

Wireless communication and the XBee modules, & their alternatives

Wireless communication is among technology's biggest contributions to mankind. Wireless communication involves the transmission of information over a distance without help of wires, cables or any other forms of electrical conductors. The transmitted distance can be anywhere between a few meters (for example, a television's remote control) and thousands of kilometres (for example, radio communication).

Some of the devices used for wireless communication are cordless telephones, mobiles, GPS units, wireless computer parts, and satellite television.

Advantages

Wireless communication has the following advantages:

- i. Communication has enhanced to convey the information quickly to the consumers.
- ii. Working professionals can work and access Internet anywhere and anytime without carrying cables or wires wherever they go. This also helps to complete the work anywhere on time and improves the productivity.
- iii. Doctors, workers and other professionals working in remote areas can be in touch with medical centres through wireless communication.
- iv. Urgent situation can be alerted through wireless communication. The affected regions can be provided help and support with the help of these alerts through wireless communication.
- v. Wireless networks are cheaper to install and maintain.

Disadvantages

The growth of wireless network has enabled us to use personal devices anywhere and anytime. This has helped mankind to improve in every field of life but this has led many threats as well.

Wireless network has led to many security threats to mankind. It is very easy for the hackers to grab the wireless signals that are spread in the air. It is very important to secure the wireless network so that the information cannot be exploited by the unauthorized users. This also increases the risk to lose information. Strong security protocols must be

created to secure the wireless signals like WPA and WPA2. Another way to secure the wireless network is to have wireless intrusion prevention system.

Types of wireless communication

The different types of wireless communication technologies include:

i. Infrared (IR) wireless communication:

IR wireless communication communicates data or information in devices or systems through infrared (IR) radiation. Infrared is electromagnetic energy at a wavelength that is longer than that of red light.

Working:

IR wireless is used for short and medium-range communications and security control. For IR communication to work, the systems mostly operate in *line-of-sight mode* which means that there must be no obstruction between the transmitter (source) and receiver (destination).

Infrared is used in television remote controls and security systems.

In the electromagnetic spectrum, infrared radiation lies between microwaves and visible light, therefore, they can be used as a source of communication.



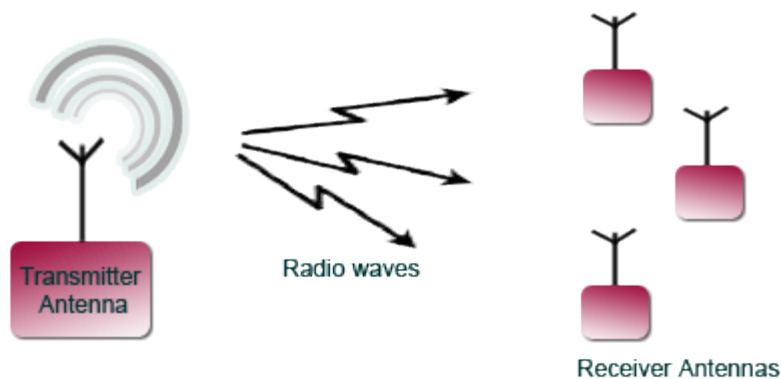
Working of Infrared Wireless Communication

A photo LED transmitter and a photodiode receptor are required for successful IR communication. The LED transmitter transmits the infrared signal in the form of non-visible light, which is captured and retrieved as information by the photo receptor. In this way, the information between the source and the target is transferred.

The source and/or destination can be laptops, mobile phones, televisions, security systems and any other device that supports wireless communication.

ii. Broadcast Radio

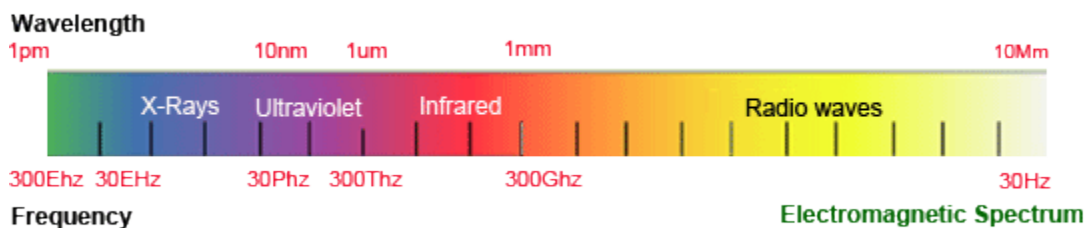
Basically an audio broadcasting service, radio broadcasts sound through the air as radio waves. It uses a transmitter to transmit radio waves to a receiving antenna. To broadcast common programming, stations are linked to the radio networks. The broadcast occurs either in syndication or simulcast (simultaneous broadcast) or both. Radio broadcasting can also be done via cable FM, the internet and satellites. A radio broadcast sends data over long distances (across countries) at up to 2 megabits per second (AM/FM Radio).



What is Broadcast Radio

Working:

Radio waves are electromagnetic signals transmitted by an antenna. Radio waves have different frequency segments, and you will be able to pick up an audio signal by tuning into a specific frequency segment.



Let us take an example of a radio station. When the Radio Jockey says “You are listening to 93.7 FM”, what he actually means is that signals are being broadcasted at a frequency

of 97.3 megahertz, which in turn means that the transmitter at the station is oscillating at a frequency of 93,700,000 cycles per second.

When you wish to listen to 93.7 FM, all you have to do is tune the radio to accept that particular frequency and you will receive flawless audio reception.

iii. Microwave Radio

Microwave transmission involves the transfer of voice and data through the atmosphere as super high-frequency radio waves called microwaves. Microwave transmission is mainly used to transmit messages between ground-based stations and satellite communications systems.

Working:

Microwave transmission mainly uses radio waves whose wavelengths are conveniently measured in small units such as centimeters. Microwaves belong to the radio spectrum ranges of roughly 1.0 gigahertz (GHz) to 30 GHz.

Antennas used in microwave transmissions are of convenient sizes and shapes. Microwave transmission depends on line-of-sight in order to work properly. For two way communications to take place, two frequencies are used. However, this does not require two antennas because the frequencies can be dealt with one antenna at both ends.

The distance covered by microwave signals relies on the height of the antenna. Each antenna is built with a fitted repeater to regenerate the signal before passing it on to the next antenna in line. The ideal distance between each antenna is approximately 25 miles.

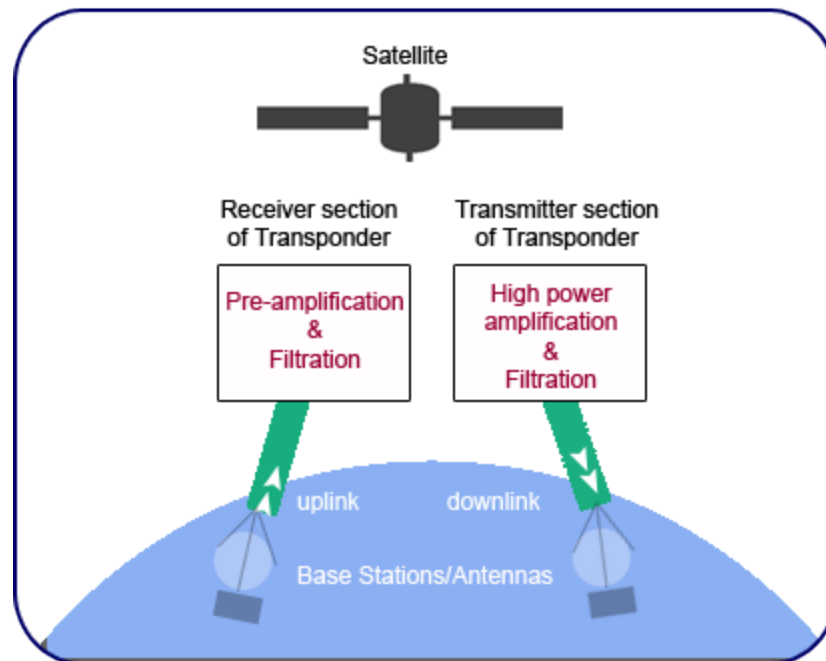
The main drawback of microwave signals is that they can be affected by bad weather, especially rain.

iv. Communications Satellites

A communication satellite is an artificial satellite used specifically as a communication transmitter/receiver in orbit. It behaves like a radio relay station above the earth to receive, amplify, and redirect analog and digital signals carried on a specific radio frequency.

Working:

Data is passed through a satellite using a transponder which is a signal path. Most satellites have between 24 to 72 transponders, with a single transponder capable of transmitting and receiving 155 million bits of information per second. This huge capability makes communication satellites an ideal medium for transmitting and receiving all kinds of content, including audios and videos.



How Satellite Communication works

Satellites transmit information by using frequency bands known as C-band and the higher Ku-band. In the near future, the use of a much higher frequency band known as Ka-band is expected to increase.

Applications of Wireless Communication

Television Remote Control – Modern televisions use wireless remote control. Currently radio waves are also used.

Wi-Fi – This is a wireless local area network that establishes internet connection with the portable computers.

Security systems – For homes and office buildings, hard wired implementation security systems are replaced by the Wireless technology.

Cellular Telephone – Radio waves are used to facilitate the operator to make phone calls from any place on the earth. [CDMA](#), GSM, and [3G](#) are examples of the advancement made by wireless communication in the domain.

Wireless energy transfer – A process where a power source transmits electrical energy to electrical load which does not have built-in power source wirelessly.

Computer Interface Devices – Computer hardware manufacturers had realized that having so many wires to communicate between devices would confuse the consumer. So they switched to wireless technology to facilitate their consumers, thus making it easy to mediate between a computer and other peripherals including mouse and keyboard. Earlier, such units required bulky, highly limited transceivers but recent generations of computer peripherals use compact and high-quality wireless devices such as [Bluetooth](#) for communication. These days, wireless devices have become very common and are preferred for their ease of handling and reliability. In reality, wireless-enabled devices have a slightly slower response time than conventional wired devices. This issue is being addressed by manufacturers and will be taken care of in the near future. Initial concerns that had risen regarding the security of wireless keyboards have also been taken care of with the advent of technology.

The Arduino XBee shield

The Arduino Xbee shield allows your Arduino board to communicate wirelessly using Zigbee. It was developed in collaboration with [Libelium](#). **Digi XBee** is the brand name of a family of form factor compatible radio modules from [Digi International](#). The first XBee radios were introduced under the **MaxStream** brand in 2005 and were based on the IEEE 802.15.4-2003 standard designed for point-to-point and star communications at over-the-air baud rates of 250 kbit/s

Two models were initially introduced — a lower cost 1 mW **XBee** and the higher power 100 mW **XBee-PRO**. Since the initial introduction, a number of new XBee radios have been introduced and an ecosystem of wireless modules, gateways, adapters and software has evolved.

The XBee radios can all be used with the minimum number of connections — power (3.3 V), ground, data in and data out ([UART](#)), with other recommended lines being Reset and Sleep. Additionally, most XBee families have some other flow control, input/output (I/O), analog-to-digital

converter (A/D) and indicator lines built in. A version called the programmable XBee has an additional on-board processor for user's code.

The programmable XBee and a [surface-mount](#) version of the XBee radios were both introduced in 2010.

There are lots of different types of modules which we are going to go over, but one of the nice things about these is that all the modules regardless of the series or type have similar pinouts. Power, ground, and your TX/RX lines are in the same place making the chips pretty interchangeable for most of the simpler applications. Some of the more advanced features are not always compatible, but for starters its not something to worry about. Now that you are ready to start learning about XBee and what it all means here is a breakdown of the XBee world.

What's an XBee, what's a Zigbee, what's a Bumblebee?

XBee – According to Digi “XBee modules are embedded solutions providing wireless end-point connectivity to devices. These modules use the IEEE 802.15.4 networking protocol for fast point-to-multipoint or peer-to-peer networking. They are designed for high-throughput applications requiring low latency and predictable communication timing.” So basically XBee is Digi's own Zigbee based protocol. In layman's term they are wicked cool, and fairly easy to use wireless modules.

Zigbee - An alliance and a standard of cost and energy efficient mesh networks. XBee uses the Zigbee standard and adds to it and wraps it up in their own neat little package.

Bumblebee - A tuna company, an insect of the family Apidae and the genus Bombus, or a small yellow Autobot. Whichever definition you choose they are fairly awesome and completely irrelevant to our talk of wireless XBee modules.

Using the XBee with Arduino

The Xbee shield allows an Arduino board to communicate wirelessly using Zigbee. It is based on the [Xbee module from MaxStream](#). The module can communicate up to

100 feet indoors or 300 feet outdoors (with line-of-sight). It can be used as a serial/usb replacement or you can put it into a command mode and configure it for a variety of broadcast and mesh networking options. The shields breaks out each of the Xbee's pins to a through-hole solder pad. It also provides female pin headers for use of digital pins 2 to 7 and the analog inputs, which are covered by the shield (digital pins 8 to 13 are not obstructed by the shield, so you can use the headers on the board itself).

A Simple Example

You should be able to get two Arduino boards with Xbee shields talking to each other without any configuration, using just the standard Arduino serial commands (described in the [reference](#)).

To upload a sketch to an Arduino board with a Xbee shield, you'll need to put both jumpers on the shield to the "USB" setting (i.e. place them on the two pins closest to the edge of the board) or remove them completely (but be sure not to lose them!). Then, you can upload a sketch normally from the Arduino environment. In this case, upload the Communication | Physical Pixel sketch to one of the boards. This sketch instructs the board to turn on the LED attached to pin 13 whenever it receives an 'H' over its serial connection, and turn the LED off when it gets an 'L'. You can test it by connecting to the board with the Arduino serial monitor (be sure it's set at 9600 baud), typing an H, and pressing enter (or clicking send). The LED should turn on. Send an L and the LED should turn off. If nothing happens, you may have an Arduino board that doesn't have a built-in LED on pin 13 (see the [board index](#) to check for sure), in this case you'll need to supply your own.

Once you've uploaded the Physical Pixel sketch and made sure that it's working, unplug the first Arduino board from the computer. Switch the jumpers to the Xbee setting (i.e. place each on the center pin and the pin farthest from the edge of the board). Now, you need to upload a sketch to the other board. Make sure its jumpers are in the USB setting. Then upload the following sketch to the board:

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.print('H');
  delay(1000);
```



```
Serial.print('L');  
delay(1000);  
}
```

When it's finished uploading, you can check that it's working with the Arduino serial monitor. You should see H's and L's arriving one a second. Turn off the serial monitor and unplug the board. Switch the jumpers to the Xbee setting. Now connect both boards to the computer. After a few seconds, you should see the LED on the first board turn on and off, once a second. (This is the LED on the Arduino board itself, not the one on the Xbee shield, which conveys information about the state of the Xbee module.) If so, congratulations, your Arduino boards are communicating wirelessly. This may not seem that exciting when both boards are connected to the same computer, but if you connect them to different computers (or power them with an external power supply - being sure to switch the power jumper on the Arduino board), they should still be able to communicate.

A Few Notes

You can use any of the standard Arduino serial commands with the Xbee shield. With the shield's jumpers in the Xbee position, the print and println commands will send data over the Xbee shield and the USB connection (i.e. to other Xbee shields and to the computer at the same time). In this configuration, however, the board will only receive data from the Xbee shield not from the USB connection (you'll need to switch the jumpers to allow the board to receive data from the computer).

The Xbee module on the shield is set up to work at 9600 baud by default, so unless you reconfigure it, you'll need to make sure you're passing 9600 to the Serial.begin() command in your sketch.

To allow your computer to communicate directly with the Xbee shield, connect it to an Arduino board whose microcontroller has been removed and place its jumpers in the USB configuration. Then you can send data to and receive data from the Xbee module from any terminal program. This allows you, for example, to see the data that the module is receiving from other Xbee shields (e.g. to collect sensor data wirelessly from a number of locations).

Configuring the Xbee Module

You can configure the Xbee module from code running on the Arduino board or from software on the computer. To configure it from the Arduino board, you'll need to have the jumpers in the Xbee position. To configure it from the computer, you'll need to have the jumpers in the USB configuration and have removed the microcontroller from your Arduino board.

To get the module into configuration mode, you need to send it three plus signs: `+++` and there needs to be at least one second before and after during which you send no other character to the module. Note that this includes newlines or carriage return characters. Thus, if you're trying to configure the module from the computer, you need to make sure your terminal software is configured to send characters as you type them, without waiting for you to press enter. Otherwise, it will send the plus signs immediately followed by a newline (i.e. you won't get the needed one second delay after the `+++`). If you successfully enter configuration mode, the module will send back the two characters 'OK', followed by a carriage return.

| Send Command | Expected Response |
|------------------|---------------------------|
| <code>+++</code> | <code>OK<CR></code> |

Once in configuration mode, you can send AT commands to the module. Command strings have the form `ATxx` (where `xx` is the name of a setting). To read the current value of the setting, send the command string followed by a carriage return. To write a new value to the setting, send the command string, immediately followed by the new setting (with no spaces or newlines in-between), followed by a carriage return. For example, to read the network ID of the module (which determines which other Xbee modules it will communicate with), use the `'ATID` command:

| Send Command | Expected Response |
|--------------------------------|-----------------------------|
| <code>ATID<enter></code> | <code>3332<CR></code> |

To change the network ID of the module:

| Send Command | Expected Response |
|------------------------------------|---------------------------|
| <code>ATID3331<enter></code> | <code>OK<CR></code> |

Now, check that the setting has taken effect:

| Send Command | Expected Response |
|--------------------------------|-----------------------------|
| <code>ATID<enter></code> | <code>3331<CR></code> |

Unless you tell the module to write the changes to non-volatile (long-term) memory, they will only be in effect until the module loses power. To save the changes permanently (until you explicitly modify them again), use the ATWR command:

| Send Command | Expected Response |
|--------------|-------------------|
| ATWR<enter> | OK<CR> |

To reset the module to the factory settings, use the ATRE command:

| Send Command | Expected Response |
|--------------|-------------------|
| ATRE<enter> | OK<CR> |

Note that like the other commands, the reset will not be permanent unless you follow it with the ATWR command.

Jumper Settings

The Xbee shield has two jumpers (the small removable plastic sleeves that each fit onto two of the three pins labelled Xbee/USB). These determine how the Xbee's serial communication connects to the serial communication between the microcontroller (ATmega8 or ATmega168) and FTDI USB-to-serial chip on the Arduino board.

With the jumpers in the Xbee position (i.e. on the two pins towards the interior of the board), the DOUT pin of the Xbee module is connected to the RX pin of the microcontroller; and DIN is connected to TX. Note that the RX and TX pins of the microcontroller are still connected to the TX and RX pins (respectively) of the FTDI chip - data sent from the microcontroller will be transmitted to the computer via USB as well as being sent wirelessly by the Xbee module. The microcontroller, however, will only be able to receive data from the Xbee module, not over USB from the computer.

With the jumpers in the USB position (i.e. on the two pins nearest the edge of the board), the DOUT pin the Xbee module is connected to the RX pin of the *FTDI chip*, and DIN on the Xbee module is connected to the TX pin of the FTDI chip. This means that the Xbee module can communicate directly with the computer - however, *this only works if the microcontroller has been removed from the Arduino board*. If the microcontroller is left in the Arduino board, it will be able to talk to the

computer normally via USB, but neither the computer nor the microcontroller will be able to talk to the Xbee module.

Networking

The Arduino XBee shield can be used with different XBee modules. The instructions below are for the XBee 802.15.4 modules (sometimes called "Series 1" to distinguish them from the Series 2 modules, although "Series 1" doesn't appear in the official name or product description).

Addressing

There are multiple parameters that need to be configured correctly for two modules to talk to each other (although with the default settings, all modules should be able to talk to each other). They need to be on the same network, as set by the ID parameter (see "Configuration" below for more details on the parameters). The modules need to be on the same channel, as set by the CH parameter. Finally, a module's destination address (DH and DL parameters) determine which modules on its network and channel will receive the data it transmits. This can happen in a few ways:

- If a module's DH is 0 and its DL is less than 0xFFFF (i.e. 16 bits), data transmitted by that module will be received by any module whose 16-bit address MY parameter equals DL.
- If DH is 0 and DL equals 0xFFFF, the module's transmissions will be received by all modules.
- If DH is non-zero or DL is greater than 0xFFFF, the transmission will only be received by the module whose serial number equals the transmitting module's destination address (i.e. whose SH equals the transmitting module's DH and whose SL equals its DL).

Again, this address matching will only happen between modules on the same network and channel. If two modules are on different networks or channels, they can't communicate regardless of their addresses.

Configuration

Here are some of the more useful parameters for configuring your Xbee module. For step-by-step instructions on reading and writing them, see the [guide to the Xbee shield](#). Make sure to prepend `AT` to the parameter name when sending a

command to the module (e.g. to read the `ID` parameter, you should send the command `AT+ID`).

| <i>Command</i> | <i>Description</i> | <i>Valid Values</i> | <i>Default Value</i> |
|-------------------------------------|---|---|---|
| <code>ID</code> | The network ID of the Xbee module. | 0 - 0xFFFF | 3332 |
| <code>CH</code> | The channel of the Xbee module. | 0x0B - 0x1A | 0X0C |
| <code>SH</code> and <code>SL</code> | The serial number of the Xbee module (<code>SH</code> gives the high 32 bits, <code>SL</code> the low 32 bits). Read-only. | 0 - 0xFFFFFFFF (for both <code>SH</code> and <code>SL</code>) | different for each module |
| <code>MY</code> | The 16-bit address of the module. | 0 - 0xFFFF | 0 |
| <code>DH</code> and <code>DL</code> | The destination address for wireless communication (<code>DH</code> is the high 32 bits, <code>DL</code> the low 32). | 0 - 0xFFFFFFFF (for both <code>DH</code> and <code>DL</code>) | 0 (for both <code>DH</code> and <code>DL</code>) |
| <code>BD</code> | The baud rate used for serial communication with the Arduino board or computer. | 0 (1200 bps) 1 (2400 bps) 2 (4800 bps) 3 (9600 bps) 4 (19200 bps) 5 (38400 bps) 6 (57600 bps) 7 (115200 bps) | 3 (9600 baud) |

Note: although the valid and default values in the table above are written with a prefix of "0x" (to indicate that they are hexadecimal numbers), the module will not include the "0x" when reporting the value of a parameter, and you should omit it when setting values.

Here are a couple more useful commands for configuring the Xbee module (you'll need to prepend `AT` to these too).

| <i>Command</i> | <i>Description</i> |
|-----------------|---|
| <code>RE</code> | Restore factory default settings (note that like parameter changes, this is not permanent unless followed by the <code>WR</code> command). |
| <code>WR</code> | Write newly configured parameter values to non-volatile (long-term) storage. Otherwise, they will only last until the module loses power. |
| <code>CN</code> | Exit command mode now. (If you don't send any commands to the module for a few seconds, command mode will timeout and exit even without a <code>CN</code> command.) |

Different types of the XBee boards, antennas etc...

- **XBee Series 1 (also called XBee 802.15.4)** - These are the easiest to work with, they don't need to be configured, although they can benefit from it. Because they are easy to work with we recommend these especially if you are just starting out. For point to point communication these modules work as well as the Series 2 but without all the work. A Series 1 module won't say Series 1 on it, but it also won't say Series 2. If it doesn't say then your module is a Series 1. Series 1 and Series 2/ZB hardware are NOT compatible. Don't try to mix and match, don't even think about it, it won't work, not even close. Nope, stop thinking about it...! [Datasheet](#)
- **XBee Znet 2.5 (Formerly Series 2) Retired** - These are the fun ones. Series 2 modules must be configured before they can be used. They can run in a transparent mode or work with API commands, but this all depends on what firmware you configure these with. These also can run in a mesh network making them highly configurable and awesome modules. It also makes them harder to use modules. These modules are in no way compatible with the Series 1 modules so stop thinking about trying! These modules are no longer sold but are being replaced with the mostly compatible ZB modules. [Datasheet](#)
- **ZB (the current Series2ish module)** - Basically the Znet2.5 hardware with new firmware. Meaning they can also run in a transparent mode or work with API commands. They can also run in a mesh network making them highly

configurable and awesome modules. You can grab the new firmware and upgrade them yourself. The firmware between the two is not compatible (but is easily interchangeable) so you will have to pick which firmware you want to use on your network and stick with it. These are often called Series 2 modules, so if you hear someone talking about Series 2, they might be talking about these. It may not be the correct term, but it does distinguish these from the Series 1 modules which is usually all people want to know. These modules will not work in any way shape or form with the Series 1 so stop thinking about it. Stop it! [Datasheet](#)

- **2B(the even more current Series2ish module)** - These new modules improve on the hardware of the Series 2 modules improving things like power usage. They run the ZB firmware but because the hardware has been changed they can no longer run the Znet2.5 firmware. So if you are looking to add this to an existing 2.5Znet network beware. Currently some of our boards are 2B and others are ZB
- **900MHz** - Technically not a series but it is a family just like the others. The 900s can run 2 different types of firmware, the DigiMesh firmware and the Point to Multipoint firmware. Digi actually sells both modules, the hardware is the same just with different firmware. Sparkfun only sells the point to multipoint version, but hey, you can change the firmware yourself. These modules should be more or less plug and play but of course can benefit from all the cool features you can configure.
- **XSC** - Basically these are 900 modules that sacrifice data rate for range. The regular 900 modules have a data rate of 156KBps (the others are all around 250Kbps) but the XSC module is only about 10Kbps. On the other hand if you attach a high gain antenna you can get a range of about 15 miles and 6 miles with a regular antenna. These modules do not require configuration out of the box and have some other differences including a different command set so make sure you check out the datasheet.
- **XSC S3B** - This is an upgraded version of the XSC modules which is less power-hungry than the previous generation despite having a higher selectable transmitting power of 250mW. This higher Tx power allows for line-of-sight range up to 28 miles with the right antenna. The S3B modules also feature higher-throughput than the previous generation XSC modules.


Antenna, Antennas, Antennae...

- **Chip Antenna** – Basically a small chip that acts as an antenna. Quick, easy, cheap, not in the way. These are being phased out in favor of trace antennas, which are essentially the same but printed directly to the circuit board.

- **Wire Antenna** – Well its a small wire sticking up, a little more of what you think of when you think of antenna.
- **u.FL Antenna** – A tiny connector to connect your own antenna, this is great if your object is in a box and you want your antenna outside the box.
- **RPSMA Antenna** – A bigger connector to connect your own antenna, once again great if your object is in a box and you want your antenna outside the box.
- **Trace Antenna** – Also called a PCB antenna, these are formed directly on the module with conductive traces. They perform about the same as wire antennas.

Regular, Pro and other things






- **Regular vs Pro** - There are a few difference between the regular XBees and the XBee Pros. The Pros are a bit longer, use more power and cost more money. That's pretty much it. The greater power means longer range (1 mile instead of 300ft) so if you need the range or like to spend more money, then use the Pros, otherwise stick with the regular models. You can mix and match these on the same network.
- **900 vs 2.4** - Most of the Xbee modules operate at 2.4GHz, but there are a few that operate at 900MHz. Basically 900MHz can go a lot farther with a high gain antenna (up to 15miles for the Pro modules and a high gain antenna). Also the lower the frequency the greater penetration the signal has. 900MHz is also not allowed in many countries (although there are 868MHz versions available from Digi that are allowed in many other countries). You can NOT mix and match these on the same network.




| XBee Device | Range | Power Consumption | Frequency | Protocol | Tx Power | Data Rate | Antenna |
|---|--------|-------------------|-----------|----------|----------|-----------|---------|
|  XBee 1mW Chip Antenna - Series 1 | 300 Ft | 50mA @ 3.3v | 2.4GHz | 802.15.4 | 1mW | 250kbps | Chip |

| XBee Device | Range | Power Consumption | Frequency | Protocol | Tx Power | Data Rate | Antenna |
|--|--------|-------------------|-----------|-------------|----------|-----------|-------------------|
|  XBee 1mW U.FL Connection - Series 1 | 300 Ft | 50mA @ 3.3v | 2.4GHz | 802.15.4 | 1mW | 250kbps | Ext./Not Included |
|  XBee 1mW Wire Antenna - Series 1 | 300 Ft | 50mA @ 3.3v | 2.4GHz | 802.15.4 | 1mW | 250kbps | Wire |
|  XBee 1mW Trace Antenna - Series 1 | 300 Ft | 50mA @ 3.3v | 2.4GHz | 802.15.4 | 1mW | 250kbps | PCB |
|  XBee 2mW PCB Antenna - Series 2 | 400 Ft | 40mA @ 3.3v | 2.4GHz | ZigBee Mesh | 2mW | 250kbps | PCB |
|  | 400 Ft | 40mA @ 3.3v | 2.4GHz | ZigBee Mesh | 2mW | 250kbps | Ext./Not Included |

| XBee Device | Range | Power Consumption | Frequency | Protocol | Tx Power | Data Rate | Antenna |
|---|--------|-------------------|-----------|-------------|----------|-----------|-------------------|
| <u>XBee 2mW RPSMA - Series 2</u> | | | | | | | |
|  <u>XBee 2mW U.FL Connection - Series 2</u> | 400 Ft | 40mA @ 3.3v | 2.4GHz | ZigBee Mesh | 2mW | 250kbps | Ext./Not Included |
|  <u>XBee 2mW Wire Antenna - Series 2</u> | 400 Ft | 40mA @ 3.3v | 2.4GHz | ZigBee Mesh | 2mW | 250kbps | Wire |
|  <u>XBee Pro 63mW PCB Antenna - Series 2B</u> | 1 Mile | 295mA @ 3.3v | 2.4GHz | ZigBee Mesh | 63mW | 250kbps | PCB |
|  <u>XBee Pro 63mW RPSMA - Series 2B</u> | 1 Mile | 295mA @ 3.3v | 2.4GHz | ZigBee Mesh | 63mW | 250kbps | Ext./Not Included |

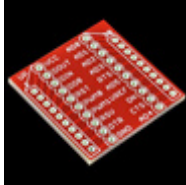


| XBee Device | Range | Power Consumption | Frequency | Protocol | Tx Power | Data Rate | Antenna |
|---|--------|-------------------|-----------|-------------|----------|-----------|-------------------|
|  XBee Pro 50mW U.FL Connection - Series 2 | 1 Mile | 295mA @ 3.3v | 2.4GHz | ZigBee Mesh | 50mW | 250kbps | Ext./Not Included |
|  XBee Pro 63mW Wire Antenna - Series 2B | 1 Mile | 295mA @ 3.3v | 2.4GHz | ZigBee Mesh | 63mW | 250kbps | Wire |
|  XBee Pro 60mW PCB Antenna - Series 1 | 1 Mile | 215mA @ 3.3v | 2.4GHz | 802.15.4 | 60mW | 250kbps | PCB |
|  XBee Pro 60mW U.FL Connection - Series 1 | 1 Mile | 215mA @ 3.3v | 2.4GHz | 802.15.4 | 60mW | 250kbps | Ext./Not Included |
|  | 1 Mile | 215mA @ 3.3v | 2.4GHz | 802.15.4 | 60mW | 250kbps | Wire |




| XBee Device | Range | Power Consumption | Frequency | Protocol | Tx Power | Data Rate | Antenna |
|---|----------|-------------------|-----------|-------------|----------|-----------|-------------------|
| XBee Pro 60mW Wire Antenna - Series 1 | | | | | | | |
|  XBee Pro 900 RPSMA | 6 Miles | 210mA @ 3.3v | 900MHz | Multi-Point | 50mW | 156kbps | Ext./Not Included |
|  XBee Pro 900 U.FL Connection | 6 Miles | 210mA @ 3.3v | 900MHz | Multi-Point | 50mW | 156kbps | Ext./Not Included |
|  XBee Pro 900 Wire Antenna | 6 Miles | 210mA @ 3.3v | 900MHz | Multi-Point | 50mW | 156kbps | Wire |
|  XBee Pro 900 XSC RPSMA | 15 Miles | 256mA @ 3.3v | 900MHz | Multi-Point | 100mW | 9.6kbps | Ext./Not Included |
|  | 15 Miles | 256mA @ 3.3v | 900MHz | Multi-Point | 100mW | 9.6kbps | Ext./Not Included |

| XBee Device | Range | Power Consumption | Frequency | Protocol | Tx Power | Data Rate | Antenna |
|--|----------|-------------------|-----------|-------------|----------|---------------------|-------------------|
| <u>XBee Pro 900 XSC U.FL</u> (retired) | | | | | | | |
|  <u>XBee Pro 900 XSC Wire</u> (retired) | 15 Miles | 256mA @ 3.3v | 900MHz | Multi-Point | 100mW | 9.6kbps | Wire |
|  <u>XBee Pro 900 XSC S3B Wire</u> | 28 Miles | 215mA @ 3.3v | 900MHz | Multi-Point | 250mW | 10 or 20 Kbps | Wire |
|  <u>XTend 900 1W RPSMA</u> | 40 Miles | 730mA @ 5v | 900MHz | Multi-Point | 1W | 9,600 or 115,200bps | Ext./Not Included |

Xbee Compatible Devices

Here's a table of stuff we sell that you can plug an XBee into!

| XBee Board | Function | Compatible With |
|---|---|--|
|  Breakout Board for XBee Module | Breaks out XBee header to breadboard-friendly 0.1"-spaced headers for easy prototyping. | Series 1/2 and Pro Series XBee Modules |
|  XBee Shield | Makes it easy to add XBee wireless communication to Arduino based projects! | Series 1/2 and Pro Series XBee Modules |
|  XBee Explorer Regulated | Handles 3.3V regulation, signal conditioning, and basic activity indicators (Power, RSSI and DIN/DOUT activity LEDs) making it easier to work with XBee radios. | Series 1/2 and Pro Series XBee Modules |

| XBee Board | Function | Compatible With |
|---|---|--|
|  XBee Explorer USB | Simple to use, USB to serial base unit for the XBee line. | Series 1/2 and Pro Series XBee Modules |
|  XBee Explorer Dongle | Like the USB Explorer, except you can plug it directly into your computer's USB port! | Series 1/2 and Pro Series XBee Modules |
|  LilyPad XBee | Sewable breakout for XBee Modules, so you can add XBee wireless functionality to your e-textiles project! | Series 1/2 and Pro Series XBee Modules |

Glossary of Terms:

Range: The range of an XBee device is affected by several factors including the transmit power of the device, the type of antenna connected and the surrounding obstacles or conditions. The range listed here reflects the maximum range of the device in ideal conditions and in open air, line-of-sight. Expect this distance to be smaller if you're trying to communicate indoors or through walls, trees or other barriers.

Power Consumption: This represents the amount of power that the device will typically consume during transmission, your system should be capable of sourcing at least this much current and then some to avoid erratic behavior or brown-out conditions.

Frequency: The operating frequency of the device will affect its range and penetrative force as well as its tendency for interference. Lower frequencies require larger

antennas to be effective but they also have greater penetrating power when it comes to transmitting through walls and barriers.

Protocol: This is the language that the device "speaks" when transmitting and receiving data. XBee modules are designed to communicate using a specific protocol, although certain devices can be made to use a different protocol by changing the firmware. Series 1 modules are designed to use 802.15.4 which is a point-to-point communication protocol. This is great for networks which contain only a transmitter and a receiver or multiple receivers. The Series 2 modules are set up for the ZigBee Mesh protocol which is a mesh network standard, this is great if you have a lot of 'nodes' that need to all talk to each other. Series 1 modules aren't capable of mesh networking and can't communicate with other devices running the ZigBee Mesh firmware, however, Series 2 modules are backwards compatible and can be firmware configured for point-to-point networking. 900MHz modules use a different protocol altogether for multi-point networks.

Tx Power: The Tx (Transmit) Power is the amount of power that the device actually broadcasts. While this is closely tied to range, it isn't the only factor. This number is important to keep in mind when selecting an antenna for a device to ensure that you comply with your local radio communication laws. The power listed on this chart is the maximum output power and it can be adjusted in firmware in case you need to dial it back.

Data Rate: The speed at which the device can communicate over the air will effect not only how much data you can push over the network at once, but also how reliably the device will communicate at long distance. Slower transmission rates can be beneficial if your network spans a large distance, a fact exploited by the XSC line of devices. The speeds listed here are maximums and can be adjusted in firmware for several of these devices.

Antenna: The type of antenna, if any, that the module comes equipped with. There are several things to keep in mind when it comes to selecting the proper antenna or antenna connector for your project. A chip antenna is small and easy to enclose, but it doesn't give the best gain. A wire antenna is simple and effective but it's also not as small as the chip antenna and can be more difficult to incorporate into your design; For that reason, most new XBee modules feature a trace antenna instead. Trace or PCB antennas are made from conductive traces on the module itself and have performance comparable to wire antennas while taking up less space. If you're building your wireless device into an enclosure it can be beneficial to attach an external antenna, this can be achieved either by U.FL or RPSMA connection. U.FL is the type of connector often found on the wireless adapters in laptop computers and other small devices, routers and larger devices often have RPSMA connectors.

Remember that whenever you add an external antenna to a device you change the gain of the transmitter, so be sure you stay in compliance with your local radio communication regulations.

More XBee Modules and their comparisons

As of March 2016, the XBee radio family consists of

- XBee 802.15.4 — The initial [point-to-point topology](#) or [star topology](#) module running the [IEEE 802.15.4](#) protocol
- XBee-PRO 802.15.4 — A higher power, longer range version of the XBee 802.15.4
- XBee DigiMesh 2.4 — A 2.4 GHz XBee module that uses DigiMesh, a sleeping mesh networking protocol developed by Digi International
- XBee-PRO DigiMesh 2.4 — A higher power, longer range version of the XBee DigiMesh 2.4
- XBee ZB — An XBee module that incorporates the [ZigBee](#) PRO mesh networking protocol
- XBee-PRO ZB — A higher power, longer range version of the XBee ZB
- XBee ZB SMT — A surface mount XBee running the ZigBee protocol
- XBee-PRO ZB SMT — A higher power, longer range version of the XBee ZB SMT
- XBee SE — An XBee ZB module that incorporates the security cluster for the [ZigBee Smart Energy](#) public profile
- XBee-PRO SE — A higher power, longer range version of the XBee SE
- XBee-PRO 900HP — A 900 MHz XBee-PRO module with up to 28 mile range with high-gain antenna that supports DigiMesh networking protocol
- XBee-PRO 900 (Legacy) — A [900 MHz](#) proprietary point-to-point and star topology module, not recommended for new design
- XBee-PRO XSC (S3B) — A 900 MHz module compatible over the air with the Digi 9XStream radios
- XBee-PRO DigiMesh 900 (Legacy) — A 900 MHz module that uses DigiMesh, not recommended for new design (see XBee-PRO 900HP for new designs)
- XBee-PRO 868 — An [868 MHz](#) 500 mW long-range module that supports proprietary point-to-point and star, for use in Europe
- XBee 865/868LP — An 868 MHz XBee module that uses DigiMesh, available in Surface Mount form-factor (also configurable to [865 MHz](#) for use in India)
- XBee ZigBee (S2C) — Incorporates an upgrade to the transceiver chip, replacing the [Silicon Labs](#) EM250 with the Silicon Labs EM357, effectively adding more RAM, more flash, faster clock speed and lowering the current draw.^[1]
- XBee-PRO ZigBee (S2C) — A higher power, longer range version of the XBee ZigBee (S2C)

Current products comparison

| Model | XBee ZigBee (S2C) | XBee-PRO 900HP | XBee 802.15.4 | XBee DigiMesh 2.4 | XBee 868LP | XBee-PRO XSC | XTend 900 MHz | XBee Wi-Fi |
|-----------------------------|--|------------------------|--|--|-------------------------|------------------------------|------------------------------|--------------------------------|
| Available form factors | Through-hole and surface mount | Through-hole | | | Surface mount | Through-hole | | Through-hole and surface mount |
| Frequency | 2.4 GHz | 900 MHz | 2.4 GHz | 2.4 GHz | 868 MHz | 900 MHz | | 2.4 GHz |
| Maximum line-of-sight range | XBee ZigBee: 1.2 km XBee-PRO ZigBee: 3.2 km | 14 km (at 10 kbit/s) | XBee 802.15.4: 90m XBee-PRO 802.15.4: 1.6 km | XBee DigiMesh 2.4: 90m XBee-PRO DigiMesh 2.4: 1.6 km | 4 km (W/ 2dBi antenna) | 45 km (W/ high gain antenna) | 64 km (W/ high gain antenna) | N/A |
| Data rate | RF: 250kbit/s Serial: 1Mbit/s | 10kbit/s or 200kbit/s | 250kbit/s | 250kbit/s | 10kbit/s or 80kbit/s | 10kbit/s or 20kbit/s | 10kbit/s or 125kbit/s | 72Mbit/s |
| Communication protocol | ZigBee | Proprietary | 802.15.4 | Proprietary | | | | 802.11b/g/n |
| Maximum transmit power | XBee ZigBee: 6.3 mW (8dBm) XBee-PRO ZigBee: 63 mW (18dBm) | 250 mW (24dBm) | 1 mW (0dBm) | XBee DigiMesh 2.4: 1 mW (0dBm) XBee-PRO DigiMesh 2.4: 63 mW (18dBm) | 25 mW (14dBm) | 250 mW (24dBm) | 1000 mW (30dBm) | 16dBm |
| Maximum receiver | XBee ZigBee: -102 dBm XBee-PRO ZigBee: -101 dBm | -110dBm (at 10 kbit/s) | XBee 802.15.4: -92 dBm XBee-PRO 802.15.4: -92 dBm | XBee DigiMesh 2.4: -92 dBm XBee-PRO DigiMesh 2.4: -100 dBm | -106 dBm (at 10 kbit/s) | -107 dBm (at 10 kbit/s) | -110dBm (at 9600 bit/s) | -93 dBm |

| | | | | | | | | |
|--|--|------------------------|---|---|---------------|----------------|--|--------------------|
| sensitivity | | | 4: -100 dBm | | | 19200 bit/s) | | |
| ADC inputs | 4 (10-bit) | | | 6 (10-bit) | 4 (10-bit) | | | 4 (12-bit) |
| Supply voltage | XBee ZigBee: 2.1 to 3.6 VDC XBee-PRO ZigBee: 2.7 to 3.6 VDC | 2.1 to 3.6VDC | 2.8 to 3.4VDC | 2.8 to 3.4VDC | 2.7 to 3.6VDC | 2.4 to 3.6VDC | 2.8 to 5.5VDC | 3.14 to 3.46VDC |
| Max. current consumption (transmitting) | XBee ZigBee: 59 mA XBee-PRO ZigBee: 120 mA | 229 mA | XBee 802.15.4: 45 mA XBee-PRO 802.15.4: 215 mA | XBee DigiMesh 2.4: 45 mA XBee-PRO DigiMesh 2.4: 340 mA | 62 mA | 215 mA | 710 mA, 900 mA or 55 mA (depends on model) | 309 mA |
| Max. current consumption (receiving) | XBee ZigBee: 42 mA XBee-PRO ZigBee: 45 mA | 44 mA | XBee 802.15.4: 50 mA XBee-PRO 802.15.4: 55 mA | XBee DigiMesh 2.4: 50 mA XBee-PRO DigiMesh 2.4: 55 mA | 41 mA | 26 mA | 35 mA (at 5V) or 40 mA (at 3.3V; depends on model) | 100 mA |
| Max. current consumption (sleep) | < 1 μ A | 2.5 μ A | < 10 μ A | < 50 μ A | 2.3 μ A | 2.5 μ A | < 147 μ A or 2.5 μ A (depends on model) | < 6 μ A |
| Certified regions | US, CA, EU, AU, BR, JP | US, CA, AU, BR, MX, SG | US, CA, EU, AU, BR, JP | US, CA, EU, AU, BR, JP | EU | US, CA, AU, BR | US, CA, AU | US, CA, EU, AU, BR |

| | | | | | | | | |
|-------------------------------|--|--------------------------------|--|--|--|------------------------------|---|--|
| I/O | 15 | | | 13 | | | | 10 |
| Operating temperature | - 40 °C to 85 °C | | - 40 °C to 85 °C | | -40 °C to 85 °C | | | - 30 °C to 85 °C |
| Antenna types options | Through-hole: PCB, RPSMA, U.FL, wire SMT: PCB, RF pad, U.FL | U.FL, RPSMA, wire | U.FL, RPSMA, chip , wire | | U.FL, RF pad, PCB (for 10 kbit/s only) | RPSMA, U.FL, wire | MMCX , RPSMA or U.FL, RF pad (depends on model) | Through-hole: PCB, RPSMA, U.FL, wire SMT: PCB, U.FL, RF pad |
| Channels | XBee ZigBee: 16 XBee-PRO ZigBee: 15 | FHSS | XBee 802.15.4: 16 XBee-PRO ZigBee: 12 | XBee DigiMesh 2.4: 16 XBee-PRO DigiMesh 2.4: 12 | 30 ^[12] | FHSS | FHSS (50 channels) | 13 |
| Serial data interfaces | UART , SPI | | UART | | UART, SPI | | UART | UART, SPI |
| Product page | XBee ZigBee S2C | XBee-PRO 900HP | XBee 802.15.4 | XBee DigiMesh 2.4 | XBee 868LP | XBee-PRO XSC | XBee XTend 900MHz | XBee Wi-Fi |

Important softwares and resources for XBee

[X-CTU software](#) - This is what you need to configure the XBee modules.

[XBee Series 1 product page](#) - The product page for the Series 1 module.

[XBee ZB product page](#) - The product page for the current Series2ish module.

[Government Regulations](#) - Wireless communication has different restrictions in different countries. The ones we sell are all fine for use in the US, but for more information check out Digi's information on what XBees are acceptable where.

[Building Wireless Sensor Networks](#) - great book on Series 2ish modules. This is an amazing book on XBees, it covers everything from configuring the modules to using the I/Os and sleep functions. It also has projects throughout the book to help you put to use what you've learned.

Alternatives to the XBee Wireless modules

Option-1:

WiFi

The home WiFi access points on the market usually incorporate a router plus DHCP server for the client side. They support a set of Ethernet and WiFi hosts on one side (the in-home network side), all of these in the same IP subnet, and they have an Ethernet port to the WAN.

Used in infrastructure mode, you might connect several of these home WiFi access points to a central layer 2 switch, or for that matter, even to a central router, configure each WiFi router's WAN port with its own IP address, and set up your own little routed network?

If the WAN side of each access point connects to a central layer 2 switch, then you will have to configure the WAN ports of each WiFi router with an IP address in the same IP subnet as the other WiFi routers' WAN ports. You would be creating a backbone network, in short, to interconnect all WiFi routers.

The WiFi routers are easy enough to configure. They come with default settings that facilitate connection to a home WAN modem/router, but you can easily reconfigure those default settings. The instructions explain how, but it's going to be menu-driven and straightforward.

And if you really mess up and can't get anything to work, there is a button that returns all settings to the default settings, so you can start over.

The end devices can be PCs/laptops, tablets, and smartphones, all of which now have WiFi built in (or you can use a USB/WiFi dongle if necessary). These end hosts can configure automatically, using the WiFi routers' DHCP servers.

For utilizing the wifi option we can use the esp8266 modules to communicate either between two arduinos or between an Arduino and the cloud server via the internet network etc, also the size of the esp8266 is smaller than the xbee and the antenna is embedded on the chip itself so it is a good alternative to the xbee modules.



The ESP8266 arduino compatible module is a **low-cost Wi-Fi chip** with **full TCP/IP capability**, and the amazing thing is that this little board has a **MCU (Micro Controller Unit) integrated** which gives the possibility to **control I/O digital pins** via simple and almost pseudo-code like programming language. This device is produced by Shanghai-based Chinese manufacturer, **Espressif Systems**.

This chip was first time seen in **August 2014**, in **ESP-01** version module, made by **AI-Thinker**, a third-party manufacturer. This little module allows the MCU to connect to WiFi network and create simple TCP/IP connections. His **His very low price (1.7\$ – 3.5\$)** and the incredible small size attracted many **geeks** and **hackers** to explore it and use it in a **large variety of projects**. Being a true success, **Espressif** produces now many versions having different dimensions and technical specifications. One of the successors is **ESP32**. You can find over the internet hundreds of projects and various implementations like home automation, data logging solutions, **robotics**, controlling things over the internet, even drones or copters.

ESP8266-01 Technical specifications

- 32-bit RISC CPU: Tensilica Xtensa LX106 running at 80 MHz **
- 64 KiB of instruction RAM, 96 KiB of data RAM
- External QSPI flash – 512 KiB to 4 MiB* (up to 16 MiB is supported)
- IEEE 802.11 b/g/n Wi-Fi
- Integrated TR switch, balun, LNA, power amplifier and matching network
- WEP or WPA/WPA2 authentication, or open networks
- 16 GPIO pins **
- SPI, I²C,
- I²S interfaces with DMA (sharing pins with GPIO)
- UART on dedicated pins, plus a transmit-only UART can be enabled on GPIO2
- 1 10-bit ADC

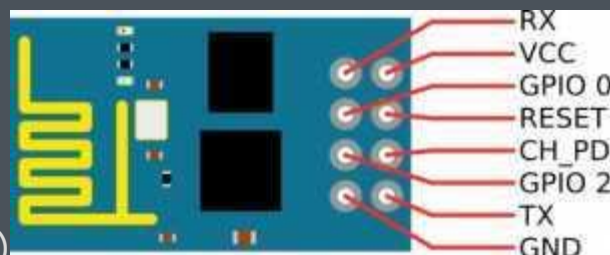
*** CPU and flash clock speeds can be raised via overclocking on some devices and the 16 I/O are not available in all versions.*

ESP8266 Arduino module comes with **PCB trace antenna** which seems to have a **very good coverage** (*I saw a demonstration with more than **1km range!!!***). Other version can have **on-board ceramic antenna or an external connector** which allows you to attach external WiFi antennas modules. ESP-01 has only 6 active pins, although the MCU can support up to 16 I/O. Board dimensions are **14.3 x 24.8 mm**.



Over the internet i found that ESP8266 arduino module, version 01, is sold in two or more versions, which at first glance seem quite the same. After buying both of them i saw that there is a difference in size of the flash memory. You may encounter issues while flashing if you don't make the proper settings according to board specifications.

Although the board default has 2 available GPIOs, you can do some workarounds and use other MCU available pins if you have the proper soldering tools. I managed to use GPIO 16 in order to wake up the device after DEEP SLEEP mode (*explained later in SLEEP MODES*).



Module pin description (pinout)

Pins are arranged in two rows, having 4 on each row. Some models have pin description on the PCB, which make it simple. On the top row you can find following pins from the left to the right:

1. **GND** (Ground from power supply)
2. **GPIO2** (Digital I/O programmable)
3. **GPIO0** (Digital I/O programmable, also used for BOOT modes)
4. **RX** – UART Receiving channel

On the bottom (second row) you can find:

1. **TX** – UART Transmitting channel

2. **CH_PD** (enable/power down, must be pulled to 3.3v directly or via resistor)
3. **REST** – reset, must be pulled to 3.3v)
4. **VCC** -3.3v power supply

Power supply and current consumption

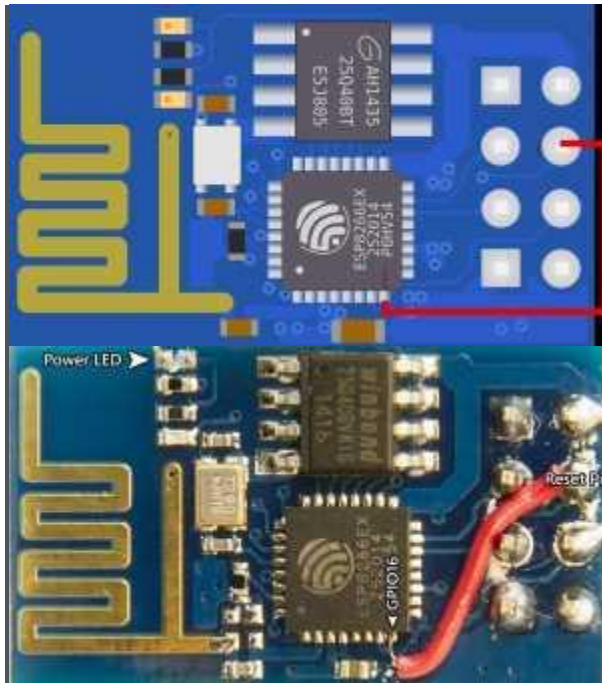
All esp8266 arduino compatible modules must be powered with **DC current** from any kind of source that can deliver **stable 3.3V** and **at least 250mA**. Also **logic signal** is rated at **3.3v** and the RX channel should be protected by a 3.3v divisor step-down. You should be careful when using this module with Arduino or other boards which supplies 5v, because this module usually **do not come with overpower protection** and can be easily destroyed.

Here is the declared power consumption from Espressif:

| Parameters | Typical | Unit |
|---|---------|------|
| Tx802.11b, CCK 11Mbps, P OUT=+17dBm | 170 | mA |
| Tx 802.11g, OFDM 54Mbps, P OUT =+15dBm | 140 | mA |
| Tx 802.11n, MCS7, P OUT =+13dBm | 120 | mA |
| Rx 802.11b, 1024 bytes packet length , -80dBm | 50 | mA |
| Rx 802.11g, 1024 bytes packet length, -70dBm | 56 | mA |
| Rx 802.11n, 1024 bytes packet length, -65dBm | 56 | mA |
| Modem-Sleep | 15 | mA |
| Light-Sleep | 0.9 | mA |
| Deep-Sleep | 10 | uA |
| Power Off | 0.5 | uA |

If you are going to use **ESP-01** in a project that is **powered by batteries or by solar power** it is mandatory to know everything about **ESP8266 arduino Sleep modes**. Current version offers 3 different sleep modes which can be triggered programmatically. ESP8266WiFi library offers specific functions to call sleep modes which can take settings parameters that change the callback jobs after wake-up like waking up with RF module powered off or on. The most important mode is **DEEP_SLEEP** because of the very low power consumption rates during sleep. Deep sleep mode is very common in projects that do data-logging at specific intervals and idle between measurements. In order to take advantage of this mode when using esp8266 arduino compatible module, ESP-01 standard, you need to make a little workaround and connect REST pin with the GPIO16 pin (which is not available in default 6 six pins).

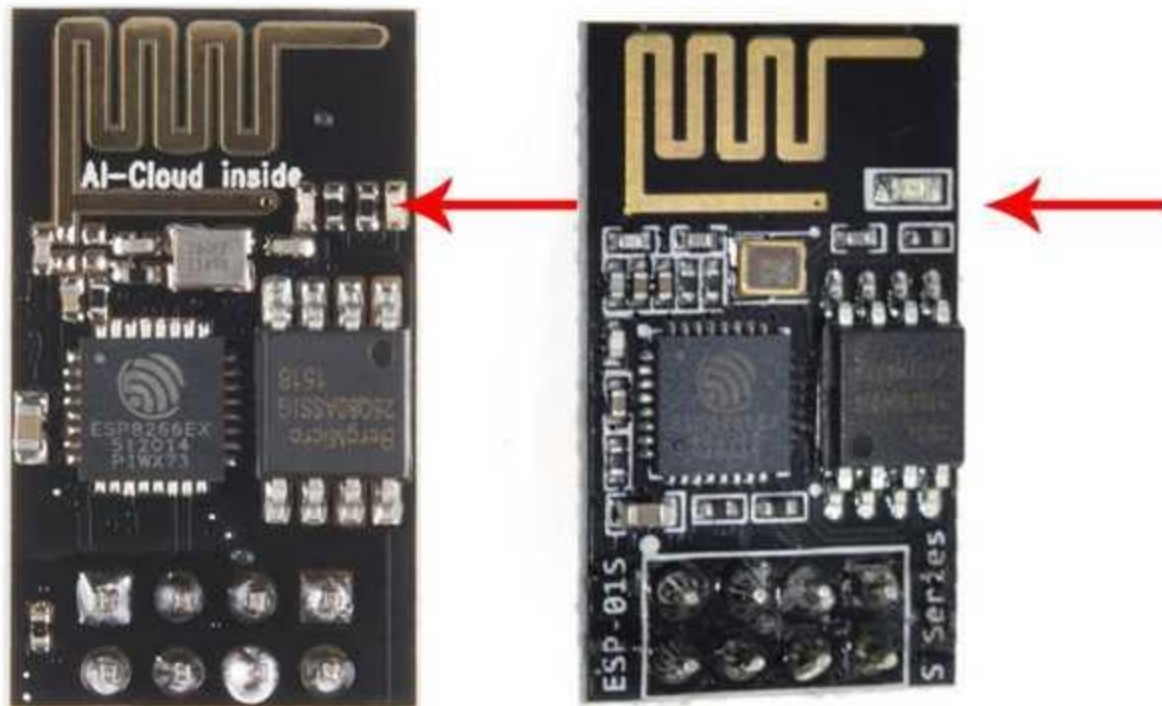
Here is an example how to do it



After doing this connection you can use the following command to trigger the deep sleep mode:

```
ESP.deepSleep(sleepTimeInSeconds * 1000000);
```

Another thing that you need to know about the esp8266 arduino 01 version is that usually comes with **two LEDs**, a **red** one for power, and a **blue** one for data transmitting. The **red LED is always on** when the **module is powered** on and the blue led blinks during serial activities. For newer versions the producers eliminated the RED pin because of the continuous power consumption, so if you are going to buy one, **try to find a version which has only the serial blue LED** especially if you are going to use a **battery power supply** in your project. The differences can be seen in the following pictures:



While in the left picture you can see **two smd LEDs** near the antenna, one for power indicator and one for data indicator, the right module has a **single LED** just to indicate the data transmission. PCB design can differ from one to another because producers tend to reduce the costs by cutting down the materials.

Talking with **ESP-01** (AT / LUA / Arduino)

ESP8266-01 gives you **many methods to communicate** with it **through the RX/TX pins** or **over the air (OTA)**. The differences are not only in hardware but can be also in what kind of **firmware is flashed out of the box**. No matter what firmware comes default installed, **you** should be able to **flash your preferred firmware** by following the firmware flashing instruction from the datasheet. This module can be programmed using **LUA code, Arduino code or directly through AT commands** and this gives us more freedom when embedding this device in our projects. Also there are few **python firmware modes** but i haven't had the chance to test them. I personally choose to work with **Arduino** because of the past experience and **tones of libraries available**.

As it comes, **out of the box**, this module **is ready to talk** via **AT commands** without any other extra settings or configurations. There are many software applications which you can use to communicate via **AT** and have tones of **ready made tools** and **functions** which will make everything easier. I used **ESPlorer** and i totally recommend it, you can find it [here](#). After booting, **to be able to use AT** commands, **module** should **display "ready"** on the serial monitor.

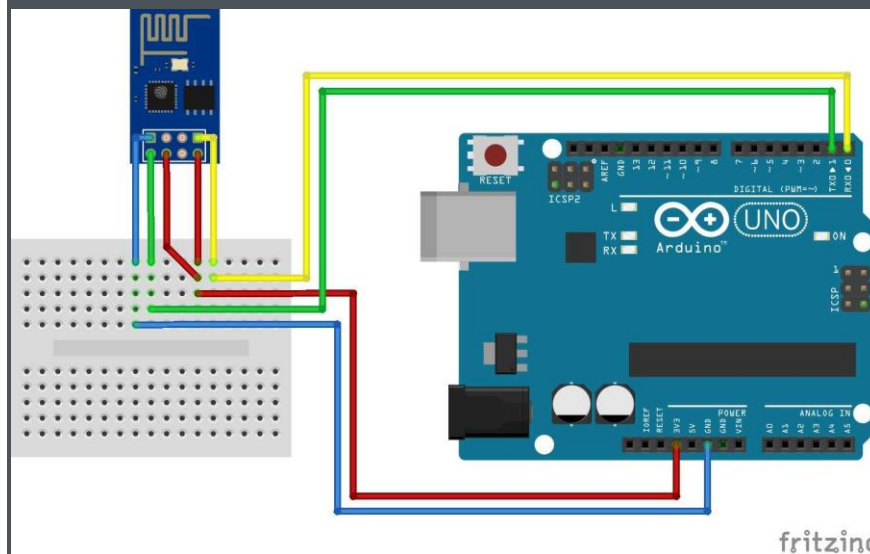
Few basic AT commands examples:

- AT – response OK

- AT+CWLAP – list nearby available WiFi networks
- AT+GMR – check the firmware version
- AT+CWJAP="<access_point_name>","<password>" – join WiFi network using credentials
- AT+CIFSR – get current allocated IP address

See here a complete list with [AT instruction set](#).

In order to be able to talk with the ESP8266 arduino compatible module, you need to choose a way to **connect it with your computer**. You can communicate with the module via **standard Serial communication RS232** by using an **Arduino** board as a **proxy/bridge**, by default Arduino having a USB to Serial converter integrated. If you are a beginner in development boards I totally recommend you [one of the best Arduino books by Jeremy Blum](#), the best combination of a formally trained electronics engineer and a [Maker/Hacker](#). **Arduino Uno** differs from all preceding boards in that it does not use the **FTDI USB-to-serial** driver chip. Instead, it features the *Atmega16U2* (Atmega8U2 up to version R2) programmed as a USB-to-serial converter. In order to use Arduino as a bridge, first you need to load an empty program on it. After doing that, you need to make the following connections in order to work:



| UNO | ESP-01 |
|------|--------------|
| RX | RX |
| TX | TX |
| 3.3v | VCC |
| GND | GND |
| RST | 3.3v / float |

| | |
|-------|--------|
| UNO | ESP-01 |
| CH_PD | 3.3v |

After that you should be able to see data and send AT commands in Serial Monitor by selecting Arduino's COM port, **setting** a proper **baudrate**, default should be **115200**, and make the additional settings to read "**Both NL & CR**".

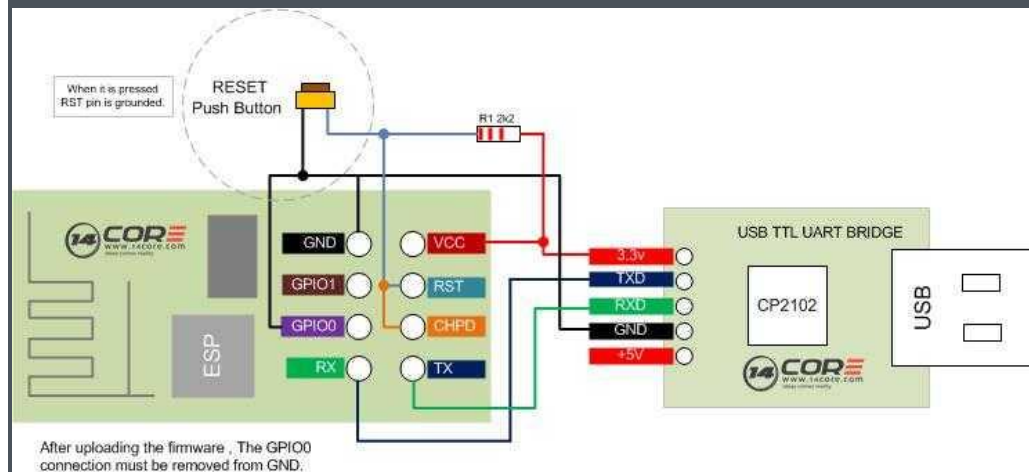
Firmware Over The Air (FOTA) solution in every embedded **DIY** or commercial project is a highly desirable if not a required feature today, when every project core needs to be scalable. So the possibility to upload your code from a remote computer via Wi-Fi connection rather than Serial is a **high advantage in every project**. First you need FOTA needs prerequisites. First firmware upload needs to be done via Serial, and if the OTA routines are correctly implemented in the program next uploads can be done over the air. Because the module needs to be exposed wirelessly, the chance of being hacked and loaded with maleficent code exists. You can improve your security by setting custom port and password. Check functionalities from the [ArduinoOTA library](#) that may help you to boost security. Because of the complexity of this procedure we will cover the full story in a future article, but for now be aware that this option exists and it works pretty good.

Another way to connect the esp8266 arduino module to computer is to use a **TTL or FTDI USB-to-serial** dedicated module. There are plenty of them on the market and there are quite cheap, but make no mistake, here quality does matter. You may encounter problems when working with it if ending up with a cheap one because of the differences in connections and also driver compatibility. Most used TTL / FTDI converters chips are **CH340G**, **CP2102** and **FT232RL**. I personally used the first two ones and I have no problem when loading programs. Following connections need to be done:

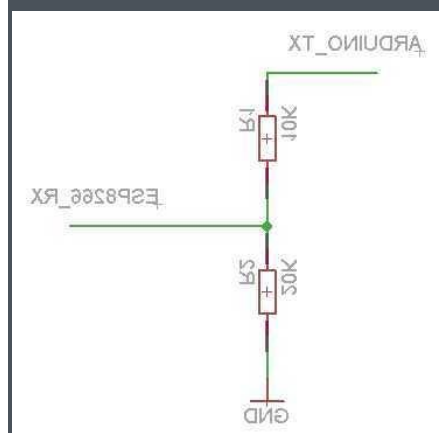
| ESP-01 | TTL / FTDI |
|--------|--------------|
| RX | TX |
| TX | RX |
| VCC | 3.3v |
| GND | GND |
| RST | 3.3v / float |
| CH_PD | 3.3v |

I highly recommend you not to use the TTL 3.3v power supply because most of them are not able to provide enough power to handle the esp8266 arduino compatible device. The embedded voltage regulator used on these modules are not the happiest choice and you may get in trouble if it cannot support ESP peeks. If you choose to use an **external power supply** don't forget to setup a **common ground** in order to have a working circuit. You can find TTL modules that have TX rated at 3.3v, if not, you should step-down the TX channel to

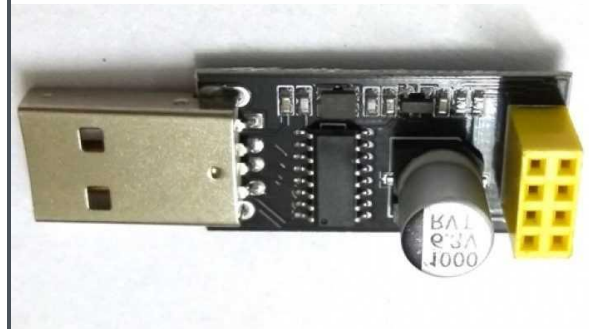
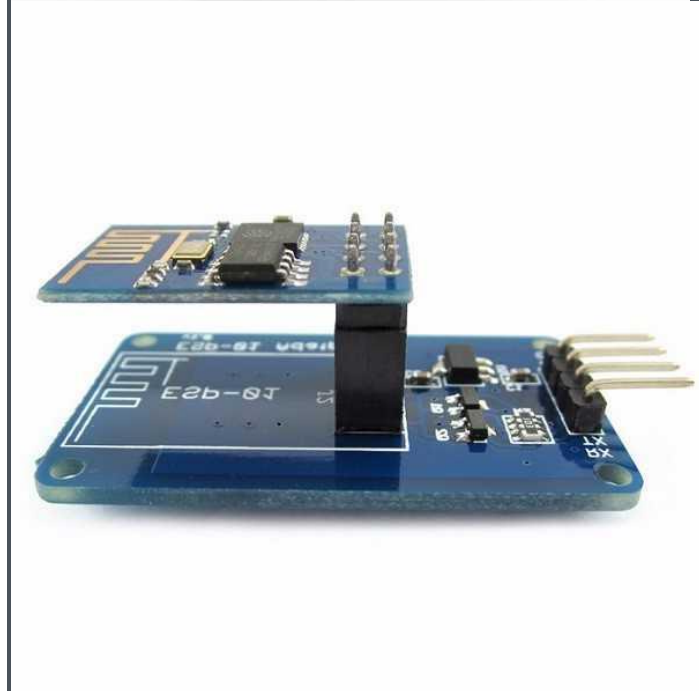
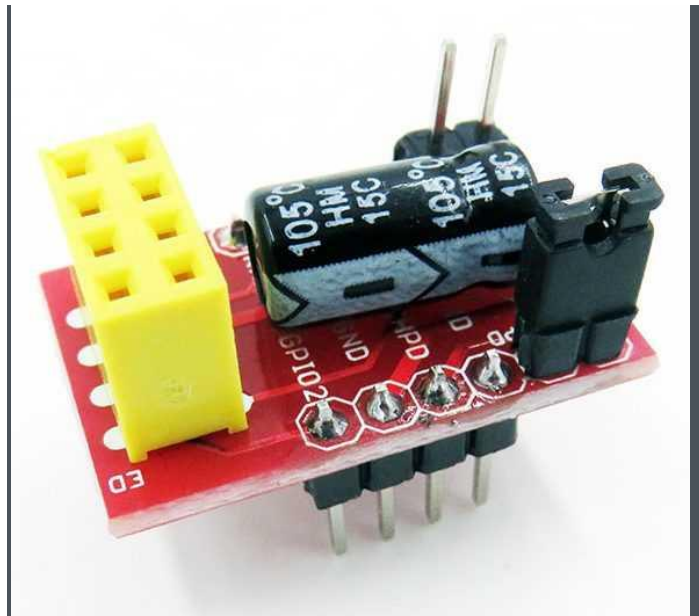
protect your ESP-01 module. You can see bellow a wiring scheme between ESP-01 and CP2102 which includes a reset button connected to ground, and also GPIO0 for boot switch.



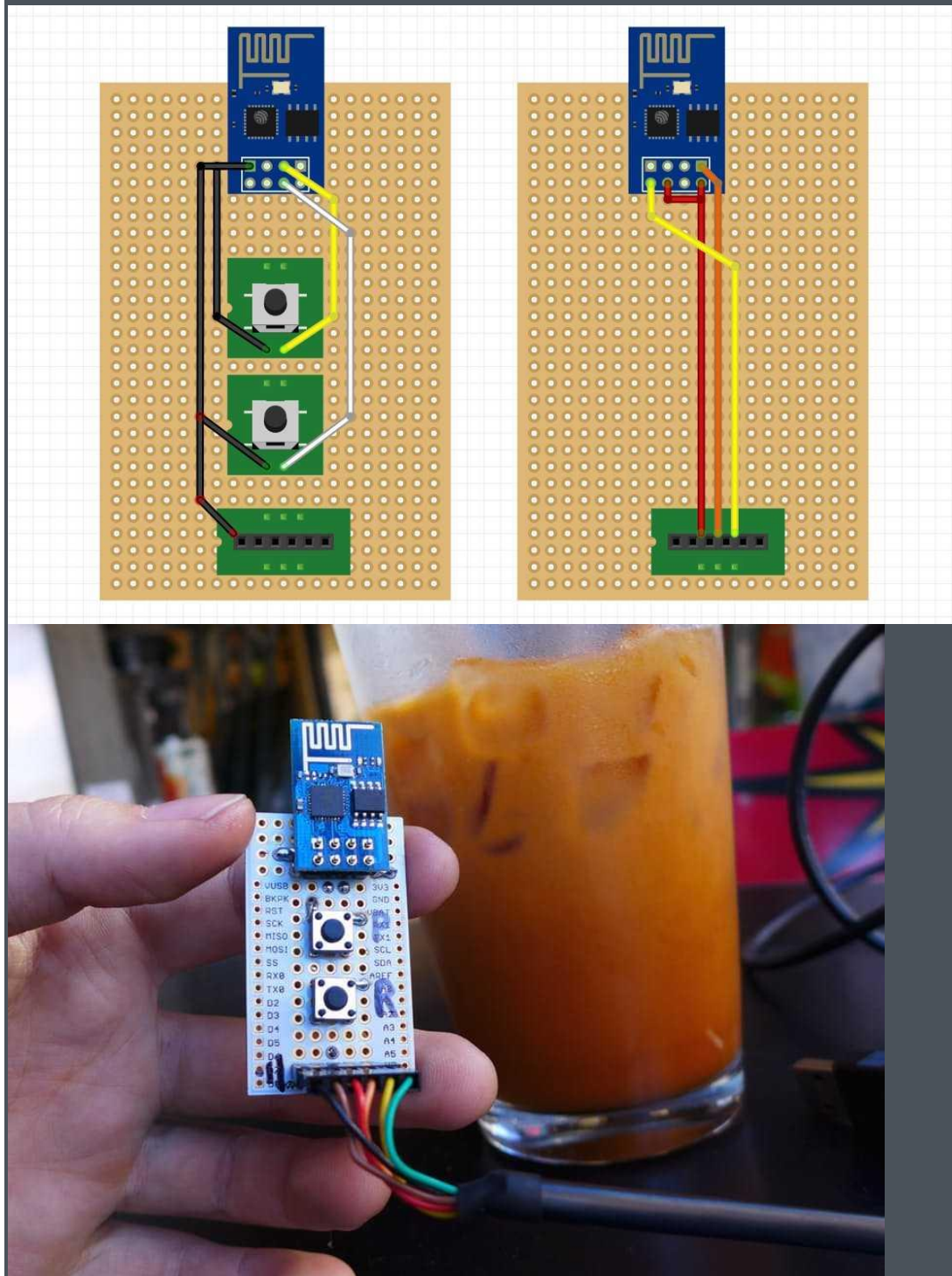
Here is a simple 3.3v divisor sketch using resistors:

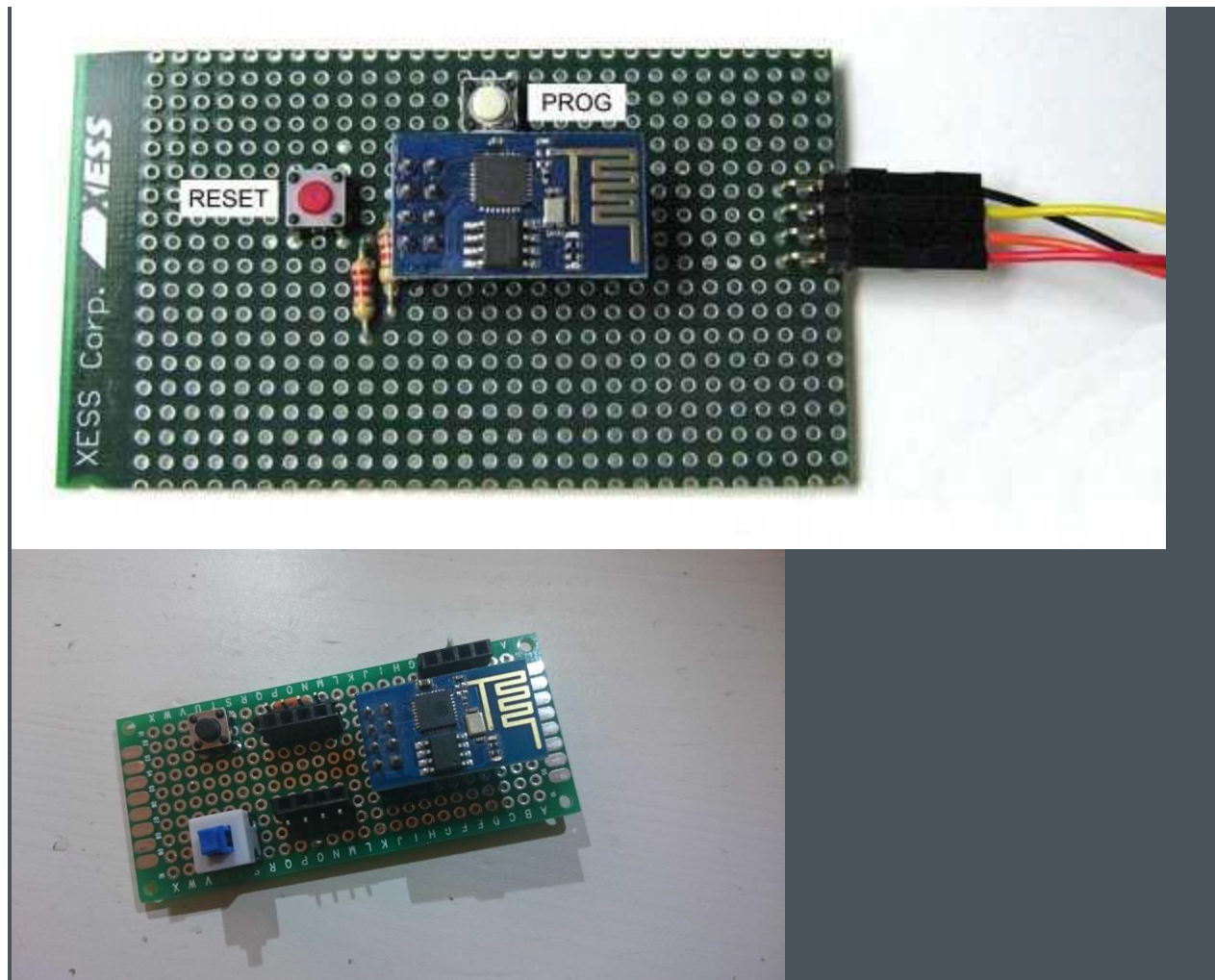


The most comfortable way to communicate with the ESP-01 is by using a **dedicated ESP01 programmer module**, which can be bought also from the same producers and the costs are very low. This kind of devices can have buttons integrated for RESET and BOOT switching and also come with a dedicated slot. All you need to do is to plug your esp and your done.



You can also build your own programmer if you have a bit of skills, you don't need to be an expert. Here are few homemade tryouts:





In order to **setup your Arduino IDE** to work with your esp8266 arduino compatible module you need to make the following steps:

1. Connect your ESP8266-01 Module to PC
2. Open your [Arduino IDE](#)
3. Go to File -> Preferences
4. Add [this link](#) to Additional Board Manager
5. Go to Tools -> Board Manager
6. Find ESP8266 board set and activate it
7. Select Generic ESP8266 board from Tools->Boards
8. Choose your programmer COM port
9. You are ready to go!



Now, to be able to download the program to your ESP-01 module, you first need to put your device in the **proper BOOT mode** (*Download code from UART*). ESP8266-01 have the following **boot modes**:

| MTDO / GPIO15 | GPIO0 | GPIO2 | Mode | Description |
|---------------|-------|-------|-------|-------------------------|
| L | L | H | UART | Download code from UART |
| L | H | H | Flash | Boot from SPI Flash |
| H | X | X | SDIO | Boot from SD-card |

Note that L = LOW (putting to Ground / -3.3v) and H = HIGH (putting to 3.3v)

After resetting the module in *Download code from UART* you should see a message containing "**boot mode:[1,6]**" in the serial monitor, if you are on the correct baudrate. A wrong baudrate setting will display garbage text / characters or nothing at all. After that you should be able to upload your sketch to ESP8266. When upload is done, module should reset itself. Don't forget to **pull HIGH the GPIO** or the module will get in *Download mode again* and you will not be able to see it working. The module can be rebooted at anytime by pulling REST pin to LOW. After each reset it will follow the boot sequence and program loading.

Once the **ESP8266** board is installed and activated in [Arduino IDE](#), you will be able to include **all ESP WiFi libraries and examples** that comes with the package. The most used library is **ESP8266WiFi** which offers many implementation examples like **WiFiClient**, **WiFiServer**, **WiFiAccessPoint** etc. You can find allot of projects examples over the internet, I for example, found great ideas on [arduino.cc projecthub](#). Here is a simple **Arduino blink example** which you can use to test the esp module with the built in LED:

```
/*
ESP8266 Arduino Blink by Simon Peter
Blink the blue LED on the ESP-01 module
This example code is in the public domain

The blue LED on the ESP-01 module is connected to GPIO1
```

```
(which is also the TXD pin; so we cannot use Serial.print() at the same
time)

Note that this sketch uses LED_BUILTIN to find the pin with the internal LED
*/

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);    // Initialize the LED_BUILTIN pin as an
  output
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, LOW);  // Turn the LED on (Note that LOW is the
  voltage level
                                   // but actually the LED is on; this is
  because
                                   // it is active low on the ESP-01)
  delay(1000);                     // Wait for a second
  digitalWrite(LED_BUILTIN, HIGH); // Turn the LED off by making the voltage
  HIGH
  delay(2000);                     // Wait for two seconds (to demonstrate
  the active low LED)
}
```

Of course, after that you can try a more complex example by loading a **ESP8266 Arduino WiFi Client example** program that sends data via WiFi to the data.sparkfun.com iot platform:

```
/*
 * This sketch sends data via HTTP GET requests to data.sparkfun.com
  service.
 *
 * You need to get streamId and privateKey at data.sparkfun.com and paste
  them
 * below. Or just customize this script to talk to other HTTP servers.
 * ESP8266 Arduino example
 */

#include <ESP8266WiFi.h>

const char* ssid      = "your-ssid";
const char* password  = "your-password";

const char* host = "data.sparkfun.com";
const char* streamId = ".....";
const char* privateKey = ".....";

void setup() {
  Serial.begin(115200);
  delay(10);

  // We start by connecting to a WiFi network

  Serial.println();
```

```
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

int value = 0;

void loop() {
    delay(5000);
    ++value;

    Serial.print("connecting to ");
    Serial.println(host);

    // Use WiFiClient class to create TCP connections
    WiFiClient client;
    const int httpPort = 80;
    if (!client.connect(host, httpPort)) {
        Serial.println("connection failed");
        return;
    }

    // We now create a URI for the request
    String url = "/input/";
    url += streamId;
    url += "?private_key=";
    url += privateKey;
    url += "&value=";
    url += value;

    Serial.print("Requesting URL: ");
    Serial.println(url);

    // This will send the request to the server
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
        "Host: " + host + "\r\n" +
        "Connection: close\r\n\r\n");
    unsigned long timeout = millis();
    while (client.available() == 0) {
        if (millis() - timeout > 5000) {
            Serial.println(">>> Client Timeout !");
            client.stop();
            return;
        }
    }
}
```

```
// Read all the lines of the reply from server and print them to Serial
while(client.available()){
  String line = client.readStringUntil('\r');
  Serial.print(line);
}

Serial.println();
Serial.println("closing connection");
}
```

Or if you need to make a server in your network, you can try **ESP8266 Arduino Wifi Server** example program:

```
/*
 * This sketch demonstrates how to set up a simple HTTP-like server.
 * The server will set a GPIO pin depending on the request
 *   http://server_ip/gpio/0 will set the GPIO2 low,
 *   http://server_ip/gpio/1 will set the GPIO2 high
 * server_ip is the IP address of the ESP8266 Arduino module, will be
 * printed to Serial when the module is connected.
 */

#include <ESP8266WiFi.h>

const char* ssid = "your-ssid";
const char* password = "your-password";

// Create an instance of the server
// specify the port to listen on as an argument
WiFiServer server(80);

void setup() {
  Serial.begin(115200);
  delay(10);

  // prepare GPIO2
  pinMode(2, OUTPUT);
  digitalWrite(2, 0);

  // Connect to WiFi network
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
}
```

```
// Start the server
server.begin();
Serial.println("Server started");

// Print the IP address
Serial.println(WiFi.localIP());
}

void loop() {
  // Check if a client has connected
  WiFiClient client = server.available();
  if (!client) {
    return;
  }

  // Wait until the client sends some data
  Serial.println("new client");
  while(!client.available()){
    delay(1);
  }

  // Read the first line of the request
  String req = client.readStringUntil('\r');
  Serial.println(req);
  client.flush();

  // Match the request
  int val;
  if (req.indexOf("/gpio/0") != -1)
    val = 0;
  else if (req.indexOf("/gpio/1") != -1)
    val = 1;
  else {
    Serial.println("invalid request");
    client.stop();
    return;
  }

  // Set GPIO2 according to the request
  digitalWrite(2, val);

  client.flush();

  // Prepare the response
  String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE
HTML>\r\n<html>\r\nGPIO is now ";
  s += (val)? "high": "low";
  s += "</html>\n";

  // Send the response to the client
  client.print(s);
  delay(1);
  Serial.println("Client disconnected");

  // The client will actually be disconnected
  // when the function returns and 'client' object is destroyed
}
```

```
}
```

And in the last example an **ESP8266 Arduino WiFi Access Point** which hosts a **web server** is created:

```
/* Create a WiFi access point and provide a web server on it.
ESP8266 Arduino example
*/

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

/* Set these to your desired credentials. */
const char *ssid = "ESPap";
const char *password = "thereisnospoon";

ESP8266WebServer server(80);

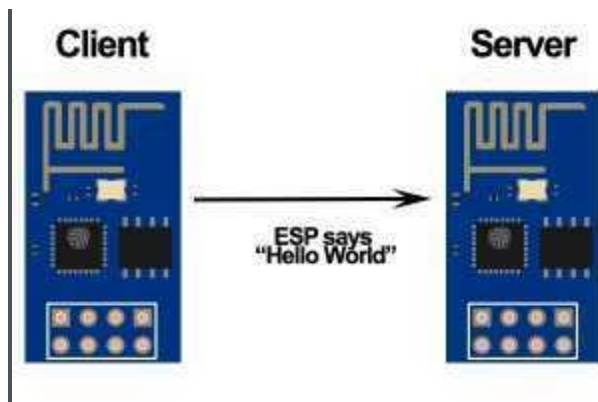
/* Just a little test message.  Go to http://192.168.4.1 in a web browser
 * connected to this access point to see it.
 */
void handleRoot() {
    server.send(200, "text/html", "<h1>You are connected</h1>");
}

void setup() {
    delay(1000);
    Serial.begin(115200);
    Serial.println();
    Serial.print("Configuring access point...");
    /* You can remove the password parameter if you want the AP to be
open. */
    WiFi.softAP(ssid, password);

    IPAddress myIP = WiFi.softAPIP();
    Serial.print("AP IP address: ");
    Serial.println(myIP);
    server.on("/", handleRoot);
    server.begin();
    Serial.println("HTTP server started");
}

void loop() {
    server.handleClient();
}
```

Linking two esp8266 to interface two arduinos with each other:



The working principle is quite simple. *It works almost as with humans.* In order to connect two ESP8266, at least one of them must be configured as **Access Point** to give the other one possibility to discover it. Once you've connected them together you then need a way to make them talk each other. As humans do, while one is talking, the other one needs to listen. The advantage of ESP8266 is that in the same time, can be both Client and Access Point / Server.

"The same time" is actually more **theoretically** because this module has **single threaded MCU**, so it can only do a certain thing at 1 point, but the fact that when you receive a request while being in AP / Server mode, you can then immediately connect to next AP and make a request as a client is amazing.

Two ESP8266 communication Arduino code examples

Below you can see the code I uploaded in one of my ESP8266 client. It reads temperature and humidity from a DHT22 and sends it to the AP ESP, and then sleeps for 5 minutes. I also used a static network configuration in order to win some time by not waiting for DHCP. For request handling I've used **ESP8266HTTPClient** library which I found being much simpler and logic and I recommend it.

```
/*
Geekstips.com
IoT project - Communication between two ESP8266 - Talk with Each Other
ESP8266 Arduino code example
*/
#include <DHT.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#define DHTPIN 2

// AP Wi-Fi credentials
const char* ssid = "*****";
const char* password = "*****";

// Local ESP web-server address
String serverHost = "http://192.168.4.1/feed";
String data;
// DEEP_SLEEP Timeout interval
int sleepInterval = 5;
// DEEP_SLEEP Timeout interval when connecting to AP fails
int failConnectRetryInterval = 2;
int counter = 0;
```

```
float h;
float t;
// Static network configuration
IPAddress ip(192, 168, 4, 4);
IPAddress gateway(192, 168, 4, 1);
IPAddress subnet(255, 255, 255, 0);

DHT dht(DHTPIN, DHT22);
WiFiClient client;

void setup() {
  ESP.eraseConfig();
  WiFi.persistent(false);
  Serial.begin(74880);
  Serial.println();
  Serial.println("*****");
  Serial.println("*****");
  Serial.println("***** BEGIN *****");
  Serial.println("- start DHT sensor");
  dht.begin();
  delay(500);
  Serial.println("- set ESP STA mode");
  WiFi.mode(WIFI_STA);
  Serial.println("- connecting to wifi");
  WiFi.config(ip, gateway, subnet);
  WiFi.begin(ssid, password);
  Serial.println("");
  while (WiFi.status() != WL_CONNECTED) {
    if(counter > 20){
      Serial.println("- can't connect, going to sleep");
      hibernate(failConnectRetryInterval);
    }
    delay(500);
    Serial.print(".");
    counter++;
  }

  Serial.println("- wifi connected");
  Serial.println("- read DHT sensor");
  readDHTSensor();
  Serial.println("- build DATA stream string");
  buildDataStream();
  Serial.println("- send GET request");
  sendHttpRequest();
  Serial.println();
  Serial.println("- got back to sleep");
  Serial.println("*****");
  Serial.println("*****");
  hibernate(sleepInterval);
}

void sendHttpRequest() {
  HTTPClient http;
  http.begin(serverHost);
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");
  http.POST(data);
}
```



```
    http.writeToStream(&Serial);
    http.end();
}

void readDHTSensor() {
    delay(200);
    h = dht.readHumidity();
    t = dht.readTemperature();
    if (isnan(h) || isnan(t)) {
        t = 0.00;
        h = 0.00;
    }
    Serial.println("- temperature read : "+String(t));
    Serial.println("- humidity read : "+String(h));
}

void buildDataStream() {
    data = "temp=";
    data += String(t);
    data += "&hum=";
    data += String(h);
    Serial.println("- data stream: "+data);
}

void hibernate(int pInterval) {
    WiFi.disconnect();
    ESP.deepSleep(10 * 600000 * pInterval, WAKE_RF_CAL);
    delay(100);
}

void loop() {}
```

For the Access Point I used the following code:

```
/*
Geekstips.com
IoT project - Communication between two ESP8266 - Talk with Each Other
ESP8266 Arduino code example
*/
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
// IoT platform Credentials
String apiKey = "*****";
const char* logServer = "api.thingspeak.com";

// Internet router credentials
const char* ssid = "*****";
const char* password = "*****";

ESP8266WebServer server(80);

void setup() {
    Serial.begin(74880);
    WiFi.mode(WIFI_AP_STA);
```

```
    setupAccessPoint();
}
// Handling the / root web page from my server
void handle_index() {
    server.send(200, "text/plain", "Get the f**k out from my server!");
}

// Handling the /feed page from my server
void handle_feed() {
    String t = server.arg("temp");
    String h = server.arg("hum");

    server.send(200, "text/plain", "This is response to client");
    setupStMode(t, h);
}

void setupAccessPoint(){
    Serial.println("*** SETUP ACCESS POINT ***");
    Serial.println("- disconnect from any other modes");
    WiFi.disconnect();
    Serial.println("- start ap with SID: " + String(ssid));
    WiFi.softAP(ssid, password);
    IPAddress myIP = WiFi.softAPIP();
    Serial.print("- AP IP address is :");
    Serial.print(myIP);
    setupServer();
}

void setupServer(){
    Serial.println("*** SETUP SERVER ***");
    Serial.println("- starting server :");
    server.on("/", handle_index);
    server.on("/feed", handle_feed);
    server.begin();
};

void setupStMode(String t, String v){
    Serial.println("*** SETUP STATION MODE ***");
    Serial.println("- disconnect from any other modes");
    WiFi.disconnect();
    Serial.println();
    Serial.println("- connecting to Home Router SID: *****");
    WiFi.begin("*****", "*****");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println();
    Serial.println("- succesfully connected");
    Serial.println("- starting client");

    WiFiClient client;

    Serial.println("- connecting to Database server: " + String(logServer));
    if (client.connect(logServer, 80)) {
        Serial.println("- succesfully connected");
        String postStr = apiKey;
```

```
    postStr += "&field1=";
    postStr += String(t);
    postStr += "&field2=";
    postStr += String(v);
    postStr += "\r\n\r\n";
    Serial.println("- sending data...");
    client.print("POST /update HTTP/1.1\n");
    client.print("Host: api.thingspeak.com\n");
    client.print("Connection: close\n");
    client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");
    client.print("Content-Type: application/x-www-form-urlencoded\n");
    client.print("Content-Length: ");
    client.print(postStr.length());
    client.print("\n\n");
    client.print(postStr);
  }
  client.stop();
  Serial.println("- stopping the client");
  /** If your ESP does not respond you can just
  *** reset after each request sending
  Serial.println("- trying reset");
  ESP.reset();
  **/
}

void loop() {
  server.handleClient();
}
```

As you can see, it starts in AP_ST mode and configures the web server and the main event listeners. On the main page listener ("/feed"), the `setupStMode()` function is called, which **connects to the main Internet router**, takes the readings as parameters and **send** them via **http request to the thingspeak IoT** platform. After that it continues to listen for other clients in AP mode. If you encounter problems after doing a complete send, use `ESP.reset()`; to quickly get back in the AP mode.

Option 2:

RF Modules+ Arduino

Description

These RF modules are very popular among the Arduino tinkerers. The 433MHz is used on a wide variety of applications that require wireless control.

These modules are very cheap and you can use them with any microcontroller (MCU).

Specifications RF 433MHz Receiver

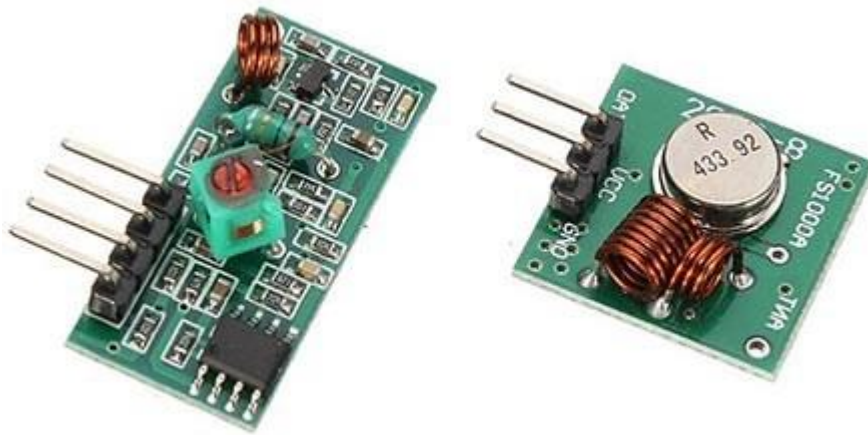
- Frequency Range: 433.92 MHz
- Modulation: ASK
- Input Voltage: 5V
- Price: \$1 to \$2

Specifications RF 433MHz Transmitter

- Frequency Range: 433.92MHz
- Input Voltage: 3-12V
- Price: \$1 to \$2

Where to buy?

You can purchase these modules from eBay for just a few dollars. [Click here to see RF 433MHz module on eBay.](#)



Arduino with RF 433MHz Modules

You need the following components to make this circuit:

- 2x Arduino ([eBay](#))
- RF 433MHz Receiver/Transmitter ([eBay](#))
- Breadboard ([eBay](#))

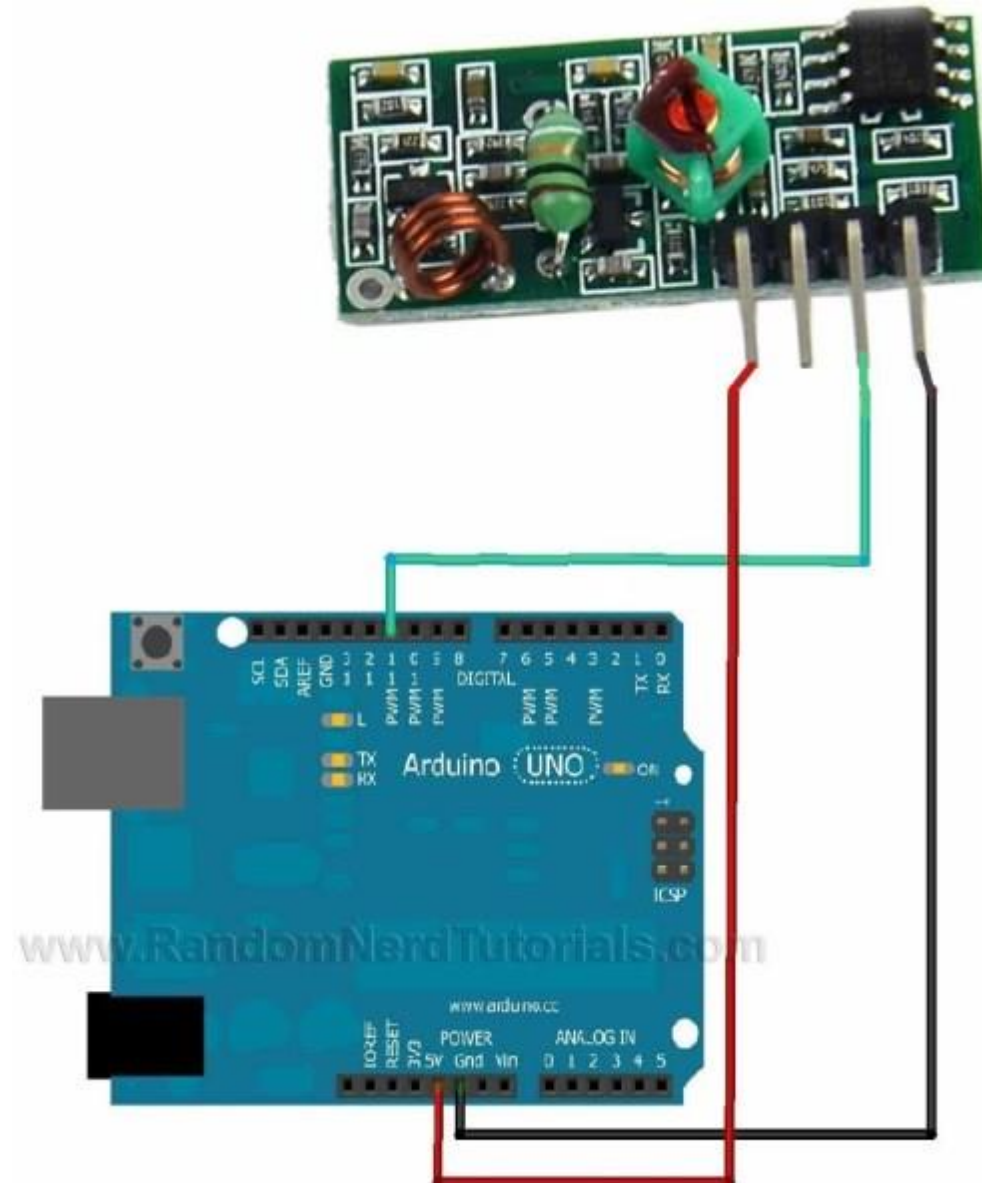
Library download

Here's the library you need for this project:

1. Download the [RadioHead library](#)
2. Unzip the RadioHead library
3. Install the RadioHead library in your Arduino IDE
4. Restart your Arduino IDE

The RadioHead library is great and it works with almost all RF modules in the market. You can read more about this project [here](#).

Receiver Circuit



Follow the circuit above for your receiver. Then upload the code below.

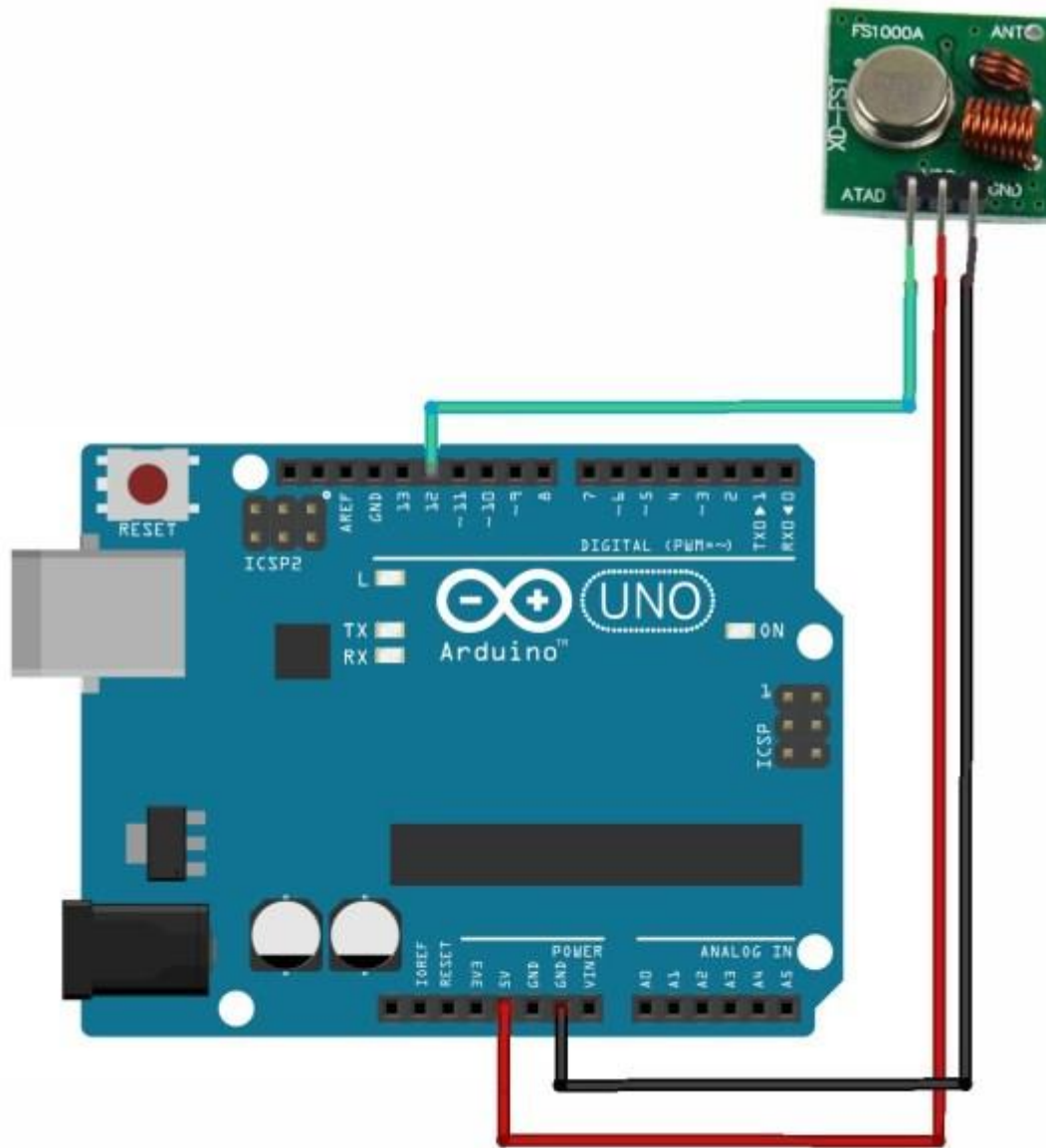
```
#include <RH_ASK.h>
#include <SPI.h> // Not actually used but needed to compile

RH_ASK driver;

void setup()
{
    Serial.begin(9600);          // Debugging only
    if (!driver.init())
        Serial.println("init failed");
}

void loop()
{
    uint8_t buf[12];
    uint8_t buflen = sizeof(buf);
    if (driver.recv(buf, &buflen)) // Non-blocking
    {
        int i;
        // Message with a good checksum received, dump it.
        Serial.print("Message: ");
        Serial.println((char*)buf);
    }
}
```

Transmitter Circuit



Follow the

circuit above for your transmitter. Then upload the code below.

```
#include <RH_ASK.h>
#include <SPI.h> // Not actually used but needed to compile
```

```
RH_ASK driver;
```

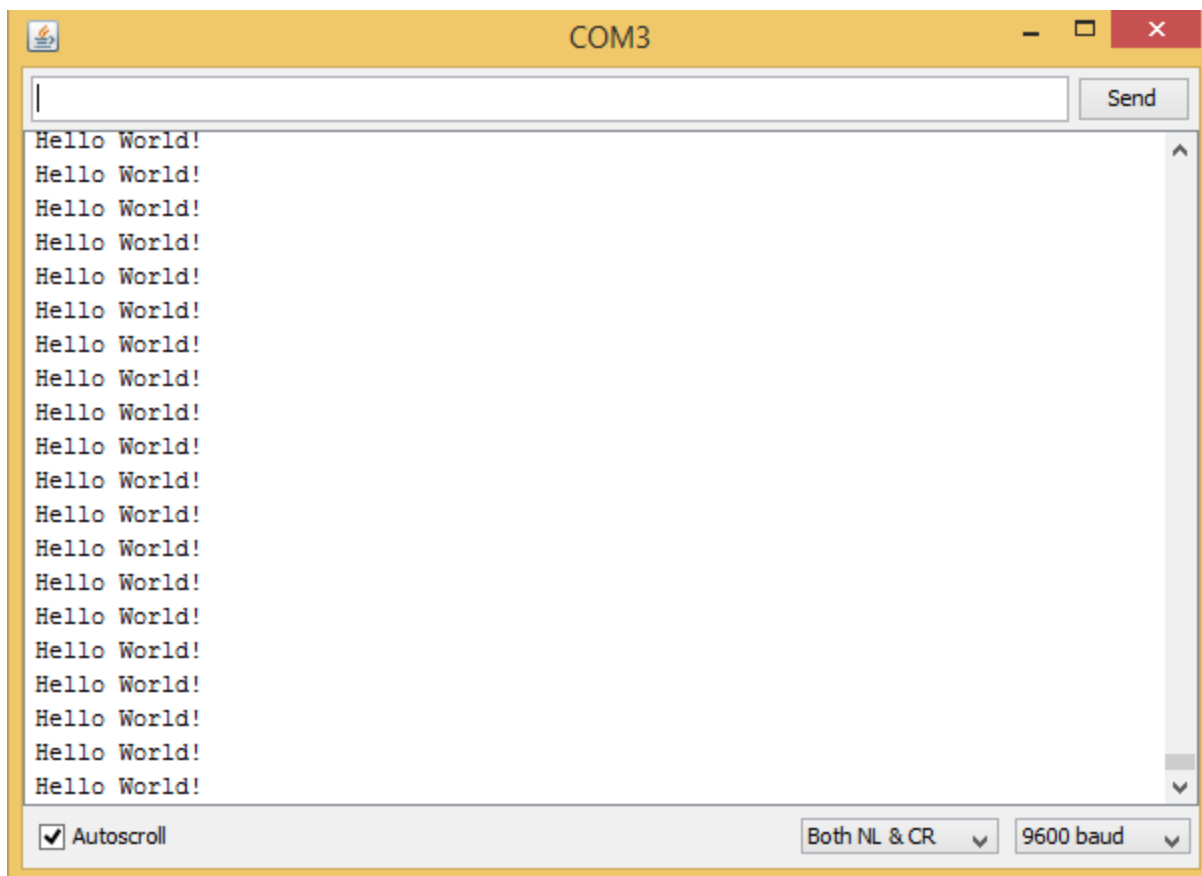
```
void setup()
{
    Serial.begin(9600);          // Debugging only
    if (!driver.init())
        Serial.println("init failed");
}
```

```
void loop()
{
```

```
const char *msg = "Hello World!";  
driver.send((uint8_t *)msg, strlen(msg));  
driver.waitPacketSent();  
delay(1000);  
}
```

Demonstration

In this project the transmitter is sending a message “Hello World!” to the receiver via RF. Those messages are being displayed in the serial monitor from the receiver. Here’s what you should see in your Arduino IDE serial monitor.



Conclusion

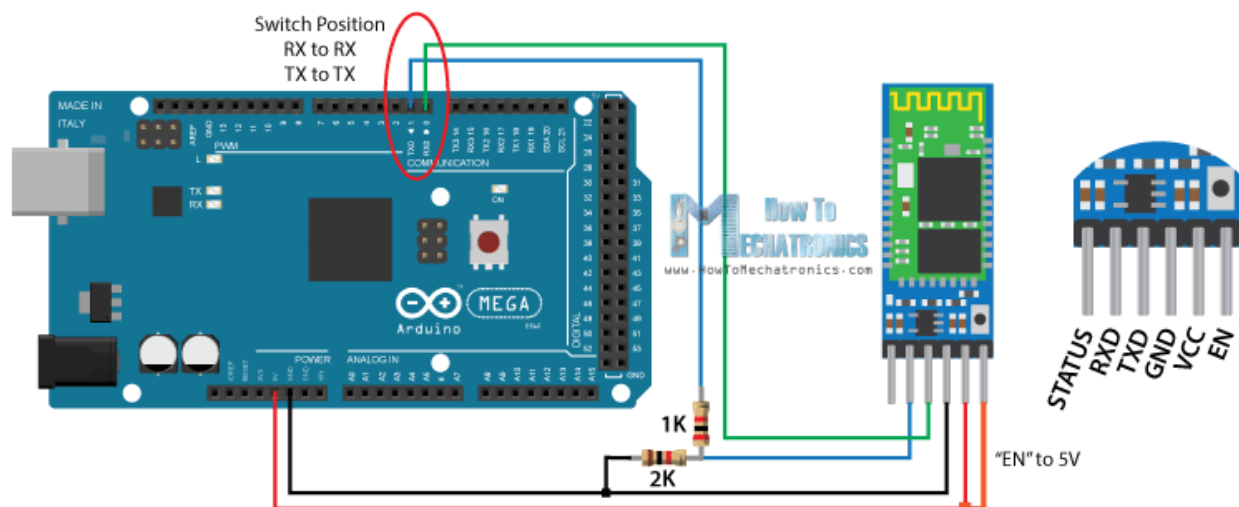
You need to have some realistic expectations when using this module. They work very well when the receiver and transmitter are close to each other. If you separate them too far you'll lose the communication.

The communication range will vary. It depends on how much voltage that you're supplying to your transmitter module, RF noise in your environment and if you're using an external antenna.

Option-3: **Using Bluetooth Modules (HC-05) Master and slave pair**

Configuring the HC-05 Bluetooth Module – AT Commands

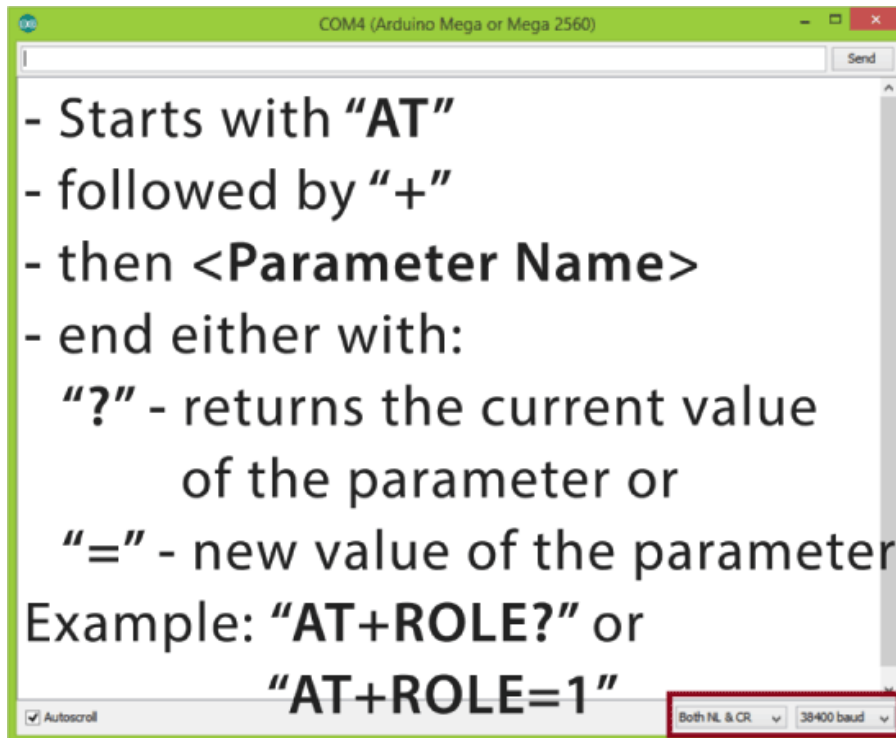
For this tutorial we need to configure both modules. In order to do that we need to switch to AT Command Mode and here's how we will do that. First we need to connect the Bluetooth module to the Arduino as the circuit schematics explained in the previous tutorials. What we need to do additionally is to connect the "EN" pin of the Bluetooth module to 5 volts and also switch the TX and RX pins at the Arduino Board.



So the RX pin of the Arduino needs to be connected to the RX pin of the Bluetooth module, through the voltage divider, and the TX pin of the Arduino to the TX pin of the Bluetooth module. Now while holding the small button over the "EN" pin we need to power the module and that's how we will enter the command mode. If the Bluetooth module led is flashing every 2 seconds that means that we have successfully entered in the AT command mode.

After this we need to upload an empty sketch to the Arduino but don't forget to disconnect the RX and TX lines while uploading. Then we need to run the Serial Monitor and there select "Both NL and

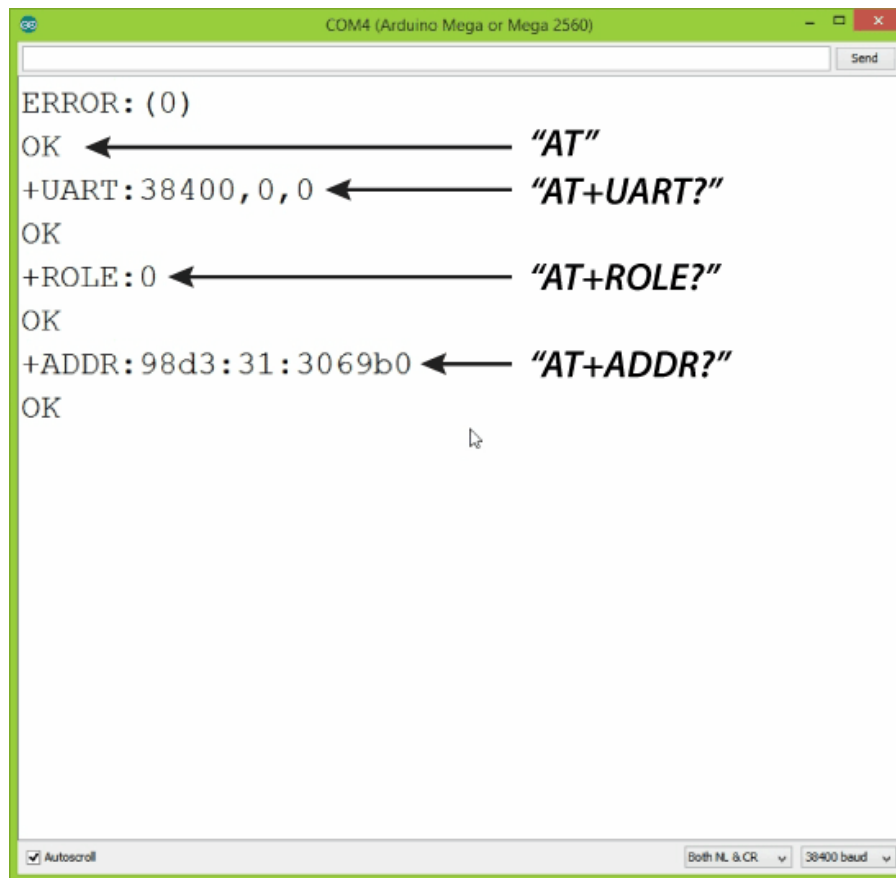
CR", as well as, "38400 baud" rate which is the default baud rate of the Bluetooth module. Now we are ready to send commands and their format is as following.



All commands start with "AT", followed by the "+" sign, then a <Parameter Name> and they end either with the "?" sign which returns the current value of the parameter or the "=" sign when we want to enter a new value for that parameter.

Slave Configuration

So for example, if we type just "AT" which is a test command we should get back the message "OK". Then if we type "AT+UART?" we should get back the message that shows the default baud rate which is 38400. Then if we type "AT+ROLE?" we will get back a message "+ROLE=0" which means that the Bluetooth device is in slave mode. If we type "AT+ADDR?" we will get back the address of the Bluetooth module and it should look something like this: **98d3:34:905d3f**.



The screenshot shows a serial terminal window titled "COM4 (Arduino Mega or Mega 2560)". The window contains the following text:

```
ERROR: (0)
OK
+UART:38400,0,0
OK
+ROLE:0
OK
+ADDR:98d3:31:3069b0
OK
```

Arrows point from the command strings on the right to the corresponding response lines on the left:

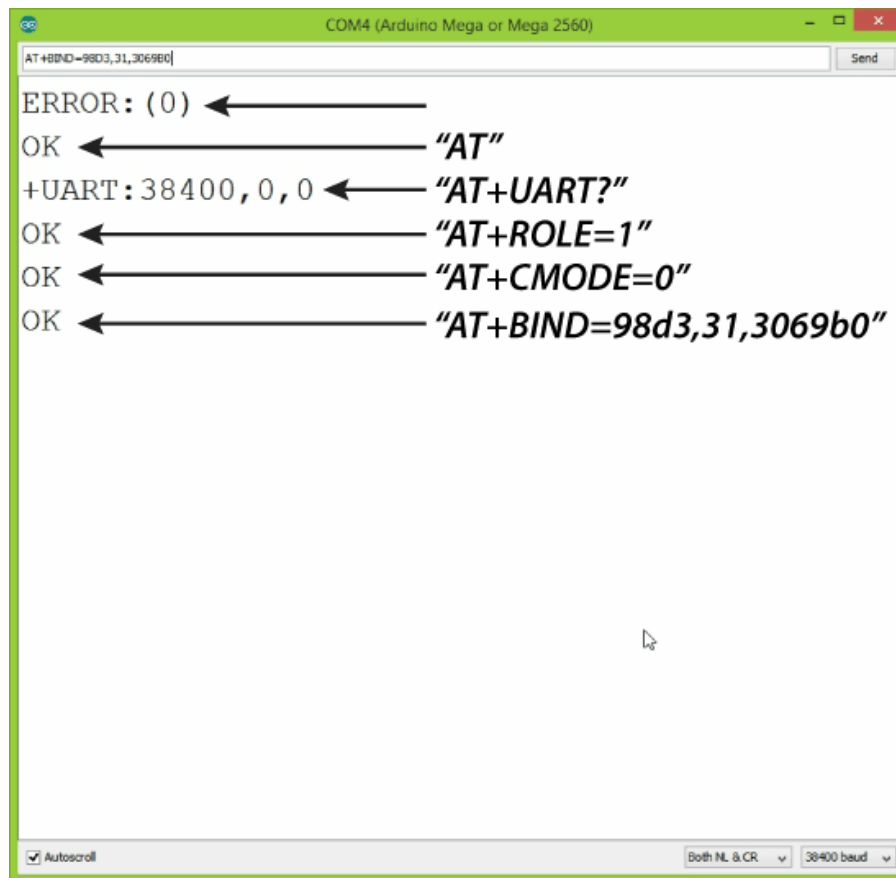
- "AT" points to the first "OK".
- "AT+UART?" points to "+UART:38400,0,0".
- "AT+ROLE?" points to "+ROLE:0".
- "AT+ADDR?" points to "+ADDR:98d3:31:3069b0".

At the bottom of the window, there is a checkbox for "Autoscroll" which is checked, and a dropdown menu for "Both NL & CR" with "38400 baud" selected.

Now we need to write down this address as we will need it when configuring the master device. Actually that's all we need when configuring the slave device, to get its address, although we can change many different parameters like its name, baud rate, pairing password and so on, but we won't do that for this example.

Master Configuration

Ok now let's move on and configure the other Bluetooth module as a master device. First we will check the baud rate to make sure it's the same 38400 as the slave device. Then by typing "AT+ROLE=1" we will set the Bluetooth module as a master device. After this using the "AT+CMODE=0" we will set the connect mode to "fixed address" and using the "AT+BIND=" command we will set the address of the slave device that we previously wrote down.



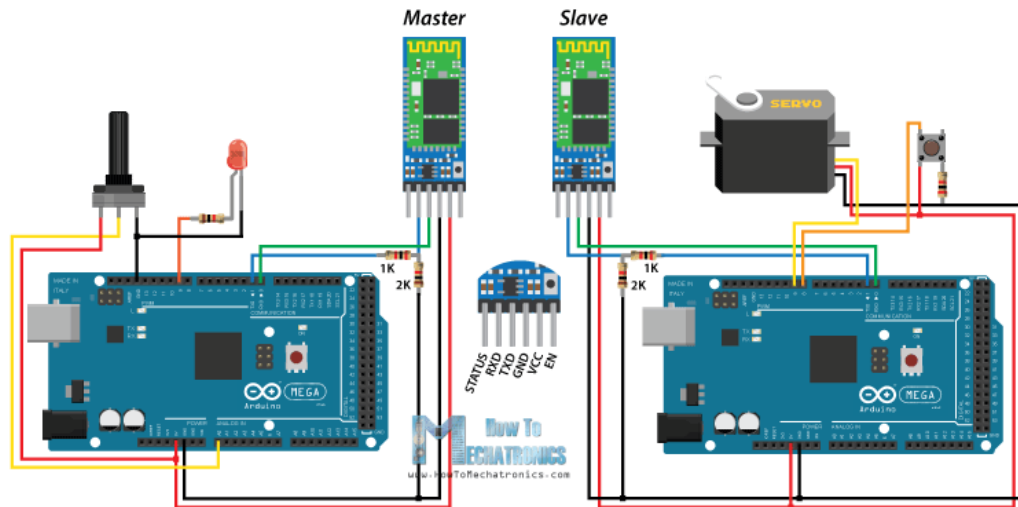
```
COM4 (Arduino Mega or Mega 2560)
AT+BIND=98d3,31,3069b0
Send
ERROR: (0)
OK
+UART:38400,0,0
OK
OK
OK
OK
AT+BIND=98d3,31,3069b0
```

Note here that when writing the address we need to use commas instead of colons. Also note that we could have skipped the previous step if we entered "1" instead of "0" at the "AT+CMODE" command, which makes the master to connect to any device in its transmission range but that's less secure configuration. Here you can find a complete list of commands and parameters: [HC-05 AT Commands List](#)

Nevertheless, that's all we need for a basic configuration of the Bluetooth modules to work as a master and slave devices and now if we reconnect them in normal, data mode, and re-power the modules, in a matter of seconds the master will connect to the slave. Both modules will start flashing every 2 seconds indicating a successful connection.

Communcation Between Two HC-05 Bluetooth Module Example

Ok so now we are ready make the practical example for this tutorial. Here's the circuit schematics. We will use a potentiometer, at the master, to control a servo motor at the slave. And vice versa, we will use a push button, at the slave, to control a LED at the master.



You can get the components needed for this Arduino tutorial from any of the sites below:

- HC-05 Bluetooth Module [Amazon](#) / [Banggood](#) / [GearBest](#) / [DealExtreme](#) / [ICStation](#)
- Arduino Board [Amazon](#) / [Banggood](#) / [GearBest](#) / [DealExtreme](#) / [ICStation](#)
- Servo Motor..... [Amazon](#) / [Banggood](#) / [GearBest](#) / [DealExtreme](#) / [ICStation](#)
- Potentiometer..... [Amazon](#) / [Banggood](#) / [GearBest](#) / [DealExtreme](#) / [ICStation](#)
- 3x 220 Ohms resistors..... [Amazon](#) / [Banggood](#) / [GearBest](#) / [DealExtreme](#) / [ICStation](#)
- Breadboard and Jump Wires [Amazon](#) / [Banggood](#) / [GearBest](#) / [DealExtreme](#) / [ICStation](#)

**Please note: These are affiliate links. I may make a commission if you buy the components through these links.*

I would appreciate your support in this way!

Arduino Source Codes

Description: So first we need to define the pins and some variables needed for the program. In the setup section, at the master, we set the LED pin as output and set it low right away, as well as, start the serial communication at 38400 baud rate. Similar, at the slave, we set the button pin as input, define the servo to which pin is connected and start the serial communication with the same baud rate.

In the loop section, in both code, with the `Serial.available()` function we will check whether there is available data in the serial port to be read and using the `Serial.read()` function we will read and store the data into the "state" variable. So if the master receive the character '1' which is sent from the slave when the button state is high, or the button is pressed, the LED will be on. Else if the character is '0' the LED will be off.

As for the servo motor control, first at the master, we read the potentiometer value and map it into a suitable range for the servo from 0 to 255. This value is sent to the slave which use it to rotate the servo motor accordingly. That's all we need and here's the demonstration of the example.

Master Code:

```
1.  /*
2.   * How to configure and pair two HC-05 Bluetooth Modules
3.   * by Dejan Nedelkovski, www.HowToMechatronics.com
4.   *
5.   * == MASTER CODE ==
6.   */
7.
8.  #define ledPin 9
9.
10. int state = 0;
11. int potValue = 0;
12.
13. void setup() {
14.   pinMode(ledPin, OUTPUT);
15.   digitalWrite(ledPin, LOW);
16.   Serial.begin(38400); // Default communication rate of the Bluetooth module
17. }
18.
19. void loop() {
20.   if(Serial.available() > 0){ // Checks whether data is coming from the serial port
21.     state = Serial.read(); // Reads the data from the serial port
22.   }
23.   // Controlling the LED
24.   if (state == '1') {
25.     digitalWrite(ledPin, HIGH); // LED ON
26.     state = 0;
27.   }
28.   else if (state == '0') {
29.     digitalWrite(ledPin, LOW); // LED ON
30.     state = 0;
31.   }
32.   // Reading the potentiometer
33.   potValue = analogRead(A0);
34.   int potValueMapped = map(potValue, 0, 1023, 0, 255);
35.   Serial.write(potValueMapped); // Sends potValue to servo motor
36.   delay(10);
37. }
```

Slave Code:

```
1.  /*
2.   * How to configure and pair two HC-05 Bluetooth Modules
3.   * by Dejan Nedelkovski, www.HowToMechatronics.com
4.   *
5.   * == SLAVE CODE ==
6.   */
7. 
```

```
8.  #include <Servo.h>
9.  #define button 8
10.
11. Servo myServo;
12. int state = 20;
13. int buttonState = 0;
14.
15. void setup() {
16.  pinMode(button, INPUT);
17.  myServo.attach(9);
18.  Serial.begin(38400); // Default communication rate of the Bluetooth module
19. }
20.
21. void loop() {
22.  if(Serial.available() > 0){ // Checks whether data is coming from the serial port
23.  state = Serial.read(); // Reads the data from the serial port
24.  }
25.  // Controlling the servo motor
26.  myServo.write(state);
27.  delay(10);
28.
29.  // Reading the button
30.  buttonState = digitalRead(button);
31.  if (buttonState == HIGH) {
32.  Serial.write('1'); // Sends '1' to the master to turn on LED
33.  }
34.  else {
35.  Serial.write('0');
36.  }
37. }
```