Exploring the Espressif ESP8266 and ESP32 with Micropython

Science Oxford Python in Education Conference 2024

John Gouk (johngouk@gmail.com)

# Introduction

The ESP32 and ESP8266\* chips from Espressif are WiFi-enabled, and in the case of ESP32, Bluetooth-enabled SOCs ([Systems on a chip](https://en.wikipedia.org/wiki/System_on_a_chip)).

* They come with enough RAM and Flash (non-volatile RAM) to run the [Micropython](https://micropython.org/) port of Python, which is more than enough to do some amazing things, especially when they are connected to the Internet and can send and receive data.
* They are ideal components to enable the Internet of Things - their GPIO ports allow connection of pretty much all the standard sensors and displays, and they are powerful enough to support a simple webserver for configuration or data display.

I have instrumented my Daikin Heat Pump with an ESP32, and it has provided great feedback on its configuration, as well as heat generated vs. energy consumed.

\* I use the terms “ESP32” and “ESP8266” fairly loosely in this discussion; Espressif supports a myriad of versions of each family e.g. the latest ESP8xxx is ESP8684, but the general points apply; also, stocks of chips of all eras are available

**Note: There are lots of different SoCs available, RP2040, RPi Pico, etc. I’m talking about ESPs because I’ve used them!**

# Choice of IDE

There are many IDEs ([Integrated Development Environment](https://en.wikipedia.org/wiki/Integrated_development_environment)) that enable development in Python on SOCs, like ESPs, microbits and so on. If you have a favourite Python IDE, like Mu, it may well already support SOC development – Mu does, for example. I am currently using [Thonny](https://thonny.org/), largely because it supports

* Viewing/manipulating SOC and computer filesystems at the same time, including easy uploads/downloads to/from SOC/ESP
* Editing multiple files simultaneously, on both computer and SOC (super handy!!)
* A version of Python REPL, which is great – try out the commands on ESP32 instantly!

A screenshot of a computer

Description automatically generated

The screenshot shows the main features:

* File system windows on the LH side
* Editing multiple files in central window
* Shell window at the bottom
* Help and Assistant windows on the RH side – you can close these to increase useful windows, they are really only useful with CPython, which Thonny also supports well
* At the very bottom RH side, you can see the current environment choice and USB connection to the ESP board

# ESP Development Boards

Although ESPs are available as bare chips, this is a pretty useless object for mere mortals. You’re better to get a [development board](https://en.wikipedia.org/wiki/Single-board_microcontroller), which includes

* a serial-USB interface to communicate with the chip (very handy!)
* easily accessible sockets or solder points exposing the various hardware ports/pins
* a power supply point, usually combined with the USB socket to take 5V from the USB

These are easily found on [Amazon](https://www.amazon.co.uk/s?k=esp32+development+board) (if you don’t mind Jeff’s gross self-indulgence at your expense), [AliExpress](https://www.aliexpress.com/w/wholesale-esp32-development-board.html) (slow but super cheap), [EBay](https://www.ebay.co.uk/sch/i.html?_nkw=esp32+development+board) and other outlets. I’ve just done a simple search for “ESP32 development board” on these sites to get these links.

# ESP8266 vs ESP32

The [ESP8266EX](https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf) (or its [predecessors](https://en.wikipedia.org/wiki/ESP8266)) was the original Espressif WiFi-enabled SoC introduced in 2013. It has a single core, 160Kb of RAM (around 50Kb left when wifi-connected in Station mode), maybe 4Mb of Flash, 1.5 UARTs for serial comms, and 17 GPIO (General Purpose Input Output) pins.

Its big brother, the [ESP32](https://en.wikipedia.org/wiki/ESP32) family, has two cores, 520Kb of on-chip RAM, a variable amount of external Flash (16Mb anyone??), 2 UARTs, and 32 GPIO pins. It also has a low-power coprocessor, which allows for a very low power sleep mode – handy if you’re a battery powered device! It also has SPI and I2C in hardware, which is very handy for connecting smart sensors…

If you don’t have a power supply problem (like not enough!), I’d go with the ESP32, on the basis that you can run much bigger Python programs on it, and it doesn’t cost any more in the kind of usable packages available.

# Connecting ESPs to your computer

For the physical connection, this is in principle pretty simple – use an appropriate USB data cable to connect the dev board USB socket to your computer USB. The only potential issue is USB drivers – some USB UARTs (the bit that allows the dev board to talk USB) require updated or different drivers, especially the WCH family. If this happens, check out their [driver download page](https://www.wch-ic.com/downloads/category/30.html) – you are looking for something like CHxxxSER.ZIP[\_MAC|\_LINUX|.[EXE|ZIP], depending on your OS and the chip type. Give it a go before you try to fix it!

Now you need to connect your IDE to the new USB/serial port. Open the IDE of choice – my example will be Thonny, but Mu and others work the same kind of way. Use Thonny/Preferences and the Interpreter tab – see below. Select the ESP32 or ESP8266 interpreter – Thonny starts with CPython if you want to try it with that, running some scripts on your computer first. Select the port the ESP is connected to – that’s the hardest part sometimes! Look in Device Manager USB on Windows to see what’s connected, if nothing like a serial device, then maybe you need additional drivers. A screenshot of a computer

Description automatically generated

Since this is the first time, click on “Install or update MicroPython (esptool)”. This will install the MP interpreter on the ESP, which is required to make this all work. Use the pop-up window to configure the right device for the MP version:

A screenshot of a computer

Description automatically generated

Select the same port as the previous panel, the ESPxx(xx) you are using, and if an ESP32, the correct variant. Use the most recent stable version (1.23 at doc time). Press Install… a progress bar and status messages appear to the left as the Espressif Python-based esptool writes the new code to the ESP. Done! You’re ready to go.

# MicroPython Docs

You’ll need the documentation at https://docs.micropython.org/en/latest/index.html . You’ll need to read the standard Python lib info, the MP lib info, and the ESP32/ESP8266 port-specific info to make progress beyond the simplest things, especially if you want to use chip-specific features. Although MP is cool, the limited space on the chips means lots of the more esoteric Python libraries didn’t make the cut… but that doesn’t prevent lots of interesting possibilities.

# Demos

I’ve set up some sample and demo code in <https://github.com/johngouk/scienceoxfordpython>, which I’ll be talking through on the walk-up session. The items are intended to show some of the most interesting, useful or tricky features of the ESP I’ve encountered recently – I’m sure there are many more!

## WiFi Connections

Connecting to wifi with an ESP is incredibly simple, as MP provides a library network that does it all. simpleConnect.py illustrates this (replace ssid and password with appropriate values):

import network

# Connection

w = network.WLAN(network.STA\_IF)

w.active(True)

w.connect("ssid", "password")

while not (w.isconnected()):

pass

ipaddr = w.ifconfig()[0]

print('Connected! IP: ' + ipaddr) # You’re connected!

print('Hostname:', network.hostname())

It’s as simple as that. The while loop is to allow the ESP to connect – this can take some seconds. Now you can ping the ESP from a terminal prompt on your computer at the IP address or using the hostname + “.local” – the ESP RTOS and Python implement the mDNS functions when you connect.

I’ve provided a simple class that does this, and also sets the RealTime Clock (RTC) on an ESP32.

## Logging

I don’t know about you, but I like programmes to tell me what’s happening, at an appropriate level for the current stage of development – super verbose when you start, dire errors or warnings only when you’re in production.