Discovering Tiny Snakes

IoT development without the need to compile
(mostly)

Quick: MicroPython vs. CircuitPython?

MicroPython

circuit python

Why is this different?

PartialUpdateExample

```
// PartialUpdateExample : example for Waveshare 1.54", 2.31" and 2.9" e-Paper and the same e-papers from Dalian Good Display Inc.
//
// Created by Jean-Marc Zingg based on demo code from Good Display for GDEP015OC1.
//
// The e-paper displays are available from:
//
// https://www.aliexpress.com/store/product/Wholesale-1-54inch-E-Ink-display-module-with-embedded-controller-200x200-Communicate-via-SPI-interface-Supports/21623
//
// http://www.buy-lcd.com/index.php?route=product/product&path=2897_8363&product_id=35120
// or https://www.aliexpress.com/store/product/E001-1-54-inch-partial-refresh-Small-size-dot-matrix-e-paper-display/600281_32815089163.html
//

// Supporting Arduino Forum Topics:
// Waveshare e-paper displays with SPI: http://forum.arduino.cc/index.php?topic=487007.0
// Good Dispay ePaper for Arduino : https://forum.arduino.cc/index.php?topic=436411.0

// mapping suggestion from Waveshare 2.9inch e-Paper to Wemos D1 mini
// BUSY -> D2, RST -> D4, DC -> D3, CS -> D8, CLK -> D5, DIN -> D7, GND -> GND, 3.3V -> 3.3V

// mapping suggestion from Waveshare 2.9inch e-Paper to generic ESP8266
// BUSY -> GPIO4, RST -> GPIO2, DC -> GPIO0, CS -> GPIO15, CLK -> GPIO14, DIN -> GPIO13, GND -> GND, 3.3V -> 3.3V

// mapping suggestion for ESP32, e.g. LOLIN32, see .../variants/.../pins_arduino.h for your board
// NOTE: there are variants with different pins for SPI ! CHECK SPI PINS OF YOUR BOARD
// BUSY -> 4, RST -> 16, DC -> 17, CS -> SS(5), CLK -> SCK(18), DIN -> MOSI(23), GND -> GND, 3.3V -> 3.3V

// mapping suggestion for AVR, UNO, NANO etc.
// BUSY -> 7, RST -> 9, DC -> 8, CS-> 10, CLK -> 13, DIN -> 11

// include library, include base class, make path known
#include <GxEPD.h>

// select the display class to use, only one
//#include <GxGDEP015OC1/GxGDEP015OC1.cpp>    // 1.54" b/w
//#include <GxGDE0213B1/GxGDE0213B1.cpp>      // 2.13" b/w
//#include <GxGDEH029A1/GxGDEH029A1.cpp>      // 2.9" b/w
//#include <GxGDEW042T2/GxGDEW042T2.cpp>      // 4.2" b/w
// these displays do not fully support partial update
```

Node32s, 80MHz, 921600 on /dev/ttyUSB0

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:4732
load:0x40078000,len:7496
load:0x40080400,len:5512
entry 0x4008114c
I (388) cpu_start: Pro cpu up.
I (389) cpu_start: Single core mode
I (389) heap_init: Initializing. RAM available for dynamic allocation:
I (392) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (398) heap_init: At 3FFC4F48 len 0001B0B8 (108 KiB): DRAM
I (405) heap_init: At 3FFE0440 len 00003BC0 (14 KiB): D/IRAM
I (411) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (417) heap_init: At 40091448 len 0000EBB8 (58 KiB): IRAM
I (424) cpu_start: Pro cpu start user code
I (218) cpu_start: Starting scheduler on PRO CPU.
Setting up LEDs
Setting up Buttons
Setting up Sensor I2C
Setting up BME280
Setting up TSL2591
bme_values[0]: 2172 - 21.72C
bme_values[1]: 25929420 - 1012.86hPa
bme_values[2]: 44558 - 43.51%
tsl_values[0]: 48
tsl_values[1]: 21
All Good
Initialize the Board LED as a PWM... Success
To break hit <ctrl>+c then enter: breathTimer.deinit()
OSError: [Errno 2] ENOENT
MicroPython v1.9.4-560-g185716514 on 2018-09-20; ESP32 module with ESP32
Type "help()" for more information.
>>>
```

Why is this different?

- Quick, iterative, development
- Most of the advantages of Python
- 0 to blinking LED very quick
- Mostly no need to compile anything
- Lots of default functionality, and upip (library / package management!)

Why is this possible?

- Same reason IoT is becoming ubiquitous
  - Low power MCUs and CPUs are getting more powerful, and cheaper at the same time.
- ESP32 on the SensorNode cost $5.10 to place on the board.
  - Dual Core
  - Wifi (802.11b/g/n up to 150Mbps 2.4GHz)
  - Bluetooth (v4.2 BR/EDR & BLE)
  - 4MB of flash
  - 520KB RAM
- There’s lots of competition in this space

How to get started

- Serial Drivers
  - Linux: Assuming your distro isn’t terrible, you are done
  - Windows / Mac: Install the Silicon Mechanics CP2104 (https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers)
- Download / Install esptool
  - This requires Python
  - Linux: distro packages are available
  - Windows / Mac: use pypi to install
- Download MicroPython and Upload it to the board
  - esptool.py --chip esp32 --port /dev/ttyUSB0 erase_flash && \
    esptool.py --chip esp32 --port /dev/ttyUSB0 write_flash -z 0x1000 <path to micropython .bin>

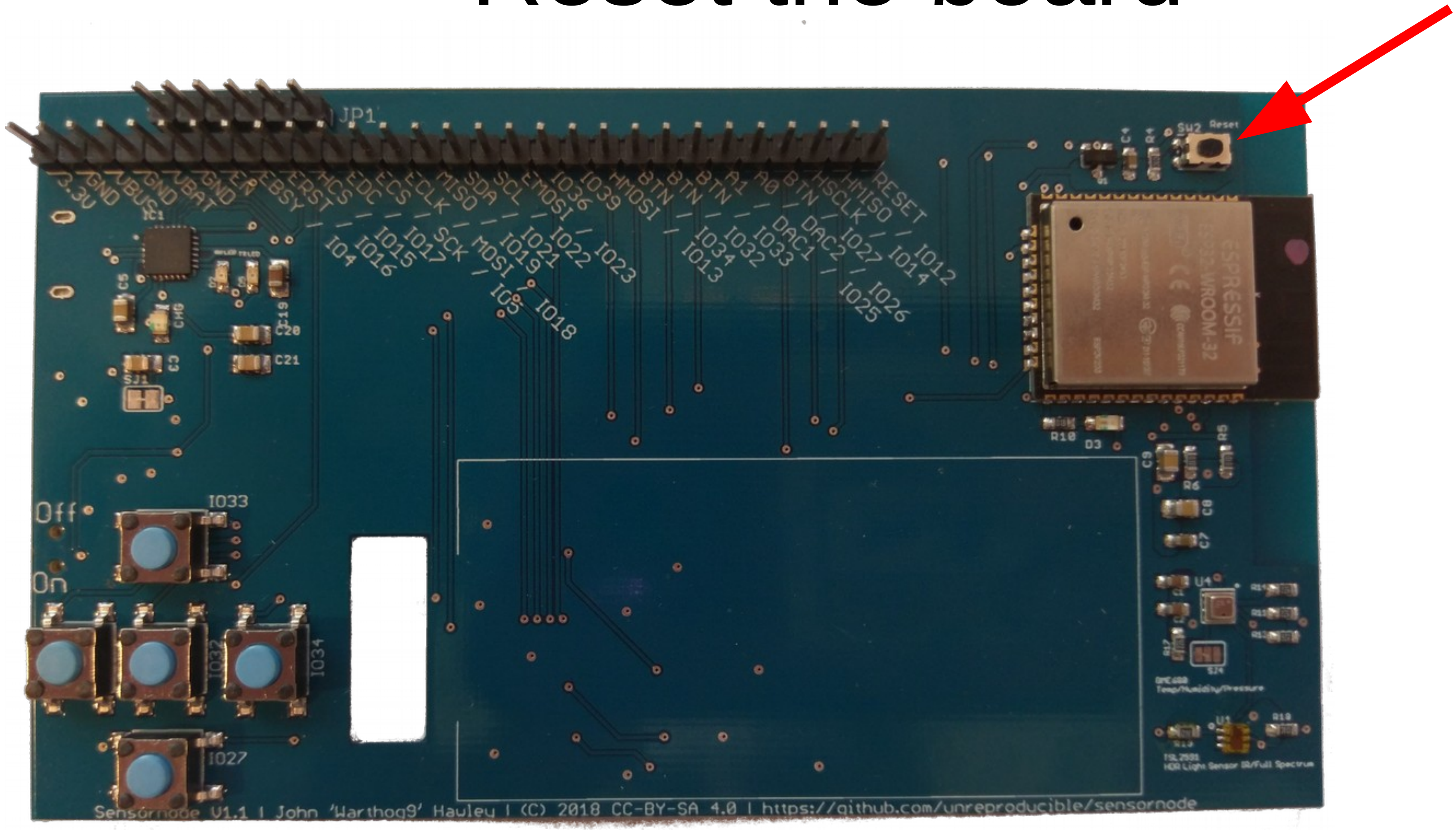
Breaking down the flash commands

```
esptool.py
    --chip esp32                  # Identifies which chip varient we are dealing with
    --port /dev/ttyUSB0           # Identifies which port the serial device is on
    erase_flash                   # Erases the flash area of the chip (not including the boot loader area)
&&
esptool.py
    --chip esp32                  # Identifies which chip varient we are dealing with
    --port /dev/ttyUSB0           # Identifies which port the serial device is on
    write_flash                   # Indicates to write to the flash chip
    -z 0x1000                     # Indicates WHERE on the flash chip to write to
    <path to micropython .bin>    # What to flash to the chip
```

Open up the serial console

- Minicom: minicom -D /dev/ttyUSB0 --baudrate 115200 (to exit <ctrl>c-q)
- Screen: screen /dev/ttyUSB0 115200n8 (to exit <ctrl>c-A \)
- Windows: use PuTTY

Reset the board



On the serial console...

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:4732
load:0x40078000,len:7496
load:0x40080400,len:5512
entry 0x4008114c
I (388) cpu_start: Pro cpu up.
I (389) cpu_start: Single core mode
I (389) heap_init: Initializing. RAM available for dynamic allocation:
I (392) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (398) heap_init: At 3FFC4F48 len 0001B0B8 (108 KiB): DRAM
I (405) heap_init: At 3FFE0440 len 00003BC0 (14 KiB): D/IRAM
I (411) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (417) heap_init: At 40091448 len 0000EBB8 (58 KiB): IRAM
I (424) cpu_start: Pro cpu start user code
I (218) cpu_start: Starting scheduler on PRO CPU.
Setting up LEDs
Setting up Buttons
Setting up Sensor I2C
Setting up BME280
Setting up TSL2591
bme_values[0]: 2172 - 21.72C
bme_values[1]: 25929420 - 1012.86hPa
bme_values[2]: 44558 - 43.51%
tsl_values[0]: 48
tsl_values[1]: 21
All Good
Initialize the Board LED as a PWM... Success
To break hit <ctrl>+c then enter: breathTimer.deinit()
OSError: [Errno 2] ENOENT
MicroPython v1.9.4-560-g185716514 on 2018-09-20; ESP32 module with ESP32
Type "help()" for more information.
>>>
```

Now to blink an LED!

- Type the following:

```
from machine import Pin, Signal
import machine
pin_led_board   = 0
_led_board = Pin(pin_led_board, Pin.OUT)
led_board = Signal( _led_board, invert=True )
led_board.off()
led_board.on()
```

Some interesting things to note

- boot.py
  - executed on every start, good for setting up the board (good place for wifi settings for example)
- main.py
  - Run after boot.py, think of it like the autoexec.bat
- It’s possible to upload more files to the board
  - Ampy - https://github.com/adafruit/ampy
- Tab completion works in the repl prompt
- <ctrl>+e at the repl prompt puts you into “paste” mode, so you can paste a longer set of code into the buffer to execute

Lets get more advanced….

- Read from the BME280
  - Upload bme280.py to the board
  - Setup I2C in python
  - Attach bme280 to the I2C bus
  - Read some data

The Code:

```
from machine import Pin, I2C
import machine
import bme280

pin_i2c_scl     = 22
pin_i2c_sda     = 21

bme280_address  = 0x77

print("Setting up Sensor I2C")

sensor_i2c = I2C( scl=Pin(pin_i2c_scl), sda=Pin(pin_i2c_sda) )

print("Setting up BME280")

bme = bme280.BME280( i2c=sensor_i2c, address=bme280_address )

bme.values
```

Adding the TSL2591 to the BME280

- Import the TSL2591 driver
- Attach the driver to the same I2C

The Code

```
from machine import Pin, I2C
import machine
import bme280
import tsl2591

pin_i2c_scl     = 22
pin_i2c_sda     = 21

bme280_address  = 0x77

print("Setting up Sensor I2C")

sensor_i2c = I2C( scl=Pin(pin_i2c_scl), sda=Pin(pin_i2c_sda) )

print("Setting up BME280")

bme = bme280.BME280( i2c=sensor_i2c, address=bme280_address )

bme.values

tsl = tsl2591.Tsl2591()
tsl.get_full_luminosity()
```

Where to go from here

- Setup Wifi in client mode
  - Then run:

```
import socket
addr_info = socket.getaddrinfo("towel.blinkenlights.nl", 23)
s = socket.socket()
s.connect(addr)
while True:
    data = s.recv(500)
    print(str(data, 'utf8'), end='')
```

  # the several enters above matter for the loop levels

- Setup Wifi in AP mode (note: it can do both simultaneously, albeit slowly)
- Install uMQTT and export sensors over MQTT
- Explore the “test” scripts included
- Put files on the sdcard
- Enjoy the board

Links to more resources

- https://github.com/unreproducible/tinysnakes
- https://docs.micropython.org/en/latest/esp8266/tutorial/intro.html
(note: most of the ideas are the same, the boards ARE different)
- https://boneskull.com/micropython-on-esp32-part-1/
- https://www.cnx-software.com/2017/10/16/esp32-micropython-tutorials/

Any questions before you start this on your own?

John ‘Warthog9’ Hawley | warthog9@eaglescrag.net | @warty9