Dedicated To

My Family

Who supported me thick and thinPreface

First, Congratulations on investing your hard-earned money on this book. Who should read this book past this page?? A Techie by heart, A Hobbyist by nature & A Hacker by chance. Someone looking forward to have a rocking time with this magical box, labeled Raspberry Pi.

The Author was as excited as you are right now to begin his hacking spree, but it was when I faced numerous challenges (both on Hardware and Software level), that I decided to capture my daily learning’s in the form of simple notes on notepad. And then, it all started, with my collection of simple notes to capture new learning’s, which has now taken the shape of a book.

The basic idea behind publishing this book, is to cut short the plethora of information available on various websites on how-to build System Image for Raspberry-Pi, and to help readers quickly jump to the practical stuff that matters, without wasting time and energy on.

Everything said and done, I would personally love to hear from you, your suggestions to make this book even better.

Regards,

Sudhanshu Gupta

Founder, CEO

[www.softwaresunleashed.com](http://www.softwaresunleashed.com)

E-mail: softwares.unleashed@gmail.com

# About the Author

Sudhanshu Gupta (Founder – Softwares Unleashed), is Bachelors in Electronics & Tele-Communications & M.S. in Softwares Systems, and has been associated with Telecom domain and Embedded Software development since 2004.

He had worked with Major Industry giant(s), LG, Infineon, Intel to name a few. Sudhanshu during his stint with the corporate world, has contributed to numerous success stories of Big OEMs (LG, Samsung, Nokia) …cutting short the list.

He is now on a fast track to take his passion forward, i.e. Application Development and Sharing his Technical Knowledge for the benefit of others.

# Table of Contents

About the Author 3

Table of Contents 4

About Raspberry Pi 6

Support Repo – RPI\_IOT\_KERNEL 7

FIRMWARE 7

LINUX 7

TOOLS 7

SUPPORT 7

Preparing Raspberry Pi for First Boot 8

Hardware Requirements 8

Install Raspbian on Your Pi and Connect to It Over SSH 8

Third Party Operating System Images 9

Image Flashing Softwares 10

Setting Wi-Fi Up via Command Line 11

GETTING WIFI NETWORK DETAILS 11

ADDING THE NETWORK DETAILS TO THE RASPBERRY PI 11

UNSECURED NETWORKS 12

HIDDEN NETWORKS 12

ADDING MULTIPLE WIRELESS NETWORK CONFIGURATIONS 13

Enable SSH (with Keyboard and Monitor attached) 14

Enable SSH on a Headless Raspberry Pi 14

Safely Shutting down RPi 15

Step by Step Guide (Building Custom Linux Kernel) 16

1) Get the kernel source 16

2) Get the cross-compiler 16

3) Install necessary packages 17

4) Code Compilation 18

5) Preparing the SYSTEM IMAGE 20

6) Transfer the Kernel Image 21

7) Copy Device Tree Blobs (Recent Kernels Only) 22

8) Building the Device Drivers (Modules) 22

9) Updating the GPU (Graphics Processing Unit) firmware 22

10) Transfer the firmware 23

Understanding Device Tree 24

DEVICE TREES 25

1.1: BASIC DTS SYNTAX 25

1.2: AN ASIDE ABOUT /INCLUDE/ 27

Transferring files between VirtualBox and Host OS 27

Shared Folder between Host & Guest OS 27

Step 1: Share a folder on the host OS 28

Step 2: Mount the folder in the guest OS 28

Secure Copy (SCP) between Host & Guest OS. 28

Using a USB card reader 29

IoT & RaspberryPi 30

1) Remote Temperature Monitor using ESP8266 MCU Node & RPi 30

Hardware Requirements 30

Hardware Configuration 31

Testing the Sensor – Dallas DS18B20 32

Trouble Shooting Guide 36

Troubleshooting Power Problems 36

How to test the F3 polyfuse 37

Red power LED is on, green LED does not flash, nothing on display 38

How to Allow Incoming SSH (SCP) connections on MAC OS? 40

Connection refused error when trying to SCP 40

Custom Kernel Image Doesn’t Boot 41

Wi-Fi Dongle Issues – Doesn’t camp to Wi-Fi Network 42

USB Stick not detected on RPI 46

Installing LoLin ESP8266 Board Configs in Arduino IDE 47

Flash Sketch Code (LED Blink Test) in ESP8266 using Arduino IDE 52

Error While Flashing on Arduino / ESP8266 Wi-Fi Node MCU 54

Further Reading 55

Legends 56

# About Raspberry Pi

The Raspberry Pi is a series of credit card-sized single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. Now over four years old, the Raspberry Pi, a cheap credit card sized computer, has taken the computing and DIY world by storm.

The original model became far more popular than anticipated, selling outside of its target market for uses such as robotics.

Per the Raspberry Pi Foundation, over 5 million Raspberry-Pi(s) have been sold before February 2015, making it the best-selling British computer.

#### Overview

Several generations of Raspberry-Pi(s) have been released. The first generation (Raspberry Pi 1 Model B) was released in February 2012. It was followed by a simpler and inexpensive model Model-A. In 2014, the foundation released a board with an improved design in Raspberry Pi 1 Model B+. The model laid the current "mainline" form-factor. Improved A+ and B+ models were released a year later. A cut down "compute module" was released in April 2014, and a Raspberry Pi Zero with smaller size and limited input/output (I/O) and general-purpose input/output (GPIO) abilities was released in November 2015 for US$5. The Raspberry Pi 2, which added more RAM, was released in February 2015. Raspberry Pi 3 Model B released in February 2016 is bundled with on-board Wi-Fi and Bluetooth. As of December 2016, Raspberry Pi 3 Model B is the newest mainline Raspberry Pi. These boards are priced between US$5–35.

All models feature a Broadcom system on a chip (SoC), which includes an ARM compatible central processing unit (CPU) and an on-chip graphics-processing unit (GPU, a Video Core IV). CPU speed ranges from 700 MHz to 1.2 GHz for the Pi 3 and on board memory range from 256 MB to 1 GB RAM. Secure Digital (SD) cards are used to store the operating system and program memory in either the SDHC or MicroSD sizes. Most boards have between one and four USB slots, HDMI and composite video output, and a 3.5 mm phone jack for audio. Lower level output is provided by several GPIO pins which support common protocols like I²C. The B-models have an 8P8C Ethernet port and the Pi 3 has on board Wi-Fi 802.11n and Bluetooth.

The Foundation provides Raspbian, a Debian-based Linux distribution for download, as well as third party Ubuntu, Windows 10 IOT Core, RISC OS, and specialized media center distributions. [8] It promotes Python and Scratch as the main programming language, with support for many other languages. The default firmware is closed source, while an unofficial open source is available.

# Support Repo – RPI\_IOT\_KERNEL

This guide comes equipped with support material to speed up and enhance your learning experience, by collating all the necessary tools, kernel, scripts at a single repo. Just in case we missed it, you need to have an account created on [www.github.com](http://www.github.com) to access this support repo.

Git Repo Link: <https://github.com/softwaresunleashed/rpi_iot_kernel.git>

Create an empty folder and type in following command on shell prompt (and wait for few mins):

$ git clone --recursive https://github.com/softwaresunleashed/rpi\_iot\_kernel.git

Note: GIT is a pre-requisite to this step, hence, if GIT isn’t installed yet on your environment, execute the following command to install GIT on your system

$ sudo apt-get install git

This repo contains the following sections:

1. Firmware
2. Linux
3. Tools
4. Support

## FIRMWARE

Official RPi GPU / Firmware (bootloader) are part of this folder.

## LINUX

Based on Official RPi Linux 4.9.x kernel. This kernel includes kernel modifications, scripts, drivers etc. done for learning concepts which are detailed throughout this book.

## TOOLS

Based on Official RPi toolchain package. This toolchain shall be helpful in compiling code or generating RPi understandable binary format.

## SUPPORT

All the support files, misc. scripts, that will assist you throughout this book are added in this folder.

# Preparing Raspberry Pi for First Boot

## Hardware Requirements

1. Raspberry Pi (of course absolute yes!!)
2. Power adapter (your micro usb phone charger will do).
3. Memory Card (MMCSD Class 10 preferred)
4. Wi-Fi dongle (USB) (good to have)
5. Ethernet Cable to connect to network. (If you don’t have a Wi-Fi dongle)
6. USB Keyboard (optional)
7. HDMI Monitor (optional)

You could hook your Raspberry Pi up to a keyboard and monitor and set things up that way, or you can connect to your Pi over SSH and run every step from the comfort of your laptop. Although my personal favourite is SSH method, which is much easier than begging your friend for a random monitor.

## Install Raspbian on Your Pi and Connect to It Over SSH

For the First ever time, you need to insert or flash (as developers like to call it) a Pre-Built on to a SD (Rpi2) / MicroSD card (Rpi3) and insert it in into the designated slots on our magic machine.

You’ll need to choose from one of the following options to flash onto your Raspberry Pi.

Two versions of Pre-Built (and more reliable) OS images for Rpi.

1. NOOBS / NOOBS Lite [ <https://www.raspberrypi.org/downloads/noobs/>]
2. Raspbian Lite / Pixel [ <https://www.raspberrypi.org/downloads/raspbian/> ]

On a newly installed Rasbian image (This is what I have been using most of my time), try connecting to RPI board via the following default credentials:

Username: pi (Default)

Password: raspberry (Default)

Note: You would need keyboard + HDMI monitor for first time login, since SSH is *not* enabled by default.

Lately, there have been substantial development on OS front, and several Third-party vendors have also come up with charming Linux distros apart from Noobs and Raspbian. In case you are in a mood to experiment, choose one variant that suits your taste buds from the section that follows on Third Party OS Images.

## Third Party Operating System Images

You may be interested in trying the following distros for Raspberry Pi, in addition to the ones mentioned above:

1. Ubuntu Mate [ <https://ubuntu-mate.org/raspberry-pi/> ]
2. Snappy Ubuntu Core [ <https://developer.ubuntu.com/core/get-started/raspberry-pi-2-3> ]
3. Windows 10 IOT Core [ <https://developer.microsoft.com/en-us/windows/iot/getstarted> ]
4. OSMC – Open Source Media Center [ <https://osmc.tv/> ]
5. LibreELEC [ <https://libreelec.tv/> ]
6. PiNet – Centralized Raspberry Pi Classroom [ <http://pinet.org.uk/> ]
7. RISC OS [ <https://www.riscosopen.org/content/downloads/raspberry-pi> ]
8. Rpi Weather Station [ <https://downloads.raspberrypi.org/weather_station/images/weather_station-2016-03-24/> ]

## Image Flashing Softwares

To flash / burn above OS images to memory card, we need certain specialized software(s) that takes OS Image file as input, converts them into a format that is understandable by target board, and finally copy the contents onto the memory card.

Following are the list of various softwares that can be used to burn RPi OS images onto memory card.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tool Name** | **Windows** | **Linux** | **OSX** | **Link** |
| Etcher | Y | Y | Y | https://etcher.io/ |
|  |  |  |  |  |

## Setting Wi-Fi Up via Command Line

[https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md]

This method is suitable if you don't have access to the graphical user interface normally used to set up Wi-Fi on the Raspberry Pi. It is particularly suitable for use with a serial console cable if you don't have access to a screen or wired Ethernet network. Note also that no additional software is required; everything you need is already included on the Raspberry Pi.

### GETTING WIFI NETWORK DETAILS

To scan for Wi-Fi networks, use the command

$ sudo iwlist wlan0 scan.

This will list all available Wi-Fi networks, along with other useful information.

Look out for:

1. 'ESSID:"testing"' is the name of the Wi-Fi network.
2. 'IE: IEEE 802.11i/WPA2 Version 1' is the authentication used. WPA2, the newer and more secure wireless standard which replaces WPA. This guide should work for WPA or WPA2, but may not work for WPA2 enterprise. You'll also need the password for the wireless network. The ESSID (ssid) for the examples below is ‘testing’ and the password (psk) is ‘testingPassword’.

### ADDING THE NETWORK DETAILS TO THE RASPBERRY PI

Open the wpa-supplicant configuration file in Nano:

$ sudo nano /etc/wpa\_supplicant/wpa\_supplicant.conf

Go to the bottom of the file and add the following:

network={

ssid="testing"

psk="testingPassword"

}

The password can be configured either as the ASCII representation, in quotes as per the example above, or as a pre-encrypted 32-byte hexadecimal number. You can use the wpa\_passphrase utility to generate an encrypted PSK. This takes the SSID and the password, and generates the encrypted PSK. With the example from above, you can generate the PSK with

$ wpa\_passphrase "testing" "testingPassword.

The output is as follows.

network={

ssid="testing"

#psk="testingPassword"

psk=131e1e221f6e06e3911a2d11ff2fac9182665c004de85300f9cac208a6a80531

}

Note that the plain text version of the code is present, but commented out. You should delete this line from the final wpa\_suplicant file for extra security.

### UNSECURED NETWORKS

If the network you are connecting to does not use a password, the wpa\_supplicant entry for the network will need to include the correct key\_mgmt entry. e.g.

network={

ssid="testing"

key\_mgmt=NONE

}

### HIDDEN NETWORKS

If you are using a hidden network, an extra option in the wpa\_supplicant file, scan\_ssid, may help connection.

network={

ssid="yourHiddenSSID"

scan\_ssid=1

psk="Your\_wifi\_password"

}

You can verify whether it has successfully connected using

$ ifconfig wlan0.

If the inet addr field has an address beside it, the Raspberry Pi has connected to the network. If not, check your password and ESSID are correct.

### ADDING MULTIPLE WIRELESS NETWORK CONFIGURATIONS

On recent versions of Raspbian, it is possible to set up multiple configurations for wireless networking. For example, you could set up one for home and one for school.

For example, if you have two networks in range, you can add the priority option to choose between them. The network in range, with the highest priority, will be the one that is connected.

network={

ssid="HomeOneSSID"

psk="passwordOne"

priority=1

id\_str="homeOne"

}

network={

ssid="HomeTwoSSID"

psk="passwordTwo"

priority=2

id\_str="homeTwo"

}

## Enable SSH (with Keyboard and Monitor attached)

[ <https://www.raspberrypi.org/documentation/remote-access/ssh/> ]

SSH can be enabled manually from the desktop:

1. Launch Raspberry Pi Configuration from the Preferences menu
2. Navigate to the Interfaces tab
3. Select Enabled next to SSH
4. Click OK

Alternatively, **raspi-config** can be used:

1. Enter **sudo raspi-config** in a terminal window
2. Select **Interfacing** Options
3. Navigate to and select **SSH**
4. Choose **Yes**
5. Select **Ok**
6. Choose **Finish**

## Enable SSH on a Headless Raspberry Pi

[ <https://www.raspberrypi.org/documentation/remote-access/ssh/> ]

For headless setup, SSH can be enabled by placing a file named **ssh**, without any extension, onto the boot partition (/boot) of the SD card. The file can be added by offloading the memory card from RPi and mounting it on Host Machine (Windows / Mac).

When the Pi boots, it looks for the ssh file. If it is found, SSH is enabled, and the file is deleted.

The content of the file does not matter: it could contain text, or nothing at all.

## Safely Shutting down RPi

Don’t mess with the wires / accessories while RPi is still powered on. Shutdown the Pi and wait for any blinking LEDs to stop and disconnect power before trying to connect / disconnect wires.

You can use either of the commands:

$ sudo shutdown -h now

Or

$ sudo halt

# Step by Step Guide (Building Custom Linux Kernel)

Building the system image for raspberry pie is easier than as depicted on various websites. This step-by-step guide will get your brand-new Raspberry Pi box up and running in no time (minus the complex jargon involved). RPi development is best done on an Ubuntu (Debian distro) with access to root privileges on shell. The guide assumes you have access to a shell with root privileges on an Ubuntu laptop / desktop.

NOTE: MAC OSX users may install Virtual Box with the latest Ubuntu distro installed. Then follow the instructions in the subsequent pages. Link to Virtual Box for MAC (<https://www.virtualbox.org/wiki/Downloads>). Also, install the extension pack provided on the website. Extension packs provide a bridge between your host environment (MAC OSX) and the guest environment (Linux Ubuntu)

## 1) Get the kernel source

First things first, we need to pull Linux Kernel source from WWW and store it locally on our hard drive.

~ # mkdir rpi\_4\_x\_xx

~ # cd rpi\_4\_x\_xx

~/rpi\_4\_x\_xx # git clone git://github.com/raspberrypi/linux.git

where x & xx are the major and minor release numbers for Raspberry-Pi Linux kernel. Latest RPI Linux Kernel version as of this writing was 4.9.y.

**Quick Repo Sync Tip:**

If you wish to save some time and download just the latest and greatest Linux Kernel available as of date, append “--depth=1” to the git clone command above. Believe me, it saves a lot of download time (and your broadband data cost), since most of the branching information is redundant to a hobbyist. However, if you are one of those geeky minds who would like to dwell into each commit that has ever been done on the repo, feel free to omit “—depth=1”.

## 2) Get the cross-compiler

Since, we shall be building Raspberry Pi system image on a machine, which would be running a CPU with different architecture, we would require to download a Cross-Compiler. A Cross-Compiler is a program that generates code for a target device (Raspberry Pi in this case), although it is executing on a completely different machine with completely different architecture.

Cross compiling from Linux (pre-built bmc2708\_armv6kz compiler)

~ # mkdir rpi\_tools

~ # cd rpi\_tools

~/rpi\_tools # git clone git://github.com/raspberrypi/tools.git <enter>

NOTE: Install git command if not already installed via the following command

# apt-get install git

## 3) Install necessary packages

* On Ubuntu shell execute the following command (requires root privileges)

# apt-get install gcc-arm-linux-gnueabi make ncurses-dev

* Some Editors are handful, if not already installed

# apt-get install vim-gnome

* For compiling 32bit kernel on 64-bit VM (e.g. Virtual Box), following library needs to be installed.

# apt-get install lib32z1-dev

* Installing QT libs for using Graphical Interface to select Kernel config options ($ make xconfig)

# apt-get install qt4-dev-tools libqt4-dev libqt4-core libqt4-gui

# apt-get install qtlib4\*

## 4) Code Compilation

#### 1#. Go to the build directory on PC

# cd <path\_to\_kernel\_source\_directory>/linux/

#### 2#. Firstly, ensure your build directory is clean:

<path\_to\_kernel\_source\_directory>/linux/ # make mrproper

#### 3#. Define Cross-Compiler & Processor Architecture

From this point on, if you are cross-compiling, please substitute <your\_compiler> with your compiler binary prefix (e.g. <your\_compiler>=arm-bcm2708hardfp-linux-gnueabi- or arm-linux-gnueabihf-) as each compiler will be named slightly differently. Check your toolchain folder for the compiler you wish to use to compile Linux Kernel for RPi.

Debug Note: Do *not* forget the hyphen (-) at the end of the compiler name. This has caused lots of time wasted and errors resulting in “xxxxxxxxx command not found”.

If you are building on the RPi (although we won’t recommend that as build process would be quite slow), remove ARCH=arm CROSS\_COMPILE=<your\_compiler> from each command.

#### 4#. Kernel Config File

You will want to get a working kernel configuration (.config) to start from.

If you are one of those lucky few, who got RPi with a pre-installed image loaded on the memory card, you can get the config file by executing following command on the shell (on the RPi):

/raspberryPi\_root\_folder # zcat /proc/config.gz > .config

& then copy .config file to your build directory (on PC).

PS: /proc/config.gz isn’t available on Raspbian Jessi distro. Support is available on Wheezy Raspbian distro only.

OR

**Alternatively**, the default configuration is available in the downloaded kernel source in <path\_to\_kernel\_src\_dir>/linux/arch/arm/configs/bcmrpi\_defconfig.

Copy (& rename) bcmrpi\_defconfig to .config in the build directory

OR

**Alternatively**, execute the following command on shell

<kernel\_src\_dir> $ export ARCH=arm

<kernel\_src\_dir> $ export CROSS \_COMPILE=arm-bcm2708hardfp-linux-gnueabi-

<kernel\_src\_dir> $ make bcmrpi\_defconfig

#### 5#. Setting Build Environment

Ensure that your configuration file is up-to-date by executing the following command...Textual, Sequential access to configuration parameters. (**Quite tedious)**

# make ARCH=arm CROSS\_COMPILE=<your\_compiler> oldconfig

OR

Optionally, if you want to tweak the configuration Graphically & more Organized -- **Better**), run this command on shell:

# make ARCH=arm CROSS\_COMPILE=<your\_compiler> <config\_targets>

Use any one of the following <config\_targets>:

config :- Update config using a Line-oriented program

nconfig :- Update config using a ncurses menu based program

menuconfig :- Update config using a menu based program

**xconfig :- Update config using a QT based front-end**

gconfig :- Update config using a GTK based front-end

NOTE: The configuration info is stored in ".config" file on exit from the configuration menu.

The file is in the "build artifacts folder" (if mentioned explicitly by the macro KBUILD\_OUTPUT)

e.g. In the build script we can mention the build output folder as

export KBUILD\_OUTPUT=\_build\_output\_folder

#### 6#. Let the Build Begin

Once you have made necessary changes in the Linux Kernel of RPi, you can trigger the build with the following command and have a cup of coffee or your lunch. This shall take time for the first fresh build. Incremental builds there-after shall be much less time consuming.

# make –j<N> ARCH=arm CROSS\_COMPILE=<your\_compiler>

If you are on a multi-core machine, you can make the build faster by appending -j<N> to the build command above. Where ‘N’ is the number of cores on your system plus one.

Quick Tip: Don’t bother to clean object files, in case changes are made only in source files…A change in header file, deserves a cleaner build with object files and library files deleted manually.

## 5) Preparing the SYSTEM IMAGE

Once your linux kernel is successfully built, you need to pack the kernel such that Raspberry Pi likes to have it. Follow the steps below…

#### 1#. Get Build Tools

Because of the way, the memory addresses are arranged in the Broadcom SoC (The CPU used on Raspberry Pi), you will need to prepare the compiled image, before uploading it to Memory Card.

If you haven't got the tools directory from the GIT repo, do so now:

# cd ~/rpi\_tools/compiler/tools

~/rpi\_tools/compiler/tools # git clone git://github.com/raspberrypi/tools.git

OR

DOWNLOAD\_FROM\_LINK\_TO\_TAR\_BALL:

https://github.com/raspberrypi/tools/

#### 2#. Make Image

In the toolchain set, there is a folder called mkimage. Enter this directory, and then run the following:

$ cd ~/rpi\_iot\_kernel/tools

$ ./mkimage/imagetool-uncompressed.py <kernel\_build\_dir>/arch/arm/boot/zImage

Update for Latest Raspberry Pi Kernel Versions :

With recent Rpi Kernels, the process of creating the image has changed as below

$ cd ~/rpi\_iot\_kernel/linux

$ ./scripts/mkknlimg ./arch/arm/boot/zImage kernel.img

Location of "kernel.img”:

Above command will output a file called "kernel.img" (in the same folder where the python script "imagetool-uncompressed.py" is located.)

Quick Tip [1]: Above python script expects boot-uncompressed.txt file to be present in the same folder as the imagetool-uncompressed.py script. Hence, to get rid of any errors, we need to be execute the python script from the “mkimage” folder, so that boot-uncompressed.txt is available to the python script.

Quick Tip [2]: If you get error regarding "python2" not available, try creating a soft link to python2 as follows: (not sure why this is needed, but it worked for me)

# ln -s /usr/bin/python2.6 /usr/bin/python2

## 6) Transfer the Kernel Image

Copy your new kernel.img file into the RPi boot partition, though preferably as a new file (such as kernel\_new.img) just in case it doesn't work. If you're building on the RPi, just copy the file to /boot.

If you use a different filename, edit **config.txt** change the kernel line:

---

# Comment out the below line

#kernel=kernel.img

# Add this new line in /boot/config.txt

kernel=kernel\_new.img

---

## 7) Copy Device Tree Blobs (Recent Kernels Only)

For relatively newer RPi kernels, copy Device Tree blobs onto the SD Card.

$ sudo cp arch/arm/boot/dts/\*.dtb /boot

$ sudo cp arch/arm/boot/dts/overlays/\*.dtb\* /boot/overlays

$ sudo cp arch/arm/boot/dts/overlays/README /boot/overlays

## 8) Building the Device Drivers (Modules)

Now you need to transfer the Device Drivers (aka Modules).

In the build directory, run the following (substituting <modules\_path> for a folder somewhere (e.g. ~/modules):

<path\_to\_kernel\_source\_directory>/linux/ # make ARCH=arm CROSS\_COMPILE=<your\_compiler> modules\_install INSTALL\_MOD\_PATH=<modules\_path>

The contents of this directory should then be copied into the RPi root directory.

NOTE: If you have rebuilt the new kernel with exactly the same version as the one that's running, you'll need to remove the old modules first. Ideally this should be done offline by mounting the SD card on another system.

## 9) Updating the GPU (Graphics Processing Unit) firmware

Your RPi should now be ready to boot the new kernel. However, at this point it's recommended (not necessary) that you update your GPU firmware and libraries. This is required if you've just moved from 3.2 to 3.6 as the firmware interface has changed.

The "firmware" and "boot files" should be updated at the same time to ensure that your new kernel works properly

"master" - This is the version of firmware currently used in Raspbian (i.e. it works with the 3.2 kernel).

"next" - This is a development branch which provides a newer GPU firmware to work with the updated drivers in the 3.6 kernel.

For the "master" branch:

<path\_to\_folder\_where\_firmware\_is\_to\_be\_stored> # git clone git://github.com/raspberrypi/firmware.git

For the "next" branch:

<path\_to\_folder\_where\_firmware\_is\_to\_be\_stored> # git fetch git://github.com/raspberrypi/firmware.git next:refs/remotes/origin/next

## 10) Transfer the firmware

In case you are using a Virtual Box setup on Windows / Mac environment, take a pause to refer the section “Transferring files between VirtualBox and Host OS” , before continuing with the following section.

a) Firstly, update the required boot files in the RPi boot directory with those you've downloaded. These are:

-1- bootcode.bin

-2- fixup.dat

-3- start.elf

Next, you need to copy the VC libraries over.

There are two copies of this: one for **hard float** and one for **soft float**.

To find the correct one that you should be using, run the following command (substituting the program name for your compiler binary as required):

# arm-none-linux-gnueabi-gcc -v 2>&1 | grep hard

If something prints out, and you can see **--with-float=hard**, you need the hard float ones.

NOTE: The current version of Raspbian uses hard float.

b) Remove the /opt/vc directory from the RPi root, then:

For hard float, copy vc from the hardfp/opt directory into /opt in the RPi root directory

Otherwise copy vc from the top-level opt directory into /opt in the RPi root directory.

# Understanding Device Tree

[ <https://www.raspberrypi.org/documentation/configuration/device-tree.md> ]

Raspberry Pi's latest kernels and firmware, including Raspbian and NOOBS releases, now use a Device Tree (DT) to manage some resource allocation and module loading by default. This was implemented to ease the problem of multiple drivers contending for system resources, and to allow HAT modules to be auto-configured.

The current implementation is not a pure Device Tree system – there is still some board support code that creates some platform devices – but the external interfaces (I2C, I2S, SPI), and the audio devices that use them, must now be instantiated using a Device Tree Blob (DTB) passed to the kernel by the loader (start.elf).

The main impact of using Device Tree is to change from "everything on", relying on module blacklisting to manage contention, to "everything off unless requested by the DTB". In order to continue to use external interfaces and the peripherals that attach to them, you will need to add some new settings to your config.txt.

Here are few examples :

===============================

# Uncomment some or all of these lines to enable the optional hardware interfaces

#dtparam=i2c\_arm=on

#dtparam=i2s=on

#dtparam=spi=on

# Uncomment one of these lines to enable an audio interface

#dtoverlay=hifiberry-amp

#dtoverlay=hifiberry-dac

#dtoverlay=hifiberry-dacplus

#dtoverlay=hifiberry-digi

#dtoverlay=iqaudio-dac

#dtoverlay=iqaudio-dacplus

#dtoverlay=audioinjector-wm8731-audio

# Uncomment this to enable the lirc-rpi module

#dtoverlay=lirc-rpi

# Uncomment this to override the defaults for the lirc-rpi module

#dtparam=gpio\_out\_pin=16

#dtparam=gpio\_in\_pin=17

#dtparam=gpio\_in\_pull=down

===============================

## DEVICE TREES

A Device Tree (DT) is a description of the hardware in a system. It should include the name of the base CPU, its memory configuration, and any peripherals (internal and external). A DT should not be used to describe the software, although by listing the hardware modules it does usually cause driver modules to be loaded. It helps to remember that DTs are supposed to be OS-neutral, so anything which is Linux-specific probably shouldn't be there.

A Device Tree represents the hardware configuration as a hierarchy of nodes. Each node may contain properties and subnodes. Properties are named arrays of bytes, which may contain strings, numbers (big-endian), arbitrary sequences of bytes, and any combination thereof. By analogy to a filesystem, nodes are directories and properties are files. The locations of nodes and properties within the tree can be described using a path, with slashes as separators and a single slash (/) to indicate the root.

### 1.1: BASIC DTS SYNTAX

Device Trees are usually written in a textual form known as Device Tree Source (DTS) and stored in files with a .dts suffix. DTS syntax is C-like, with braces for grouping and semicolons at the end of each line. Note that DTS requires semicolons after closing braces: think of C structs rather than functions. The compiled binary format is referred to as Flattened Device Tree (FDT) or Device Tree Blob (DTB), and is stored in DTB files.

The following is a simple tree in DTS format:

/dts-v1/;

/include/ "common.dtsi";

/ {

node1 {

a-string-property = "A string";

a-string-list-property = "first string", "second string";

a-byte-data-property = [0x01 0x23 0x34 0x56];

cousin: child-node1 {

first-child-property;

second-child-property = <1>;

a-string-property = "Hello, world";

};

child-node2 {

};

};

node2 {

an-empty-property;

a-cell-property = <1 2 3 4>; /\* each number (cell) is a uint32 \*/

child-node1 {

my-cousin = <&cousin>;

};

};

};

/node2 {

another-property-for-node2;

};

This tree contains:

* a required header: /dts-v1/.
* The inclusion of another DTS file, conventionally named \*.dtsi and analogous to a .h header file in C - see An aside about /include/ below.
* a single root node: /
* a couple of child nodes: node1 and node2.
* some children for node1: child-node1 and child-node2.
* a label (cousin) and a reference to that label (&cousin): see Labels and References below.
* several properties scattered through the tree.
* a repeated node (/node2).

**Properties** are simple key-value pairs where the value can either be empty or contain an arbitrary byte stream. While data types are not encoded in the data structure, there are a few fundamental data representations that can be expressed in a Device Tree source file.

Text strings (NUL-terminated) are indicated with double quotes:

string-property = "a string";

Cells are 32-bit unsigned integers delimited by angle brackets:

cell-property = <0xbeef 123 0xabcd1234>;

Arbitrary byte data is delimited with square brackets, and entered in hex:

binary-property = [01 23 45 67 89 ab cd ef];

Data of differing representations can be concatenated using a comma:

mixed-property = "a string", [01 23 45 67], <0x12345678>;

Commas are also used to create lists of strings:

string-list = "red fish", "blue fish";

### 1.2: AN ASIDE ABOUT /INCLUDE/

The **/include/** directive results in simple textual inclusion, much like C's **#include** directive, but a feature of the Device Tree compiler leads to different usage patterns. Given that nodes are named, potentially with absolute paths, it is possible for the same node to appear twice in a DTS file (and its inclusions). When this happens, the nodes and properties are combined, interleaving and overwriting properties as required (later values override earlier ones).

# Transferring files between VirtualBox and Host OS

Like multiple users, you may be using Virtual Box VM (running Ubuntu on top of it), to build Kernel images. Once Kernel image is built, you may have to transfer the image and firmware to memory card, before it can be deployed onto RPi.

To do this there are couple of methods :

1. Shared Folder between Host & Guest OS
2. Mapping Card Reader to Virtual SD Card (. vmdk)
3. Secure copy between Host & Guest OS.
4. Using a USB card reader

## Shared Folder between Host & Guest OS

[ https://ryansechrest.com/2012/10/permanently-share-a-folder-between-host-mac-and-guest-linux-os-using-virtualbox/]

### Step 1: Share a folder on the host OS

1. In VirtualBox, click your OS on the left and click on **Settings**.
2. Click on the **Shared Folders** tab.
3. Click on the **folder with the plus** on the right.
4. **Browse to a folder** of your choice in the folder path.
5. **Enter a folder name** with no spaces e.g. “Share”.
6. Check **Auto-mount** and **Make Permanent**, if available.
7. Click on **OK**.

### Step 2: Mount the folder in the guest OS

1. **Create a folder** in your guest OS that you want to share.
2. **Open up Terminal**.
3. **Type in** id and press ENTER— remember that ID.
4. **Switch to the** root **user** using sudo su and enter your password.
5. **Browse to the** etc **folder** using cd /etc.
6. **Edit the** rc.local **file** using vi rc.local.
7. Move your **cursor right above** exit 0 and **press the letter “i”** on your keyboard to insert text.
8. **Type in the following command on shell**:  sudo mount -t vboxsf -o uid=1000,gid=1000 Share /home/username/Documents/Share
   * 1000 should match the ID you noted down earlier.
   * Share should match the folder name from step 1.
   * username should match your Linux username.
   * /Documents/Share should be the absolute path of the new folder you created.
9. **Now hit “ESC”**, type :wq and hit ENTER to save and quit the file editing.

After you restart the guest OS, your shared folder will be automatically mounted.

## Secure Copy (SCP) between Host & Guest OS.

This is my favorite method (and easiest too). Works like a charm for me every time. In a Linux environment, for both security and ease of use, ssh is the best way to go. SSH, SSHFS, SCP, and SFTP as you list are all just different services built on top of the SSH protocol.

SCP is very easy to use, it works just like CP but you can provide user and machine names in the path.

So, we might do a CP like cp ~/rpi\_iot\_kernel/ ~/rpi\_iot\_kernel\_backup/, but we could just as easily do

$ scp ~/rpi\_iot\_kernel/linux/kernel.img [user@host\_pc:~/temp/kernel.img](mailto:user@host_pc:~/temp/kernel.img)

to send it to the host computer (on which your Virtual Box Guest OS is running).

That's it - we don't need to set anything up. You'll be prompted for the account password on the other machine if you don't have certificate or some other authentication set up (scp shares those settings with ssh, of course).

## Using a USB card reader

Another one of my favorite. Simply insert a USB Card Reader with SD Card, it is auto mounted in Guest OS (e.g. Ubuntu running inside Virtual Box). Once SD Card shows in Guest OS, we can simply exchange file(s) using any in-built file explorer.

# IoT & RaspberryPi

## 1) Remote Temperature Monitor using ESP8266 MCU Node & RPi

[ <https://openhomeautomation.net/connect-esp8266-raspberry-pi> ]

Have you ever dreamt of controlling your home’s cooling/heating equipment, just to make that perfect ambience by the time you reach your home? All this and more, without you clicking a single button.

Monitoring the temperature of your home remotely, and that too without your intervention could be a bliss, not to forget the optimized communication between various IoT enabled devices, that help minimize your electricity bills.

Following pages will take you through a step by step procedure of setting up the various IoT devices involved and getting them to interact.

**Summary**

We will connect a Temperature sensor (Dallas DS18B20) to an ESP8266 board and access the data via Wi-Fi. However, here, we are going to grab that data from a Raspberry Pi and make RPi display the data on a simple graphical interface.

To do so, we will run a simple web server on the ESP8266 chip. The Raspberry Pi will then access this data via Wi-Fi and display it graphically. The wonderful thing is that you can make the Raspberry Pi the ‘hub’ of your home, with several ESP8266-based devices connected to it.

### Hardware Requirements

1. Temperature Sensor (Adafruit DHT11, Dallas DS18B20)
2. ESP8266 chip (which makes Temperature sensor IoT enabled)
3. Raspberry Pi board
4. Bread board (Optional)
5. Jumper wires

**Software Requirements**

1. Arduino IDE (Latest version)

[ <https://www.arduino.cc/en/Main/Software> ]

1. Raspbian operating system installed on RPi board
2. Dallas Temperature library

You can install it from the Arduino library manager, that you can access from **Sketch > Include Library > Manage Libraries** inside the Arduino IDE…Search for “**DALLAS**” & “**OneWire**”

1. Node.js on RPi

[ <https://learn.adafruit.com/node-embedded-development/installing-node-dot-js> ]

### Hardware Configuration

Once Software installation is complete, we can now go ahead with connecting ESP8266 with Temperature sensor. (Refer to Temperature Sensor Datasheet for correct pin configurations - Dallas DS18B20 – used in this instance).

|  |  |
| --- | --- |
| ESP8266 MCU Node | Dallas DS18B20 Breakout Board |
|  | |

=== Dallas Temperature Sensor Datasheet ===

<https://github.com/softwaresunleashed/rpi_iot_kernel/blob/devel/documents/Datasheets/TemperatureSensor/Dallas/DS18B20.pdf>

===

=== LoLin ESP8266 Wi-Fi Board Setup ===

Refer to Datasheet – Chapter THREE for setting up LoLin ESP8266 Wi-Fi module (used by author for this project) in Arduino IDE. In case you are using any other ESP8266 module in your project, you can skip this section and can refer to the corresponding datasheet.

<https://github.com/softwaresunleashed/rpi_iot_kernel/blob/devel/documents/Datasheets/ESP8266/esp8266-V10.pdf>

OR

Refer to section – “**Installing LoLin ESP8266 Board Configs in Arduino IDE**” in this book.

===

|  |  |
| --- | --- |
| **ESP8266 Chip** | **Temperature Sensor** |
| 3.3V | Vcc pin (Right) |
| GND | GND (Left) |
| GPIO5 (D1) | Data Out pin (Middle) |

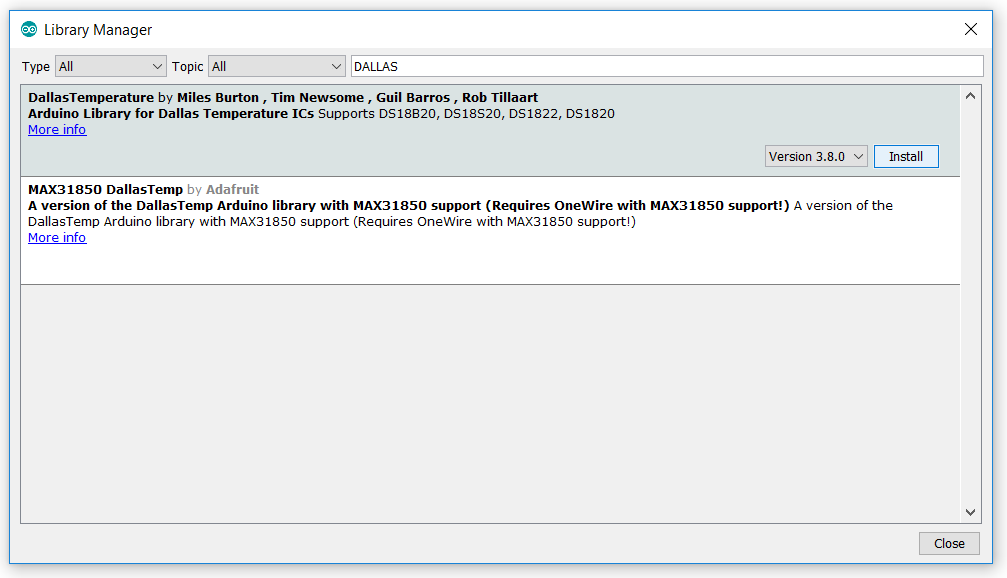
Make sure that you connected everything according to the instructions above, or you won’t be able to continue.

### Testing the Sensor – Dallas DS18B20

We will now test the sensor. Again, remember that we are using the Arduino IDE, so we can code just like we would do using an Arduino board. Here, we will simply print the value of the temperature inside the Serial monitor of the Arduino IDE.

Remember to Install “**DALLAS**” Temperature Library & “**OneWire**” Library (if not already done). Refer to snapshot below.

**Dallas Library**



**One Wire**



**Complete code for this part:**

// Code for testing Temperature Sensor over Serial Port

#include <OneWire.h>

#include <DallasTemperature.h>

// Data wire is plugged into pin 2 on the ESP8266 MCU Node

// ‘5’ refers to GPIO05 which is pin 2(refer to ESP8266 pinout

// snapshot above

#define ONE\_WIRE\_BUS 5

// Setup a oneWire instance to communicate with any OneWire

// devices (not just lcMaxim/Dallas temperature ICs)

OneWire oneWire(ONE\_WIRE\_BUS);

// Pass our oneWire reference to Dallas Temperature.

DallasTemperature sensors(&oneWire);

void setup(void)

{

// start serial port

Serial.begin(115200);

Serial.println("Dallas Temperature IC Control Library Demo");

// Start up the library

sensors.begin();

}

void loop(void)

{

// call sensors.requestTemperatures() to issue a global temperature

// request to all devices on the bus

Serial.print(" Requesting temperatures...");

// Send the command to get temperatures

sensors.requestTemperatures();

Serial.println("DONE");

Serial.print("Temperature for Device 1 is: ");

Serial.print(sensors.getTempCByIndex(0));

// You can have more than one IC on the same bus.

// 0 refers to the first IC on the wire

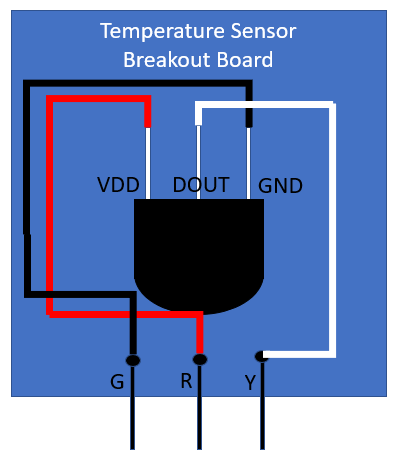
delay(2000);

}

You can see that all the measurement part is contained inside the loop() function, which makes the code inside it repeats every 2 seconds. Then, we read data from the Dallas DS18B20 sensor, print the value of the temperature on the Serial port.

**Trouble Shoot**

If you are not getting some valid (deg Celsius) temperature, refer to Troubleshooting section – “Temperature sensor reporting invalid or constant temperature”. Refer to block diagram below for Temperature Sensor IC to Header mapping.



There are few temperature sensor breakout boards available in the market, like the one used in IoT projects section.

Here the mapping between Temp IC and Headers may not be straight forward. You have to ensure the IC pins to external header’s mapping using “Continuity” feature on any standard multimeter.

If “Continuity” feature isn’t available on your multimeter, you can check for resistance between IC pins and header pins. The two pins that are connected shall show minimal (close to Zero) resistance.

**Accessing the Sensor via Wi-Fi**

At this point, we are sure that the sensor is working, and that data can be read by the ESP8266 chip. Now, we are going to build the sketch that will connect to your WiFi network, and then make the measurements accessible by the Raspberry Pi.

As this sketch is quite long, I will only detail the most important parts here. Complete code can be found on git repo or end of this section.

It starts by including the required libraries for this project:

**#include <ESP8266WiFi.h>**

**#include <ESP8266WebServer.h>**

**#include <aREST.h>**

**#include <OneWire.h>**

**#include <DallasTemperature.h>**

Then, you need to set up your own WiFi network name & password in the code:

**const** **char**\* ssid = "your\_wifi\_network\_name";

**const** **char**\* password = "your\_wifi\_network\_password";

We also declare the aREST API, that will help us control the board remotely from the Raspberry Pi:

aREST rest = aREST();

Now, we also declare one variable that will contain the measurements made by the sensor:

**int** temperature;

After that, we create a web server on port 80:

WiFiServer **server**(80);

Then, inside the setup() function of the sketch, we connect the ESP8266 to the WiFi network:

WiFi.begin(ssid, password);

**while** (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected");

Then, we expose the two measurement variables to the aREST API, so they can be access from the outside:

rest.variable("temperature", &temperature);

After that, we set the name & ID of the board:

rest.set\_id("1");

rest.set\_name("sensor\_module");

Then, we start the server, and print the IP address on the Serial port:

*// Start the server*

server.begin();

Serial.println("Server started");

*// Print the IP address*

Serial.println(WiFi.localIP());

Inside the loop() function of the sketch, we check if a client is connected to the ESP8266, and handle the request:

WiFiClient client = server.available();

**if** (!client) {

return;

}

**while**(!client.available()){

delay(1);

}

rest.handle(client);

Then, we read data from the sensor:

// Send the command to get temperatures

temperature = sensors.getTempCByIndex(0);

Follow the instructions we saw in the previous section to upload the code to the board. Then, open the Serial monitor to get the IP address of the board, you will need it soon.

You would need to install “aREST” library to compile the code shown in next section.



**Complete code for this part:**

// Code for testing Temperature Sensor over Web Interface

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

#include <aREST.h>

#include <OneWire.h>

#include <DallasTemperature.h>

//------------------------------------------

//WIFI

const char\* ssid = "your\_wifi\_network\_name";

const char\* password = "your\_wifi\_network\_password";

aREST rest = aREST();

int temperature;

//------------------------------------------

//HTTP

// The port to listen for incoming TCP connections

#define LISTEN\_PORT 80

// Create an instance of the server

WiFiServer server(LISTEN\_PORT);

// Data wire is plugged into pin 2 on the ESP8266 MCU Node

// ‘5’ refers to GPIO05 which is pin 2(refer to ESP8266 pinout)

#define ONE\_WIRE\_BUS 5

// Setup a oneWire instance to communicate with any OneWire devices

// (not just lcMaxim/Dallas temperature ICs)

OneWire oneWire(ONE\_WIRE\_BUS);

// Pass our oneWire reference to Dallas Temperature.

DallasTemperature sensors(&oneWire);

void setup(void)

{

// start serial port

Serial.begin(115200);

Serial.println("Dallas Temperature IC Control Library Demo");

Serial.println("");

// trigger wi-fi connection

WiFi.begin(ssid, password);

// wait for wifi connection to complete.

Serial.print("Connecting to SSID -- " + ssid);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected");

// expose the two measurement variable(temperature) to the aREST API

// so they can be accessed from outside

rest.variable("temperature",&temperature);

// set the name & ID of the board:

rest.set\_id("1");

rest.set\_name("sensor\_module");

// Start the server

server.begin();

Serial.println("Server started");

// Print the IP address

Serial.println(WiFi.localIP());

// Start up the temperature sensor library

sensors.begin();

}

void loop(void)

{

// call sensors.requestTemperatures() to issue a global temperature

// request to all devices on the bus

sensors.requestTemperatures();

// Send the command to get temperatures

temperature = sensors.getTempCByIndex(0);

// check if a client is connected to the ESP8266, and handle the request

WiFiClient client = server.available();

if (!client) {

return;

}

while(!client.available()){

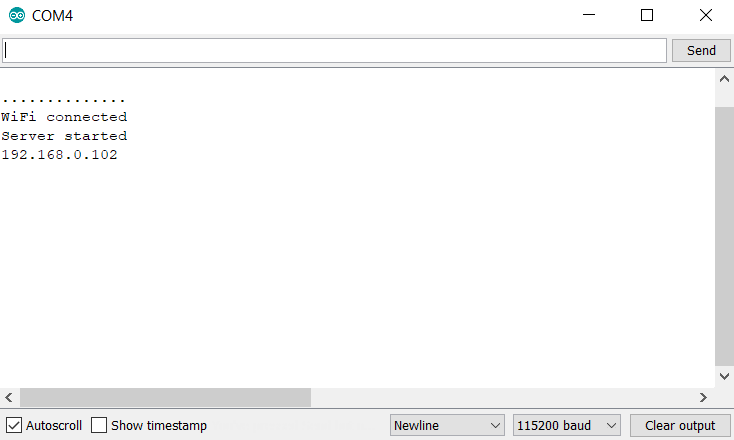
delay(1);

}

rest.handle(client);

}

**Snapshot of Server Listening for Temperature Requests from Clients**



### Connecting the Board to Your Raspberry Pi over Air Interface

We shall now see how to connect the ESP8266 board to your Raspberry Pi, so it can display the data in a nice graphical interface.

Download and place all the files into a folder on your Pi,

$ mkdir -p iot\_projects/remote\_temp\_monitor

$ cd iot\_projects/remote\_temp\_monitor/

Link of Folder to copy from GitHub: [https://github.com/softwaresunleashed/rpi\_iot\_kernel/blob/devel/iot\_projects\_src\_code/01\_iot\_based\_temp\_monitor/RPI/web\_interface](https://github.com/softwaresunleashed/rpi_iot_kernel/blob/devel/iot_projects_src_code/01_iot_based_temp_monitor/RPI/web_interface.tar.gz)

After you have copied the above folder from Git Repo, navigate to this folder with a terminal, and type:

$ sudo npm install express jade arest

**Troubleshoot:**

If “npm”package isn’t installed on RPi, then you will get following error message.

sudo: npm: command not found

This error can be fixed by installing NPM package on RPi by executing following command on RPi shell

$ sudo apt-get update

$ sudo apt-get dist-upgrade

$ sudo apt-get install npm

**Be patient, these steps can take a while**. After that, type:

$ sudo nodejs app.js

This will start the application on your Raspberry Pi. Then, navigate to the URL of your Raspberry Pi on port 3000, for example:

http://192.168.0.100:3000

You should immediately see the interface showing the temperature readings made by the ESP8266 board:

The only thing that you need to change to run this interface is the IP address of your ESP8266 board inside the app.js file:

// Replace 192.168.0.100 with IP address of ESP8266 board

rest.addDevice('http','192.168.0.100');

# Trouble Shooting Guide

There would be times, when our friendly device RPi is distressed and may give you hard time going around. Following section provides real life problems encountered and Tips & Tricks on how they were fixed.

Priceless!!!

## Troubleshooting Power Problems

[ [http://elinux.org/R-Pi\_Troubleshooting - Troubleshooting\_power\_problems](http://elinux.org/R-Pi_Troubleshooting#Troubleshooting_power_problems) ]

If you think you have a problem with your power supply, it is a good idea to check the actual voltage on the Raspberry Pi circuit board. Two test points labelled TP1 and TP2 are provided on the circuit board to facilitate voltage measurements.

Use a multimeter which is set to the range 20 volts DC (or 20v =). You should see a voltage between 4.75 and 5.25 volts. Anything outside this range indicates that you have a problem with your power supply or your power cable.

Note: Even if the multimeter shows the correct voltage, you may have some power supply problems. A multimeter only displays the average voltage. If there are very short-lived dips or spikes in the voltage, these will not be shown by the multimeter. It is best to measure voltage when Pi is busy.

If your voltage is low, it could be:

* The power supply produces too low a voltage
* The power supply cannot supply enough current, which results in a voltage drop. Make sure Power supply is labelled as at least 700mA. (Some cheap power supplies don't deliver what is labelled).
* The Micro USB power cable is low quality. Some Micro USB cables have very thin conductors, resulting in enough voltage drop for RPi to fail even if the power supply itself is fine.
* Attached USB devices want too much power. The Pi is only designed for up to 100mA USB devices. A USB device wanting more than that will cause a voltage drop.
* The F3 Polyfuse could be blown or bad, see below for how to test.

Note: keyboards with LCD displays, built in USB hubs, backlights, etc. are likely to be problematic. Try to use a basic one. Wi-Fi dongles are also unlikely to work when directly connected. Connect high powered USB devices to a powered USB hub.

Try booting without HDMI, Ethernet or USB device plugged in, and see if the voltage improves.

## How to test the F3 polyfuse

1. Remove all the things plugged into your Raspberry Pi, including SD card.
2. Locate the TP2 test point on the top of the board.
3. Turn your board over and find the TP2 test point on the bottom of the board. One lead of your multi-meter will always be on the TP2 point on the bottom of the board for all tests.
4. Plug your power supply into the micro usb port and power your board.
5. Place one lead of your multi-meter on the TP2 point on the bottom of the board and one lead on the side of the F3 fuse closest to the edge of the board. Note the voltage. This is the voltage coming into your RPi from your power supply.
6. Keeping one lead on TP2, move the other lead to the side of F3 closest to the SD card slot. This is the voltage coming out of the F3 fuse.



If the voltage is different by more than about 0.3v you probably have an issue with the F3 fuse.

When polyfuses "blow" their resistance increases dramatically, thereby limiting the voltage that can pass through them. If your power problem suddenly appeared after your board was known to be working fine, it is probable the fuse is just "blown" and will return to normal. Polyfuses recover from the tripped state to near their normal value in a few minutes, but do take some hours to fully recover so leave it unpowered and check it again in a little while. If your power problem has been since the first time you plugged in your board, the fuse was probably bad when it arrived and should be returned to place you purchased it.

Also, on a related issue, do note that if you do not power the PI in the "official manner", that is through its micro-USB port, but use any alternative way (such as through the GPIO header, the test points TP1 and TP2), but also by back-powering it, you are bypassing the PI's input polyfuse protection device! This can have extreme consequences if ever you manage to put more than 6V on the PI, even for a very short period. As this causes the overvoltage device D17 on the PI to trigger and short the 5V supply! Without the polyfuse limiting the current through D17, it will burn out, probably melting the PI's enclosure with it, (if you have any) and possibly causing a fire-hazard. It will probably also create a permanent short of the 5V supply! So be warned, and if you use back power make sure your hub or its PSU has a fuse to prevent this from happening. If not, add your own fuse.

## Red power LED is on, green LED does not flash, nothing on display

[ [http://elinux.org/R-Pi\_Troubleshooting - Red\_power\_LED\_is\_on.2C\_green\_LED\_does\_not\_flash.2C\_nothing\_on\_display](http://elinux.org/R-Pi_Troubleshooting#Red_power_LED_is_on.2C_green_LED_does_not_flash.2C_nothing_on_display) ]

A faintly glowing steady green LED means **no boot code has ever been executed**, as almost the first thing the boot code does is to turn the faint glow off! When flashing/blinking, the green LED should be as bright as the red LED.

There is a difference between the quad core Pi 2 (BCM2836) and the other models if there is no SD card inserted, or the SD card is improperly formatted. The former will have both red and green lights on bright and steady. The latter will have only the red light on.

This indicates:

* The Raspberry Pi cannot find a valid image on the SD card. Turn the board over to check that the card is inserted correctly; the insertion force is much larger than for some laptops.
* Check that you have correctly written a Raspberry Pi image to the card by using a MAC or PC and browse for the following files:
  + bootcode.bin
  + fixup.dat
  + start.elf amongst others
* Did you have admin rights when you used the SD-card writer software? Without it the software might go through the motions without doing anything!
* Older images do not load boot code for revB boards with the Hynix chip. Use release 2013-02-09 (?) or later. (I observe a single blip on the green activity LED)
* It is also possible that the image you are writing to the card is corrupt, as downloads do occasionally end up corrupted or truncated. You can check with a checksum utility to verify the integrity of the download.
* The SD card may itself have an issue. (This one I faced myself, my RPi board dint boot up due to the broken MicroSD converter).
* Try with no cables connected except the USB power lead, and SD card inserted. You should see flashing of the OK light for ~20 seconds. If that helps, plug in cables one at a time to identify which is interfering with boot.
* Confirm the USB cable is properly seated in the power slot. The red power LED does not necessarily mean it is fully connected.
* Look at the SD card holder on the Raspberry Pi carefully. At first glance it may look fine but the contacts must be springy and they must protrude at least 2mm as measured from the lower edge of the holder to the top of the contact bulge. This happens due to the solder process and the type of holder used. Some of the solder residue falls into the contact cavity restricting the springiness and the height that the contact protrudes. You can fix this yourself but remember you can void your warranty. The contacts are delicate so be careful. Insert a needle pin under the contact bulge and pull lightly up until the one end of the contact unclips. Clean the cavity where the contact unclipped from of any solder or other residue by blowing into the cavity. Clip the contact back into the cavity by lightly pushing it into the cavity. Do this for all the contacts.
* Ensure that when your SD Card is fully inserted that the longer metal spring contacts (one clearly visible on the end of the slot, and one hidden in the side nearest the power connector) are closed. These are used to detect the presence of an SD Card therefore if no contact is made then the Raspberry Pi won't attempt to access the card.
* Check carefully for any cracks or damage to the SD Card slot, if the sides are damaged then the card may not be making proper contact with the pins (can usually confirm this if your Raspberry Pi boots if you manually hold the SD Card in position).
* The voltage is too low (below 5 V), try a different power supply and/or cable. The R-Pi needs a supply rated for 700 mA or more. Some supplies labelled as such cannot provide their rated current while maintaining 5V.
* If for whatever reason the main polyfuse F3 has been overheated previously it may happen that it hasn't completely recovered, in which case, if you turn the PI on, a considerable amount of energy from the power supply is lost in the fuse and doesn't reach the PI. Try if the polyfuse seems hot.
* Some problems have been reported if the ambient temperature is low that might be related to micro-fractures, fissures in solder or other issues. Try warming the Raspberry Pi with a hair dryer for just a few seconds (do not use excessive heat or you may cause irreversible damage!) and reconnect the power.

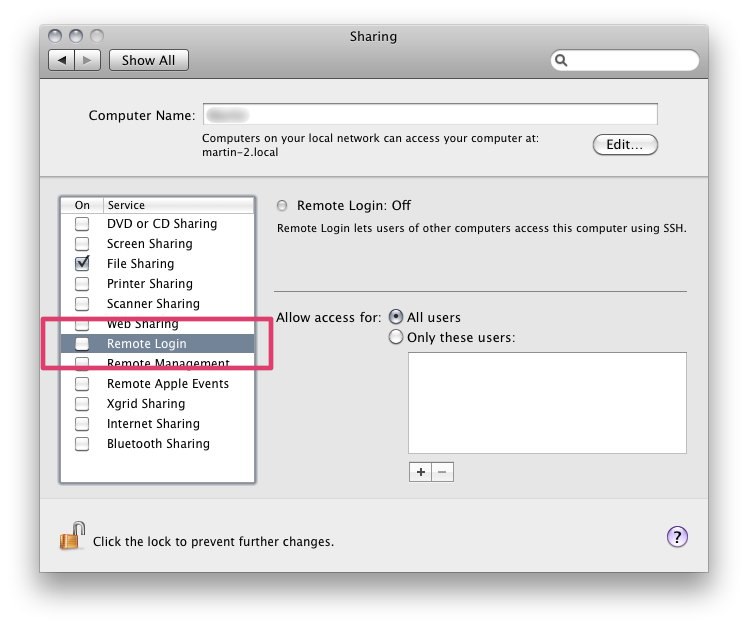
## [How to Allow Incoming SSH (SCP) connections on MAC OS?](https://apple.stackexchange.com/questions/2419/how-can-i-allow-incoming-ssh-scp-connections)

## Connection refused error when trying to SCP

Mac OS

To allow SSH incoming connections to your Mac, you need to enable Remote Login in Sharing Preferences pane. See image below.

Here’s a picture:



## Custom Kernel Image Doesn’t Boot

[ <https://www.raspberrypi.org/forums/viewtopic.php?t=103087&p=713260> ]

Check the ELF headers of your custom kernel using the following command:

$ file kernel.img

If you get something like “kernel.img : data”, this could mean that your image creation method isn’t compatible with the latest kernel formats (Of course, this was the format used by earlier RPi kernel versions).

Follow the steps below, to generate a compatible kernel image with latest kernel version.

With recent RPi Kernels, the process of creating the kernel image has changed as described below.

$ cd ~/rpi\_iot\_kernel/linux

$ ./scripts/mkknlimg ./arch/arm/boot/zImage kernel\_new.img

$ file kernel\_image\_new.img

The above command, checks the ELF headers of newly generated kernel image

Expected Result :

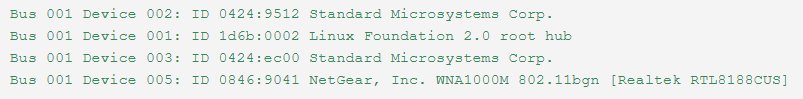
kernel\_new.img: Linux kernel ARM boot executable zImage (little-endian)

Finally, Copy the kernel\_new.img generated above, to /boot folder on Raspberry Pi and reboot. Remember to change /boot/config.txt to boot from the newly created kernel.img

## Wi-Fi Dongle Issues – Doesn’t camp to Wi-Fi Network

[ <https://www.raspberrypi.org/forums/viewtopic.php?f=28&t=44044> ]

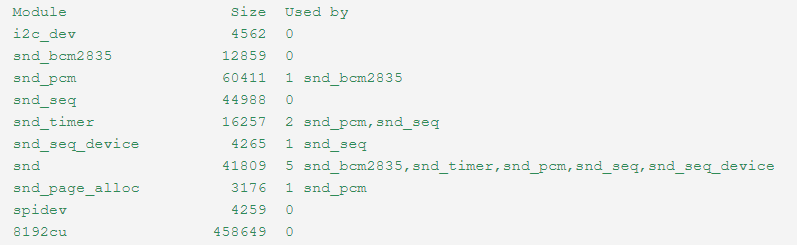
1. **Check whether the Wi-Fi stick is being picked up at all**  
   Run a 'lsusb' in a terminal and you should see your Wi-Fi stick in the output of the command.



Possible Outcomes:

* + If you can see it, it means that at least the stick is most likely working.
  + If you can't see it, then the stick is not being picked up.
  + If you're using a powered hub, try the stick in another computer
  + If you're NOT using a powered hub, try a powered hub first.

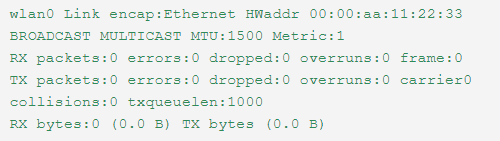
1. **Check whether the modules loaded**Run a ‘lsmod’ and see if the module for the stick is loaded. The module name will change depending on the stick that is being used. For example, if you have the above RTL8188CUS Wi-Fi adapter, then lsmod will show the following output (notice the 8192cu module is loaded):



Possible Outcomes:

* For RT5370 sticks, you'll see a rt2800usb module in there instead. If the module is there, then the drivers are being loaded correctly.
* If NOT, ensure that you copied the Modules (/lib/modules) folder that were generated while building the RPi Kernel.

1. **Check the Wi-Fi NIC**  
   Since the stick is there and drivers are loaded, check to make sure the stick is being picked up  
     
   Run an **ifconfig -a** and check the output. You should see a 'wlan0' in the output.



Possible Outcomes:

* If you do not see this not then you may be missing some firmware for your device.
* The firmware packages for the more common devices are as follows:  
  **firmware-ralink** for Ralink devices (RT5370/RT3070/etc.)  
  **firmware-realtek** for Realtek devices (RTL8188CUS/etc.)  
  **firmware-atheros** for Atheros devices.
* **apt-get install <firmware-package>** will install the firmware on Raspbian.

Once you've installed the firmware, unplug and plug the stick back

in to see if **ifconfig -a** can see it now.

* If not, try a reboot as well.

1. **Check the WiFi configuration**

If you can see wlan0 when you run ifconfig -a but you can't get online, you'll need to make sure the adapter is getting an IP address, and is connected to a WiFi Network.

For Raspbian:  
Check /etc/network/interfaces and ensure that you have the WiFi details configured.

This can be done in either the interfaces file, or it can be linked to another file via the wpa-roam or wpa-conf parameters in /etc/network/interfaces

Verify the details for the network are correct, SSID and PSK need to be exactly right for the WiFi NIC to connect to a network.

If you have changed the details, run **sudo ifdown wlan0 && sudo ifup wlan0** to use the new settings.  
  
These settings will be applied when the WiFi starts up.

**4a) WiFi Not Connecting on Startup**  
If the WiFi works when you run ifup wlan0, but doesn't come up when you reboot the Pi, we'll need to check the file /etc/network/interfaces again.

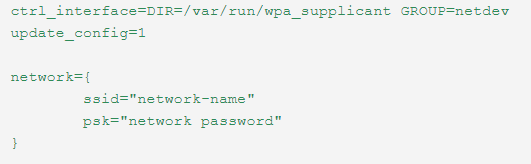
Look for any lines that have auto at the beginning, and see if you see one that has 'auto wlan0'.

If not, then add that line in. This line directs the Pi to connect to bring up the WiFi NIC automatically, which in turns triggers the WiFi NIC to connect to the WiFi network.

**4b) How do I configure the WiFi Settings**  
By default, Raspbian has in /etc/network/interfaces this line:



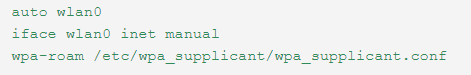
The contents of the file should look something like this:



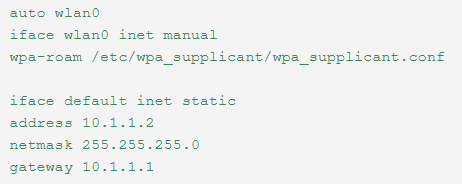
If it doesn't, or if it has the wrong details, updating the details here will let the WiFi Adapter connect.

**4c) How do I configure WiFi to use a Static IP address**

In /etc/network/interfaces, there should be these lines



To make it use a static IP address, we need to add a few lines, and change dhcp to static, and then it should look like this:



That block will bring up wlan0 on startup, connect to the WiFi network that's detailed in wpa\_supplicant.conf, and set an address of 10.1.1.2

**4d) Make Sure The WiFi Network Can Be Detected**

If the Pi still won't connect, make sure the Pi is actually seeing the WiFi Network  
  
This can be done with the wpa\_cli command, but this needs to be run as root



That should show a list of available WiFi networks, and if yours isn't on the list, then you may need to adjust the location of the Pi or the WiFi router.

## USB Stick not detected on RPI

There have been times, when the Memory Stick (Pen Drive or any other fancy name you call it by), isn’t mounted on RPI board. This happens in spite of the number of times you format the pen drive on your Windows or Mac machine. Of course, there is mostly no trouble observed if you are using a Linux machine to format the media.

Solution:

* 1. Insert the pen drive in RPI USB slot
  2. Boot to RPI shell prompt
  3. If memory stick is functioning (working one), the device would be seen under /dev/sd[x]
  4. Confirm [x] via typing following command

**$ dmesg | grep sd**

Let’s say only one USB storage is connected, so you may see something like **/dev/sda**

* 1. Let’s now make a filesystem on previously identified pen drive

$ mkfs.msdos -I /dev/sda

* 1. Mount the device using following command

$ mount -t vfat /dev/sda /mnt

* 1. Pen drive contents are now available under /mnt folder

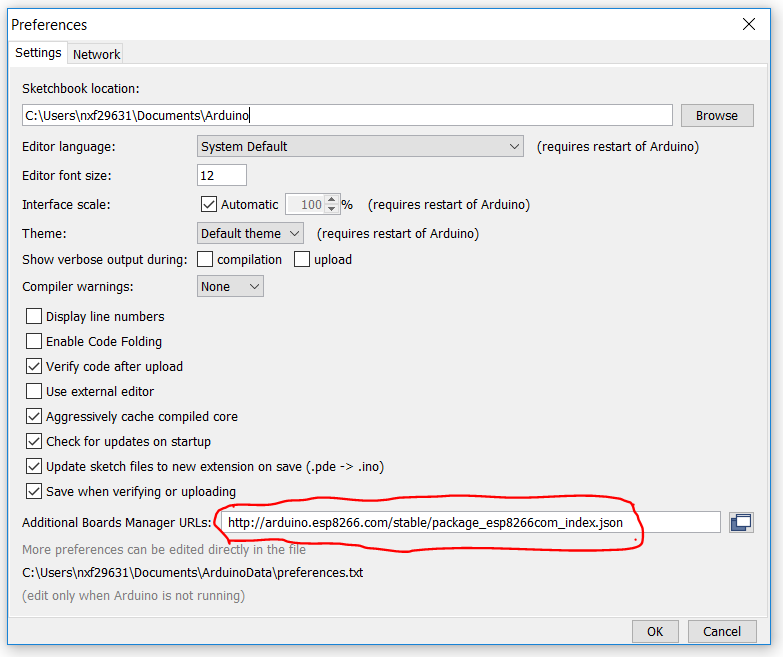
## Installing LoLin ESP8266 Board Configs in Arduino IDE

Follow below step(s) to install ESP8266 library to work in Arduino IDE environment.

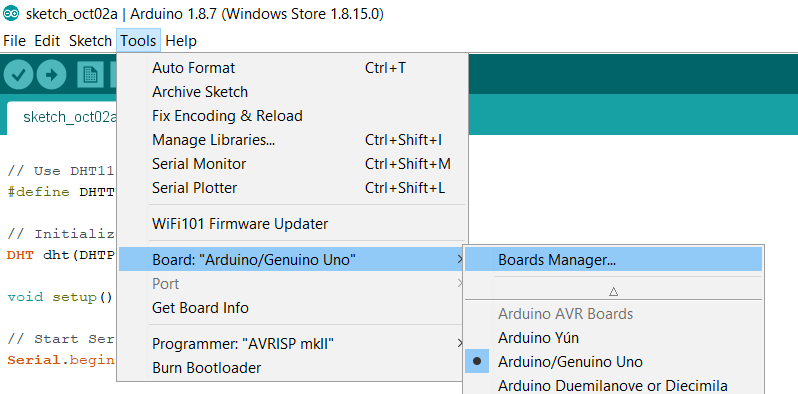
1. Install the latest version of Arduino IDE. <https://www.arduino.cc/en/Main/Software>

2. Install the ESP8266 Board Package

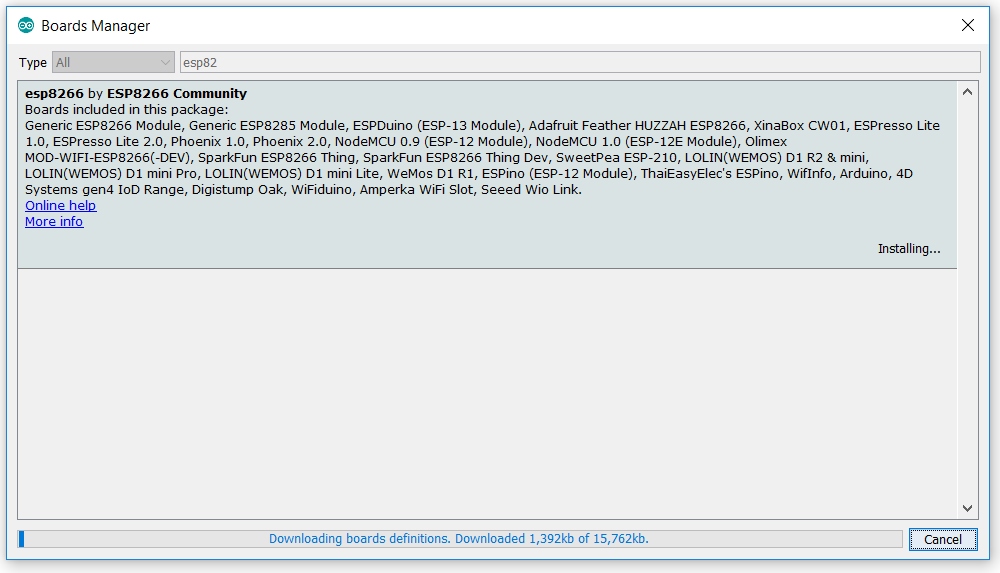
Enter **http://arduino.esp8266.com/stable/package\_esp8266com\_index.json** into Additional Board Manager URLs field in the Arduino IDE preferences.

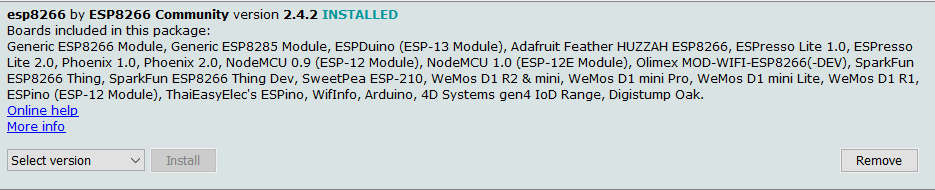


3. Next, use the Board manager to install the ESP8266 package.



4. Search for “esp8266”, and install **“esp8266” by ESP8266 Community**



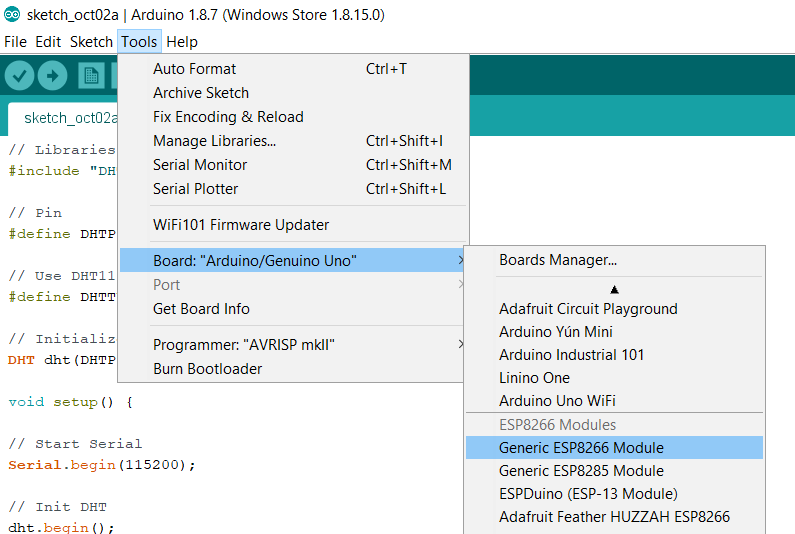


**Precautionary Note:**

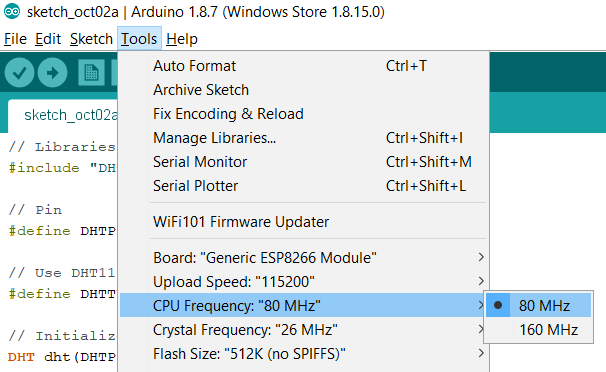
If the you don’t see any search results OR the download doesn’t start for some time, ensure that you are not behind an Office Firewall or VPN connection. (I had to close down my Cisco VPN, before the download started for me.)

5. Restart Arduino IDE.

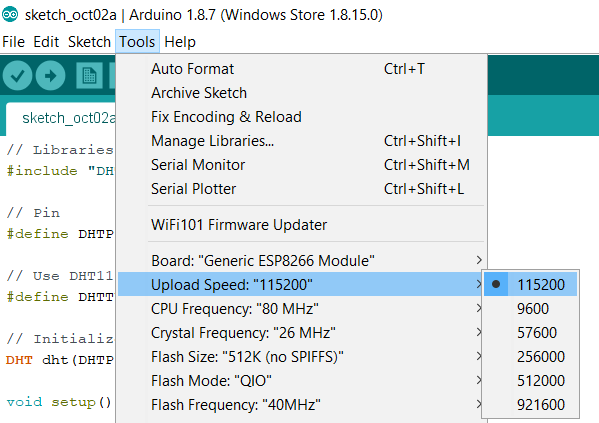
When you've restarted Arduino IDE, select ‘**Generic ESP8266 Module**’ from the ‘Tools’ -> ‘Board:’ dropdown menu.

****

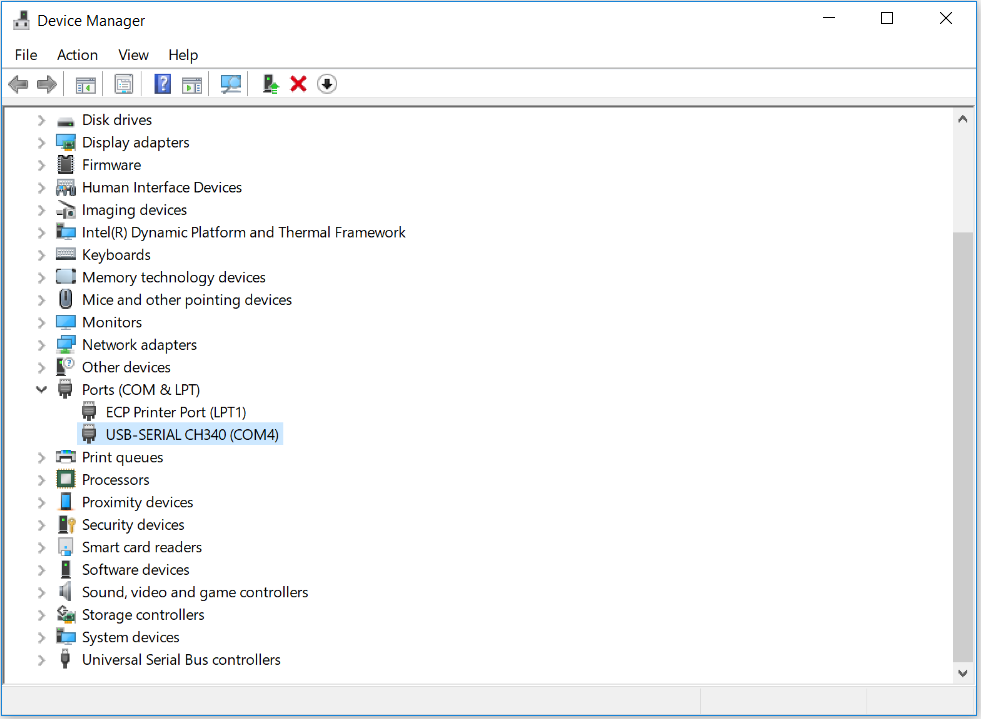
**6.** Select 80 MHz as the CPU frequency (you can try 160 MHz overclock later)

****

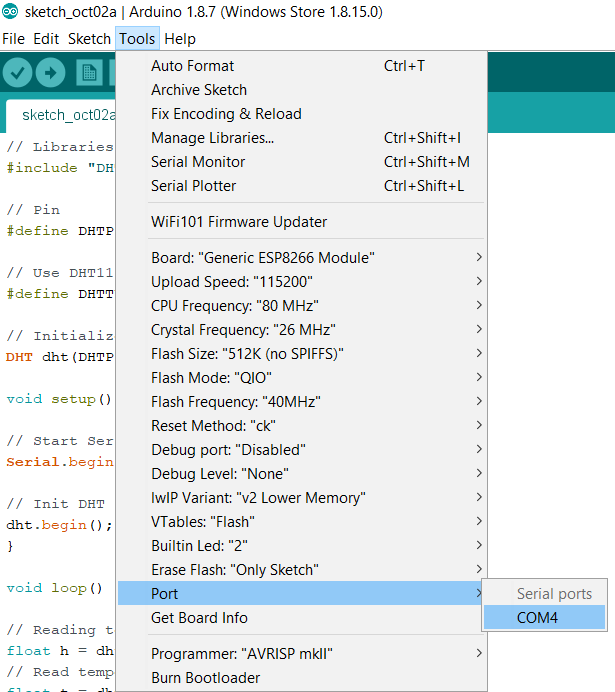
**7.** Select ‘115200’ baud upload speed is a good place to start - later on you can try higher speeds

****

**8.** Go to your Windows ‘Device Manager’ to find out which Com Port ‘USB-Serial CH340’ is assigned to.



**9.** Select the matching COM/serial port for your CH340 USB-Serial interface.

****

Note::

If this is your first time using CH340 “**USB-to-Serial**” interface, please install the driver first before proceeding with the above Com Port setting.

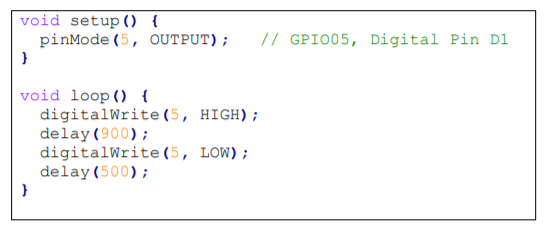
The CH340 driver can be download from the below site:

<https://github.com/softwaresunleashed/rpi_iot_kernel/tree/devel/documents/Datasheets/ESP8266/drivers>

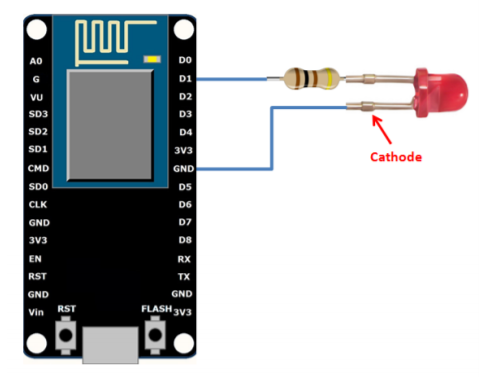
## Flash Sketch Code (LED Blink Test) in ESP8266 using Arduino IDE

Following section explains how to upload a sketch / code to Arduino based ESP8266 MCU Node.

Enter this code into the sketch window. A sketch window is where you type in your Arduino based code.



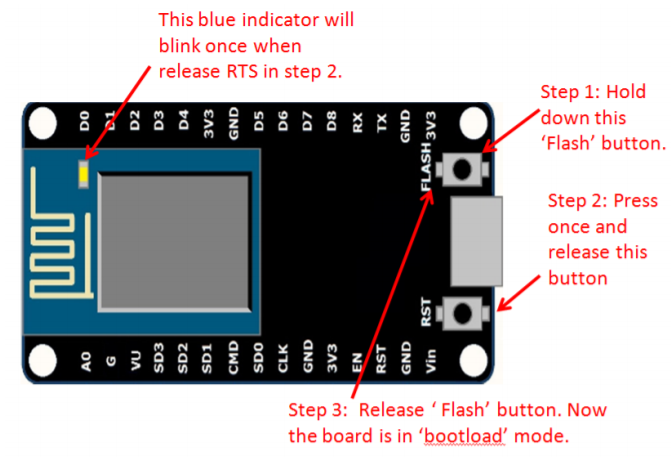
**Connect a LED as shown in Figure.**



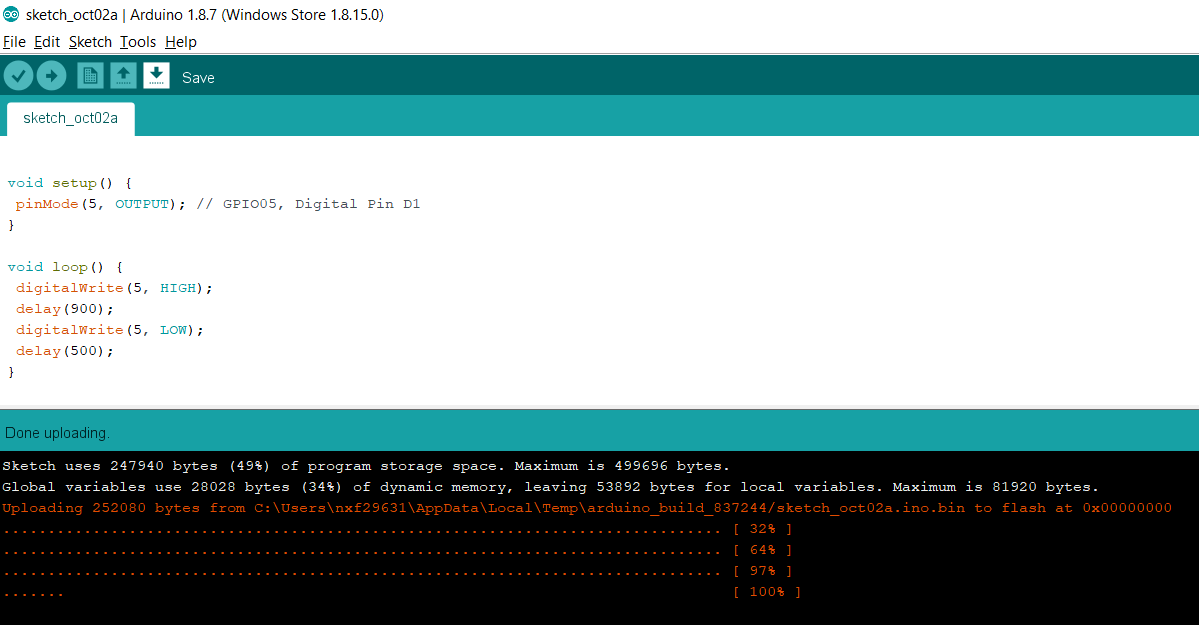
**Steps to upload code:**

Put the board into bootload mode. You'll have to do this before each upload. There is no timeout for bootload mode, so you don't have to rush!

1. Hold down the ‘Flash’ button.
2. While holding down ‘ Flash’, press the ‘RST’ button.
3. Release ‘RST’, then release ‘Flash’
4. When you release the ‘RST’ button, the blue indication will blink once, this means its ready to bootload.



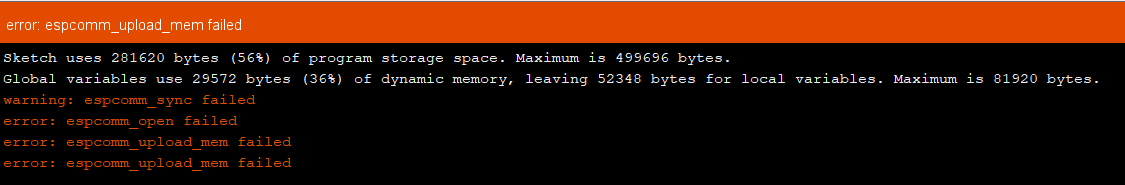
**Successful Flash Snapshot**



## Error While Flashing on Arduino / ESP8266 Wi-Fi Node MCU

**Problem:**

While flashing code on Arduino / ESP8266, if we are continuously getting the following error.



**Solution:**

Check the following –

* 1. Serial console or any other application is not currently using the COM port where Arduino based ESP8266 is connected.
  2. There is no active connection to ESP8266 board, ie no slave device is connected to the board which may interfere in download process.
  3. Isolate ESP8266 board and then Unplug and Plug power cable (USB cable) to ESP8266 board, and try flashing code again.

## Temperature sensor reporting invalid or constant temperature

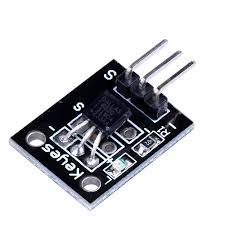
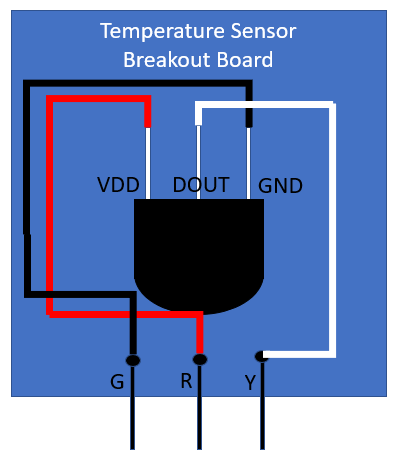
If temperature sensor is not reporting valid temperature on serial port, following could be the possible reason(s). Check if :

1. Power (Voltage / Current) is applied as per required values mentioned in the Temperature sensor datasheet. Typically the voltage requirement is 3.3V to 5V.
2. Connections are proper. There are few temperature sensor breakout boards available in the market, like the one used in IoT projects section (snapshot below).

Here the mapping between Temp IC and Headers may not be straight forward. You have to ensure the IC pins to external header’s mapping using “Continuity” feature on any standard multimeter.

If “Continuity” feature isn’t available on your multimeter, you can check for resistance between IC pins and header pins. The two pins that are connected shall show minimal (close to Zero) resistance.

Refer to block diagram below for mapping

## Installing node.js on Raspberry Pi

[ <https://learn.adafruit.com/node-embedded-development/installing-node-dot-js> ]

Once connected to RPi via SSH, you can move on to installing the latest stable version of node.js.

## Adding the Package Repository

The first step will be to add the apt.adafruit.com package repository to your Pi's /etc/apt/sources.list file. You can do this by running the following command.

$ curl -sLS https://apt.adafruit.com/add | sudo bash

Next, we will install the latest version of node.js using *apt-get*.

$ sudo apt-get install node

If everything went as expected, we can check the installed version of node.js by running **node -v**.  At the time of this writing, the latest stable version of node.js was *v0.12.0.*

$ pi@raspberrypi ~ $ node -v

v0.12.0

# Further Reading

<http://elinux.org/RPi_Kernel_Compilation>

<http://elinux.org/RPiconfig>

<https://www.raspberrypi.org/>

<https://www.raspberrypi.org/resources/learn/>

<https://www.raspberrypi.org/documentation/linux/kernel/building.md>

<https://www.raspberrypi.org/documentation/remote-access/ssh/>

<http://www.howtogeek.com/276468/how-to-use-a-raspberry-pi-as-a-networked-time-machine-drive-for-your-mac/>

<http://www.howtogeek.com/138281/the-htg-guide-to-getting-started-with-raspberry-pi/all/>

<https://superuser.com/questions/326211/best-way-to-transfer-files-over-a-lan-between-two-linux-computers>

# Legends

RPi – Raspberry Pi

SoC – System on Chip

VC – Video Core