Micropython

Christo Goosen

Micropython

Python for microcontrollers and constrained environments

About Me

Christo Goosen

Work

Work for VOSS Solutions

Python Dev and QA in Performance testing team.

VOSS Solutions

VOSS Solutions is a British founded multinational unified communications and collaboration corporation, presently headquartered in Richardson, Texas, United States, that provides service delivery and management software.

OWASP Cape Town

- https://www.owasp.org/index.php/Cape_Town
- http://www.meetup.com/OWASP-Cape-Town-Chapter-Meetup/

Contact Details:

Email:

- **christo@christogoosen.co.za**
- **christo.goosen@owasp.org**

Github: https://github.com/c-goosen

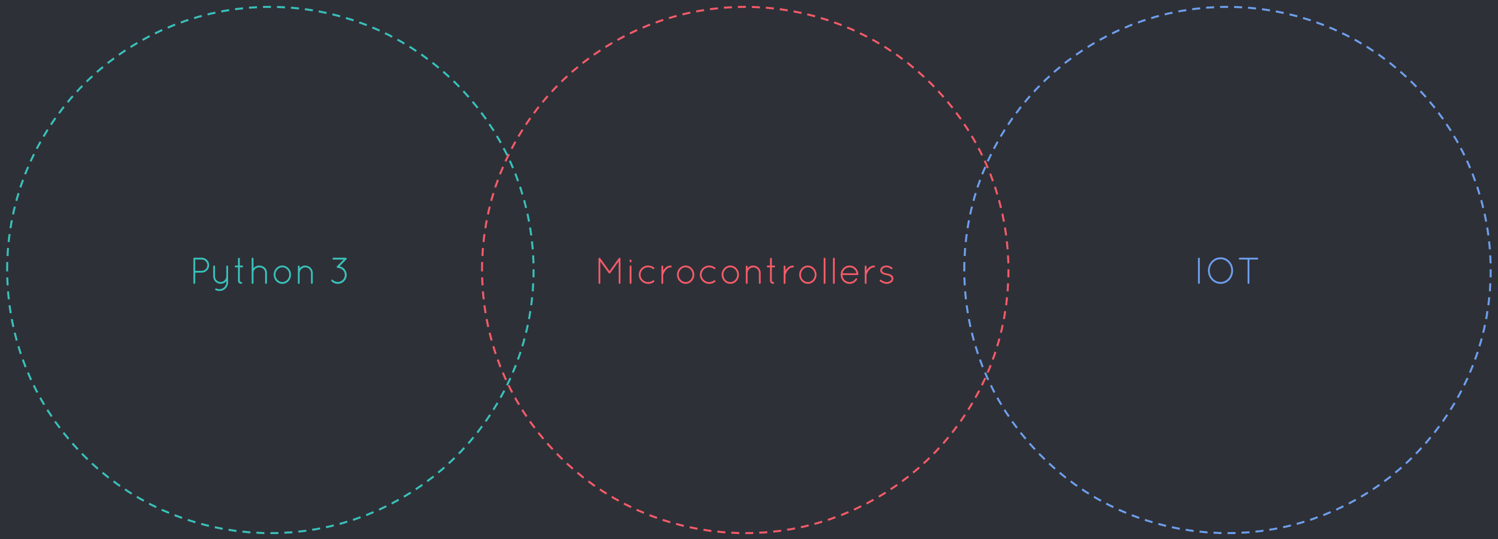
Website: http://www.christogoosen.co.za/

@OWASP_CPT

Outline

- Why Micropython?
- What is Micropython?
- Micropython Hardware API
- Micropython vs Arduino
- Micropython Libraries
- Micropython Framework
- Building Micropython
- Micropython for Unix
- Pyboard
- ESP8266
- Q&A

Why Micropython?



Why Micropython?

Why python(micro)?

- High-level language
- Fast development or prototyping
- Big open source community
- Easy to learn and read
- Powerful features
- It's not Javascript/Lua/C

Why not CPython/PyPy, etc.?

- RAM
- Microcontrollers need optimized code for constrained resources
- CPython Integer operations, method calls and for loops expensive in RAM
- Some CPython libraries cache disk data in RAM

List of differences between micropython and Cpython:
https://github.com/micropython/micropython/wiki/Differences

Why Micropython?

Differences between micropython and CPython

- “Unlike CPython3, which uses reference-counting, MicroPython uses garbage collection as the primary means of memory management.”
- “MicroPython does not ship with an extensive standard library of modules. It's not possible and does not make sense, to provide the complete CPython3 library. Many modules are not usable or useful in the context of embedded systems, and there is not enough memory to deploy the entire library on small devices”
- MicroPython does not implement complete CPython object data model, but only a subset of it. Advanced usages of multiple inheritance, __new__ method may not work. Method resolution order is different (#525). Metaclasses are not supported (at least yet).

List of differences between micropython and Cpython:
https://github.com/micropython/micropython/wiki/Differences

So what is micropython?

“MicroPython is a lean and efficient implementation of the Python 3 programming language that includes a small subset of the Python standard library and is optimised to run on microcontrollers and in constrained environments. The MicroPython board is a small electronic circuit board that runs MicroPython on the bare metal, and gives you a low-level Python operating system that can be used to control all kinds of electronic projects. ”

Origin

Started as a kickstarter campaign (Link) with a focus on a “Micropython” board that was shipped on the successful backing of the project. The successful kickstarter campaign and open source community following widened the support of Micropython to more devices and applications beyond the pyboard.

Features

- REPL (MicroPython Interactive Interpreter Mode)
- Autocompletion
- Auto-indentation
- Paste Mode
- WEBREPL
- Hardware API
- Execute Assembler

```
@micropython.asm_thumb
def led_on():
    movwt(r0, stm.GPIOA)
    movw(r1, 1 << 13)
    strh(r1, [r0, stm.GPIO_BSRRL])
```

- The special variable _ (underscore)
  - When you use the REPL, you may perform computations and see the results. MicroPython stores the results of the previous statement in the variable _ (underscore). So you can use the underscore to save the result in a variable. For example:

Hardware API

https://github.com/micropython/micropython/wiki/Hardware-API

The API should be Pythonic, obvious and relatively minimal. There should be a close mapping from functions to hardware and there should be as little magic as possible. The API should be as consistent across peripherals (Pin, UART, I2C, ADC, etc) as possible. There should usually be only one way to do something. A method name should do exactly what it says and no more (i.e. it shouldn't be heavily overloaded).

Import machine

From machine import Pin, I2C, UART, ADC

Devices supported: Teensy, micro:bit, wiPy, bare-ARM, cc3200, esp8266

How does MicroPython compare to Arduino?

1. First is that Arduino is an entire 'ecosystem' with the Arduino IDE (i.e. the desktop application you use to write and upload sketches), the Arduino programming language (based on C/C++), and Arduino hardware like the Arduino Uno R3 board.
2. MicroPython is only a programming language interpreter and does not include an editor. Some MicroPython boards support a web-based code prompt/editor.
3. The second important difference is that the MicroPython language is interpreted instead of being compiled.
4. You can even write and run interpreted code like MicroPython directly on a board without any compiling or uploading--something that's impossible with Arduino!
5. One disadvantage of interpreted code and MicroPython vs. Arduino is that there's less performance and sometimes more memory usage when interpreting code.

Source: https://learn.adafruit.com/micropython-basics-what-is-micropython/overview

Micropython Libraries

github.com/micropython/micropython-lib

- Upip (micro pip)
- uasyncio
- Umqtt.simple & umqtt.robust
- Urllib & urllib.urequest
- Uuid
- Multiprocessing
- Itertools
- Io
- Gzip & tar
- JSON
- Argparse…..and many more

Micropython Specific Libraries

https://github.com/micropython/micropython/wiki/Hardware-API

- Machine (Accessing hardware specifics of board)
- Micropython (Internals of micropython)
- network (Networking around WLAN)
- Uctypes (access binary data in a structured way)

Framework - Picoweb

https://github.com/pfalcon/picoweb/

“Intro:

picoweb is "micro" web micro-framework (thus, "pico-framework") for radically unbloated web applications using radically unbloated Python implementation, MicroPython, https://github.com/micropython/micropython .

Features:

* Asynchronous from the start, using unbloated asyncio-like library
for MicroPython (uasyncio).
* Modest memory usage (I would say radically small memory usage, but
so far, trivial web app requires 64K (yes, kilobytes) of heap, which
is much more than I expected).
* Has API affinity with well-known Python web micro-framework(s),
thus it should be easy start if you have experience with that, and
existing applications can be potentially ported, instead of requiring
complete rewrite.

“

Has support for running on CPython 3.4.2

Picoweb

“picoweb API is roughly based on APIs of other well-known Python web frameworks. The strongest affinity is Flask, http://flask.pocoo.org, as arguably the most popular micro-framework. Some features are also based on Bottle and Django. Note that this does not mean particular "compatibility" with Flask, Bottle, or Django: most existing web frameworks are synchronous (and threaded), while picoweb is async framework, so its architecture is quite different

“

Building Micropython

Supported platforms:

- Microcontrollers (Bare-Metal, without an OS)
- UNIX
  - DEB-based systems
  - FreeBSD-based systems
  - RPM-based systems
  - Pacman-based systems
  - Gentoo-based systems
  - Mac systems

Still attempting to get a build system working on alpine Linux due to a Ubuntu docker container bloating to 4.5GB.

Guide to building:
https://github.com/micropython/micropython/wiki/Getting-Start
ed

Building Micropython

Building on Ubuntu (Unix port of micropython)

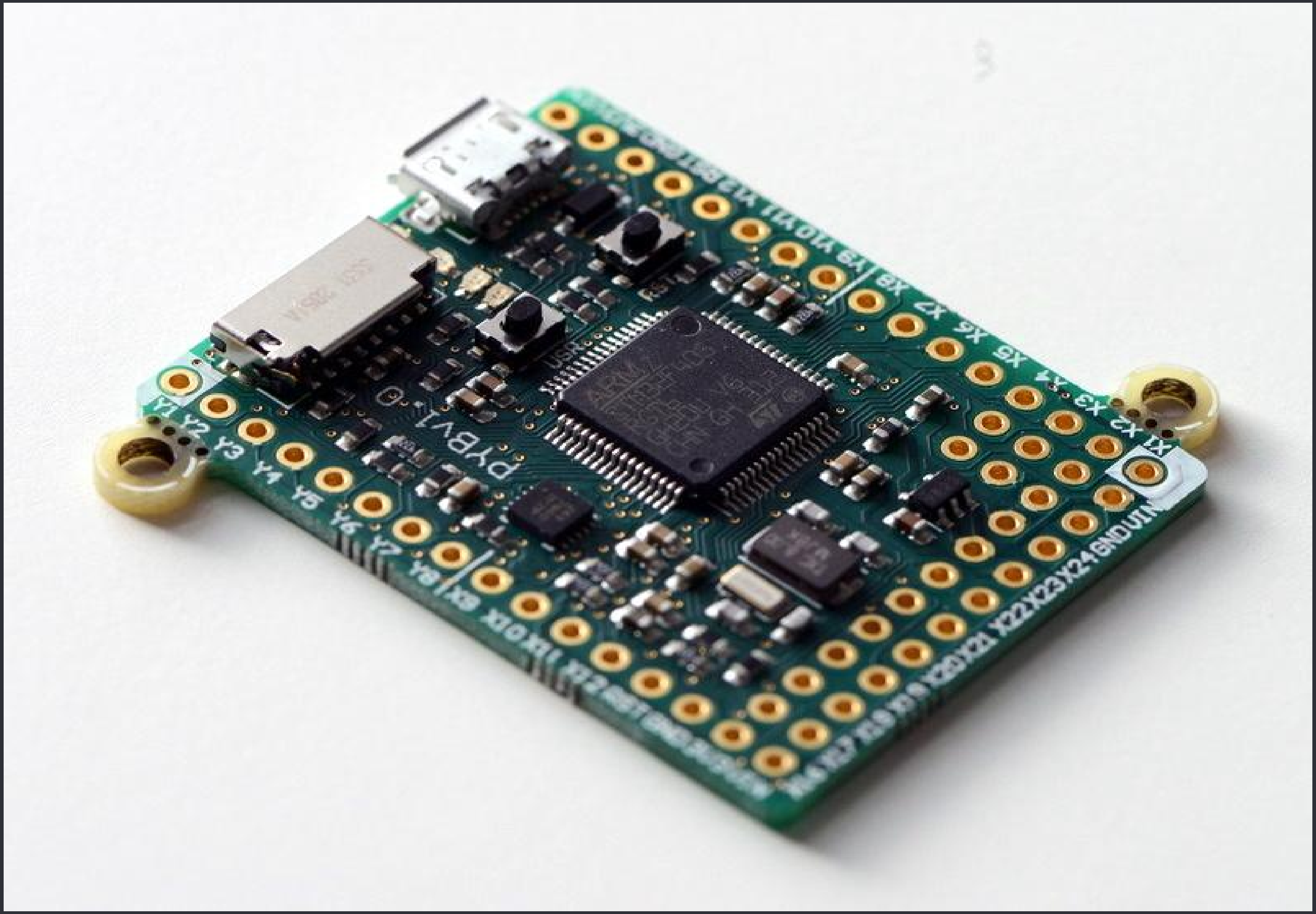
```
sudo apt-get install build-essential libreadline-dev libffi-dev git

git clone --recurse-submodules https://github.com/micropython/micropython.git

cd ./micropython/unix

make axtls

make
```

Building Micropython

Check if build was successfull:

```
root@ab58c2abf8f9:/home/micropy/micropython/unix#
./micropython
MicroPython v1.8.4-80-g0c595fa on 2016-10-01; linux version
Use Ctrl-D to exit, Ctrl-E for paste mode
>>>
```

Pyboard


PYBv1.0
USR
RST
Y1 Y2 Y3 Y4 Y5 Y6 Y7 Y8
X9 X10 X11 X12
X1 X2 X3 X4 X5 X6 X7 X8
Y9 Y10 Y11 Y12
X22 X23 X24 GND VIN
X20 X21

Pyboard

Easy programming with Pyboard

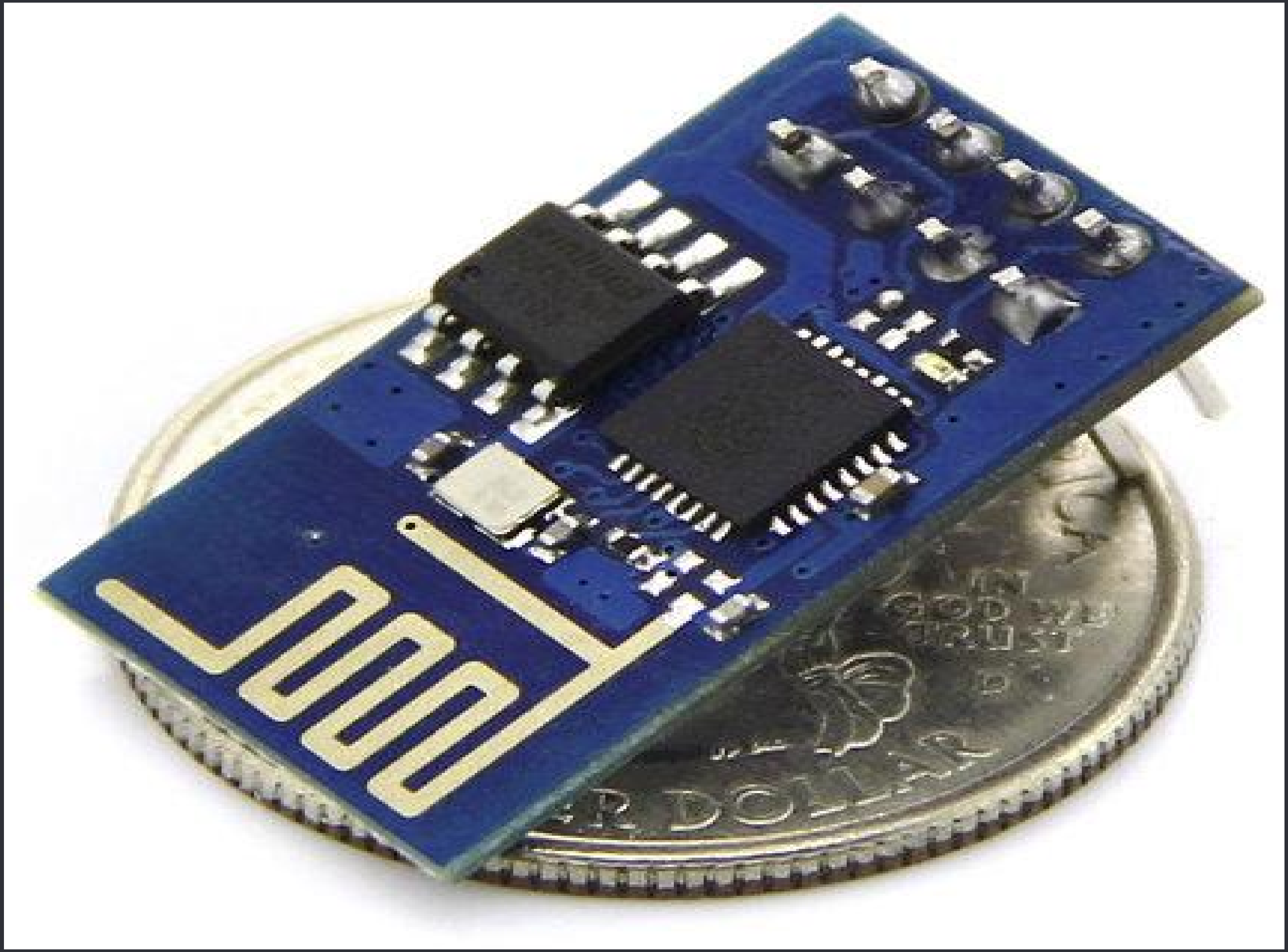
```
~import pyb
~ accel = pyb.Accel()
light = pyb.LED(3)
SENSITIVITY = 3

while True:
    x = accel.x()
    if abs(x) > SENSITIVITY:
        light.on()
    else:
        light.off()

    pyb.delay(100)
```

Pyboard

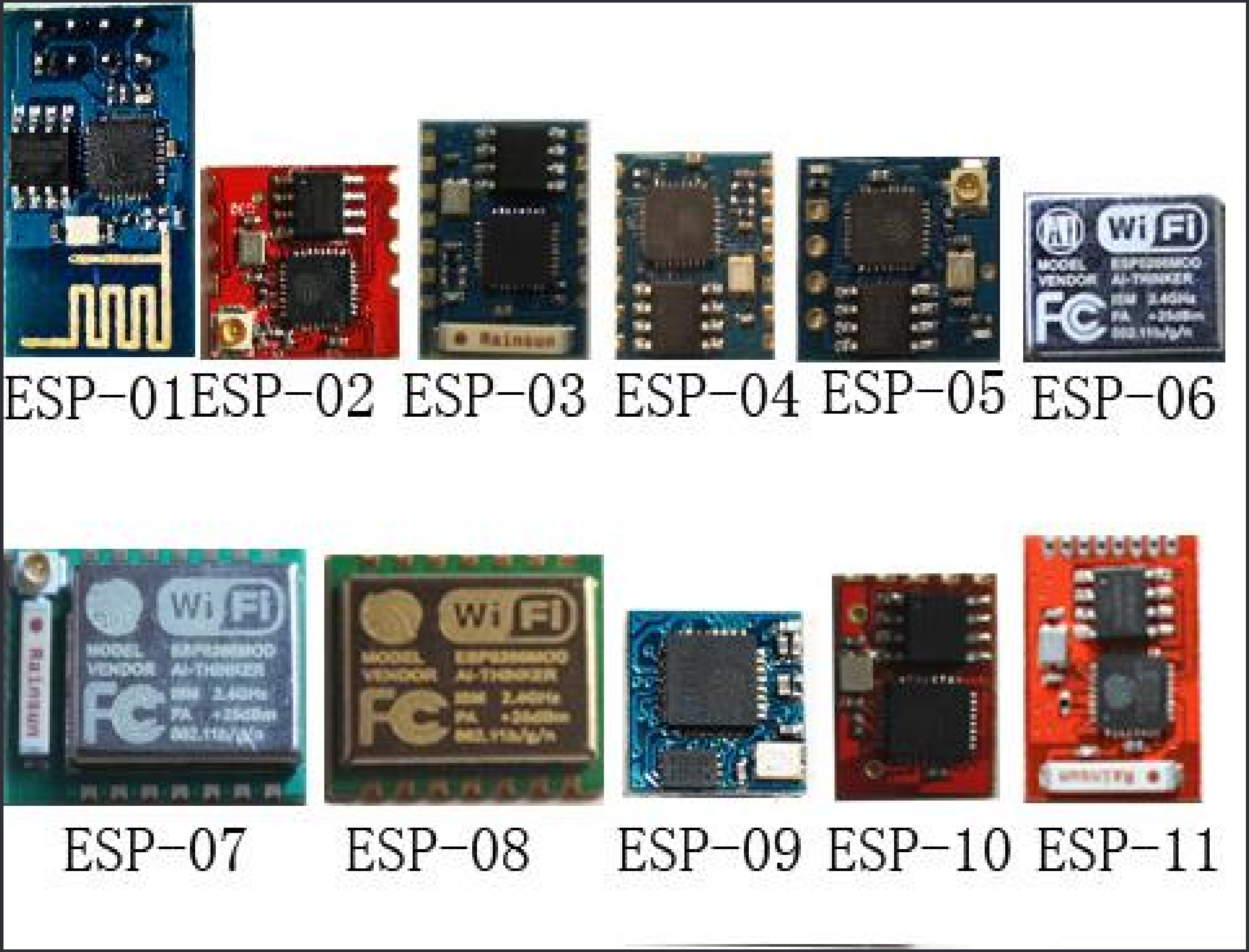
Main features of the hardware:

- STM32F411RE microcontroller
- 96 MHz Cortex M4 CPU with hardware floating point
- 512KiB flash ROM and 128KiB RAM
- Micro USB connector for power and serial communication
- Micro SD card slot, supporting standard and high capacity SD cards
- 24 GPIO on left and right edges and 5 GPIO on bottom row, plus LED and switch GPIO available on bottom row
- 1x 12-bit analog to digital converter, available on 16 pins, 4 with analog ground shielding
- 4 LEDs (red, green, yellow and blue)
- 1 reset and 1 user switch
- On-board 3.3V LDO voltage regulator, capable of supplying up to 250mA, input voltage range 3.6V to 16V
- DFU bootloader in ROM for easy upgrading of firmware

ESP8266



ESP8266

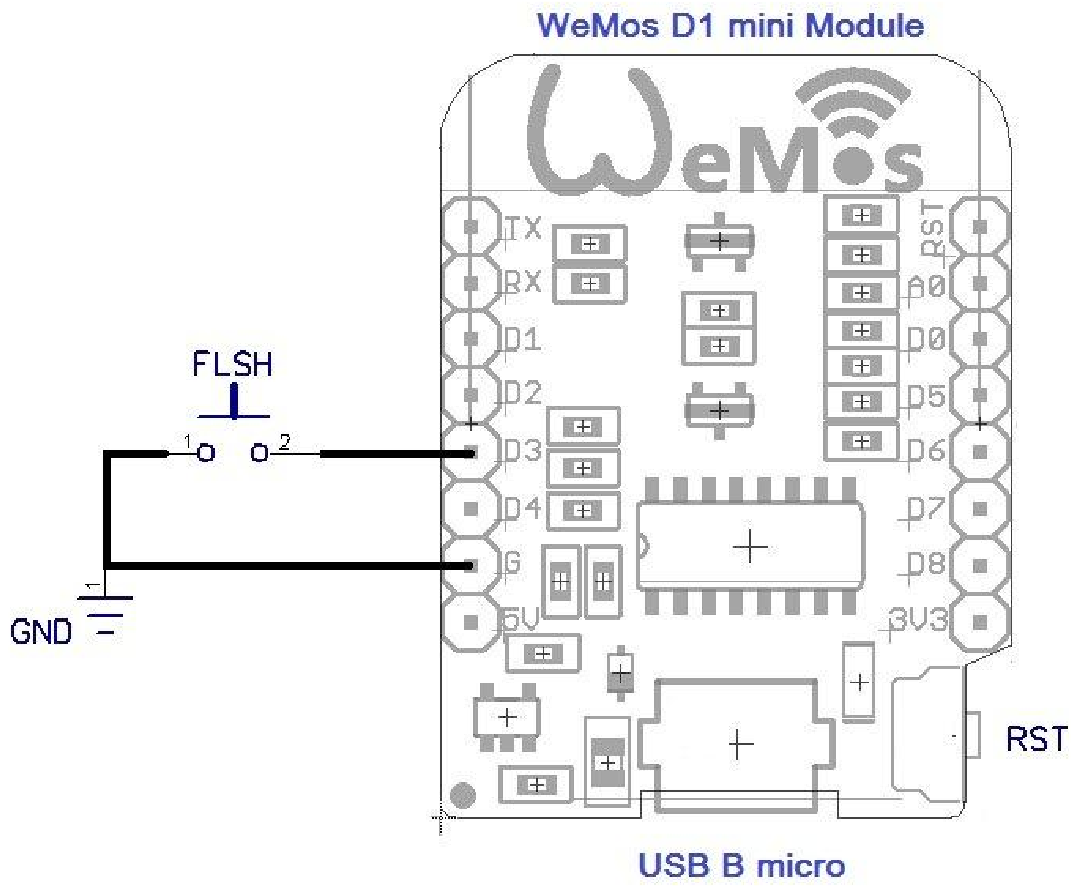

ESP-01
ESP-02
ESP-03
ESP-04
ESP-05
ESP-06
ESP-07
ESP-08
ESP-09
ESP-10
ESP-11

ESP8266

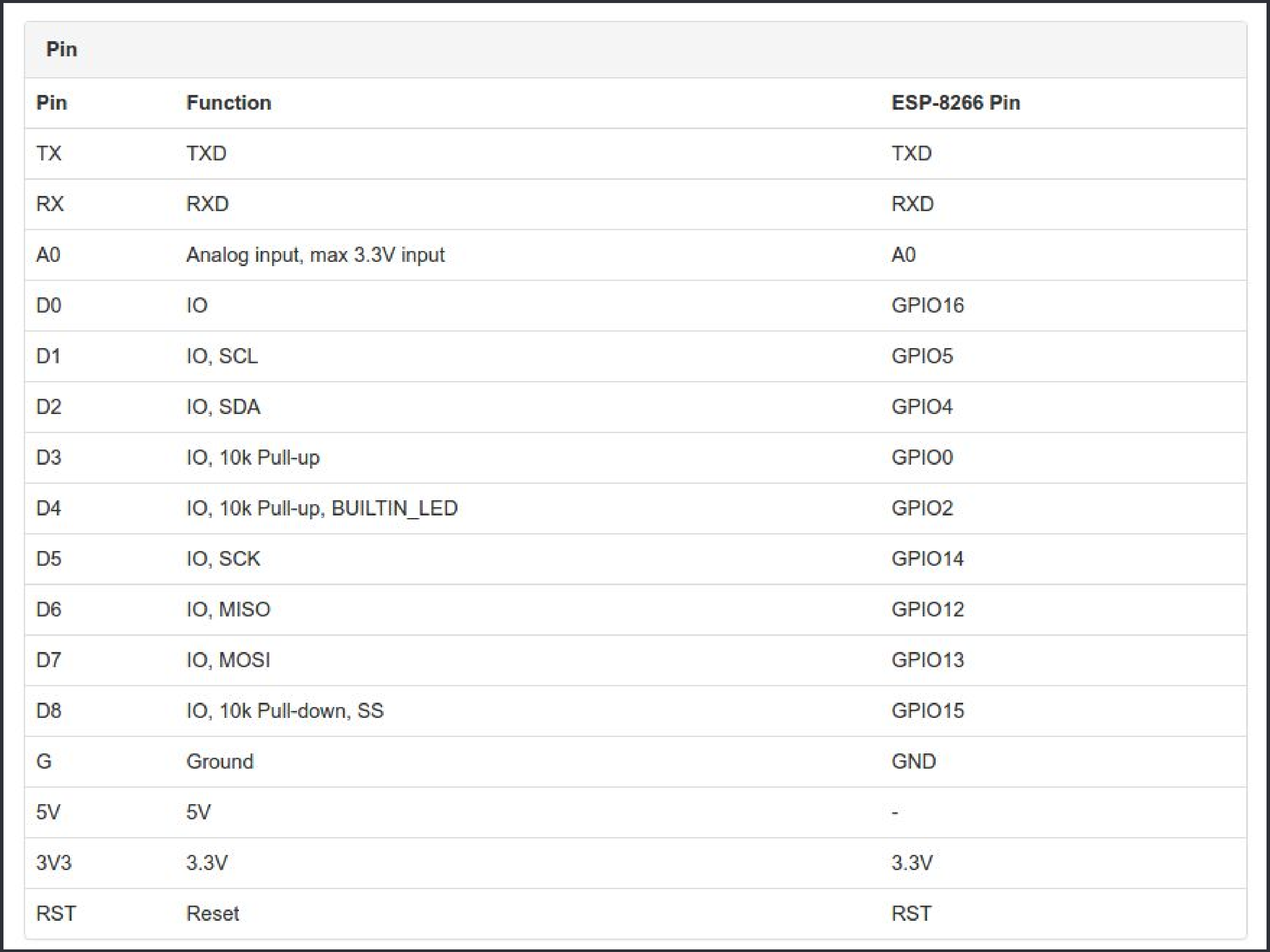
Main features of the hardware:

- 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, I C,
- I S 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

- D1 Mini - based on ESP-8266EX.



ESP8266 - D1 Mini



Pin

Pin	Function	ESP-8266 Pin
TX	TXD	TXD
RX	RXD	RXD
A0	Analog input, max 3.3V input	A0
D0	IO	GPIO16
D1	IO, SCL	GPIO5
D2	IO, SDA	GPIO4
D3	IO, 10k Pull-up	GPIO0
D4	IO, 10k Pull-up, BUILTIN_LED	GPIO2
D5	IO, SCK	GPIO14
D6	IO, MISO	GPIO12
D7	IO, MOSI	GPIO13
D8	IO, 10k Pull-down, SS	GPIO15
G	Ground	GND
5V	5V	-
3V3	3.3V	3.3V
RST	Reset	RST

ESP8266

Micropython:

- WARNING: The port is experimental and many APIs are subject to change.

- Supported features include:
    - REPL (Python prompt) over UART0.
    - Garbage collector, exceptions.
    - Unicode support.
    - Builtin modules: gc, array, collections, io, struct, sys, esp, network, many more.
    - Arbitrary-precision long integers and 30-bit precision floats.
    - WiFi support.

ESP8266

Micropython:

Features continued:

- Sockets using modlwip.
- GPIO and bit-banging I2C, SPI support.
- 1-Wire and WS2812 (aka Neopixel) protocols support.
- Internal filesystem using the flash.
- WebREPL over WiFi from a browser (clients at https://github.com/micropython/webrepl).
- Modules for HTTP, MQTT, many other formats and protocols via https://github.com/micropython/micropython-lib .
-
- Work-in-progress documentation is available at http://docs.micropython.org/en/latest/esp8266/ .

ESP8266

Building micropython for ESP8266

Step 1:

Download: https://github.com/pfalcon/esp-open-sdk.git

```
sudo apt-get install make unrar-free autoconf automake libtool gcc g++ gperf \
    flex bison texinfo gawk ncurses-dev libexpat-dev python-dev python python-serial \
    sed git unzip bash help2man wget bzip2
```

```
git clone --recursive https://github.com/pfalcon/esp-open-sdk.git /home/micropy/esp-open-sdk
```

ESP8266

Building micropython for ESP8266

Step 2:

```
~ cd esp-open-sdk
~ git submodule update --init

~ make   (This will take long, have some coffee)
```

Add to path
/home/micropy/esp-open-sdk/xtensa-lx106-elf/bin:$PATH

```
~ cd micropython
~ make -C mpy-cross
```

ESP8266

Building micropython for ESP8266

Step 3:

~ git clone --recursive **https://github.com/micropython/micropython.git** (Skip if this step already done)

~ cd micropython/esp8266

~ make axtls

~ make

ESP8266

Building micropython for ESP8266

Step 4:

To flash the firmware you can do use make

```
~ make PORT=/dev/ttyUSB0 deploy
```

Or you can use esptool (pip install esptool)

```
~ sudo esptool.py --port /dev/ttyUSB0 write_flash
--flash_mode qio --flash_size 32m 0x0
firmware-combined.bin --verify
```

Firmware is located in micropython/esp8266/build/

Micropython & ESP8266 specifics

- Don’t have access to the Pyb API
- Less modules
- machine.freq(160000000) 160Mhz

Resources:

- Alixpress
- Communica.co.za
- https://www.adafruit.com/categories/35
- www.micropython.org
- https://learn.adafruit.com/micropython-hardware-digital-i-slash-o/
- https://github.com/micropython/micropython
- http://docs.micropython.org/en/v1.8/esp8266/
- https://media.readthedocs.org/pdf/micropython-on-esp8266-workshop/latest/micropython-on-esp8266-workshop.pdf
- https://www.maker.io/en/blogs/programming-micropython-on-the-esp8266/57abec67b5c34eb398d8fe6ae6442f46

Q&A