**Objective**

The objective of this tutorial is to get familiar with embedded programming with Arduino on an ESP8266 chip, the chip that’s become super popular among the maker community for its accessibility and ease of use in the IoT space. The tutorial also gets our hands dirty with Alexa using an unofficial “hack” to get Alexa to do our bidding in the home (this technique is not meant for use in production, only for at-home demonstration). Try this at home, but not at work.

The beauty of this technique is that we can use it in our own homes to make Alexa automate almost literally anything. As an added bonus we get an insight into embedded programming with Arduino, a skill that’s maybe not so common among mainstream programmers these days. Finally, we get to work with the popular ESP8266 chip, a favorite among do-it-yourselfers; it’s an amazing little chip with ability to run all sorts of things, and a built-in wifi chip which we will need for this project. It’s what will enable the Alexa device and the chip to communicate with one another directly.

Just some background on Alexa programming: The Alexa “skills” programming model works like this:

* You speak to your Alexa
* Alexa routes your speech all the way back to Amazon’s cloud
* The speech command is routed to an Alexa “skill” (a program that runs in Amazon’s cloud)

The Alexa “skill” takes over handling of the command; normally it results in a response being sent back to the Alexa device, causing it to say something to the user in response. In the case of Alexa IoT, the command gets routed to a “device shadow” on Amazon’s cloud, which in the end results in a response being sent to some other device in your home. We are bypassing all that with our hack.

Our hack is not really a secret. We are going to make our ESP8266 “emulate” a Wemo Belkin, a device that has a special license with Amazon allowing it to communicate directly with the Alexa device, bypassing all of that Amazon cloud communication described above. Pretending to be a Wemo, our ESP8266 enjoys the privilege of being able to receive commands directly from the Alexa.

**Basic plan:**

* Listen for the Alexa device sending out probes on the local wifi network for compatible devices, and response to these probes by saying “I’m a Wemo”.
* Once trusted by the Alexa, device, listen for further commands from said device. Handle them by sending IR codes through the IR transmitter, turning our TV on/off.

**Hardware:**

[insert pics – Alexa, ESP8266 chip, adapter, IR diode]

To complete this tutorial, you’re going to need to obtain some items on your own, all of which are easy to obtain.

* Any Alexa device. I’ve developed this tutorial with an Alexa Dot. Will the tutorial work with an Echo simulator? (<https://echosim.io/>). It should! (but I haven’t tested it). Give it a try if you’re feeling adventurous (or frugal). The Alexa device costs some pocket money, but use of the echosim is free.
* An ESP8266 chip. They cost about just a few USD at the time of this writing. <http://www.ebay.com/itm/ESP8266-ESP-12E-Serial-WIFI-Wireless-Transceiver-Wireless-Module-LWIP-AP-STA-/191607430420>
* An IR (infrared) diode. You’ll need to wire this to your ESP8266 chip, and this you’ll have to do yourself. For this project we need only the sending capabilities, we don’t care about IR receiving. Be sure to wire the diode to GND and output 0, for this tutorial to work. (If you do it in any other way, that’s fine, but you’ll have to also be responsible for modifying the tutorial code accordingly) This link should help you: [LINK]. Be cognizant that due to the numbering scheme used on the ESP8266, pin 0 may be labeled as “D3”.
* A serial adapter which on one side is USB (to plug into your dev computer), and the other side fits into the ESP8266 chip.
* Some wifi to connect to

**Software:**

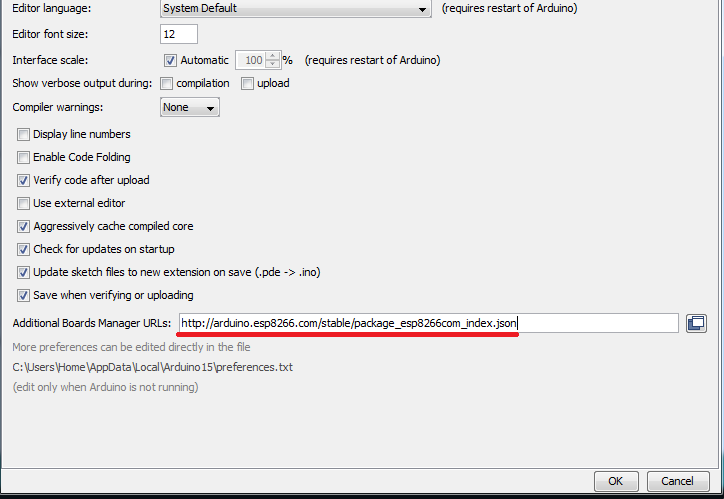
* Arduino IDE. There are versions for all major OSes, including Windows. This tutorial was developed on the Ubuntu version, but I’ve installed and used Arduino on Windows as well, no problems.
* The ESP8266 development library for Arduino. <https://github.com/esp8266/Arduino>
* Drivers: [LINKS]

**Setup:**

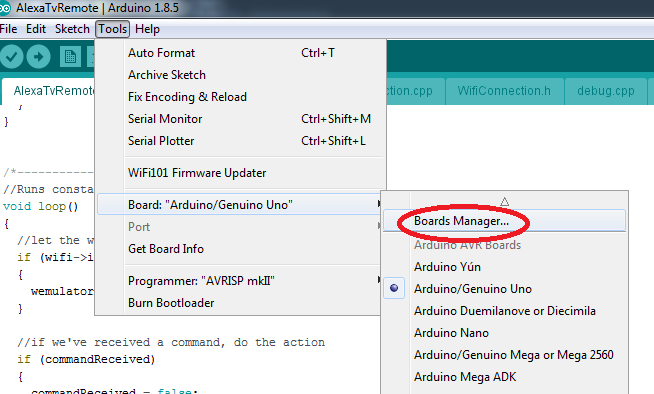
* Install Arduino IDE <https://www.arduino.cc/en/Guide/Windows>
* Install the ESP8266 library using Boards Manager [SHOW TUTORIAL]

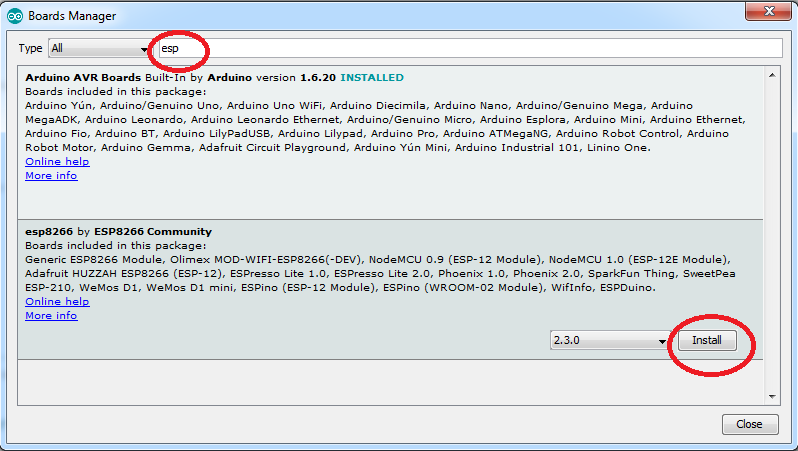
To install ESP8266 support:

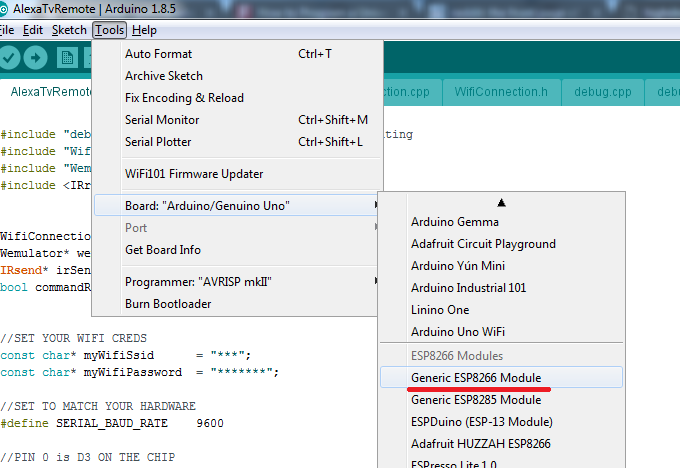
* In Arduino IDE, open File -> Preferences
* Enter this URL in “Additional Boards Manager URL”: <http://arduino.esp8266.com/stable/package_esp8266com_index.json>
* Click OK



* Go to Boards Manager (Tools -> Board: [current board] -> Boards Manager)
* In the ‘filter’ textbox, type “ESP8266”
* You should get an entry for “esp8266” now that you have the additional boards manager added. Choose it, and click “install”.
* Wait a while – it takes a while to download everything.
* Restart your Arduino IDE.
* Open Tools -> Board: -> this time scroll down to “Generic ESP8266 Module”, and select it.

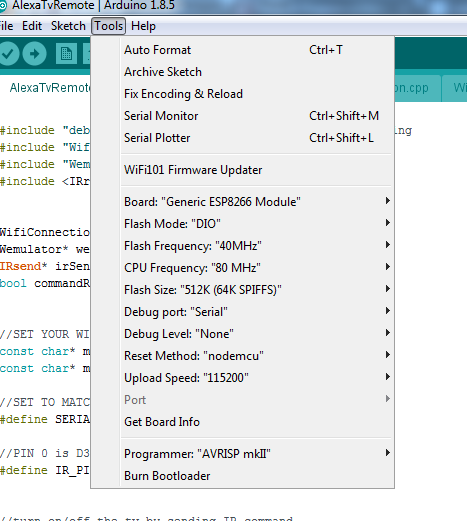






**Settings**

The image below shows typical settings, which work for me and my hardware, but may vary. You can try the settings below, but there’s a chance that you may have to adjust them based on your particular chip and adapter. Mine is nodemcu, for example, so I had to change reset method from “ck” (the default) to “nodemcu”. Also, set “debug port” to “serial” so that you can use the serial debugger.



**Prove your Setup:**

ESP8266 Hello World:

Arduino projects start with an .ino file. The .ino file defines two points of entry: setup and loop. For our “hello world”, we’re going to turn a little light on, on the ESP8266, just to verify that our code works.

//SET TO MATCH YOUR HARDWARE

#define SERIAL\_BAUD\_RATE 9600

/\*---------------------------------------\*/

//Runs once, when device is powered on or code has just been flashed

void setup()

{

//if set wrong, your serial debugger will not be readable

Serial.begin(SERIAL\_BAUD\_RATE);

}

/\*---------------------------------------\*/

//Runs constantly

void loop()

{

pinMode(3, 1);

delay(1000);

pinMode(3, 0);

delay(1000);

}

Flash the code to the chip [INSTRUCTIONS for flashing]

[TEST THAT THE SERIAL DEBUGGER IS WORKING]

Great, so that works; next we want to verify our IR output. Let’s send a signal through our IR transmitter, and verify that the signal’s coming through.

We’re going to make use of an existing IR library to help us. One of the great things about Arduino is how easy it is to snap libraries and modules in and out. Refreshing for a C++ framework.

<https://github.com/markszabo/IRremoteESP8266>

Just follow the instructions in that git repo’s README file, to install in Arduino.

This code just flashes the IR transmitter repeatedly. IR is invisible to the human eye, but there’s a pro-tip for testing it; run this code, verify (via the debugger) that it’s running on your chip, then open your mobile device’s camera. Look directly at the IR diode bulb *through your camera*. If it’s working, you should see the bulb visibly turning on and off. You can try this with any working remote control as well (e.g. a standard TV’s remote).

[CODE SAMPLE]

Begin the tutorial

If everything has worked so far, I think we can be satisfied that our basic equipment and setup and working, and we’re ready to begin the meat of the tutorial.

Connect to Wifi

First, we’re going to need to connect to the local wifi. The code below will attempt to connect to the Wifi, and reports success on connection (through the serial debugger). In the code sample, don’t forget to replace myWifiSsid’s value with the username of your wifi network, and replace myWifiPassword’s value with the correct password.

#include "debug.h" // Serial debugger printing

#include "WifiConnection.h" // Wifi connection // this file is part of my tutorial code

#include <IRremoteESP8266.h> // IR library

WifiConnection\* wifi; // wifi connection

IRsend\* irSend; // infrared sender

//SET YOUR WIFI CREDS

const char\* myWifiSsid = "\*\*\*";

const char\* myWifiPassword = "\*\*\*\*\*\*\*";

//SET TO MATCH YOUR HARDWARE

#define SERIAL\_BAUD\_RATE 9600

//PIN 0 is D3 ON THE CHIP

#define IR\_PIN 0

/\*---------------------------------------\*/

//Runs once, when device is powered on or code has just been flashed

void setup()

{

//if set wrong, your serial debugger will not be readable

Serial.begin(SERIAL\_BAUD\_RATE);

//initialize wifi connection

wifi = new WifiConnection(myWifiSsid, myWifiPassword);

wifi->begin();

//connect to wifi

if (wifi->connect())

{

debugPrint("Wifi Connected");

}

}

/\*---------------------------------------\*/

//Runs constantly

void loop()

{

}

Run the Wemo Server

Connected? Good. Now we’re getting to the meat of the project: the Wemo server. My own Wemo Emulator is included in the source files for this tutorial. Now, you can search google and find a simpler Wemo emulator. You can find one that’s written using less code, and which is easy to understand. By all means, feel free to examine, experiment, write your own, etc. That’s all part of making this tutorial your own. The reasoning behind mine is that it uses ESPAsyncTCP. Why is this good? Well, there are only so many servers (or devices) you can run on the ESP8266 using this method before it starts becoming unreliable. Unreliable in the sense that, the Alexa will start missing devices (not finding them), commands will get dropped, and performance becomes slow. I find that this number is *maximized* by use of the ESPAsyncTCP library. Without it I’ve found unreliability to creep in at around 10-12 devices; with it, I find that number ups to around 16. In case you’d like to expand this tutorial and explore the limits of what the chip can do, I’d recommend using my version. If you want to see a simpler version just for your own understanding, feel free to search “wemo emulator Arduino” on google; you should find a host of examples.

Now, we have to install the ESPAsyncTCP library. Install it just as we did the IR library; go to the git page and follow the instructions. Here’s the link:

<https://github.com/me-no-dev/ESPAsyncTCP>

This library’s included in my tutorial code as well.

#include "debug.h" // Serial debugger printing

#include "WifiConnection.h" // Wifi connection

#include "Wemulator.h" // Our Wemo emulator

#include <IRremoteESP8266.h> // IR library

WifiConnection\* wifi; // wifi connection

Wemulator\* wemulator; // wemo emulator

IRsend\* irSend; // infrared sender

//SET YOUR WIFI CREDS

const char\* myWifiSsid = "\*\*\*";

const char\* myWifiPassword = "\*\*\*\*\*\*\*";

//SET TO MATCH YOUR HARDWARE

#define SERIAL\_BAUD\_RATE 9600

//PIN 0 is D3 ON THE CHIP

#define IR\_PIN 0

/\*---------------------------------------\*/

//Runs once, when device is powered on or code has just been flashed

void setup()

{

//if set wrong, your serial debugger will not be readable

Serial.begin(SERIAL\_BAUD\_RATE);

//initialize wifi connection

wifi = new WifiConnection(myWifiSsid, myWifiPassword);

wifi->begin();

//initialize the IR

irSend = new IRsend(IR\_PIN);

irSend->begin();

//initialize wemo emulator

wemulator = new Wemulator();

//connect to wifi

if (wifi->connect())

{

//start the wemo emulator (it runs as a series of webservers)

wemulator->addCommand("tv");

wemulator->addCommand("television");

wemulator->addCommand("my tv");

wemulator->addCommand("my television");

//set the event handler for when a voice command is received

wemulator->onMessage([](unsigned char device\_id, const char\* device\_name, bool state)

{

debugPrintln("command received");

});

}

}

/\*---------------------------------------\*/

//Runs constantly

void loop()

{

//let the wemulator listen for voice commands

if (wifi->isConnected)

{

wemulator->handle();

}

}

Test by running it with Alexa. This tutorial assumes that your Alexa device is set up and installed in your home.

Test discovery:

Say to Alexa “Alexa, discover devices”.

This will cause Alexa to broadcast a UDP request on your local wifi network, scanning for Wemos and other compatible devices. This request should be received in the call to

wemulator->handle();

in the loop() function. This in turn routes it to Wemulator’s handleUDPPacket(\*) method. A response is sent out in the nextUDPResponse() method. Note the content of that response:

const char UDP\_TEMPLATE[] PROGMEM =

"HTTP/1.1 200 OK\r\n"

"CACHE-CONTROL: max-age=86400\r\n"

"DATE: Sun, 20 Nov 2016 00:00:00 GMT\r\n"

"EXT:\r\n"

"LOCATION: http://%s:%d/setup.xml\r\n"

"OPT: \"http://schemas.upnp.org/upnp/1/0/\"; ns=01\r\n"

"01-NLS: %s\r\n"

"SERVER: Unspecified, UPnP/1.0, Unspecified\r\n"

"ST: urn:Belkin:device:\*\*\r\n"

"USN: uuid:Socket-1\_0-%s::urn:Belkin:device:\*\*\r\n\r\n";

This is the code that tells Alexa “I’m a Wemo (Belkin), how can I help you?” Once Alexa receives this response, it knows and remembers that future smart-home commands may be routed to this device.

In the setup() function, note the following line:

wemulator->onMessage([](unsigned char device\_id, const char \* device\_name, bool state)

This is the callback where we will capture commands from Alexa. Once we’ve captured a command from Alexa, we can do what we like with it. In the lines before that, we’ve set up possible commands that can be used, with these lines of code:

wemulator->addCommand("tv");

wemulator->addCommand("television");

wemulator->addCommand("my tv");

wemulator->addCommand("my television");

So these are 4 separate “servers” or listeners we’re running on the chip. This sets up the ability to say to Alexa any of the following commands:

Alexa, turn on tv

Alexa, turn off tv

Alexa, turn on television

Alexa, turn off television

Alexa, turn on my tv

Alexa, turn off my tv

Alexa, turn on my television

Alexa, turn off my television

… and this is how we’ll test it. We expect that saying any of those commands should wake up our code, enter that callback, where we can do what we like with it.

**Add the IR Command**

Now that we’re receiving the command, it’s time to handle it by… turning on/off our TV. My TV is an LG, so I looked up the appropriate sequence for turning on/off, and sent that through our IR library’s sendLG function. IR encoding/decoding is a separate subject in itself, wherein a message is encoded in the modulation of a signal; it’s a specification of very precise timings, marks, and spaces. Each manufacturer tends to use its own proprietary protocol for commands, and with different timings; it’s quite interesting, and you can dig deeper by looking into the source code of that IR library, googling, etc. But to our convenience, the details of all that are taken care of for us by our IR library.

Your TV’s not an LG? Just google the correct code. Here’s the one for Samsung TVs: [GET CODE]

If you want to get really do-it-yourself, you can set up an IR receiver, point your remote (or any IR transmitter) at it, and decode the codes that it’s sending; that’s a different tutorial though.

**End to End Test**

* Place your Alexa anywhere it can hear you.
* Place your ESP8266 with attached IR diode, within remote-control range of the TV.
* Say “Alexa, discover devices”. Wait for it to report success (it should have discovered at least one device)
* Say “Alexa, turn on my TV” or “Alexa, turn off my TV”.

Alexa should understand your command (as a smarthome command, not directed to a specific skill), search for a local device to handle it, and send the command to the device (your ESP8266). Your device should receive it, and send the remote control command to the TV. You can view your diode through a mobile phone camera to ensure that it’s emitting.

Since the IR code to turn a TV off is the same as the code to turn it on, it doesn’t matter whether you give the command to turn “on” or “off”. It’s the same code, and it toggles the state. If the TV’s off it should turn on, and if on it should turn off.

**Troubleshooting**

**Are you connected to the wifi? [ADD STEPS]**

Did you enter the correct username/password into the correct variable values? Are you getting a failure msg when connecting to the wifi?

**Is your device being discovered by the Alexa? [ADD STEPS]**

**Is your device receiving the command ? [ADD STEPS]**

When you issue the command “Alexa, turn my TV on”, execution should be entering your wemulator->onMessage handler (in the .ino file). If you haven’t done so, try outputting some debug message in there to ensure that that’s firing when you issue your command. Also, try ensuring that Alexa knows about your device, by saying “Alexa, discover devices” *before* issuing your command

**Is the IR diode emitting? [ADD STEPS]**

As described before, when you think your device should be emitting, point your mobile phone’s camera at it and look at the diode through the camera. Though in real life you can’t see anything, through the camera it should appear as a normal light lighting up & flashing. If you see this, then it’s emitting… something.

**Is the IR signal reversed? [ADD STEPS]**

Your IR diode may be wired in such a way that the signal is essentially reversed. Please bear with me on my explanation, as I’m not an electronics or wiring guy, but

**Do you have the right remote control code and command? [ADD STEPS]**

If everything else seems to be going ok, but the TV’s just not obeying, it’s probably quite likely that something is wrong with the IR code. Try different function calls on that IR library interface (e.g. sendLG, sendPanasonic, sendSharp, etc.), or make sure that the one you’re using matches your hardware. It’s very unlikely that your TV’s hardware is not supported by that library, but I guess it’s technically possible.

Make sure that the code you’re sending is the right one for your hardware. You might have to do some digging on google to find the right one. If all else fails, there’s always the option of detecting the code that emits from your working remote, when you press the Power button – but that’s a different project.