  - No specific sequence of execution

Event driven toggle switch

```
gpio.mode(5,gpio.OUTPUT)
gpio.mode(4,gpio.INT)

function toggleLED ()
 WLedState = not WLedState
 if WLedState then
    gpio.write(5,gpio.HIGH)
   else
    gpio.write(5,gpio.LOW)
  end
gpio.write(1,gpio.HIGH)
  tmr.delay(100000)
  gpio.write(1,gpio.LOW)
end

WLedState = false

gpio.trig(4,"up",toggleLED)
```

Create a function to toggle state of the White LED and make a short beep

- Attach this function to LOW→HIGH change event of S_3 input pin

Pin mode is now set to INT

Event handler function
Toggle the State variable (WLedState) and make a short beep

Atttach event function to input pin and set to trigger when changed from LOW→HIGH

Supported event types	
up : LOW→HIGH	down : HIGH→LOW
low : when LOW	high : when HIGH

Timer module: periodic and timeout events

Creating a timer event

tmr.alarm(*index*, *interval*, *type*, *event_handler*)

Timer index (0 to 7)
Supports up to 7 independent timers

Time interval (ms)
at which the event should be called in milli Seconds

Periodic / Timeout
1 → repeat until stopped
0 → generate only once

Event handler
Function that handle timer event

Stop a timer from generating further events

tmr.stop(*index*)

NOTE: Once stopped timer will not generate any further events. Can be called even within a timer event handler to prevent further events

Blinking LED with timer event

```
gpio.mode(5,gpio.OUTPUT)


function toggleLED ()
 WLedState = not WLedState
 if WLedState then
    gpio.write(5,gpio.HIGH)
   else
    gpio.write(5,gpio.LOW)
  end
end

WLedState = false

tmr.alarm(1,1000,1,toggleLED)
```

Create a timer event that is called every 1 second (1000 ms)

- Attach the function that toggle LED to this event

Pin mode is now set to INT

Event handler function
Toggle the State variable (WLedState) each time called

Create a timer event that is triggered every 1000 ms repeatedly and attached LED event handler

Colour changing LED

- **Write a script that change the colour of the RGB LED every 1 second**
  - Write 3 separate event functions that will toggle each of the Red, Green and Blue LEDs when called
  - Create 3 separate timers, that trigger a periodic event at 1S, 2S and 4S intervals
  - Attach one event function to each of the timers

Each time the timer is triggered one of the RGB LEDs will toggle between ON & OFF. Since events are generated at different intervals this will create combinations of RGB colours

Colour changing LED

```
function toggleRED ()
 REDState = not REDState
 if REDState then
    gpio.write(8,gpio.HIGH)
   else
    gpio.write(8,gpio.LOW)
  end
end

function toggleGREEN ()
 GREENState = not GREENState
 if GREENState then
    gpio.write(7,gpio.HIGH)
   else
    gpio.write(7,gpio.LOW)
  end
end

function toggleBLUE ()
 BLUEState = not BLUEState
 if BLUEState then
    gpio.write(6,gpio.HIGH)
   else
    gpio.write(6,gpio.LOW)
  end
end
```

RED LED toggle

GREEN LED toggle

BLUE LED toggle

```
gpio.mode(6,gpio.OUTPUT)
gpio.mode(7,gpio.OUTPUT)
gpio.mode(8,gpio.OUTPUT)
```

Set pin mode

```
REDState = false
GREENState = false
BLUEState = false
```

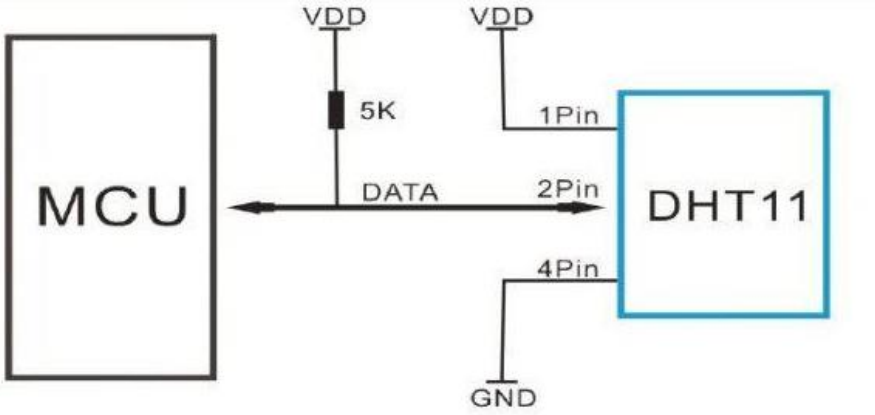
Initialize State variables

```
tmr.alarm(1,1000,1,toggleRED)
tmr.alarm(2,2000,1,toggleGREEN)
tmr.alarm(4,4000,1,toggleBLUE)
```

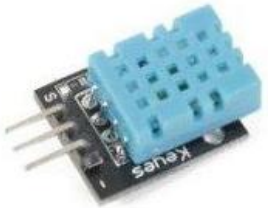
Create 3 timer events

DHT11: Temperature & Humidity Sensor

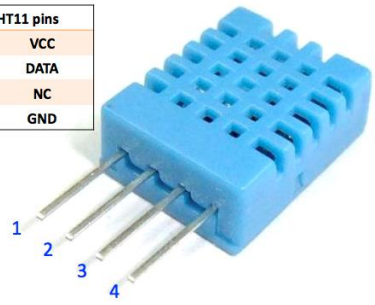
- **DHT11 is a sensor module that can read temperature and humidity**
- **Uses 1-wire interface to communicate with MCU**
- **Provide readings in digital format**
  - Read temperature in Celcius (0.1 degree)
  - Read humidity (0.1%)

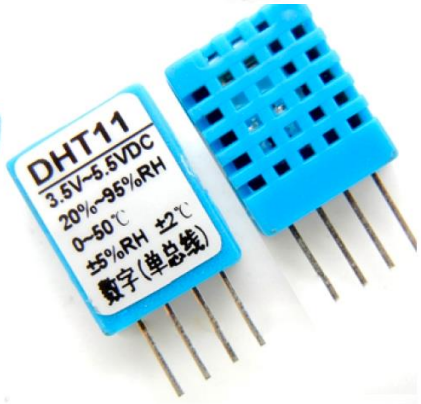


Typical Three Wire Connections for DHT11



DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND





Reading from DHT11

DHT sensor interface is in a separate library ("*dht_lib.lua*") which must be saved into device storage

```
DHT= require("dht_lib")

--read sensor and get temp / humidity
  DHT.read(2)
  tmr.delay(500000)
  t = DHT.getTemperature()
  h = DHT.getHumidity()

DHT = nil

--print results of temp conversion
print("temperature : "..(t/10).."."..(t-10*(t/10))..."C")
print("Humidity    : "..(h/10).."."..(h-10*(h/10))..."%")
```

Load "dht_lib" library

Read data from sensor

Load "dht_lib" library

Print temp & humidity after converting to a decimal point
Note: *conversion required since only integers are supported*

Print humidity and temperature in an event

- Include DHT sensor reading and printing inside an event handler
- Some special conditions are kept to prevent errors when the sensor is busy
- Call the event handler through a timer event, every 3 seconds

```
--Load the dht sensor library from device storage
DHT= require("dht_lib")

function showDHT ()
--read sensor and get temp / humidity
  DHT.read(2)
  tmr.delay(500000)
  t = DHT.getTemperature()
  h = DHT.getHumidity()
  --print results of temp conversion
  if (t ~= nil) and (h~=nil) then
    print("temperature : "..(t/10).."."..(t-10*(t/10))..."C")
    print("Humidity    : "..(h/10).."."..(h-10*(h/10))..."%")
   else
    print("waiting for sensor readings .....")
  end
end

--setup timer to print every 3 seconds
tmr.alarm(1,3000,1,showDHT)
```

Working with WiFi module

- **ESP8266 has a built in WiFi interface that can be operated in two primary modes**
  - STA mode → as a WiFi station (i.e. adapter in a client device)
  - AP mode → as a WiFi access point (with limited functionality)
- **Following steps are required in connecting and using WiFI network features (STA mode)**
  1. Setup device MAC address (optional if only one device is in use)
  2. Set WiFi mode
  3. Configure with SSID and PASSWORD of your network access point
  4. Connect to network
  5. Optional
     - Check connection status
     - Check IP address
  6. Setup a server or client
     - Write relevant event handlers

WiFi module : key functions

1. Set MAC address

wifi.sta.setmac(*<address_string>*)

<address_string> is the mac address in byte string, for example:"\024\024\024\024\024\024"

2. Set operating mode

wifi.setmode(*<mode>*)

<mode> options	
Wifi.STATION	Set to STA mode
Wifi.SOFTAP	Set to AP mode

3. Configure for a access point connection

wifi.sta.configure(*<SSID>,<PASSWORD>*)

<SSID> → *String value specifying network SSID*
<PASSWORD> → *String value contain network joining password*

WiFi module : key functions

4. Connect to the network

wifi.sta.connect()

Connect to configured network (use wifi.sta.disconnect() to disconnect from a network)

5a. Check status

wifi.sta.status()

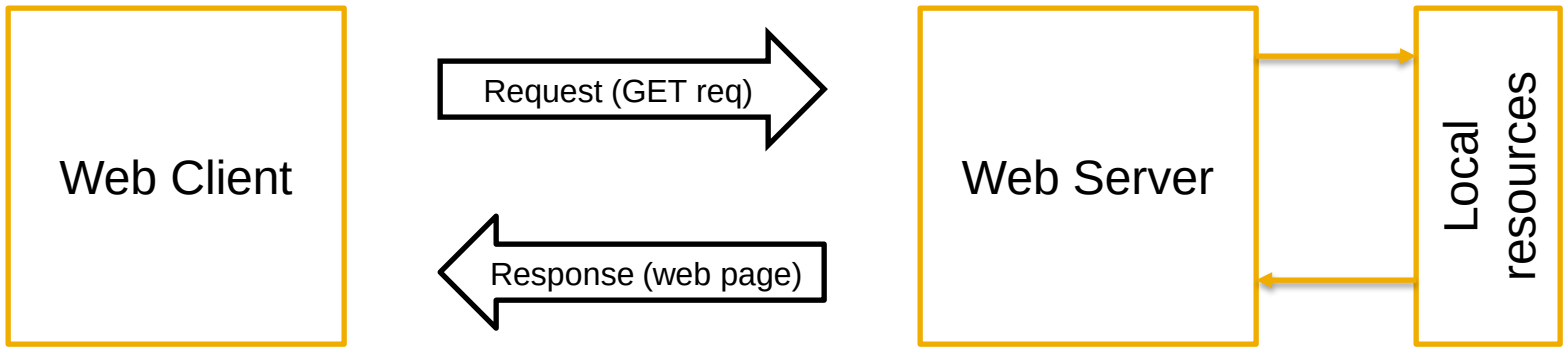
Connection status	
0	Station idle
1	Station connecting
2	Wrong password
3	AP not found
4	Connection failed
5	connected

5b. Check IP address

wifi.sta.getip()

Display <IP address>, <Subnet Mask>, <Gateway address>

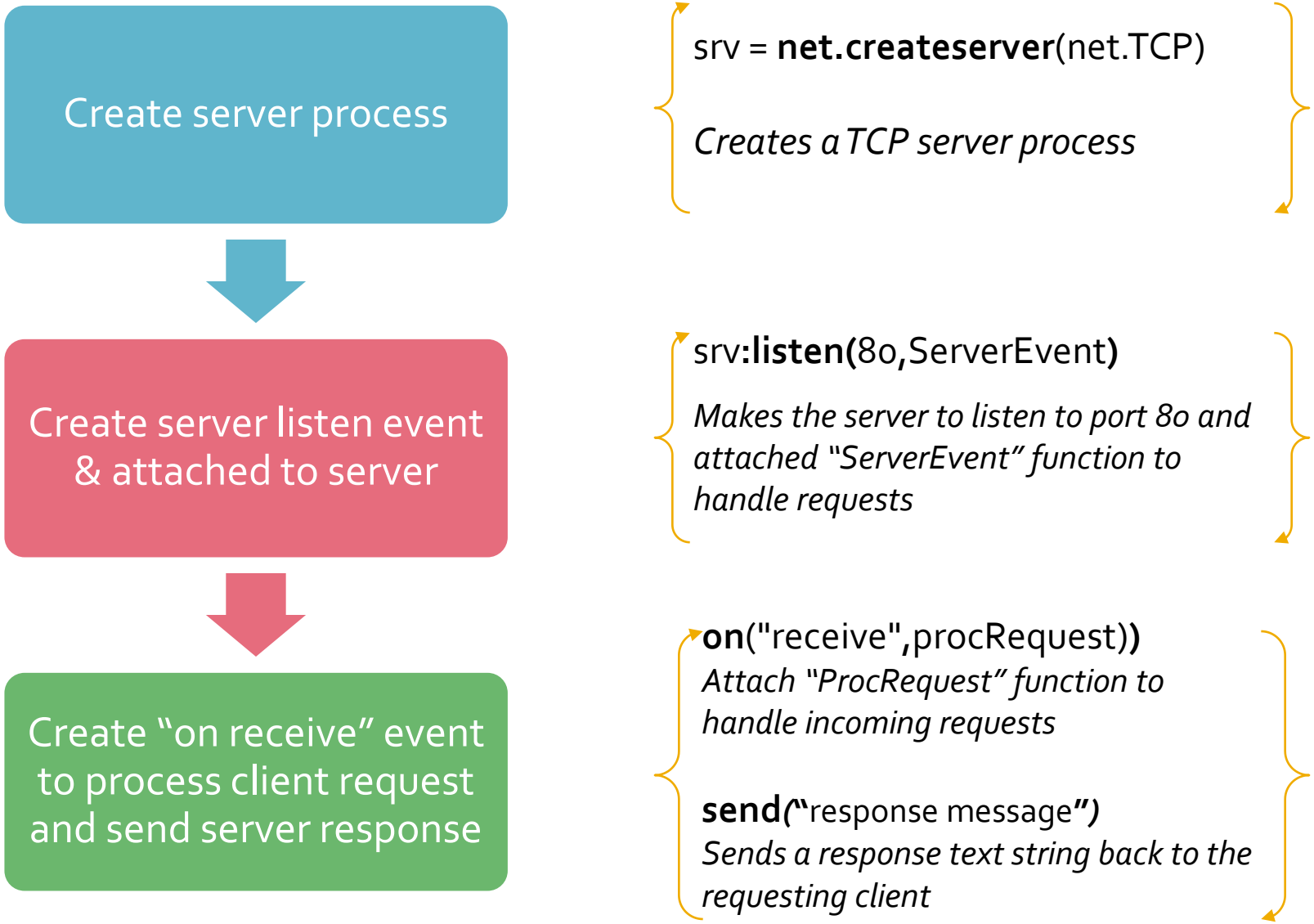
Web interaction process



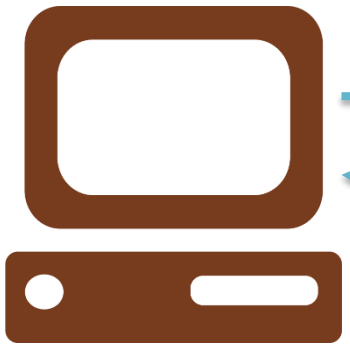

```
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type" />
<title> This is web Page 1</title>
</head>
<body>
 Contents of web page
</body>
</html>
```

```
GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: en-SG,en;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36 Edge/12.10240
Accept-Encoding: gzip, deflate
Host: 192.168.1.8
Connection: Keep-Alive
```

Creating a web server



Serving a web client request



Request (GET)

Response (HTML web page)

Server process

Listen event handler function

On-receive event handler function

1. Check the incoming message and trigger the Listen event of the request port

1. When request is received, trigger On-Receive event to process the request
2. Prepare and send the Response and complete the transaction

1. Get the request message string (payload)
2. Process request string and if applicable determine the response to be sent

Each response is a new web page sent to the client. Server does not maintain any state information.

State data (if applicable) must be included in the web page to be sent back with next request

Web display of temperature and humidity

Creating the server process

```
srv=net.createServer(net.TCP)
srv:listen(80,ListenEvent)
```

The on-receive event handler

```
function procRequest(conn,payload)
  print("----------New request ----------")
  print(payload)
end
```

On-receive event can be used to do additional server side processing based on the payload request
e.g. control a device, read a sensor etc.

The listen event handler

```
function ListenEvent (conn)
  conn:on("receive",procRequest)

  dht.read(2)
  temp = dht.getTemperature()
  hum = dht.getHumidity()

  conn:send(HTMLTop)
  conn:send(HTMLBottom)
end
```

Response must be a complete web page written in plain HTML or any other web scripting language

Page can be static read from storage or generated dynamically

Web Server: Adding control to server side

Creating the server process

```
srv=net.createServer(net.TCP)
srv:listen(80,ListenEvent)
```

The listen event handler

```
function ListenEvent (conn)
   conn:on("receive",procRequest)

  dht.read(2)
   temp = dht.getTemperature()
   hum = dht.getHumidity()

   conn:send(HTMLTop)
   …….
   conn:send(HTMLBottom)
end
```

The on-receive event handler

```
function procRequest(conn,payload)
    print(payload)
--  switch on LED if "ON" button is clicked
    i,j = string.find(payload,"SwitchON")
    if (i~=nil) then
     if (i < 50) then
      gpio.write(5,gpio.HIGH)
      gpio.write(0,gpio.HIGH)
     end
    end
--  switch off LED if "OFF" button is clicked
    i,j = string.find(payload,"SwitchOFF")
    if (i~=nil) then
     if (i < 50) then
      gpio.write(5,gpio.LOW)
      gpio.write(0,gpio.LOW)
     end
    end
end
```

Now process payload and control LED / Relay

More information

NodeMCU website	www.nodemcu.com
Lua official website	www.lua.org
NodeMCU Documentation	http://www.nodemcu.com/docs/index/
ESP8266 manufacturer	www. espressif.com
ESP8266 community forum	www.esp8266.com
Esplorer web site	esp8266.ru/esplorer/

IO pin mapping in demo board

Device	Pin index
White LED	5
RGB LED (Red)	8
RGB LED (Green)	7
RGB LED (Blue)	6
Buzer	1
Relay	0
S2	3
S3	4

Thank you