# Insteon ESP8266 IoT

**What You will Need:**

- Any ESP8266-12 module or board with an ESP8266-12 module on it.
- USB-to-Serial bridge with RTS and DTR unless the ESP8266 board has it built-in.
- 5V, 2A power supply.
- Power Jack that mates with the power supply.
- Eight 10K resisters
- Eight Pushbutton Switches
- Custom Circuit Board or hand-wired circuit board
- See individual Bills of Materials for complete parts list.

**Software apps and online services:**

- Arduino IDE, version 1.8.0 or higher
- ESP8266 Plug-in for Arduino
- Optional, OpenSCAD

**Hand tools and fabrication machines:**

- Soldering iron
- Solder, preferably with no clean flux
- Diagonal cutters
- Screwdriver
- Optional, 3D printer

**Why I did this:**

Home automation is a hot topic these days. It covers everything from window and door sensors to WiFi enabled thermostats to remote control of lights and appliances. I use Insteon devices to control some of the lights in my house and a ceiling fan/light in my bedroom. I originally chose Insteon because it is backward compatible with my older X10 devices. With the inclusion of an Insteon Hub, you can install the Insteon app on your Android or IOS device and control your Insteon devices with your phone or tablet. Insteon is also compatible with the Amazon Echo Dot, a.k.a Alexa. With Insteon and Alexa you can issue voice commands to control your Insteon devices. Very cool. So why would you need yet another way to control your Insteon devices ? The reasons are that Alexa is expensive and does not always work! Your local Amazon Echo recognizes the key phrase “Alexa” and then uses your Internet connection to forward your voice command to an Amazon server for processing. The reply is a digital command sequence that your Alexa forwards to the Insteon Hub. The Insteon Hub then performs the magic. Sometimes Alexa tells me that my Insteon Hub cannot be found or that she is having trouble connecting to the Amazon servers. The end result is that my voice commands do not work. So I have to find my phone or a tablet, wait for it to boot up and then issue the command using the Insteon app. This can be a hassle if you just got into bed and want to turn off the light. You have to get back up and locate an Android device. Ugh.

It would be so much nicer if you had a simple WiFi device with switches that could be programmed to turn on/off your Insteon devices on your bedside table. It would also be great if said device did not rely on having to send messages over the Internet to remote servers to send commands to your local Insteon Hub. This is exactly what I have designed. A simple, low cost, ESP8266 powered IoT device that sends commands to your Insteon Hub, using your local WiFi and wired network. Internet is not required and no outside servers are used.

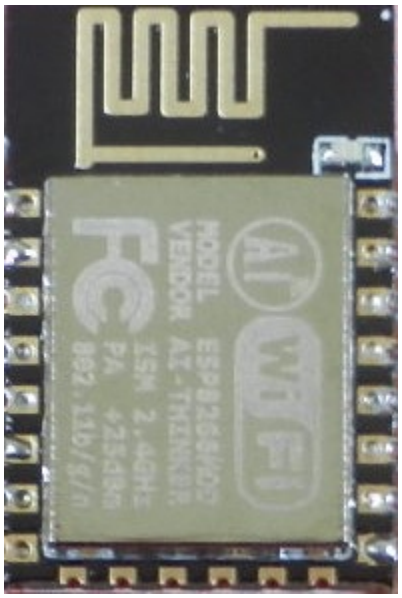
I designed my Insteon Switch Box to have up to eight pushbutton switches. Each switch can be programmed to turn on or off one Insteon device or scene. A switch may also be programmed to read the state of an Insteon device and then issue a command to change the state. The result is one switch that with one press turns on an Insteon device and a second press turns off the same device. I use an Insteon Fanlinc device to control a ceiling fan in my bedroom. A Fanlinc has two sections. One section controls the light and the other controls the fan. The fan can have three speeds and the light is dimmable. Alexa cannot work directly with an Insteon Fanlinc. In order to use a Fanlinc with Alexa you must create scenes. One scene for the light and one scene for each of the fan speeds. Because scenes are either on or off, Alexa cannot dim the Fanlinc's light. My Insteon Switch box can send commands to the Insteon Hub to turn on or off or dim the Fanlinc's light and to control the fan without the need for scenes. This is an improvement over what Alexa can do. My Insteon Switch box was not designed to control a thermostat or listen for door or window sensors. However, with Insteon On/Off plug-in modules that use a relay to turn on/off the plugged in object, it is possible to control drapes or a coffee maker, toaster or electric kettle. You merely have to press a button when you get out of bed. When you exit the shower, your coffee will be ready. All without waking up your mate by talking to Alexa.

The remainder of this document will detail how to build one of these for yourself and how to configure the software so it will work with your Insteon devices. There is also some small lessons in writing 'C' code.

**How to build one:**

I designed a custom PCB for this project. The zip archive from GitHub has a pdf of the schematic. If you have worked with the ESP8266 then you are aware that it comes in a dizzying array of forms. From bare modules that require support circuits to breadboard friendly boards with all required support circuits and a USB-to-serial bridge. The board that I designed for this project can accept the ESP8266 in four different forms and has pads to add any support circuits required by the ESP8266 you chose to use. My board does not have a USB-to-Serial bridge on board. If that is required there is a header to attach one. Your USB-to-Serial bridge must have RTS and DTR in order to program the ESP8266. Of course, you are free to handwire the switches to the ESP8266-12 of your choice. The four ESP8266 forms supported by my board are;

***ESP8266-12 module***

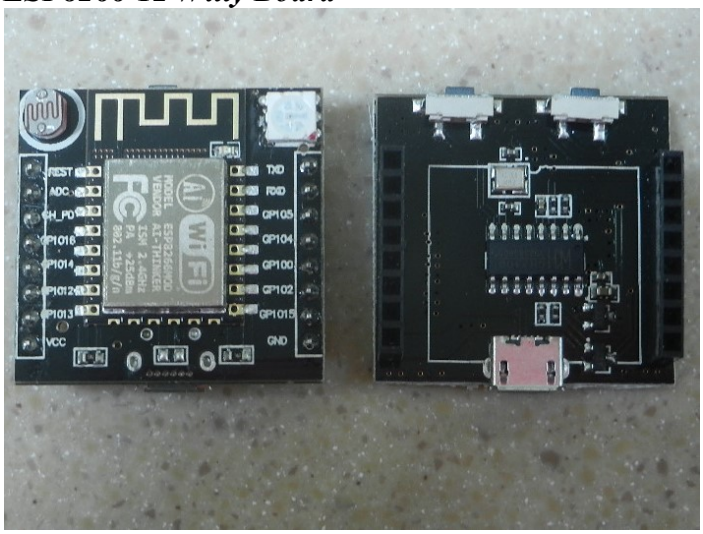


The least expensive ESP8266, but requires the most support. My board has pads for a 3.3V regulator and the 'auto-reset' circuit used for programming. You will need a USB-to-Serial bridge with RTS and DTR.

Bill of Materials

| Quantity | Reference Numbers | Description |
|---|---|---|
| 1 | SOC1 | ESP8266-12 module |
| 8 | SW1 – SW8 | Pushbutton switches, four leads, 7x4mm spacing |
| 10 | R1, R3 – R11 | 10K Ω, surface mount resistors, 1206 |
| 1 | R2 | 470 Ω, surface mount resistor, 1206 |
| 2 | R12, R13 | 1K Ω, surface mount resistors, 1206 |
| 2 | R14, R15 | 4.7K Ω, surface mount resistors, 1206 |
| 1 | VR1 | AS1117-3.3V regulator, SOT-223 |
| 2 | Q1, Q2 | MMBT2222A transistor, TO-23 |
| 2 | C1, C2 | 10uf, 16V, Tantalum, capacitors, 1206, size A |
| 1 | J5 | 3 pin right angle, male header |
| 1 | J8 | 6 pin right angle, male header |

***ESP8266-12 Witty Board***



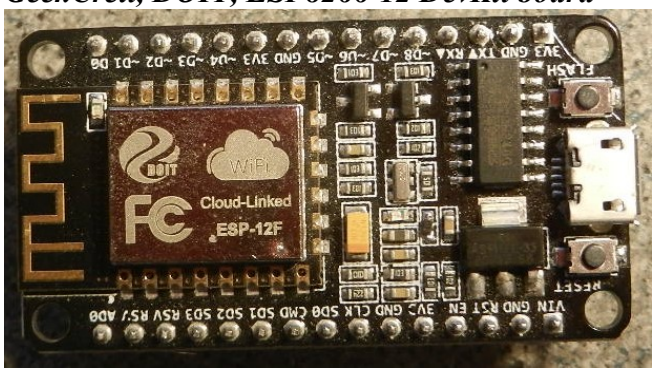

This is actually two boards that mate together. If you use this you will only need the top board that actually contains the ESP8266 and the 3.3V regulator. You can program the ESP8266 while attached to the lower board it came with and then mount it to your board or you can program it in place by using jumpers to connect to the lower board or adding the 'auto-reset' circuit and using a USB-to-Serial bridge. Do not throw away the lower board. You can always use it as a USB-to-Serial bridge. Just be aware that it provides 5V signals, not 3.3V signals like most USB-to-Serial bridges. The Witty board does not require the pulldowns on GPIO15 and the analog input as these parts are on the Witty board. You can also eliminate the pullups on Reset and GPIO0 (SW7) for the same reason.

Bill of Materials (BOM)

| **Quantity** | **Reference Numbers** | **Description** |
|---|---|---|
| 1 | | Witty ESP8266-12 board |
| 2 | J1, J2 | 8 pin female socket headers, optional |
| 8 | SW1 – SW8 | Pushbutton switches, four leads, 7x4mm spacing |
| 10 | R3 – R9, R11 | 10K Ω, surface mount resistors, 1206 |
| 2 | R12, R13 | 1K Ω, surface mount resistors, 1206 |
| 1 | VR1 | AS1117-3.3V regulator, SOT-223 |
| 2 | Q1, Q2 | MMBT2222A transistor, TO-23 |
| 2 | C1, C2 | 10uf, 16V, Tantalum, capacitors, 1206, size A |
| 1 | J5 | 3 pin right angle, male header |
| 1 | J8 | 6 pin right angle, male header |

Note:
If you do not intend to program the ESP8266 Witty board in circuit then you can eliminate; Q1, Q2, J8, R12 and R13. If you solder the Witty board in place then use the holes for J1 and J2 and you can eliminate J1 & J2. Power can be applied using the micro USB connector on the bottom of the Witty board with the ESP8266 or can be wired to J5 or J8. You can eliminate J5 if you intend to use J8 or the micro USB connector.

***GeekCreit, DOIT, ESP8266-12 DevKit board***





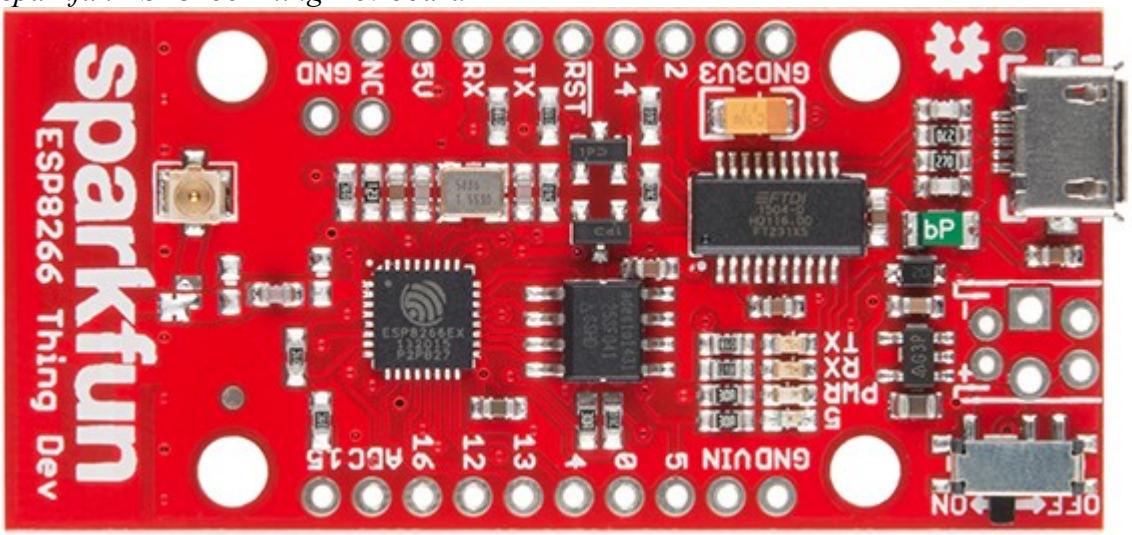
This is a single board that has everything onboard to program the ESP8266. There is a micro USB connector and an onboard USB-to-Serial bridge. No support circuits are required to use this board. Boards like this are also known as NodeMcu boards. These boards come in two widths, wide and narrow. My circuit board is laid out to accept the narrower boards. Look for one that has pin spacing of 0.9 inches. The original NodeMcu boards have 1.2 inch spacing and will not fit.

Bill of Materials

| Quantity | Reference Numbers | Description |
|---|---|---|
| 1 | | GeekCreit, DoIt ESP8266-12 board |
| 2 | J3, J4 | 15 pin female socket headers, optional |
| 8 | SW1 – SW8 | Pushbutton switches, four leads, 7x4mm spacing |
| 10 | R3 – R9, R11 | 10K Ω, surface mount resistors, 1206 |
| 1 | J5 | 3 pin right angle, male header, optional |

Note:
If you solder the GeekCreit, DoIt board in place then you can eliminate J3 & J4. You can power the system using the micro USB connector on the GeekCreit board or using J5.

*Sparkfun ESP8266 Thing Dev board*




I included this board because Sparkfun is a co-sponsor of the contest that I designed this project for. Unfortunately, the sponsors did not like my idea well enough to provide me with a free Sparkfun ESP8266 Thing Dev board. The Sparkfun board has all of the required support circuits already on it, including the USB-to-Serial bridge.

Bill of Materials

| Quantity | Reference Numbers | Description |
|---|---|---|
| 1 | | Sparkfun ESP8266 Thing Dev board |
| 2 | J6, J7 | 10 pin female socket headers, optional |
| 2 | | 10 pin male headers |
| 8 | SW1 – SW8 | Pushbutton switches, four leads, 7x4mm spacing |
| 10 | R3 – R9, R11 | 10K Ω, surface mount resistors, 1206 |
| 1 | J5 | 3 pin right angle, male header, optional |

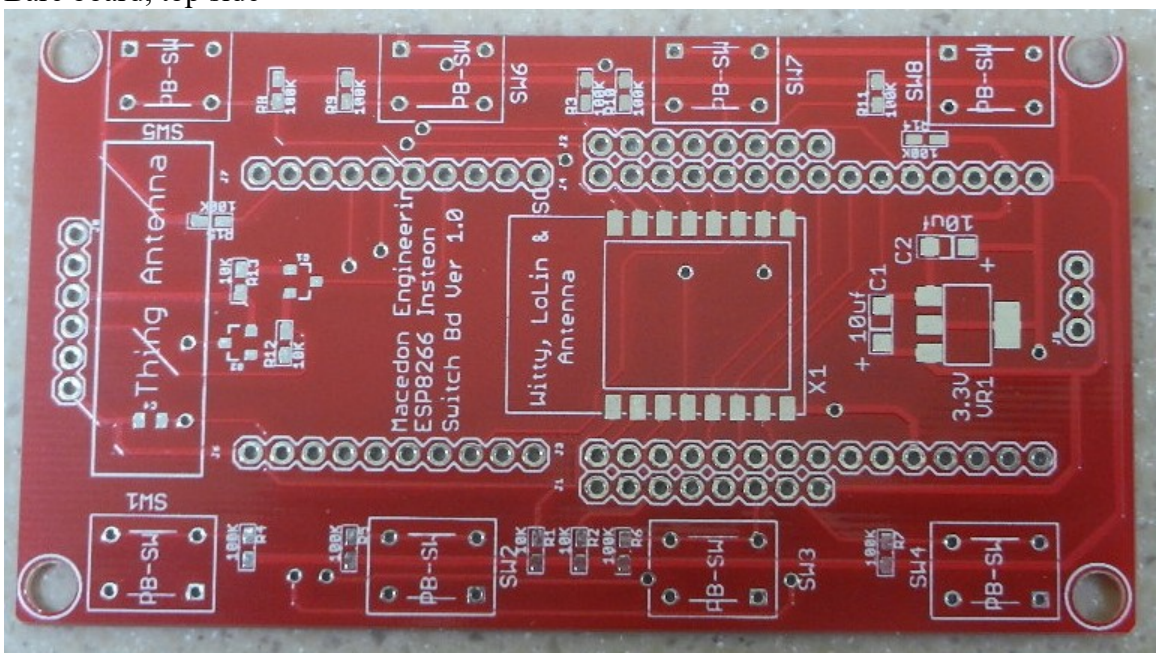
Note:
If you solder the Sparkfun ESP8266 Thing Dev board in place then you can eliminate J6 & J7. You can power the system using the micro USB connector on the Sparkfun ESP8266 Thing Dev board or using J5. The two 10 pin male headers are soldered to the bottom of the Sparkfun ESP8266 Thing Dev board (they should come with the board).

My custom circuit board has places to mount eight pushbutton switches and provides pads for a 10K pullup for each switch. The pullups are required due to the ESP8266 not having internal pullups on the GPIO pins. All of the GPIO pins except the analog pin, GPIO15, Tx and Rx are used for the pushbutton switches. The analog input and GPIO15 have pulldowns wired to them and Rx has a pullup. No pins are left floating. Always a good thing to do. Of course, you are free to handwire your own board. Then you can use whatever ESP8266 board that you have available. You can leave out the 3.3V regulator and 'auto-reset' circuits if the board you use does not require them. Mount the ESP8266 board of your choice with the antenna oriented as indicated on the circuit board. Use the set of holes or pads that fit the board that you are using. The mounting holes in the corners are sized to accept 3mm, 4-40 or 6-32 hardware.

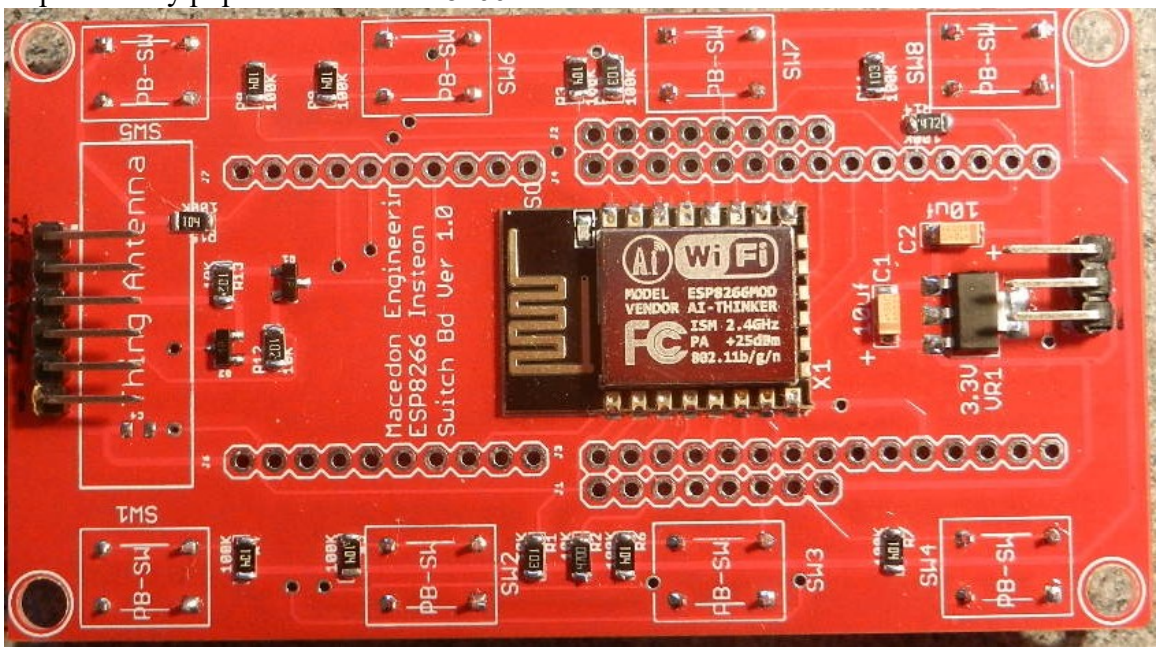
I prefer to assemble boards from the lowest height components to the tallest components. Therefore I usually start with the surface mount resistors, transistors, capacitors and regulators. Then comes the ESP8266 module. Last would be the headers and sockets. I built my boards with right angle male headers facing inward to reduce the overall height and length of the finished board. The pushbutton switches are intended to be mounted on the bottom side of the board. I have included pictures of assembled boards with different ESP8266 boards attached. The ESP8266 boards were not soldered down. They are in place to show location and orientation. They were paid scale for their modeling work.

***Gallery***

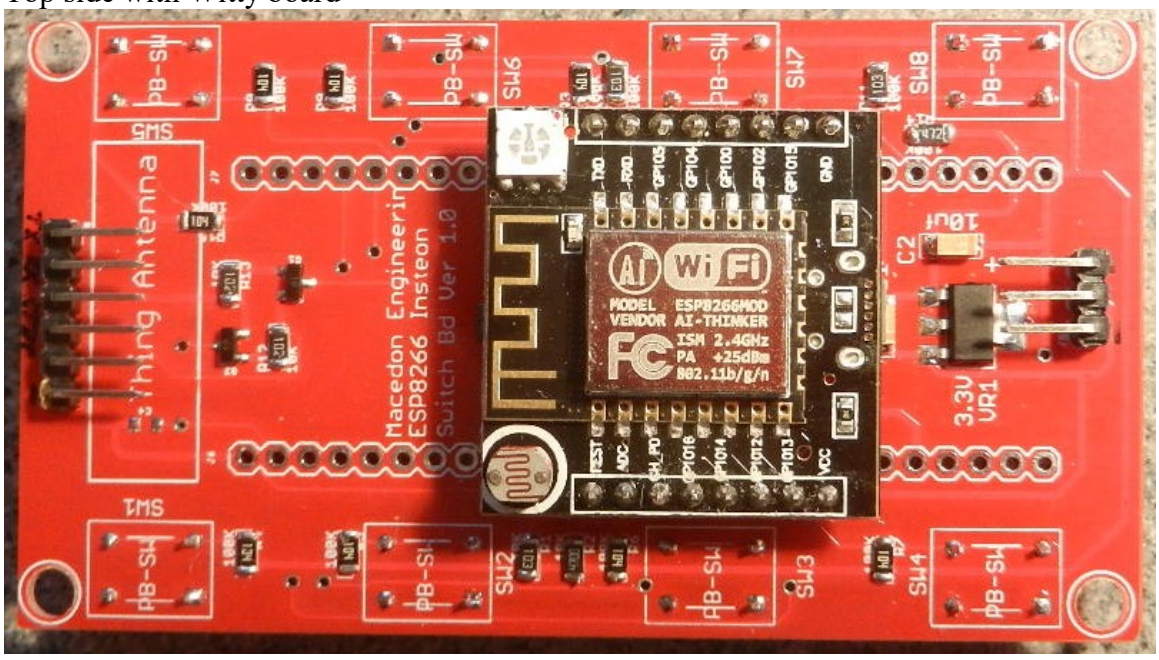
Bare board, top side



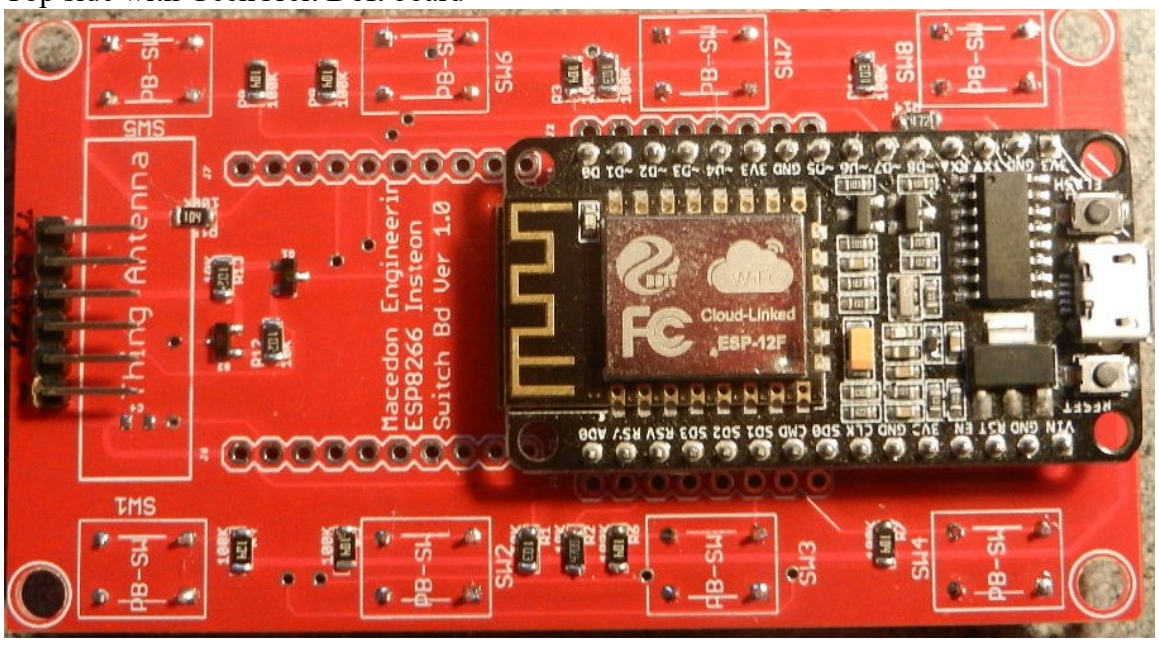

Top side fully populated with ESP8266 module



Top side with Witty board

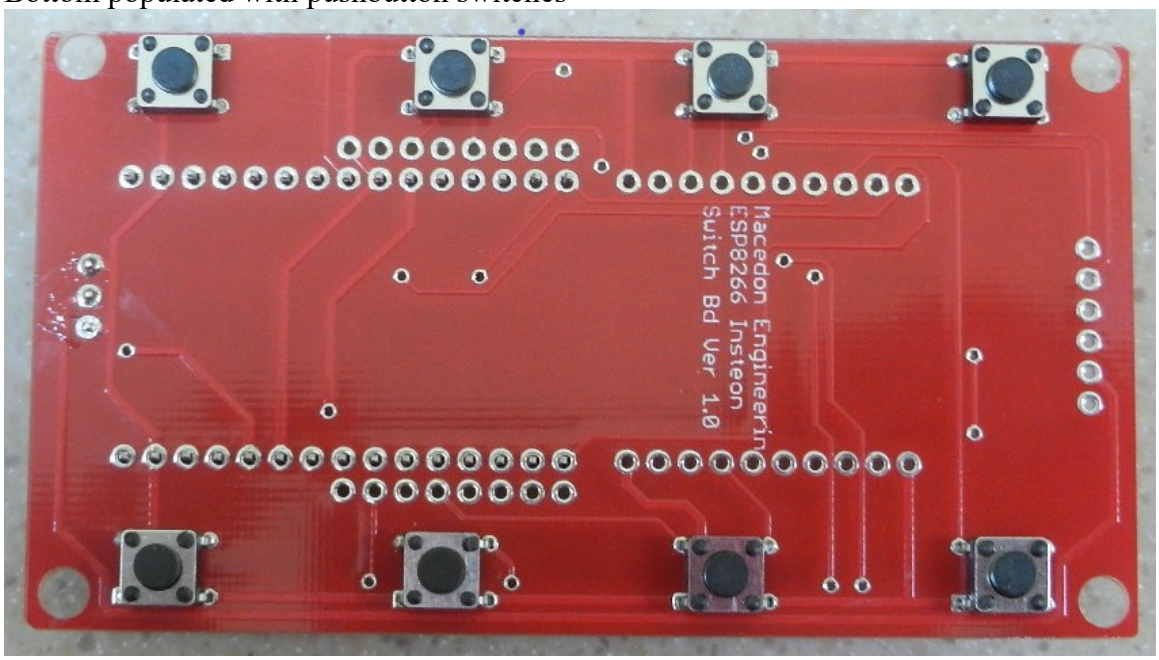



Top side with GeekCreit DoIt board




No picture of the Sparkfun ESP8266 Thing Dev board in position and orientation is available. The Sparkfun board declined to attend the photo shoot.

Bottom populated with pushbutton switches




**How it works:**

The ESP8266 at the heart of the Insteon Switch box debounces the switches and looks for a switch pressed. When it detects that a switch has been pressed it looks up what has been programmed for that switch to do and then creates a control message for the Insteon Hub. The message is sent to the Hub, which converts the message into a control command and forwards it to the targeted Insteon device. Any replies from the targeted Insteon device are sent back to the Hub and the Hub then sends the reply back to ESP8266 using the Insteon Hub message format. The reply is parsed and the relevant data is stored. The Internet is not used. Everything is done over your local network.

**How to configure the firmware for your Insteon devices:**

There are a lot of websites providing detailed instructions on how to setup the Arduino IDE, compile a program and upload it as well as how to install the ESP8266 plug-in. I am not going to repeat that information here. Just Google it.

I have defined a 'C' structure named *TSwitch*, that contains all the information needed for one pushbutton switch. The structure is as follows;

```
Typedef struct {
byte        pin;        // GPIO pin number that the switch is wired to
int16_t     state;      // the last known state of the Insteon device
uint32_t    debounce;   // switch debouncer bit array
ActionPtr   action;     // pointer to the action function assigned to the switch
char        *devId;     // pointer to the device ID string of the Insteon device
```

```
char        *cmd2;      // pointer to the cmd2 string to send for status requests
char        *cmdOn;     // pointer to the on command string
char        *cmdOff;    // pointer to the off command string
} TSwitch;
```

An array of eight *TSwitch* structures is used to hold the information for each switch. If you build one of these for yourself you will need to replace the data in some of the fields of the *TSwitch* structure for how you would like your switches to work and for the Insteon devices and scenes that you want to control. Each Insteon device has a label on it with its unique Insteon ID. I have a series of statements where I define a meaningful to me name for each Insteon device ID. You can find these in the code as;

*#define Kitchen "339EFC"*

Kitchen is the meaningful name, "339EFC" is the Insteon device ID.

The *#define* makes this a 'C' preprocessor macro. This means that wherever *Kitchen* is found in the source code the 'C' preprocessor will substitute "339EFC". *"339EFC"* is the Insteon ID string converted from the dot format on the label (33.9E.FC). I also have some *#define* statements that define the scene Ids that I might use. Scene Ids are two digit number strings such as *"10"* for scene number ten or *"07"* for scene number seven. You can get the scene numbers and Insteon Ids from the Insteon app on your phone or tablet. There are a bunch of *#define* statements that define the command strings that can be used. All of the command strings are prefixed with *Cmd* followed by a descriptive name. For example;

*#define CmdOn        "11FF"        // command to turn on a device to full brightness*

You will need to change the *#define* statements that define the Insteon Ids and scene numbers for the devices and scenes that you use. The *#define* statements for the commands should not be changed.

There is a statement; *#define PINMASK      0x00000FFF*
This statement defines the number of bits that are used to debounce a pushbutton switch. If the buttons that you use have more than 12mS of bounce then you should increase the number of bits in the mask. If you use all 32 bits and require more time then you should change the 1mS delay in the main loop to two or more milliseconds and decrease the number of bits accordingly. Twelve bits and 1mS should be adequate for most pushbutton switches.

As I stated earlier, there is an array of eight *TSwitch* structures, one for each pushbutton switch. A 'C' style initializer is defined for the array. This means that a table of entries for initializing the *TSwitch* array is defined in Flash ROM. The table is copied to the RAM *TSwitch* array automatically when the program starts up. You have to edit this initializer table to configure what you want the pushbutton switches to do and what Insteon devices/scenes each pushbutton will work with. The initializer is of the form;

{SWx, UnknownState, 0, action-to-take, device Id/scene, Cmd2, CmdOn, Cmdoff}

The individual initializers are separated by a comma (‘,’) and the whole group of eight initializers is wrapped in a pair of curly braces (‘{‘ ‘}’). This is just the way that the ‘C’ programming language does things. There must be one initializer for each entry in the array or the compiler will complain. If you leave out a curly brace or a comma the compiler will complain. That is just how compilers are, always complaining that you left something out. The pushbutton switches are arranged in the same numerical order that they are in the hardware, so SW1 in the software is SW1 on the board, is array[0]. I advise against changing the order of the switches. The first three entries in the initializer should remain unchanged. They represent the GPIO pin the pushbutton switch is wired to, the state of the Insteon device (unknown), and the state of the debouncer. The forth value is a pointer to the ‘C’ function that will be called when the switch is pressed. There are a number of action functions defined. You should choose the one that does what you want the pushbutton to do. If you are not using the pushbutton then set the action field to *NULL* (all caps). This will tell the code to ignore this pushbutton. The action functions are; *actionToggle, actionOn, actionOff, actionFanSpeed.*

*actionToggle* reads the current state of the Insteon device whose ID is in the *devId* field of the associated *TSwitch* structure and then issues a command to change to the opposite state. Use *actionToggle* to create a button that will turn something on when pressed and turn it off when pressed again. The *cmdOn* and *cmdOff* fields of the associated *TSwitch* structure are used to get the on and off commands. If the Insteon device being used is dimmable then you could use the *CmdHalf* command in place of *CmdOn*. This would cause the light to toggle between half brightness and off.

*actionOn* simply sends whatever command is in the *cmdOn* field of the associated *TSwitch* structure to the Insteon device whose ID is in the *devId* field of the associated *TSwitch* structure. Use *actionOn* when you want a pushbutton to turn an Insteon device on. If you replace the *CmdOn* with *CmdHalf* and the Insteon device is dimmable then it would come on at half brightness. If you replaced *CmdOn* with *CmdBrighten* then each time you pressed the button the brightness would increase until full brightness is achieved.

*actionOff* simply sends whatever command is in the *cmdOff* field of the associated *TSwitch* structure to the Insteon device whose ID is in the *devId* field of the associated *TSwitch* structure. Use *actionOff* when you want a pushbutton to turn an Insteon device off. If your Insteon device is dimmable then you can substitute *CmdHalf* to reduce the brightness to half or *CmdDim* to reduce the brightness in steps until it was fully off. If you used *CmdHalf* then you will need to use another button to actually turn the device off.

*actionFanSpeed* is unique in that it reads the current fan speed setting and then ups the speed to the next fan speed setting. The cycle is; off -> low -> medium -> high -> off. ActionFanSpeed does not use the cmdOn and cmdOff fields.

The next field in the initializer (*devId*) is a 'C' style character string that contains the Insteon Id of the device or the scene number that you want to control. I previously talked about the *#define* values that I setup to provide meaningful names (aliases) for my Insteon Ids. Here I use those names. You should do the same, making meaningful names for your Insteon Ids or simply put the Insteon Id string here as "123456". Keep In mind that an Insteon device Id will have six characters while a scene number will have two digits ("07").

The next field in the initializer is the *cmd2* field. This field is normally initialized with the value; *Cmd2Lights*. Change this initializer to *Cmd2Fans* if you are setting up a pushbutton to work with the fans section of a Fanlinc Insteon device.

The last two fields of the initializer are the *cmdOn* and *cmdOff* fields. These fields contain pointers to the defined Insteon command strings. Typically you would put *CmdOn* in *cmdOn* and *CmdOff* in *cmdOff*. You can put any of the *CmdXxxx* values in these fields, although *CmdStatus*, *Cmd2Lights* and *Cmd2Fans* should not be used.

One thing to note here, I defined a simple command for *CmdOn* that turns a light on full brightness. If you have dimmable light controller you can define commands that will turn on your light to a preset brightness. The light on command is "11" the "FF" in the second byte is the brightness level. *CmdHalf* is defined as "117F", it will turn your light on to half brightness. You can use any value from "01" to "FF" for the brightness level. Keep in mind that if you send a command for partial brightness to an Insteon device that is not dimmable or a scene then it will turn on at full brightness.

The *CmdOnFast* and *CmdOffFast* commands are be used to override the ramp up and ramp down setting in the Insteon app and force the device to full brightness or off without ramping. I do not use these commands because all my dimmable Insteon devices use the default ramping.

There are also *#define* statements for;

| | |
|---|---|
| SSID | – change to the ssid of your network |
| PASSKEY | – change to the WiFi WPA passkey for your network, |
| HUBUSER | – change to the user ID for your Insteon Hub |
| HUBPASSWORD | – change to the password for your Insteon Hub |
| HUBIP | – change to the IP address of your Insteon Hub |
| HTTPPORT | – you can change this if you change the port number that your Insteon Hub uses, otherwise this is the default value. |

That's it. Make the changes to the source code, recompile and download it to your ESP8266. I hope that you find it to be a useful addition to how you control your Insteon devices. If you have any problems building one for yourself or need help with the code you may contact me through HacksterIO. I will endeavor to provide as much help as I can.

I bought ten of these boards from a Chinese PCB manufacturer. I built up three. I have seven boards that are unused bare boards. If you would like one, tell me why you deserve a free board and maybe I will send you one.

I have included the Eagle files the board was made from in the GitHub archive. Feel free to have your own boards made. Ten boards cost around $15-$20.00 US and take about 3-4 weeks to arrive in the USA.

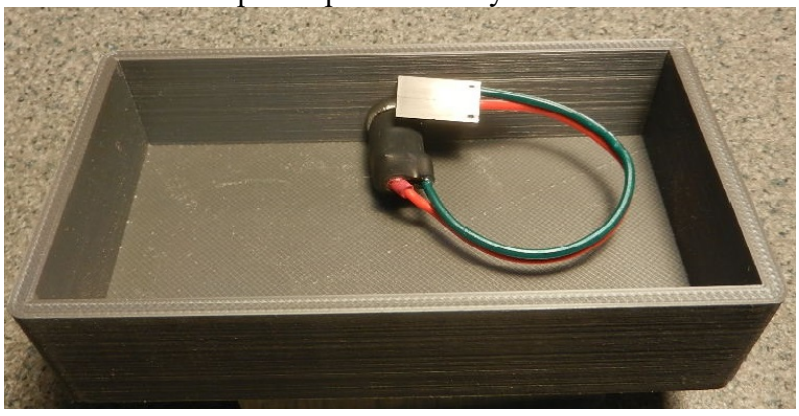
I have also designed a 3D printable case for my project board. The design was done using OpenSCAD. I have include the OpenSCAD source code and the .stl file if you would like to print your own. If you design your own case, I would be interested in seeing what you came up with. Send me a picture of the case you designed. My case uses four 3x6mm screws to hold the board to the top of the case. In the OpenSCAD source file there is a variable named SHD that can be changed to make the hole in the bosses correct for other screw sizes. Always use a ¼ inch or 6mm long screw to avoid coming through the top of the case. After printing the parts, clean off all of the easy removal pads. Insert the buttons into the holes in the top, wide part should be inside the case. Screw the circuit board to the underside of the top. Add the power jack to the case bottom. Solder wires to the power jack. Either add crimp pins and a header, or use an IDC header or solder the wires in place of J5 on the circuit board. The top just snaps into the bottom. The fit should be tight enough that it will stay put without any glue. Congratulations, pat yourself on the back and break out the Champagne. You are done.

The .stl file is for a case top, bottom, four bullet top buttons and four flat top buttons. It is very easy to modify the source code to print only the top or bottom or to change the number of bullet top & flat top buttons. I commented out the top and bottom parts and printed some buttons using red and white filament. Everything is done in a modular fashion with the lines following the *"$fn=50*" to the first "*module*" statement determining what gets printed. "*translate*" statements are used to reposition the parts. If you only print buttons then change the "*translate*" statements to move the buttons to the center of your build platform or where ever you want them.

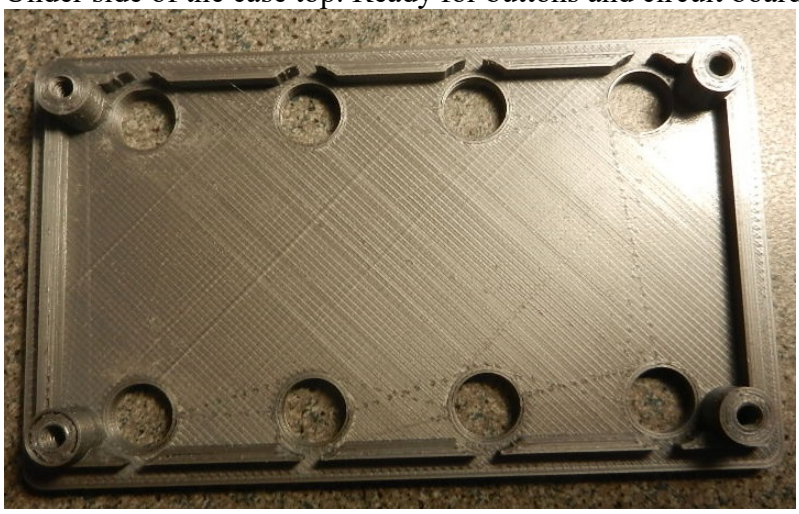
Here are some pictures of the case assembly.

Power plug assembly

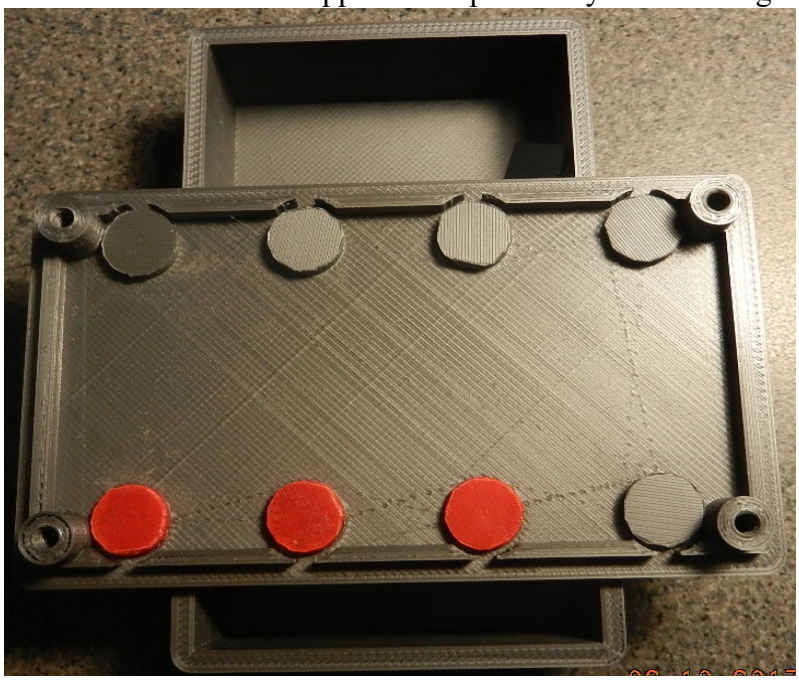
Case bottom with power plus assembly.



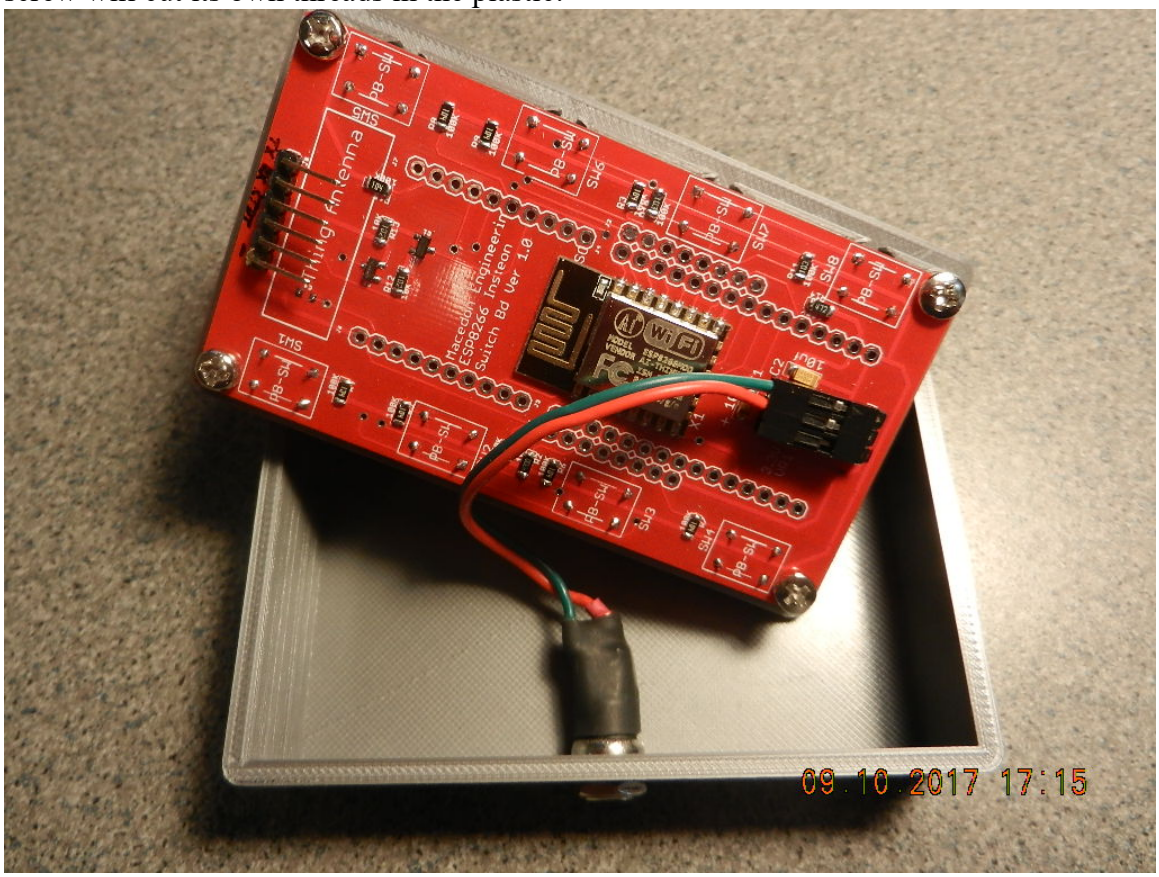
Under side of the case top. Ready for buttons and circuit board.



Case top with buttons inserted. Red buttons are in position for Sw1-Sw3. Notice how the case bottom is used to support the top so that you are not fighting with the buttons.



Screw down the circuit board. I used 6-32x1/4” screws. I tapped the holes in the bosses with a 6-32 tap. #4x1/4” or #6x1/4” round head wood screws could be used. A wood screw will cut its own threads in the plastic.




The completed project. The bullet buttons are used as a reference for finding the right button to push by feel.

While I was developing the code I had to search the Internet for tidbits on the Insteon protocol. I found that the Insteon Hub commands are not limited to just controlling Insteon devices. It is also possible to send commands to the Hub to control X10 devices. I was not interested in controlling X10 devices so I did not include any code for doing that. Insteon commands can also control an Insteon thermostat. I do not have an Insteon thermostat so I was not interested in controlling one. Like they say, these are exercises best left to the reader.

OpenSCAD is a free program that allows you to design 3D models using a simple programming metaphor.

Arduino IDE is a free program that allows you to write ‘C’ style code, compile and upload it to a variety of microcontrollers. It is based on the Programming paradigm of coding.