INDEX

INSTALLATION

1. Ubuntu

(a) Have Ubuntu image in pen-drive or a cd

(b) Take care to disable secure boot before installation

(c) After installation if windows doesn't show in the boot options then repair the GRUB from UBUNTU using *boot repair*

(d) If you use face further problems use GRUB customizer from UBUNTU

(e) Once Windows shows in boot options then change the Boot preference order from windows

(f) In Sony systems while booting use *ASSIST* key

2. openHAB

Studied openHAB, The thing system and other engines and found that among these openHAB was the best option available, since it was fully open sourced and it consisted of nearly 100+ addons. It also support MQTT binding which will be useful later on as a minimal size packet data communication protocol.

Installation of openHAB runtime

(a) Download openHAB from its Github repository https://github.com/openhab

(b) Extract the zip file to a suitable directory of your Linux system

3. Installatio-of-Apache2-,-PHP-and-MySQL-LAMP-server

About LAMP

LAMP stack is a group of open source software used to get web servers up and running. The acronym stands for Linux, Apache, MySQL, and PHP. Since the virtual private server is already running Ubuntu/openSuse, the linux part is taken care of. Here is how to install the rest.

(a) Step 1—Install Apache

To install apache, open terminal and type in these commands: **For Ubuntu** `sudo apt-get update`

```
sudo apt-get install apache2
```

For openSuse

`sudo zypper in apache2`

Firewall Adjustments

In openSuse, by default the firewall configuration blocks all traffic coming on port 80 to your machine. So if you need to allow access so that the web server can be accessed from within a LAN we need to fine tune the firewall configuration. The below step needs to be performed as root user. The supplied configurations are called apache2 and apache2-ssl. They can be enabled via YaST, by adding them to FW_CONFIGURATIONS_EXT in /etc/sysconfig/SuSEfirewall2

```
sysconf_addword /etc/sysconfig/SuSEfirewall2 FW_CONFIGURATIONS_EXT
apache2 sysconf_addword /etc/sysconfig/SuSEfirewall2 FW_CONFIGURATIONS_EXT
apache2-ssl rcSuSEfirewall2 restart
```

Starting Server

Start the server and configure it to automatically start at boot time.

`rcapache2 start chkconfig -a apache2`

(b) Step 2—Install MySQL

MySQL is a powerful database management system used for organizing and retrieving data

To install MySQL, open terminal and type in these commands:

For Ubuntu sudo apt-get install mysql-server libapache2-mod-auth-mysql php5-mysql

For openSuse

`sudo zypper install mysql-server`

To start the server

`sudo systemctl start mysql.service`

(c) Step 3—Install PHP

PHP is an open source web scripting language that is widely use to build dynamic webpages.

To install PHP, open terminal and type in this command.

For Ubuntu

`sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt`

For openSuse

`sudo zypper in php5 php5-mysql apache2-mod_php5`

Installing phpMyAdmin

For Ubuntu

`sudo apt-get install phpMyAdmin`

For OpenSuse

`sudo zypper in phpMyAdmin`

Finally restart apache so that all of the changes take effect:

For Ubuntu

`sudo service apache2 restart`

For openSuse

```
sudo systemctl start apache2.service
```

4. Broker and nodejs

Based on their use, there are two widely available MQTT brokers available on the web.

We chose Mosca because it is:

(a) MQTT 3.1 compliant.

(b) supporting various storage options for QoS 1 offline packets, and subscriptions.

(c) As fast as it is possible.

(d) Usable inside ANY other Node.js app.

Installation

Installation on a OpenSuse system (version 13.1) **Part 1** First install nodejs and npm if both are absent in your system.

Nodejs

```
sudo zypper install nodejs
```

npm

```
sudo zypper install npm
```

If everything goes right (no errors) then proceed to second part.

Part 2

Installation of Mosca

```
npm install mosca
```

If everything goes write then you are good to go.

To start the broker >`mosca`

Note: For persistence mode, broker requires mongodb/redis installed on the system. Installation codes are: >`sudo zypper install mongodb-org`

To start the database service >`sudo service mongod start`

Installation on an Ubuntu system (version 14.4) Part 1 First install nodejs and npm if both are absent in your system.

Nodejs

```
sudo apt-get update

sudo apt-get install nodejs

sudo ln -s /usr/bin/nodejs /usr/bin/node
```

npm

```
sudo apt-get install npm
```

If everything goes right (no errors) then proceed to second part.

Part 2

Installation of Mosca

```
sudo npm install debug
```

`sudo npm install mosca bunyan -g` //for installing mosca globally

```
sudo npm install daemon
```

If everything goes write then you are good to go.

To start the broker >`mosca`

Note: For persistence mode, broker requires mongodb/redis installed on the system. Installation codes are: >`sudo apt-get update` >`sudo apt-get install -y mongodb-org`

To start the database service >`sudo service mongod start`

Testing the broker

First to test the mqtt broker, we need to install a mqtt client on the system.

```
sudo npm install mqtt -g
```

Second, creating scripts for testing

Script for starting MQTT broker service, **mosca-app.js**

To start the service type >`node mosca-app.js`

Script for MQTT Publishing Client client-pub.js

Script for MQTT Publishing Client client-sub.js

Open a new Terminal and Execute above MQTT Subscription Client client-sub.js >'node client-sub.js'

Now open a differnent Terminal and Execute above MQTT Subscription Client client-pub.js >'node client-pub.js'

Successful Output From client-sub.js terminal

```
Hello!
```

Troubleshooting

If while installing mosca there are errors in fetching the links from server, do any of the following

- Check that nodejs version is not 'pre', if it is then update it to a stable non pre version.
- reinstalling nodejs and npm in root mode.
- restart your system.

While executing those scripts, if any of the nodejs script file shows error **Mqtt module not found**, then copy the scripts to a folder inside mosca directory, and try executing from there.

For **openSuse** users, open firewall in yast2 GUI. Then go to allowed service, then go to to advance option, there enter 1883 inside tcp port.

The thing system - Architecture

The steward is center of the system, monitoring and controlling all sorts of things, according to the directives it receives from various clients.

Communicating with Things

There are three different protocols that the steward uses to communicate with a thing.

1. With consumer equipment native protocol is used.
2. Things concerned with sensors reading reports uses
3. Things performing some events uses

Communication with clients

1. Simple clients like platform specific Arduino, Android, iOS and platform neutral HTML5. Code can be found on
2. Apprentices, they work autonomously.

Solenoid valves

Found a pdf explaining different types of solenoid valves and their power consumption.Holding amperes in case of latching Solenoid valves is for momentary time of around 20-50 ms reducing power consumption significantly. Here’s the link

Latching solenoid valves

Some of the specifications of the latching solenoid valve related to our applications are:

- operates at 6v dc and 650ma
- will last around 500K cycles
- operate normally till about 45 degree celcius(ambient temp)
- operating pressure upto 6 bar

Key points to note:

- Power rating varies from about 7 to 11 watts. Considering the duration of the pulse this would be a good change from solenoid valves and their continuous power consumption.
- some of the important features and working of latching valves can be found at
- Latching solenoid valves specifications matching irrigation applications: port size : 3/4’‘and G3/8’‘˜G2”
- 2 port direct acting normally closed solenoid valves : model no L2003 and L2004
- Availaible online at stores like Indiamart and Alibaba with price ranging from 3-8 US dollars per piece

Batteries

Options for Rechargeable batteries we can use :

1. lead acid battery
2. alkaline batteries
3. nickel-cadmium batteries.

Alkaline rechargeable batteries would be suitable for our low power applications. Some of the important features of these batteries are:

- They have a life of around 5 years
- They do not self discharge much(0.3%/month)
- They are compact and Easy to maintain as well.
- They are cheap and easily replaceable

Experimentation

- The batteries need to provide a minimum of 6V or higher
- They need to be 2500maH or greater
- should be able to supply a minimum of 650 ma
- Tried with Alkaline rechargeable batteries and Duracell rechargeable batteries
- Make sure to use the batteries of the same make together lest one of the batteries discharge faster and also discharge the other batteries along with it

Power consumption

1. **Solenoid Valve**

   Power rating of each latching solenoid valve ranges from 7-11 watts.The energy consumed is significantly reduced when compared to the solenoid valves without latching, as the time duration of the pulse is very less, around 50ms.

2. **ESP8266 WIFI module**

   A detailed report of its power consumption can be found later in the wiki

Tasks to accomplish

1. Procure the Latching valve and wireless module
2. Design the circuit ready for controlling the valves and check its working
3. Install necessary software and try and interface the wireless module with the comp
4. Communicate with the valve remotely

5. Install openHAB and MQTT broker
6. Control the valve using openHAB GUI
7. Work on device discovery and persistence
8. Make the PCB for the circuit and try to reduce unnecessary components
9. A detailed report on the power consumption of the setup
10. Make a compact and portable setup

ESP8266 WiFi module

ESP8266 is a low power WIFI module which is very much suitable for our irrigation application. Some of the important features of the ESP8266 are:

- Its an ultra low power module which works on 3.3V. This is ideal for IoT applications
- It has high on-chip integration
- It has GPIO support for the application devices that you want to control
- It has a 32 bit CPU
- It is very small in size
- It is a low cost module
- It can act as a standalone device without the need of any external controller

Progress stages and key points

- Installed CCS successfully but later found that this step was unnecessary.you can skip this step.Download CCS from TI website and follow this link for instructions
- A variety of AT commands are mentioned in the link which will be useful to send commanda to the ESP from the serial terminal
- The code for ESP8266 can be written in *C* or *python* with the help of the SDK which has extensible functions which are useful
- Not much help exists about coding ESP module using *Energia*
- ESP8266 also has MQTT libraries which can be included for its use
- We can use arduino IDE for programming as there is great support for Arduino from the ESP community
- All we have to do is flash the code on the ESP using FTDI and we can start programming with arduinoIDE

We chose to go with ESP8266 over CC3200 because the ESP module had all the functionalities that we needed and was cheaper, lighter and better for IoT applications.

IDE for ESP8266

1. ESP8266 comes with the NodeMCU firmware loaded and it accepts LUA script which can be loaded into the board using ESplorer IDE, but the use of this IDE lacks proper documentation.

2. Download ESplorer from the website

3. LUA is an embeddable, fast, powerful and light script. This is useful in data description and has procedural syntax. It also has some other important features like: Automatic memory management and Garbage collection

4. One other way of loading code into the ESP is using the Arduino IDE.This directly loads code into the EEPROM and the NodeMCU firmware can be loaded back in the ESP if needed

5. Clone arduino IDE fro ESP-Arduino Github repository and follow the installation procedure

MQTT in ESP8266

- The ESP8266 community has come up with a MQTT library which can be useful to implement MQTT protocol. Go to ESP-mqtt Gitub repository

- MQTT can also be implemented using ESP8266 along with Arduino board using espduino library. clone from ESP-duino github repository. This is not useful to us as it requires Arduino as an external micro-controller

- MQTT functions are already present in the nodeMCU firmware to be used

- while using Arduino IDE and additional library called *Pubsubclient* needs to be downloaded and placed in the libraries folder. Go to the pubsubclient github repository

Troubleshooting

1. After installation of Arduino if an error saying *JAVA exception not supported in headless version* pops up then you have to install the nonheadless version of JAVA.Installed open-JDK-non-headless version and configured to a new java and the installation problem was solved. Refer this

Design and Testing

After successfull installation of the forked version of arduino, testing of ESP8266 is done. The components used are:

- USB to TTL converter
- wires and jumpers
- ESP module
- 1k resistors

Before you begin:

1. Test the USB to TTL module for its working.
2. Rig up the circuit and try loading programs onto the ESP using arduino IDE. Test using sample codes.

Key points to note while flashing code

- GPIO 0 is connected to ground
- Take care of the TX-TX and TX-RX connections.The connections might be direct or cross coupled depending on the USB to TTL module used.
- The reset of the ESP module and converter are both connected together and pulled high
- Take care to set the baud rate right.115200 works just fine.
- choose the ESP board, port connected and set CPU frequency to 80 Mhz

Troubleshooting

1. If esp-sync and esp-comm failed error then recheck the above mentioned.
2. If port is greyed out then try changing ownership of port by >`sudo chown $username /dev/ttyUSB`

Testing GPIO's and wifi connection
Some of the key points observed during testing:

- While testing the GPIO pins of the ESP8266, found that the code only works if the GPIO 0 pin is removed from ground(Which is required while loading code)
- Sample codes like wifiscan are working fine. If The ESP is able to connect to the WIFI's present but the webpage hosted the server is not acccessing check authentication details

- Since there was authentication hassle with IITB-Guest wifi that we are using, we have decided to setup a local server(WAMP) and a wifi hotspot on one of our laptop. Our testing code will enable the ESP8266 to fetch the html page from the local server and display it on the console.

Test codes

1. The wifiWebServer test code was run on the ESP module and a web server was successfully hosted on the ESP.The GPIO were controlled through the Web server page
2. The solenoid test conducted and verified the working of the solenoid in the desired fashion
3. The solenoid was controlled using the web server hosted on the ESP

The circuit for controlling the solenoid valves

Components required:

- L298N Dual H-bridge
- Power supply or battery
- wires and jumpers
- Latching solenoid valve
- Arduino Mega ADK(any microcontroller)

Steps

- The GPIO pins of the arduino are controlled and given as input to the IN1,IN2 to control direction and EN1 to control the PWM pulse
- The combination of IN1 and IN2 values control the direction of motion of the coil.When IN1 is high and IN2 is low the latch opens and when the vice versa happens the latch closes
- As the H-bridge works at 12v(given to the coil) therefore the duration of the pulse should be around 20ms for optimum working
- LT293 H-bridge driver IC also can be used to test the circuit
- This is another alternative to the same, Using dual Hbridge driver DRV8841

nodeMCU firmware

Reasons for Switching from Arduino IDE to nodeMCU firmware

- Arduino IDE doesn't provide us the liberty to do tweaks at the basic level which is possible through the nodeMCU firmware
- It cant access the sleep modes of the ESP which are important for our IoT application
- checkout the nodeMCU API
- nodeMCU firmware comes inbuilt with the ESP8266 and it uses LUA script which is a fast and powerful script
- Incase you have erased the original firmware then you can reflash it by downloading the firmware here
- It can be flashed through the esptool. Download ESPTOOL here
- Follow the instructions given here

Insatllation procedure:

- Download nodeMCU latest firmware from the link above
- Download ESPTOOL
- Move to the directory containing esptool and execute the following command

  ```
  sudo python esptool.py --port /dev/ttyUSB0 write_flash 0x00000
  The_Path_To_The_NodeMCU_Firmware.bin
  ```

- Make sure you are connected to the right port and change can access it. Else change ownership using

  ```
  Sudo chown $username /dev/ttyUSB*
  ```

note: Make sure GPIO 0 is connected to ground while flashing
ESPlorer IDE

- The ESPlorer IDE is a great tool to work with the ESP8266 making the loading of code into the ESP an easy task
- Download ESPlorer here
- ESPlorer is a platform that helps you load lua scripts into the ESP. follow this for more help
- For the GPIO mapping refer this
- Refer this for some more documentation regarding nodeMCU

Troubleshooting

- ESPlorer needs JAVA 7 or higher. install JAVA 8 for ubuntu by following these steps After downloading ESPlorer make your way to the folder in which ESPlorer.jar exists and run this command in the terminal >`java -jar ESPlorer.jar` and your done
- esptool depends on for serial communication with the target device.

If you choose to install esptool system-wide by running `python setup.py install`, then this will be taken care of automatically.

If not using `setup.py`, then you'll have to install pySerial manually by running something like `pip install pyserial`, `easy_install pyserial` or `apt-get install python-serial`, depending on your platform. (The official pySerial installation instructions are This utility actually have a user interface! It uses and is rather self-documenting. Try running `esptool -h`. Or hack the script to your hearts content. The serial port is selected using the `-p` option, like `-p /dev/ttyUSB0` (on unixen like Linux and OSX) or `-p COM1` (on Windows). The perhaps not so obvious corner case here is when you run esptool in Cygwin on Windows, where you have to convert the Windows-style name into an Unix-style path (`COM1` -> `/dev/ttyS0`, and so on).

Interfacing MOSCA and ESP

- Install MOSCA on your system
- Setup the server using *mosca -v –host $hostname | bunyan*
- Burn the LUA scripts on the ESP8266
- send subscribe request from your terminal using >`node client-sub.js` (go to the apprpriate folder first)
- send publish command from the terminal >`node client-pub.js`
- see the sent data on your terminal with confirm message

Troubleshooting

ESP8266

- take care to use the delays at appropriate places so as to send out publish and subscribe requests
- LUA syntax must be kept in mind.
- Use >`tmr.alarm` instead of >`tmr.delay` wherever possible

MOSCA server

- The local IP must be changed in the the ESP and also the >`client-pub.js` and >`client-sub.js` scrits
- The topic must be the same in the >`client-pub.js` and subscribe code in the ESP

Protocols for communication

The Protocols that can be looked into are:

- MQTT
- MQTT-SN
- The skinny call control protocol(SCCP)
- 6LOWPAN
- LoRaWAN
- Constrained application protocol(COAP)
- Light-weight M2M(LWM2M)
- The simple thing protocol

Material for reading

1. Here is a Pdf explaining the LWM2M protocol. This is a very specific M2M protocol for Internet of things applications.This protocol intergrates COAP in it as well
2. COAP protocol is widely used in IOT applications. Ideal for low power and constrained memory usage applications This could make use of sleep-wakeup model as mentioned in the pdf
3. MQTT-SN is also a very lightweight sensor application protocol. MQTT-SN protocol is just an extension of this protocol where the communication of sensors and actuators is given importance.There are certain changes here and there which would be ideal for sensor communication
4. Additional features of MQTT-SN are explained here
5. Comparison and Pros and cons of MQTT and COAP at the application layer

Message queing telemetry transport protocol

Some of the important features of MQTT protocol:

- Its a low power protocol which is useful in IoT applications
- Its a low bandwidth protocol
- Even though its unreliable because it doesn't error checking overheads this makes it very lightweight
- It offers three different qualites of service(QOS)
- It is a publish/subscribe model which is ideal for our irrigations application
- ESP8266 has inbuilt support for MQTT with all the important functions
- MQTT protocol is explained in detail here
- Look through MQTT as a protocol and its API documentation
- It runs on UDP therefore doesn't need to maintain the connection information
- Simple to use
- Less processing at both ends

These are the reasons we decided to go with MQTT protocol for our application.

After successfully installing MOSCA broker on the system, you are now good to go on next step.

Running MOSCA

Configuring IP on which MOSCA has to be run.

on the terminal type >`mosca -help`, it will display all the handles which mosca currently supports we are going to use handles, -v, –host, | bunyan. *To start the server,on the terminal type >'mosca -v –host "ur ip" |bunyan* server testing can be done via two codes included in the mosca folder.

NOTE: for opensuse users, make sure that entry for tcp port 1883 is in the firewall for inbound connection.

- to run the code, first run client-sub.js on a diiferent terminal, and then run client-pub.js on a second terminal.
- message from the topic subscribed will be printed where client-sub.js has been run.

Configuring MOSCA with openHAB

After successfully running mosca on the system, it is time to start running openHAB.

- copy required addons (mqtt binding, mqtt-transport binding, persistence binding etc) from the addon zip downloaded from the link to the openhab> addon folder of your system.
- create openhab.cfg from openhab-default.cfg file and add following details
- under mqtt transport section update following lines

```
javacript mqtt:broker.url=tcp://192.168.0.100:1883 mqtt:broker.clientId=openhab
```

- your system is now ready for openHAB to start and test a demo site for glowing a led on esp8266 module.
- copy content of the folders to the respective folder in the openHAB
- Open browser and type >`localhost:8080/openhab.app?sitemap=test`
- toggle the switch and you can see led going on and off.

Table Description

Database name: iot

Tables in database:

- devices
- groups
- sensors
- tasks

Device table

Name: device

Field	Type	Null	Key	Default	Extra
id	int(255)	NO	PRI	NULL	auto_increment
name	varchar(255)	YES		NULL	
macid	varchar(30)	NO		NULL	
group	varchar(255)	YES		NULL	
status	int(1)	YES		NULL	
battery	varchar(10)	YES		NULL	
seen	timestamp	NO		CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP
action	int(1)	YES		NULL	
type	varchar(255)	YES		NULL	

Group Table

Name: groups

Field	Type	Null	Key	Default	Extra
id	int(10)	NO	PRI	NULL	auto_increment
name	varchar(255)	NO		NULL	

Sensor Table

Name: sensors

Field	Type	Null	Key	Default	Extra
id	int(255)	NO	PRI	NULL	auto_increment
name	varchar(255)	NO		NULL	

Tasks Table

Name: tasks

Field	Type	Null	Key	Default	Extra
id	int(255)	NO	PRI	NULL	auto_increment
item	varchar(25)	NO		NULL	
start	int(4)	YES		NULL	
stop	int(4)	YES		NULL	
action	int(1)	YES		NULL	

Library used

Uses php library **SSKAJE MQTT** for communicating with MOSCA broker.

About the website

After studying different home automation server, we found that, we need to create an engine, which will work in the background performing following automated tasks:

1. Device discovery –> `subscribe.php`
2. Valve and other sensors scheduling –> `tasks.php`
3. Listening for battery status –> `battery.php`
4. Listening for Moisture value –> `moisture.php`

Tasks.php is executed every minute using cron and above other files are run continuously in php cli mode.

Power Analysis

ESP8266

Modes of operation of ESP8266

- Active mode : Consumes a peak current of about 200ma. The power consumption would be 0.7W

The high speed clock is operational and sent to each block enabled by the clock control register. Lower level clock gating is implemented at the block level, including the CPU, which can be gated off using the *WAITI* instruction, while the system is on.

- Sleep mode : consumes about 0.9ma typically and the power consumption would be around 3mW

Only the RTC is operating. The crystal oscillator is disabled. Any wakeup events (MAC, host, RTC timer,external interrupts) will put the chip into the WAKEUP state.

- Wakeup mode : consumes a peak current of 60 ma during wakeup.

In this state, the system goes from the sleep states to the PWR state. Thecrystal oscillator and PLLs are enabled.

- Deep-sleep mode : consumes about 1ma and about 8uW of power in this mode

Only RTC is powered on –the rest of the chip is powered off. Recovery memory of RTC can keep basic Wi-Fi connecting information.

To be able to implement deep sleep (without adding extra hardware to generate a wake-up signal), you need to link the *RESET pin to the GPIO 16 pin.*By linking these, you lose the ability to use GPIO16, but gain an automatic wake-up from deep sleep after the number of microseconds that you specify in the `node.dsleep(time_in_us)` call.

When the ESP wakes from sleep it runs init.lua file again.

It does not require Wi-Fi connection to be maintained. It's mainly for application with long time lags between data transmission.

ESP8266 need be waked up by GPIO16 or by other gpio of external MCU. Thus, GPIO 16 or external GPIO should be connected to RST pin.

- ***note*** : During flashing : consumes a peak current of 75 ma.

WIFI sleep modes

`wifi.sleeptype()`

- Configures the sleep type for wifi modem

Note: works only in wifi.STATION mode

- Syntax: type_actual = wifi.sleeptype(type_need)

parameters

- wifi.NONE_SLEEP: wake up device from LIGHT_SLEEP or MODEM_SLEEP mode
- wifi.MODEM_SLEEP: closes wifi modem, but CPU and other peripherals still work.

It saves power to shut down the Wi-Fi Modem circuit while maintaining a Wi-Fi connection with no data transmission. It requires the CPU to be working, as in PWM or I2C applications.

- wifi.LGIHT_SLEEP: close wifi modem, Oscillator and PLL, CPU and peripheral enters "clock-pause" status.

The CPU may be suspended in applications like Wi-Fi switch. Without data transmission, the Wi-Fi Modem circuit can be turned off and CPU suspended to save power.

Note: Both are automatically added by underlying, which can sleep and be waked up automatically. There is no need of any processing in hardware.Both can adjust themselves to support the type of DTIM according to the AP that connected.

Note: Power saving modes can only be enabled in station mode

Implementaton

`node.dsleep()`

Description: Enter deep sleep mode, wake up when timed out.

Syntax: node.dsleep(us, option)

Note: This function can only be used in the condition that esp8266 RST and GPIO16 are connected together. Using sleep(0) will set no wake up timer, connect a GPIO to pin RST, the chip will wake up by a falling-edge on pin RST.

option=0, init data byte 108 is valuable; option>0, init data byte 108 is valueless.

More details as follows: 0, RF_CAL or not after deep-sleep wake up, depends on init data byte 108. 1, RF_CAL after deep-sleep wake up, there will belarge current. 2, no RF_CAL after deep-sleep wake up, there will only be small current. 4, disable RF after deep-sleep wake up, just like modem sleep, there will be the smallest current.

Parameters:

- **us**: number(Integer) or nil, sleep time in micro second. If us = 0, it will sleep forever. If us = nil, will not set sleep time.
- **option**: number(Integer) or nil. If option = nil, it will use last alive setting as default option.

Returns

- nil

Example

–do nothing
`node.dsleep()`
–sleep s
`node.dsleep(1000000)`
–set sleep option, then sleep s
`node.dsleep(1000000, 4)`
–set sleep option only
`node.dsleep(nil,4)`

Note: `node.dsleep(0)` makes the ESP sleep forever until the RST pin is brought low

Steps

- Connect the GPIO 16 to ground and connect the multimeter in series with the battery powering the module
- Change the setting of the multimeter to read current
- Add this line of code in your program

```
node.dsleep(time_in_us)
```

- After this you must see the version of NodeMCU firmware on your screen and the init.lua file running again
- If not there might be a problem with the hardware reset and other options have to be sought

WiFi sleep modes

- Checked using the functions given above in different WiFi sleep modes, But could not see any significant change in the current drawn
- The current consumed in Light_Sleep mode stays to be around 70 ma and in Modem_Sleep to be around 80 ma
- The deep sleep is a better option when occasional reception of data is required as it can go down to as low as 1 ma, maybe even to micro-amperes in some cases, but the problem of the hardware reset is yet to be rectified.
- When the ESP looses its connection with the network, it continuously scans for any available network and in the process consumes a lot of current. Hence it is meaningful to give the ESP a finite number of tries to connect and then just make it go to deepsleep mode.

Graphs showing current consumption

- Average current consumption of the ESP8266 module during its various operations over time

- Peak current consumption of the ESP8266 module during its various operations over time

- A graph showing the breakup between the average current consumed by the H-bridge(theoretical) and the ESP8266 module

Key Points

- The current consumption peaks during message reception and publish time
- The H-bridge consumes more current to turn the latch on than when turning it off
- Overall current peaks up to 45mA at an average
- The peak current has a possibility to spike up to 75mA
- There is a wide fluctuation in the current being drawn even during idle time therefore there is a possibility that the battery might drain out if the power is not switched off at regular intervals

Observations

- At idle we can observe an average current of about 20mA
- A 2500 mA will last for a maximum of 4 days

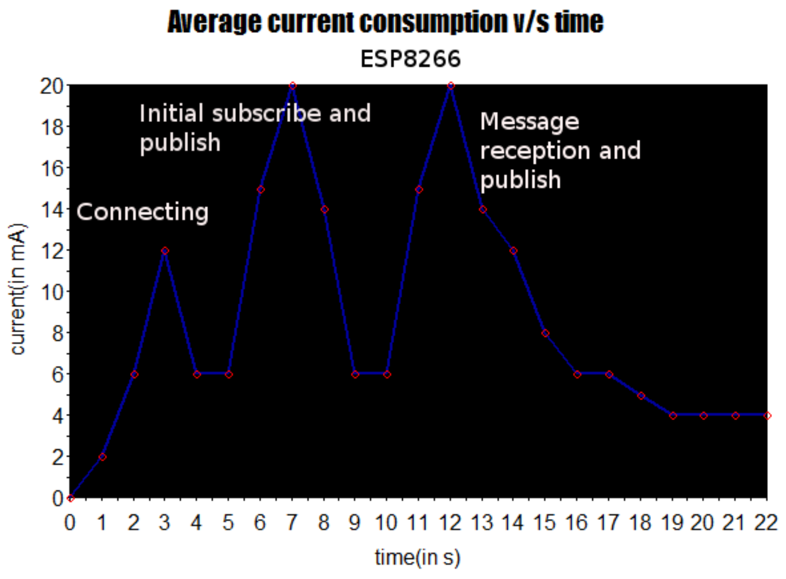

Figure 1: Average

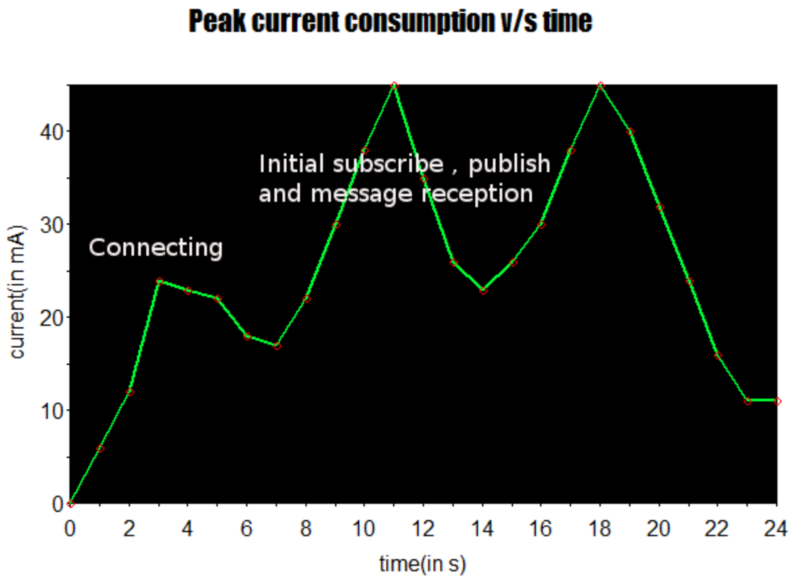

Figure 2: Peak

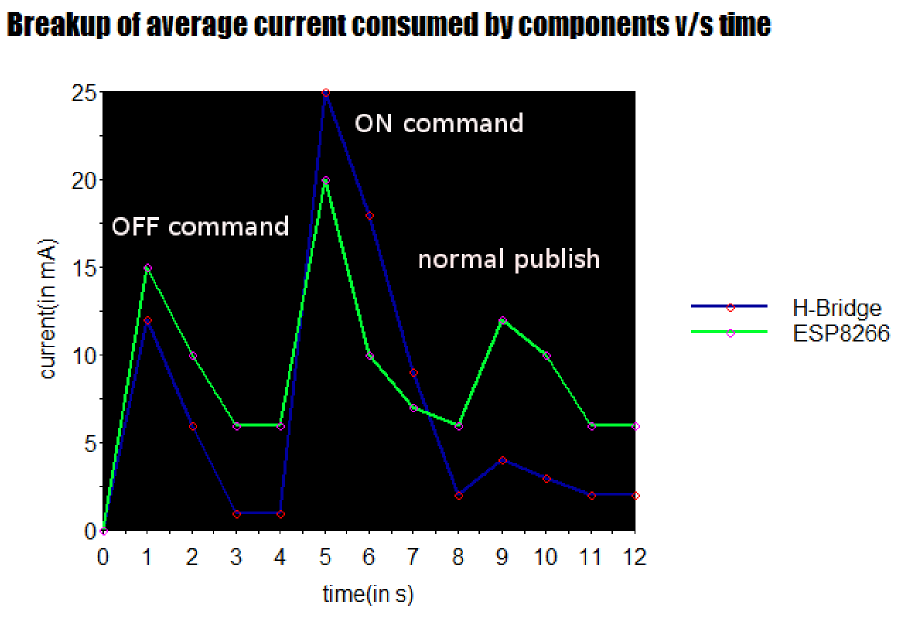

Figure 3: Breakup

- The sleep option of the ESP have to be tapped and as the `node.dsleep()` function is not working the options we can look into are:

1. Use a cheap external controller to wake the ESP8266
2. Change the ESP8266 firmware to one of the SDK versions
3. Wait for an update to the nodeMCU firmware

Troubleshooting

- Even after connecting the GPIO 16 pin to the RESET pin the ESP doesn't wake up to its initial state. It wakes up to some random condition and halts the ongoing process
- The ESP8266 has bugs with its software RESET. Therefore an external controller might be required to pull its reset pin low if one wants to explore the sleep modes of ESP

Follow this link for additional info

Future plans

In the future we could try to integrate solar power into the module. The Valve would still be driven by the battery but the ESP8266 module could be driven by Solar cells during the day and charge the battery during the night.

Problems

- The amount of current a solar cell can give is very less, about 40ma
- The ESP module requires a minimum of 350 ma. So it would take many such cells to give power to the ESP which would in-turn turn out to be costly.

Integrating Moisture sensor

YL-69 Soil Hygrometer Humidity & Soil Moisture Detection Sensor

This is an Electrical resistance Sensor. The sensor is made up of two electrodes. This soil moisture sensor reads the moisture content around it. A current is passed across the electrodes through the soil and the resistance to the current in the soil determines the soil moisture. If the soil has more water resistance will be low and thus more current will pass through. On the other hand when the soil moisture is low the sensor module outputs a high level of resistance. This sensor has both digital and analogue outputs. Digital output is simple to use but is not as accurate as the analogue output.

Specifications

- Vcc power supply : 3.3V or 5V
- Current : 35mA
- Signal output voltage : 0-4.2V

- Digital Outputs : 0 or 1
- Analog : Resistance ()
- Panel Dimension : 3.0cm by 1.6cm
- Probe Dimension : 6.0cm by 3.0cm
- GND : Connected to ground

Key Features

- The sensor comes with a small PCB board fitted with LM393 comparator chip and a digital potentiometer.
- Digipots are used mostly in scaling analog signals to be used in a microcontroller.
- Digipot output resistance is variable based on digital inputs and thus also know as resistive digital-to-analog converters (RDACs).
- Available for cheap (Rs. 150)
- Efficient in power savings as works on 3.3 volts
- Operates on a range of voltages so that the wrong voltage wont spoil the sensor
- Not bulky but light and slender, which is an advantage for many applications
- Soil moisture module is most sensitive to the ambient humidity is generally used to detect the moisture content of the soil

Implementation

- Make the PCB which encompasses the ESP module, YL-69 moisture sensor and the YL-69 PCB
- Continuously powering the sensor module will corrode the moisture sensor overtime. Therefore control the power to the moisture sensor using a GPIO pin of ESP8266
- The Analog out pin A0 of the sensor module is connected to the ADC pin of ESP8266
- The signal voltage range is 0 - 4.2V and this has to be brought to 0 - 1.2v. Voltage divider with 1K and 3K are used for the same
- Make sure to keep common grounds
- The Digital out D0 can also be used to check whether the moisture reaches beyond a certain threshold

- After the installation check the values that the sensor gives for dry, wet and humid soil and process accordingly

Schematic and PCB Design

Schematics in KICAD

Install KICAD from the ubntu software centre along with the additional addons and tools.

Here is a tutorial that will guide you in the making of schematics using

- If you want to delete, add or functionally change a component, it's best to change the schematic and repeat the entire process again
- Make changes in EESCHEMA, re-annotating the components if necessary

Netlists and footprints

- After completing the schematic, generate a netlist that contains all the components and their connections
- Save netlist in EESCHEMA (schematic editor)
- After generating the netlist associate footprints in CVPCB
- Each component has its own footprint to be associated to it
- After associating all the components to their respective footprints load the netlist in PCBNEW (backup the .brd first!)
- Sometimes the required footprints are not available in KICAD but you can make your own footprints in CVPCB

PCB designing with PCBNEW

Here is a list of tutorials for PCB making