**SAGAR PRADHAN**

Time & Frequency

JB KNOWLEDGE PARK (MDU)

MENTOR-

1. Dr. Poonam Arora

2. Dr. Anurag Gupta

**NETWORK TIME DISPLAY USING NPL NTP SERVER**

An Embedded System to display Indian Standard Time

Through Network Time Protocol

**CERTIFICATE**

This is to certify that the project entitled “**NETWORK TIME DISPLAY USING NPL NTP SERVER**” is the authentic record of work carried out by**SAGAR PRADHAN**, student of **B.Tech(Electronics and Communication), JB KNOWLEDGE PARK**at Time and Frequency Division, **CSIR- National Physical Laboratory**, New Delhi under the guidance of **Dr. Poonam Arora** and **Dr. Anurag Gupta**, during the period of March 19, 2019 to June 19, 2019.

**Dr.Poonam Arora Dr. Anurag Gupta**

Sr. Scientist Chief. Scientist

Time & FrequencyMetrology IST Division

CSIR-National Physical Laboratory CSIR-National Physical Laboratory

New Delhi, Delhi-110012 New Delhi, Delhi-110012

**ACKNOWLEDGEMENT**

First of all I would like to thank the Almighty for his constant strength, guidance and his abundant grace which enabled me to finish my project work successfully.

I would like to thank **Dr.Harshit Sinha** (**Maharishi DayanandUniversity**) for allowing me to do my internship at CSIR-National Physical Laboratory.

I would also like to thank all my faculty members of Delhi Technological University for their guidance and help.

The internship opportunity I had with **CSIR-National Physical Laboratory, New Delhi** was a great chance for learning and professional development.

I would like to express my gratitude towards my Guide **Dr. Poonam Arora** (**Sr. Scientist**, Time and Frequency Division, CSIR-National Physical Laboratory) and **Dr. Anurag Gupta** (**Chief Scientist**, Time and Frequency Division, CSIR-NPL) for giving me the opportunity to carry out this project in their supervision. I would like to thank them genuinely for encouraging for the project, that asks and challenges that were given to me which helped me to increase significantly in my knowledge. I sincerely thank him for guiding me and keeping me on the correct path to carry out the project.

I am highly thankful to the Ph.D scholar **Mrs. Shalu Goel and Mr. Navraj Poudel**(CSIR-National Physical Laboratory) for all their kind help throughout the internship.

My acknowledgement would be incomplete without expressing my sincere thanks toall the scientists and all the staff of CSIR-National Physical Laboratory, who actively helped me in every aspect by providing relevant materials, data and information pertaining to the successful completion of my project which was extremely valuable for my study both theoretically and practically.

I perceive this opportunity as a big milestone in my career development. I will strive to use the gained skills and knowledge in the best possible way and I will continue to work on their improvement in order to attain the desired career objectives.

I am indebted to my loving parents "**Pramod Pradhan**", "**Sumitra Pradhan** immensely for their love and support without whom everything is incomplete and I would also like to thanks my brother "**Manoj Pradhan**" for his constant help, emotional support, love and concern.

Finally, I would like to thank all my friends and well-wishers for their support throughout my course and project work.

**SAGAR PRADHAN**

About CSIR-NPL

CSIR-National Physical Laboratory (CSIR-NPL) is the National Measurement Institute (NMI) of India and has the mandate for establishment, maintenance and dissemination of the units of physical measurements based on the International System of units (SI units). One of the seven base SI units, second, is realized (at par to the international level), maintained and disseminated by CSIR-NPL to the whole nation as the *Indian Standard Time*(IST).

CSIR-NPL (internationally known as NPLI) is the *Timekeeper of the Nation and*has the “Primary Time Scale Ensemble”, which gives time that is traceable to the Coordinated Universal Time (UTC) provided by International Bureau of Weights and Measures (BIPM) located in Sevres, France. The IST (i.e. UTC-NPLI plus 5:30 hours) generated with an ensemble of a bank of caesium clocks and hydrogen maser has current uncertainty of  7.2 nano-seconds with respect to UTC. CSIR-NPL disseminates IST through various mechanisms, such as Common View Global Navigation Satellite System (CVGNSS), Network Time Protocol (NTP), and Telephone lines. The present App is a network time dissemination mode and shows the IST maintained by CSIR-NPL through NTP service.

**MOTIVATION**

CSIR- National Physical Laboratory is one of the most prestigious labs in India where I got the opportunity to do my summer internship program. I got to know about the different standards that are maintained here, and was really motivated to work under one of those standards. Under the guidance of **Dr. Poonam Arora** in Time and Frequency Metrology Division, I was acquainted with a totally new field of research that related to dissemination of time.

In Time and Frequency Division, Indian Standard time is disseminated through three ways which are via internet, landline connection and satellite link. But, time transfer through network time protocol using **ESP8266 wifi Module** was a real challenge for me. This kept me motivated and also gave me a platform to learn C/C++ coding.

To display IST(Indian Standard Time), I have previously designed a device with Raspberry Pi3 model to display IST time using NTP(Network Time Protocol) server of time.nplindia.org. To program Raspberry Pi3 device I have used Python programming language. The total cost of the device is around Four thousand rupees which is quite expensive for this purpose, also there are many other features of Rasp pie which are not required for the project. Therefore, we have moved to a much cheaper and cost efficient device (ESP8266 wifi module, which is based on Arduino IDE and the primary programming language use to code in this is C/C++) which basically has all the features required for the project.

**ABSTRACT**

Among all the physical quantities, time is the one which can be measured to the highest precision and accuracy. The base unit of time in System international (SI) is ‘second’ which is being realized to with an uncertainty of 10-15. The current definition of the SI second, as given by the International Bureau of Weights and Measures (BIPM) is;

“The second is duration of 9,192,631,770 periods of radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium 133 atoms.”

CSIR- National Physical Laboratory, New Delhi is an authorized signatory of the BIPM in India. As a time keeper of India, it generates, maintains and disseminates the time across the nation. Presently CSIR- NPL has five atomic clock which generates 1 pulse per second i.e. based on above definition of SI second. These atomic clocks are traceable via satellite link to BIPM which generates Universal Coordinated Time (UTC). There are many techniques through which CSIR - NPL disseminates time and frequency to all over India, Network Time Protocol is one of them.

Network Time protocol (NTP) is commonly used to synchronize clocks over network to UTC (NPLI) in wide area network such as public network. The delay asymmetry in wide area network due to routing or band width saturation is usually a dominating source of error. The uncertainty in time transfer using this method is in milliseconds. NTP is an application layer protocol and was designed by David L. Mills of University in 1985.NTP works on User Datagram protocol ((UDP), It is default port number is 123 and it uses its own binary format. NTP is used to synchronize the clocks of server and clients to Universal Coordinated Time (UTC) within few milliseconds error and maintain consistent time where high accuracy is required. Primary sources of time to NTP server can be GPS time or UTC synchronized atomic clocks.

It is the most widely used method for maintaining the accurate time over the world. NTP is one of the oldest and most used protocol

### **CONTENTS**

### **CHAPTER – 1**

1. 1.1 Clock Synchronization

1.2 Clock Synchronization Algorithm

**CHAPTER- 2**

2. 2.1 ESP8266 Wifi Module

2.2 Specifications of ESP8266

2.3 Features of ESP8266 NODEMCU?

2.4 **NodeMCU DEVKIT 1.0 Specification**

2.5 Installed Driver forEsp8266 2.6 Introduction of LCD16\*2 module

### **CHAPTER – 3**

3. 3.1 Implementation Code Apply

3.2 Result

**CHAPTER 1**

1.1 **NTP ( Network Time Protocol)**

In 1979, Network [time synchronization](https://en.wikipedia.org/wiki/Time_synchronization) technology was used in what was possibly the first public demonstration of [Internet](https://en.wikipedia.org/wiki/Internet) services running over a trans-Atlantic satellite network, at the [National Computer Conference](https://en.wikipedia.org/wiki/National_Computer_Conference) in New York. The technology was later described in the 1981 Internet Engineering Note (IEN) 173[[6]](https://en.wikipedia.org/wiki/Network_Time_Protocol#cite_note-6) and a public protocol was developed from it that was documented in [RFC](https://en.wikipedia.org/wiki/Request_for_Comments_(identifier)) [778](https://tools.ietf.org/html/rfc778). The technology was first deployed in a local area network as part of the Hello routing protocol and implemented in the [Fuzzball router](https://en.wikipedia.org/wiki/Fuzzball_router" \o "Fuzzball router), an experimental operating system used in network prototyping, where it ran for many years.

Other related network tools were available both then and now. They include the [Daytime](https://en.wikipedia.org/wiki/Daytime_Protocol) and [Time](https://en.wikipedia.org/wiki/Time_Protocol) protocols for recording the time of events, as well as the [ICMP Timestamp](https://en.wikipedia.org/wiki/ICMP_Timestamp) and IP Timestamp option ([RFC](https://en.wikipedia.org/wiki/Request_for_Comments_(identifier)) [781](https://tools.ietf.org/html/rfc781)). More complete synchronization systems, although lacking NTP's data analysis and clock disciplining algorithms, include the Unix daemon *timed*, which uses an election algorithm to appoint a server for all the clients;[[7]](https://en.wikipedia.org/wiki/Network_Time_Protocol" \l "cite_note-7) and the **Digital Time Synchronization Service** (DTSS), which uses a hierarchy of servers similar to the NTP stratum model.

In 1985, NTP version 0 (NTPv0) was implemented in both Fuzzball and Unix, and the NTP packet header and round-trip delay and offset calculations, which have persisted into NTPv4, were documented in [RFC](https://en.wikipedia.org/wiki/Request_for_Comments_(identifier)) [958](https://tools.ietf.org/html/rfc958). Despite the relatively slow computers and networks available at the time, accuracy of better than 100 [milliseconds](https://en.wikipedia.org/wiki/Millisecond) was usually obtained on Atlantic spanning links, with accuracy of tens of milliseconds on [Ethernet](https://en.wikipedia.org/wiki/Ethernet) networks.

In 1988, a much more complete specification of the NTPv1 protocol, with associated algorithms, was published in [RFC](https://en.wikipedia.org/wiki/Request_for_Comments_(identifier)) [1059](https://tools.ietf.org/html/rfc1059). It drew on the experimental results and clock filter algorithm documented in [RFC](https://en.wikipedia.org/wiki/Request_for_Comments_(identifier)) [956](https://tools.ietf.org/html/rfc956) and was the first version to describe the [client-server](https://en.wikipedia.org/wiki/Client-server_model) and [peer-to-peer](https://en.wikipedia.org/wiki/Peer-to-peer) modes. In 1991, the NTPv1 architecture, protocol and algorithms were brought to the attention of a wider engineering community with the publication of an article by [David L. Mills](https://en.wikipedia.org/wiki/David_L._Mills) in the [IEEE Transactions on Communications](https://en.wikipedia.org/wiki/IEEE_Transactions_on_Communications)

NTP

NTP is the Internet standard time synchronization protocol. The primary author and maintainer is David Mills at the University of Delaware.

The NTP code has been ported to a bewildering variety of Unix and non-Unix machines and is rigorously backwards compatible.

NTP can be run as a daemon which regularly polls a group of time servers and keeps the system clock in synch from moment to moment. Accuracy to within hundredths of a second is standard.

The NTP distribution also comes with an ntpdate program (similar to the BSD rdate program) which can be used on client machines at boot time or called every few hours out of cron to keep less critical machines in rough synchronization with the rest of the organization (a similar tool for Win32 machines is available for free from http://www.thinkman.com/dimension4/index.html).

When run in daemon mode, NTP is essentially its own proxy. The usual configuration is to have the hosts on your external network synch with accurate clock sources on the Internet and then have your internal hosts synchronize off the hosts on your external network.

Time synchronization is vitally important to your organization. From a security perspective, effective prosecution of security incidents requires accurate matching timestamps on all log files. Any discrepancies will complicate or sabotage legal proceedings.

There is also a reasonably large body of security software which requires accurate time information to work effectively. If you are a software development organization, correct time information across your NFS servers and clients can make or break your development-- particularly if you use a parallel/distributed make product.

NTP v3 is the current Internet standard version of the protocol. When initially released in 1992, NTP v2 was still in heavy use particularly on PC platforms.

The v3 release was commonly called "XNTP" to distinguish it from v2 implementations. NTP v4 has gone back to just "NTP". More info from a personal message from David Mills (5/2/2000): "NTP version 0-4 were/are real and distinct. See rfc1305 appendix [D] for historic conformance statement. Versions 1, 2 and 3 were documented in rfcs; version 4 is still wet. The reference implementations for 0-3 were on PDP11 fuzzballs.

Later reference implementations for 3, 4 are in Unix. … The original implementation which evolved to NTP was in the Hello routing algorithm used in the fuzzballs from 1979.

It was documented in an early Internet Experiment Note circa 1981-3.

I don't remember the exact rollout dates, but that doesn't matter much, since the architecture, protocol and algorithms evolved essentially in a continuous manner to the present day

**1.2 Global Architecture**

Like many other Internet-based distributed system, NTP is hierarchically-oriented. That is, there is a small core of primary time servers who set their clocks against external, highly accurate sources of time information (Cesium clocks, GPS receivers, etc.).

Below this level are secondary servers which are responsible for distributing time from the primary servers to the rest of the Internet. Secondary servers are required because, while a given NTP server can service hundreds (if not thousands) of other time servers, the number of machines requiring time synchronization on the Internet today numbers into the millions of machines.

As we will see later in the course, this hierarchical structure is continued as the NTP infrastructure permeates the individual organizations which are connected to the Internet– that is each organization synchronizes several tertiary servers to the publicly available secondary servers, and then proceeds to synchronize other hosts in their enterprise against these local tertiary machines.

Pictorially speaking, the Internet time hierarchy looks something like what is shown above. At the top level are the relatively few (somewhere between 200 and 300) "publicly available" primary servers.

Each of these machines shares time information with several dozen secondary servers– there is some overlap, but there must be thousands of publicly available secondary servers on the Internet today.

Below the secondary server level are all of the organizations with direct Internet connections. These organizations synchronize off the secondary server tier.

In other words, if your host is synching against one stratum 1 server and two stratum 2 servers, then your host is a stratum 2 server. Stratum values are dynamic- - if you suddenly lose connectivity to that stratum 1 server, then your stratum value will drop to 3.

Generally speaking, hosts will prefer time synchronization information from lower

NTP servers can either be configured in a peer-to-peer relationship or in a master/slave relationship. A peer relationship implies that both parties are aware of each other’s existence and have the other machine configured in their local NTP configuration file. However, in a master/slave relationship only the client configuration file needs to contain the server information, so the server may be unaware of the client’s existence until the client actually requests time information from the server (“pull” rather than “push”).

Clients who take time synchronization information from a server without notifying that server’s administrator are generally referred to as clock suckers. xntpd keeps moment to moment statistics on how much average variance the local clock has from the time standard.

One of the problems with your security depending upon time information from an external source is that a knowledgeable attacker may try to impersonate your external time source and skew your machine's clock. Success may allow the attacker to replay time-based security credentials against your infrastructure.

NTP has built-in algorithms for detecting and ignoring obviously bogus time information. For these to be effective, however, your server must be receiving updates from more than one external clock source

Let's say your local NTP server is getting time information from four external sources. Each source reports their notion of the current time (represented by the dots in the picture above) and your local NTP server is able to calculate a maximal error threshold for each time source, which gives a small range of potentially "correct" time values (represented by the arrowed lines).

It turns out that the actual time must lie somewhere in the intersection of the ranges reported by the various time sources. Your local NTP server searches for the largest group of intersecting time ranges from all of its potential time sources and then sets the time based on the information from those intersecting servers. In the example shown above, sources #1, #2, and #4 all intersect to varying degrees so these are valid clock sources (true chimers).

Source #3 is reporting time outside of the range of any of the other servers, so it is invalid (a false ticker)– this may be because source #3 has been compromised by an attacker, or it may just be that source #3 has a terrible system clock or is getting time from some other wildly inaccurate time source.

Again we see the hierarchical nature of good NTP architectural design. The idea is to avoid overloading low stratum servers and to keep hosts from having to leave their local LANs for time synchronization information.

This is particularly important over WAN links. Desktop clients should probably never peer with other machines, and it may be sufficient to simply use ntpdate to keep desktop machines in synch.

A list of accurate Internet clock sources is provided on the NTP Web site

Here's a simple diagram showing how an organization with a single Internet feed might get good external time information from the Internet. Three (or possibly more) local servers are deployed on some externally connected network.

Each of these machines is synchronized against a least three external secondary servers (there are three local servers in the picture above, so nine external secondary servers are required). Each of the local servers also peers with the other two local machines

**1.3Internal Time Synch Arch**

Now let's suppose that this same organization happens to have three distinct organizational units with various LAN and WAN connections between them. Each of these organizations also has one or more local NTP servers.

Each of the NTP servers in each of the different organizational units will peer with all of the other internal NTP servers. These internal servers will also peer with the servers which are getting time information from the external Internet.

For large organizations, you may wish to configure this as a master/slave relationship rather than peer-to-peer.

There will come a time when your Internet gateway or other WAN link will go down and all of your local time servers will suddenly drop to stratum 16 and their clients will stop listening to time synchronization information.

A few carefully located pseudo clocks will prevent this from happening. Generally speaking, each of your locations should have at least one master time source with a pseudo-clock definition.

Having several pseudo-clocks per site will prevent a single machine with an inaccurate system clock from distorting your network time. You must be careful, however, to set the stratum of your pseudo-clock at least a couple of points higher than the stratum that the server usually operates at– this prevents the pseudo-clock from interfering with the information that you're acquiring from the Internet.

This means that pseudo-clocks should generally be configured at strata 5-8.

Here's our internal network architecture diagram again, this time showing appropriate stratum values for the various internal and external pseudo-clocks. The pseudo-clocks on the "Outside Servers" should be configured at stratum 5– this means that if the Internet connection becomes severed then the "Outside Servers" will become stratum 6 servers (the stratum of the pseudo-clock plus one).

This implies that the internal time servers will drop to stratum 7 when the Internet link is lost. Therefore, configure the pseudo-clocks on these internal machines at stratum 8 so that the internal machines will only synchronize off of these clocks if they lose their connectivity to the rest of the company

Most almost every router on the market today is capable of being an NTP server. Many organizations choose to simply use their router infrastructure to supply time to their entire enterprise.

**1.4 CLOCK SYNCHRONISATION**

NTP uses a hierarchical, semi-layered system of time sources. Each level of this hierarchy is termed a *stratum* and is assigned a number starting with zero for the reference clock at the top. A server synchronized to a stratum *n* server runs at stratum *n* + 1. The number represents the distance from the reference clock and is used to prevent cyclical dependencies in the hierarchy. Stratum is not always an indication of quality or reliability; it is common to find stratum 3 time sources that are higher quality than other stratum 2 time sources. A brief description of strata 0, 1, 2 and 3 is provided below.

**Stratum 0**

These are high-precision timekeeping devices such as atomic clocks, GPS or other [radio clocks](https://en.wikipedia.org/wiki/Radio_clock). They generate a very accurate [pulse per second](https://en.wikipedia.org/wiki/Pulse_per_second) signal that triggers an [interrupt](https://en.wikipedia.org/wiki/Interrupt) and timestamp on a connected computer. Stratum 0 devices are also known as reference clocks.

**Stratum 1**

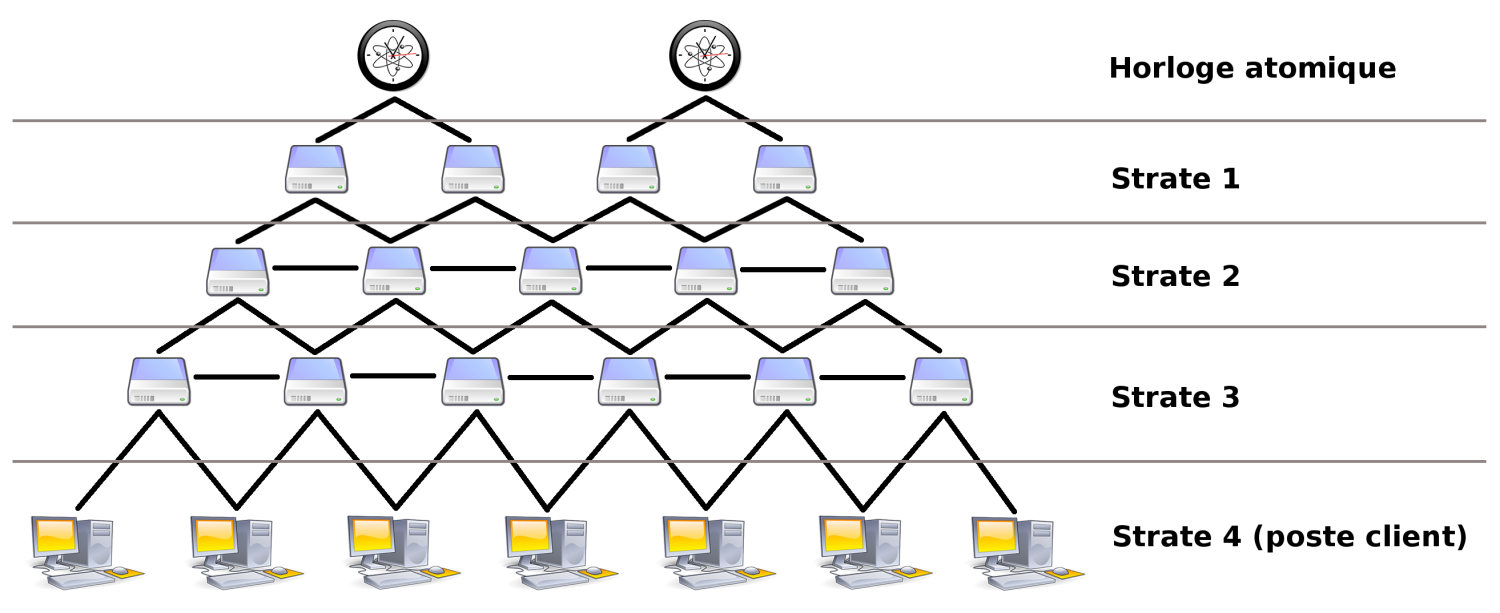
These are computers whose [system time](https://en.wikipedia.org/wiki/System_time) is synchronized to within a few microseconds of their attached stratum 0 devices. Stratum 1 servers may peer with other stratum 1 servers for [sanity check](https://en.wikipedia.org/wiki/Sanity_check) and backup. They are also referred to as primary time servers.

**Stratum 2**

These are computers that are synchronized over a network to stratum 1 servers. Often a stratum 2 computer will query several stratum 1 servers. Stratum 2 computers may also peer with other stratum 2 computers to provide more stable and robust time for all devices in the peer group.

**Stratum 3**

These are computers that are synchronized to stratum 2 servers. They employ the same algorithms for peering and data sampling as stratum 2, and can themselves act as servers for stratum 4 computers, and so on.

****The upper limit for stratum is 15; stratum 16 is used to indicate that a device is unsynchronized. The NTP algorithms on each computer interact to construct a [Bellman-Ford](https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm) shortest-path [spanning tree](https://en.wikipedia.org/wiki/Spanning_tree), to minimize the accumulated round-trip delay to the stratum 1 servers for all the clients.

*Fig.7: NTP Architecture*

**1.5 Clock Synchronization algorithm**

A typical NTP client will regularly [poll](https://en.wikipedia.org/wiki/Polling_(computer_science)) one or more NTP servers. To synchronize its clock, the client must compute its time offset and [round-trip delay](https://en.wikipedia.org/wiki/Round-trip_delay_time). Time offset *θ* is defined by

{\displaystyle \theta ={(t\_{1}-t\_{0})+(t\_{2}-t\_{3}) \over 2}}

and the round-trip delay *δ* by

*δ*  **= (t3 – t0) + (t2 – t1)**

{\displaystyle \delta ={(t\_{3}-t\_{0})-(t\_{2}-t\_{1})}}

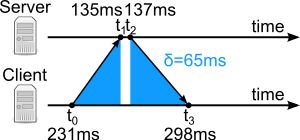
where

*t*0 is the client's timestamp of the request packet transmission,

*t*1 is the server's timestamp of the request packet reception,

*t*2 is the server's timestamp of the response packet transmission and

*t*3 is the client's timestamp of the response packet reception

****

*Fig. 8: Round Trip Delay Time*

Time offset is given by following formula:-

**Ɵ = (t1 – t2) + (t2 – t3)**

**2**

This often works well because network administration is usually a centralized corporate function and so time synchronization can be provided reasonably seamlessly throughout the entire organization.

Also routers typically appear at a "predictable" address on a given LAN (often the .1 or .254 address) and so machines can build their own NTP configuration files "on the fly" at boot time to simply synchronize against their default router.

On the minus side, your routers may already be too overloaded to be providing timing information to all of the locally attached hosts.

The router also becomes an even larger single point of failure for your LAN. Also, it's typically easier to do global updates to host-based configuration files (using tools like rdist and rsync) than it is to update the configurations on a network full of routers.

NTP allows servers to broadcast time information to local clients (rather than having the local clients continuously synchronizing against one or more central servers).

This can save some network bandwidth, though NTP is generally such a lightweight protocol that the savings is negligible.

The problem is that broadcasts are inherently unreliable– a client may regularly miss the broadcast updates and its clock may being to drift off true. Also, the client may be unable to accurately determine the delay between server and client, again resulting in inaccurate time on your client machines.

On the whole, the savings in network bandwidth (especially on modern switched network fabrics) don't seem to outweigh the disadvantages in using broadcast time synchronization.

Note that it may be unnecessary to build NTP from source– most vendor-supplied operating systems now include NTP v3 with the base OS. The build process for the Open Source NTP distribution is extremely easy thanks to a GNU configure script and significant effort on the part of the NTP porting community.

Once the configure script is completed, the administrator need only run make and then make install. By default NTP will install itself under /usr/local.

The administrator may change this path by specifying --prefix on the configure command

line: configure --prefix=/opt/ntp

**1.6 NTP -- Build process**

The xntpdc program allows the administrator to interrogate running NTP servers for statistical information and current values such as how much variance there is between the local server and the hosts it is synchronizing against.

The xntpdc program allows the administrator to query NTP servers on other hosts, including hosts at other organizations!

Try the command xntpdc –p to get information about which machines a given is synchronizing with. tickadj is used to modify various kernel parameters which control features like the granularity of the system clock. This program is not required for SYSV based systems.

One of the kernel parameters set by the tickadj program on BSD systems is the value of dosynctodr– this parameter controls the relationship between the system hardware clock and the system software clock.

This parameter sometimes needs to be tweaked on SYSV-based systems as well, but there has been some historical confusion about whether this is necessary under Solaris. From Sun's "Symptoms and Resolutions Database", SRDB #19195:

The common lore for setting up xntpd on Solaris using the freeware version included the warning to set the kernel variable dosynctodr to 0 in the /etc/system file thus: set dosynctodr=0 When using NTP on Solaris 2.6 or later, the kernel variable MUST be left at the default value of 1.

Prior to 2.6 this variable controlled whether or not to rein in the soft clock using the hardware clock, with the result that NTP and the hardware clock would fight for control of the soft clock; thus before 2.6 you had to set dosynctodr to 0.

At 2.6, every system call that adjusts the soft clock also sets the hard clock, thus while NTP controls the soft clock, the hard clock is also controlled.

Setting dosynctodr to 0 reverts the behaviour back to the pre 2.6 default behaviour, having exactly the opposite effect as that intended. Do not set dosynctodr to 0.

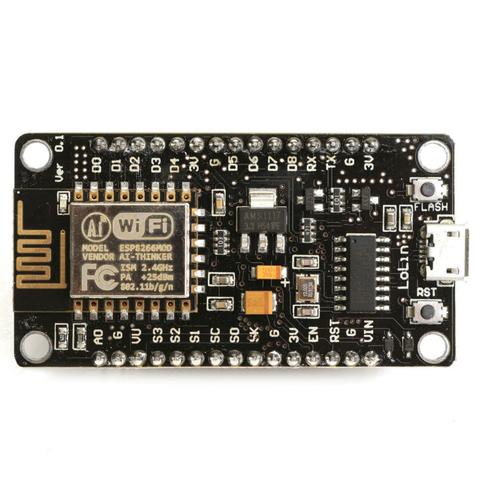
**CHAPTER 2**

**2.1 ESP8266 Wifi Module(**NodeMCU DEVKIT 1.0)

### **NodeMCU** is an open source [IoT](https://en.wikipedia.org/wiki/Internet_of_Things" \o "Internet of Things) platform. It includes [firmware](https://en.wikipedia.org/wiki/Firmware) which runs on the [ESP8266](https://en.wikipedia.org/wiki/ESP8266) [Wi-Fi](https://en.wikipedia.org/wiki/Wi-Fi) [SoC](https://en.wikipedia.org/wiki/System_on_a_chip" \o "System on a chip) from Espressif Systems, and hardware which is based on the ESP-12 module.[[6]](https://en.wikipedia.org/wiki/NodeMCU#cite_note-Espressif_Systems-6)[[7]](https://en.wikipedia.org/wiki/NodeMCU#cite_note-7)

### The term "NodeMCU" by default refers to the firmware rather than the development kits. The firmware uses the [Lua](https://en.wikipedia.org/wiki/Lua_(programming_language)" \o "Lua (programming language)) scripting language. It is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266.

### It uses many open source projects, such as lua-cjson[[8]](https://en.wikipedia.org/wiki/NodeMCU" \l "cite_note-8) and [SPIFFS](https://en.wikipedia.org/w/index.php?title=SPIFFS&action=edit&redlink=1).[[9]](https://en.wikipedia.org/wiki/NodeMCU#cite_note-spiffs-9)



*Fig 2.1 ESP8266 NodeMCU wifi module*

### **2.2 Specifications of ESP8266**

\* 802.11b/g/n protocol

\* Wifi-Direct(P2P), soft-AP

\* Intergrated TR switch, balun, LNA, power amplifier and matching network

\* +19.5dbm output power in 802.11b mode

\* Support antenna diversity

\*Power down leakage current of <10uA

\* Intergrated low power 32-bit CPU could be used as application processor

\* SDIO 2.0 SPI, UART

\* STBC, 1\*1 MIMO, 2\*1 MIMO

\* A-MPDU and A-MSDU

\* Wake up and transmit aggregation and 0.4s guard interval

\* Standby power consumptions of <1.0mW

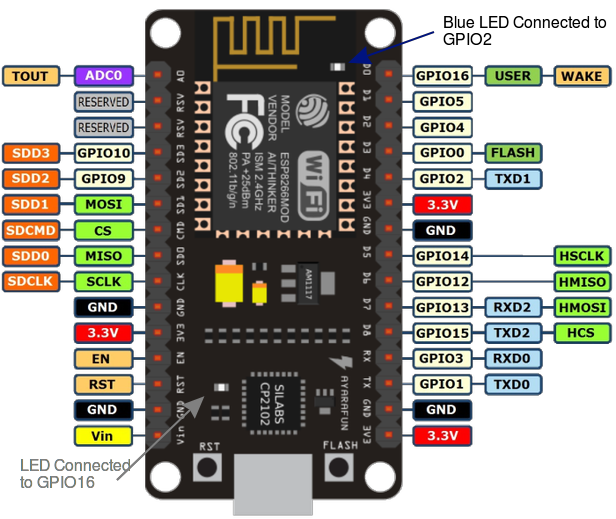
### **2.3 Features of ESP8266 NODEMCU?**

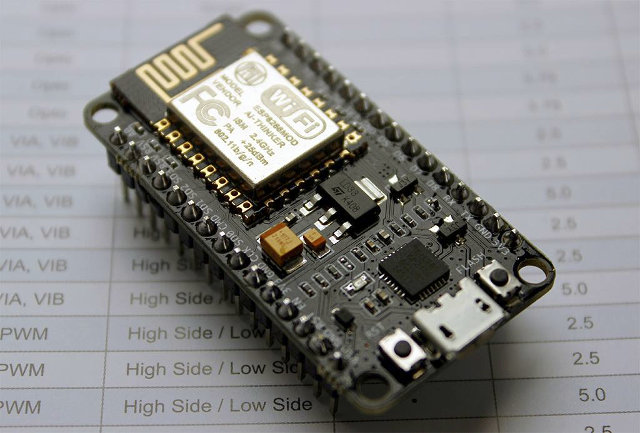
* Open-source
* Interactive
* Programmable
* Low cost
* Simple
* Smart
* WI-FI enabled
* USB-TTL included
* Plug & Play

**2.4 NodeMCU DEVKIT 1.0 Specification:**

**Developer :** ESP8266 Opensource Community  
**Type :**  Single-board microcontroller  
**Operating system :** XTOS  
**CPU :** ESP8266  
**Memory :** 128kBytes  
**Storage :** 4MBytes  
**Power By :** USB  
**Power Voltage :** 3v ,5v (used with 3.3v Regulator which inbuilt on Board using Pin VIN)  
**Code :** Arduino Cpp  
**IDE Used :** Arduino IDE  
**GPIO :** 10

**Pin Out Diagram**

****

****

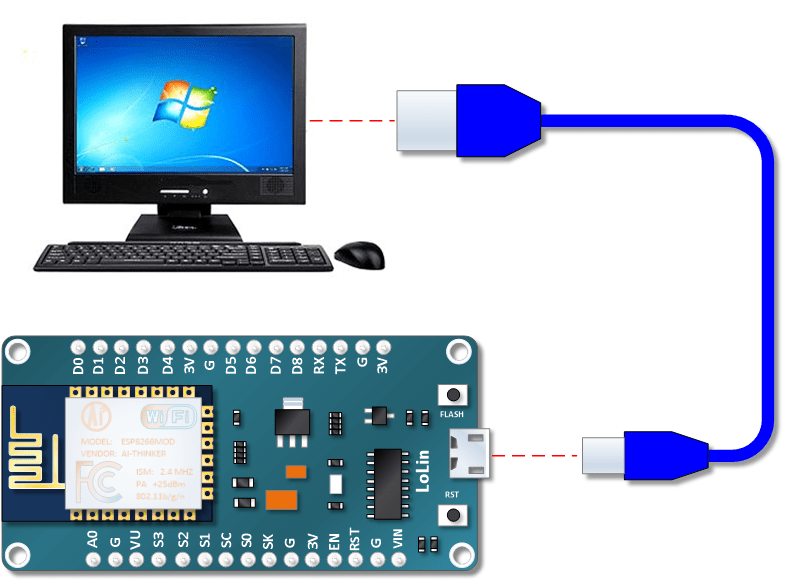
**Arduino Software**

**Arduino** is an [open-source hardware](https://en.wikipedia.org/wiki/Open-source_hardware) and [software](https://en.wikipedia.org/wiki/Open-source_software) company, project and user community that designs and manufactures [single-board microcontrollers](https://en.wikipedia.org/wiki/Single-board_microcontroller) and [microcontroller](https://en.wikipedia.org/wiki/Microcontroller) kits for building digital devices and interactive objects that can sense and control both physically and digitally.

**Connect Arduino To Esp8266 module**

### **Step 1: Connect your NodeMCU to your computer**

You need a USB micro B cable to connect the board. Once you plugged it in, a blue LED will start flashing. If your computer is not able to detect the NodeMCU board,



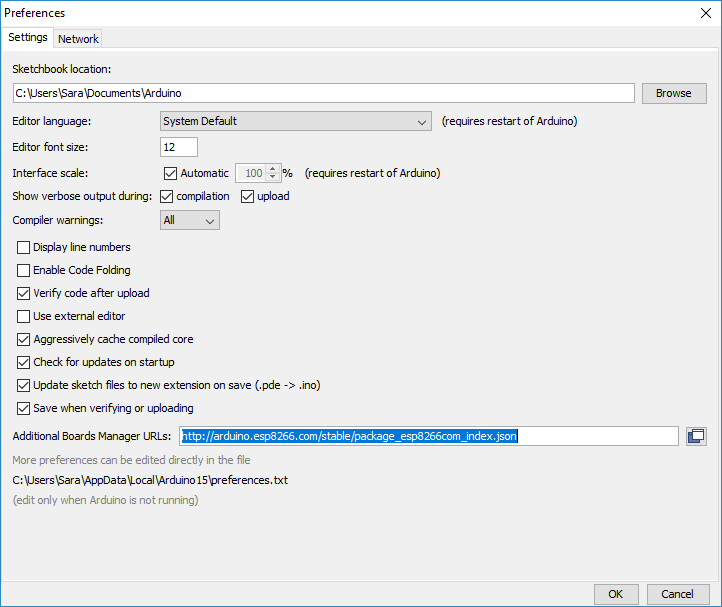
**2.5 Installed Driver for Esp8266**

CP210x USB to UART Bridge VCP Drivers

### **Step 2: Open Arduino IDE**

You need to have at least Arduino IDE version 1.6.4 to proceed with this.

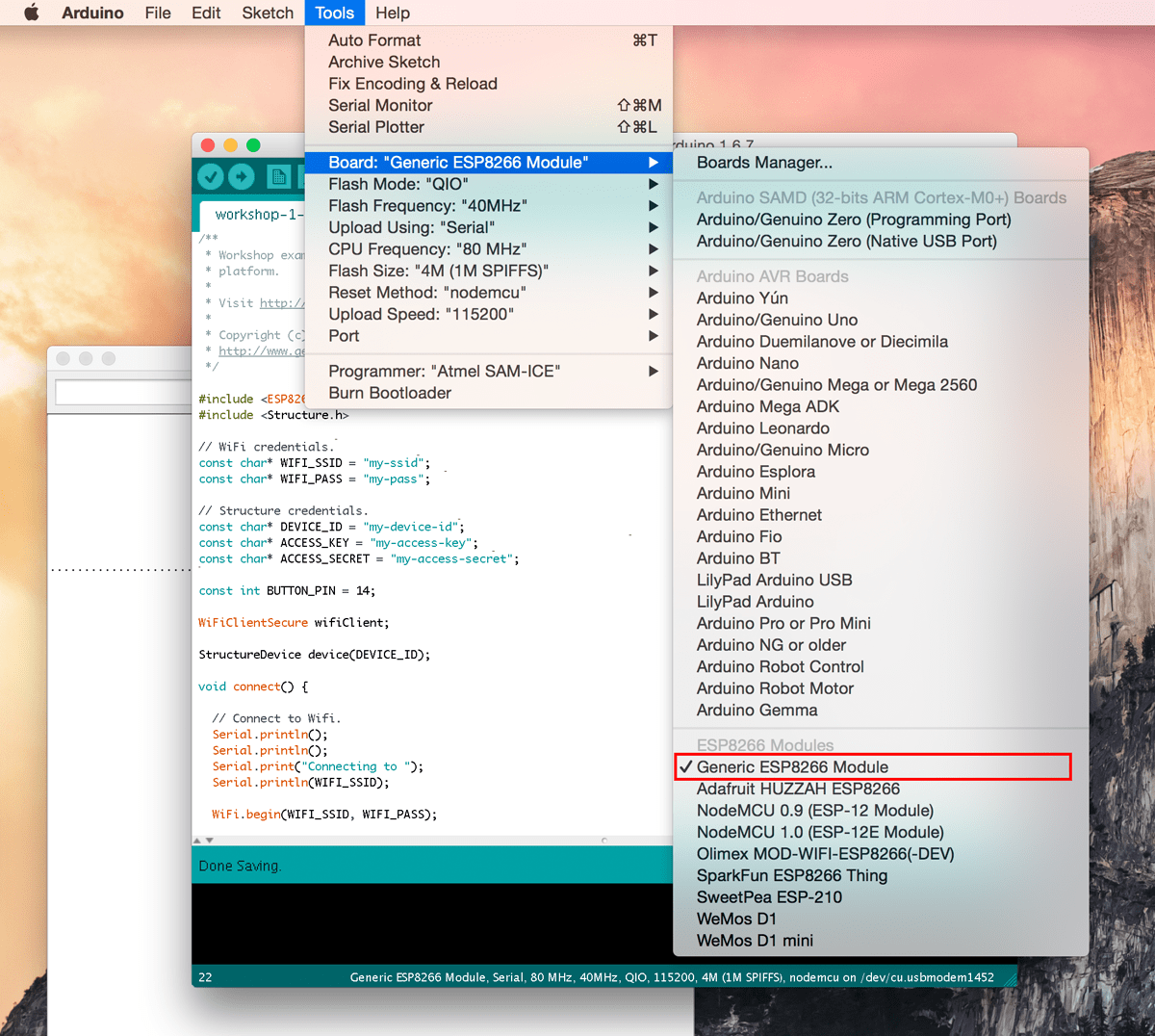
1. Go to File > Preferences

****

**2.** n the "Additional Boards Manager URLs" field, type (or copy-paste) http://arduino.esp8266.com/stable/package\_esp8266com\_index.json. Don't forget to click OK!

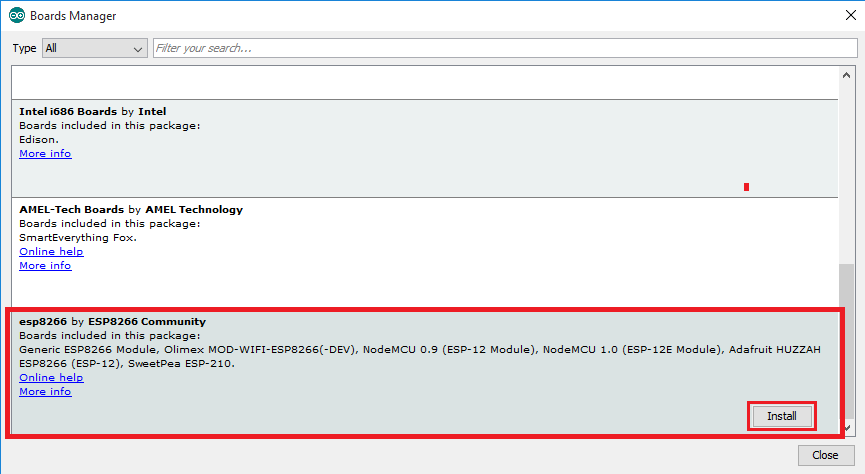
http://arduino.esp8266.com/stable/package\_esp8266com\_index.json

Step 3. Then go to  Tools > Board > Board Manager

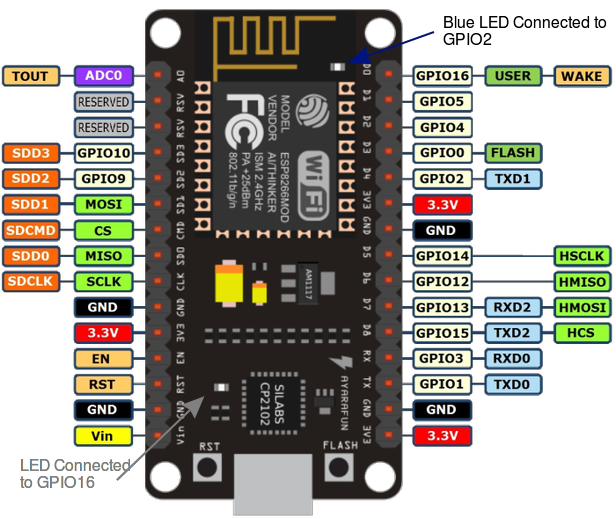


1. Type "esp8266" in the search field. The entry "esp8266 by ESP8266 Community

Click that entry and look for the install button on the lower right.

****

****

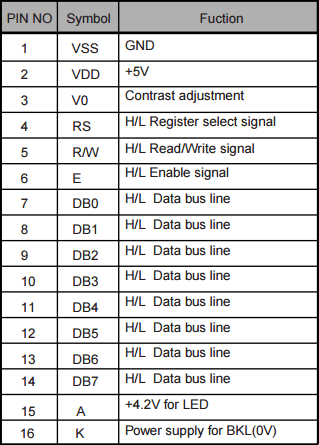
* ****

****

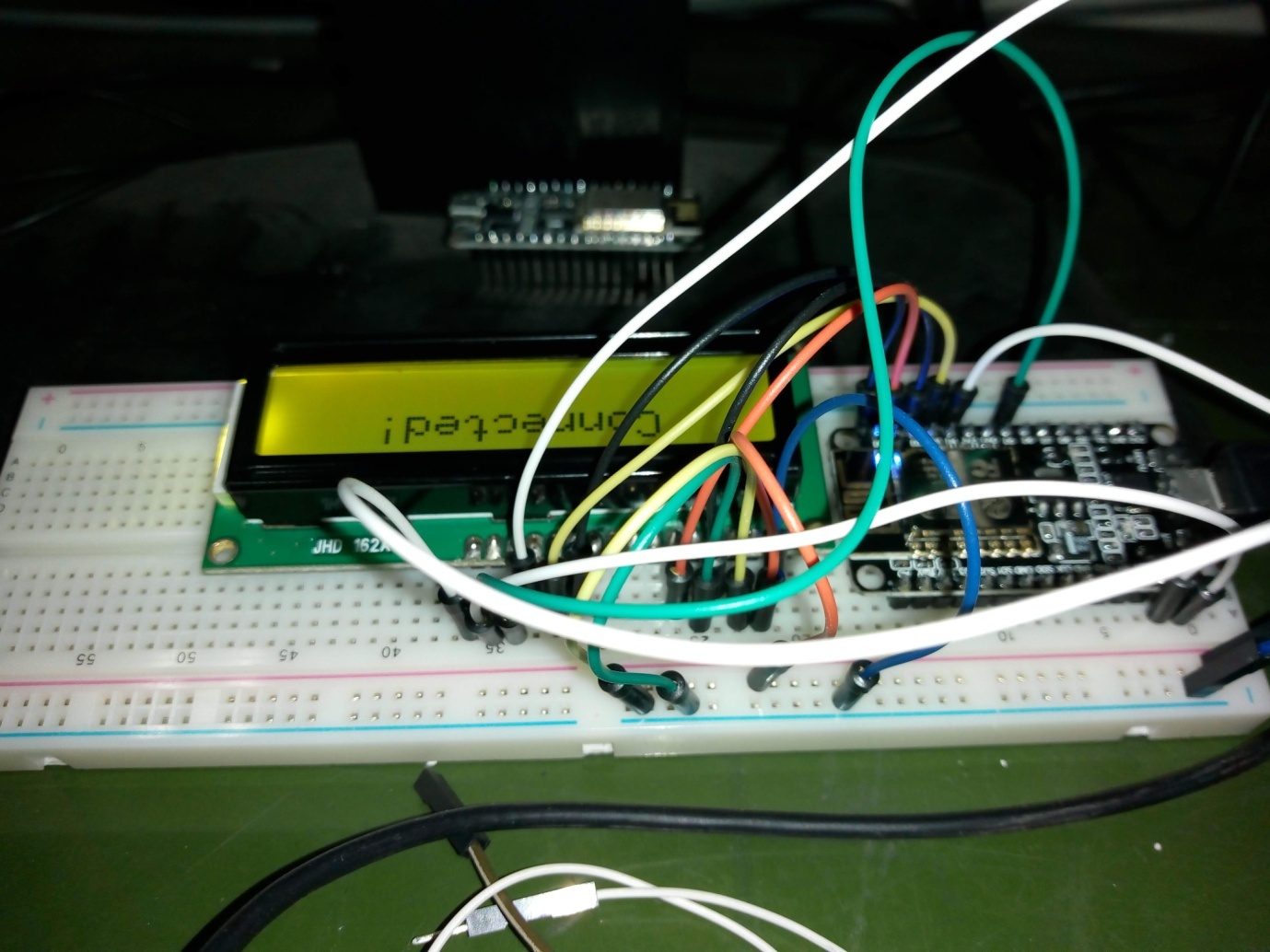
*Fig. 3: JHD 16\*2 M7 LCD*

**2.6 Introduction of LCD16\*2 module**

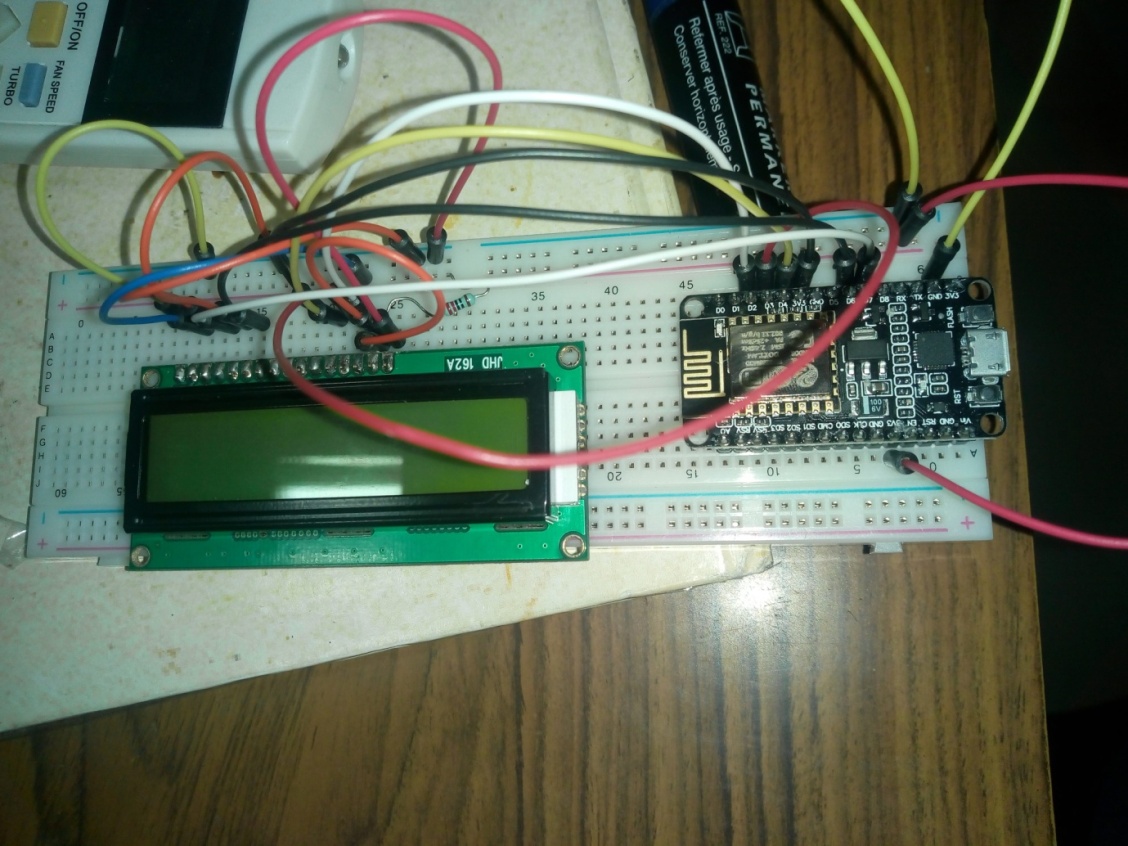
An LCD is an electronic display module which uses liquid crystal to produce a visible image. The 16\*2 LCD display is a very basic module commonly used in DIYs and circuits. The 16\*2 translates on a display 16\*2 characters per line in 2 such lines. In this LCD each character is displayed in a 5\*7 pixel matrix



*Fig. 9: LCD Pins functions*

****

*Fig. 5: ESP8266 module connect with LCD16\*2*

*Fig6.Esp8266 wifi with LCD to Display time*

**CHAPTER 3**

**3.1 Implementation code apply**

* Open Arduino IDE
* Go to File and select Preference
* Then Put this URL on Additional board manager
* (http://arduino.esp8266.com/stable/package\_esp8266com\_index.json)
* Then select tools and Go to Board Manager
* Connect LCD with ESP8266 module
* Select NodeMCU wifi module
* Run the Code in Arduino IDE

**3.2 Main Code:-**

#include <ESP8266WiFi.h>

#include <WiFiUdp.h>

#include <LiquidCrystal.h>

#include <time.h>

#ifndef STASSID

#define STASSID "SAGAR"

#define STAPSK "HONEYNINNI"

#endif

#define D0 16 // GPIO3

#define D1 5 // GPIO1

#define D2 4 // GPIO16

#define D3 0 // GPIO5

#define D4 2 // GPIO4

#define D5 14 // GPIO14

#define BUF\_LEN 256

char buffer\_time[26];

char buffer\_date[26];

struct tm\* tm\_info;

LiquidCrystal lcd(D0, D1, D2, D3, D4, D5);

const char \* ssid = STASSID; const char \* pass = STAPSK;

unsigned int localPort = 2390;

IPAddress timeServerIP;

const char\* ntpServerName = "time.nplindia.org";

const int NTP\_PACKET\_SIZE = 48;

byte packetBuffer[ NTP\_PACKET\_SIZE];

WiFiUDP udp;

int h = 0; int m = 0; int s = 0; int y = 0; int M = 0;

int offset = -1;

void setup() {

Serial.begin(115200);

lcd.begin(16, 2);

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("Connecting to ");

lcd.setCursor(0, 1);

lcd.print(ssid);

WiFi.mode(WIFI\_STA);

WiFi.begin(ssid, pass);

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

delay(500);

Serial.print(".");

}

lcd.clear();

lcd.setCursor(0, 0);

lcd.println("WiFi connected");

lcd.clear();

lcd.setCursor(0, 0);

lcd.println("NTP Sync...........");

udp.begin(localPort);

}

int MAX\_WAIT = 10, WAIT\_SEQ = MAX\_WAIT+1;

void loop() {

//Serial.println(udp\_id");

WiFi.hostByName(ntpServerName, timeServerIP);

sendNTPpacket(timeServerIP);

int cb = udp.parsePacket();

if (!cb) {

Serial.println("no packet yet");

}

else {

Serial.print("packet received, length=");

Serial.println(cb);

if(WAIT\_SEQ > MAX\_WAIT || tm\_info->tm\_sec == 59)

{

WAIT\_SEQ = 1;

Serial.print("i am called......");

udp.read(packetBuffer, NTP\_PACKET\_SIZE);

unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);

unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);

unsigned long secsSince1900 = highWord << 16 | lowWord;

Serial.print("Seconds since Jan 1 1900 = ");

Serial.println(secsSince1900);

Serial.print("Unix time = ");

const unsigned long seventyYears = 2208988800UL;

unsigned long epoch = secsSince1900 - seventyYears+19800+offset;

time\_t date\_time;

date\_time = epoch;

tm\_info = localtime(&date\_time);

}

else

{

WAIT\_SEQ++;-

tm\_info->tm\_sec++;

}

strftime(buffer\_time, 26, "%H:%M:%S", tm\_info);

strftime(buffer\_date, 26, "%d/%m/%Y", tm\_info);

lcd.clear();

lcd.setCursor(0, 0); lcd.print("IST "+String(buffer\_time));

lcd.setCursor(0, 1); lcd.print("DATE "+String(buffer\_date));

}

delay(1000);

}

void sendNTPpacket(IPAddress& address) {

Serial.println("sending NTP packet...");

memset(packetBuffer, 0, NTP\_PACKET\_SIZE);

packetBuffer[0] = 0b11100011;

packetBuffer[1] = 0;

packetBuffer[2] = 6;

packetBuffer[3] = 0xEC;

packetBuffer[12] = 49;

packetBuffer[13] = 0x4E;

packetBuffer[14] = 49;

packetBuffer[15] = 52;

udp.beginPacket(address, 123);

udp.write(packetBuffer, NTP\_PACKET\_SIZE);

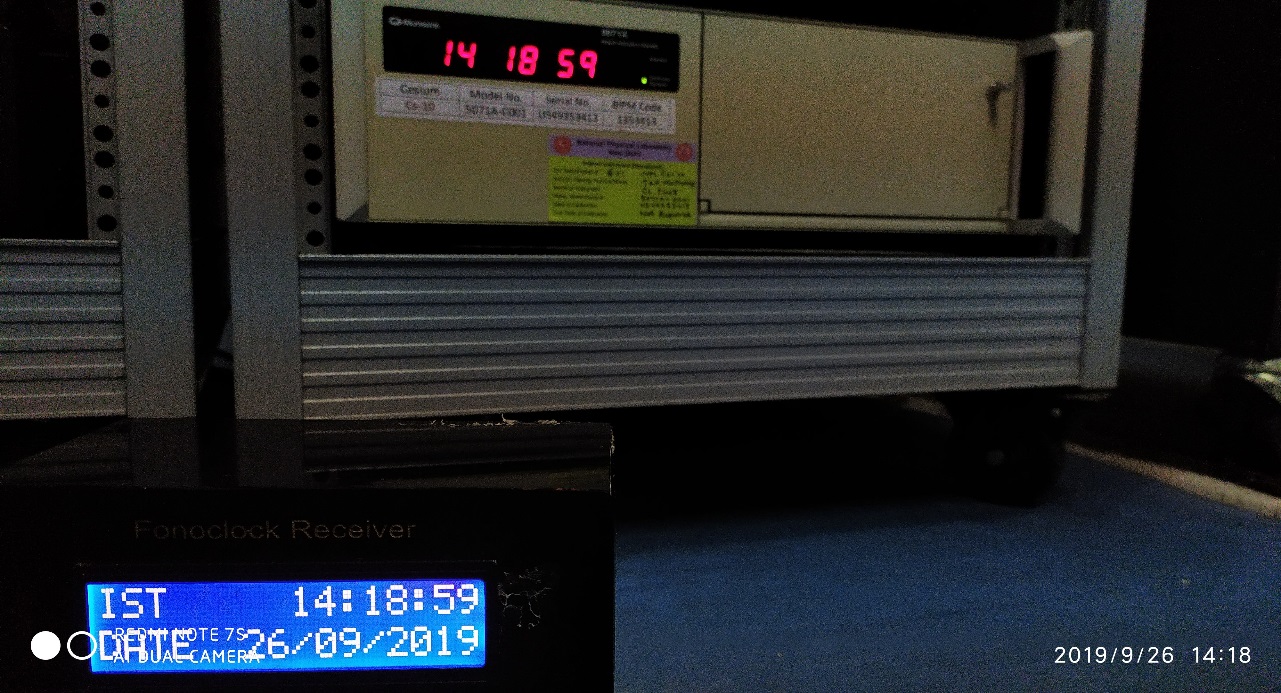
udp.endPacket();

}

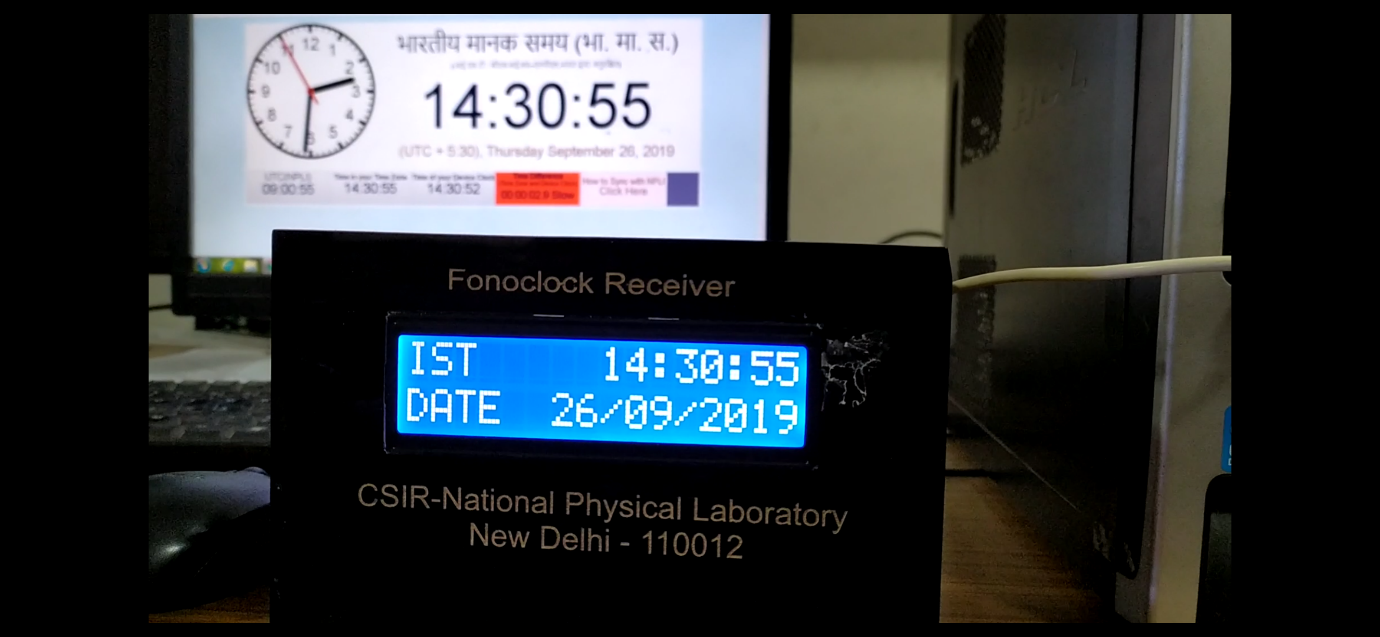
**RESULT & SCOPE**

IST (Indian Standard Time) show successfully on LCD Screen using ESP8266 wifi through NTP Server ([www.time.nplindia.org](http://www.time.nplindia.org)).

Scope is that to reduce delay of showing Indian Standard Time because reaching NTP server and getting time from it to display on a receiver produces some delay too.

****

*Fig. 10: Final Complete Showing IST (Indian Standard Time)*

**