Designing Embedded Systems using Micro Python

Class 3: Customizing the MicroPython Kernel for Production

June 12, 2019
Jacob Beningo

Presented by:

Course Overview

Topics:

- Designing Products with MicroPython
- Getting Started with the Pyboard D-Series
- **Customizing the MicroPython Kernel for Production**
- Developing Real-time Application Projects
- Testing MicroPython Projects

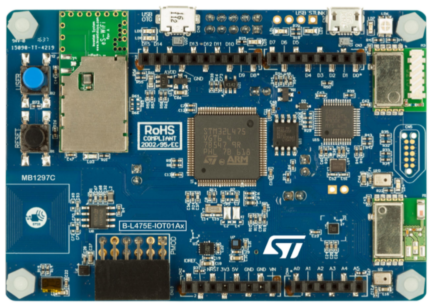
Presented by:

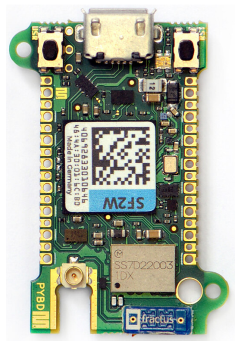
Session Overview

- Building a product
- The MicroPython Kernel
- How to customize the kernel
- Creating production code (MPY)
- Deploying the Kernel

Presented by:

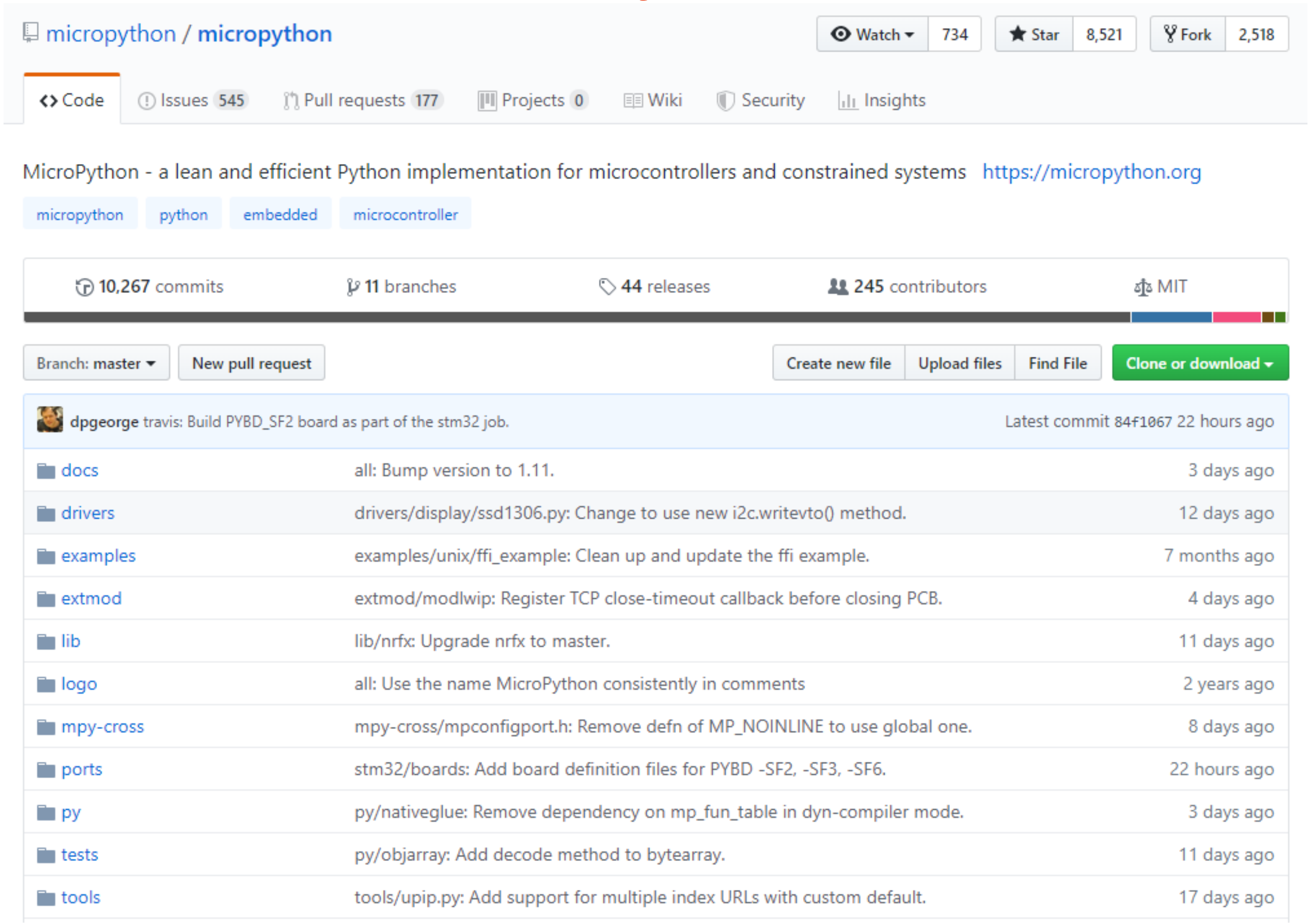
How do we customize MicroPython?





Presented by:

The MicroPython Kernel



micropython / micropython

MicroPython - a lean and efficient Python implementation for microcontrollers and constrained systems https://micropython.org

micropython python embedded microcontroller

10,267 commits · 11 branches · 44 releases · 245 contributors · MIT

dpgeorge travis: Build PYBD_SF2 board as part of the stm32 job. Latest commit 84f1067 22 hours ago

docs	all: Bump version to 1.11.	3 days ago
drivers	drivers/display/ssd1306.py: Change to use new i2c.writevto() method.	12 days ago
examples	examples/unix/ffi_example: Clean up and update the ffi example.	7 months ago
extmod	extmod/modlwip: Register TCP close-timeout callback before closing PCB.	4 days ago
lib	lib/nrfx: Upgrade nrfx to master.	11 days ago
logo	all: Use the name MicroPython consistently in comments	2 years ago
mpy-cross	mpy-cross/mpconfigport.h: Remove defn of MP_NOINLINE to use global one.	8 days ago
ports	stm32/boards: Add board definition files for PYBD -SF2, -SF3, -SF6.	22 hours ago
py	py/nativeglue: Remove dependency on mp_fun_table in dyn-compiler mode.	3 days ago
tests	py/objarray: Add decode method to bytearray.	11 days ago
tools	tools/upip.py: Add support for multiple index URLs with custom default.	17 days ago

Downloading the MicroPython

1) Open a terminal
2) Install gcc toolchain:

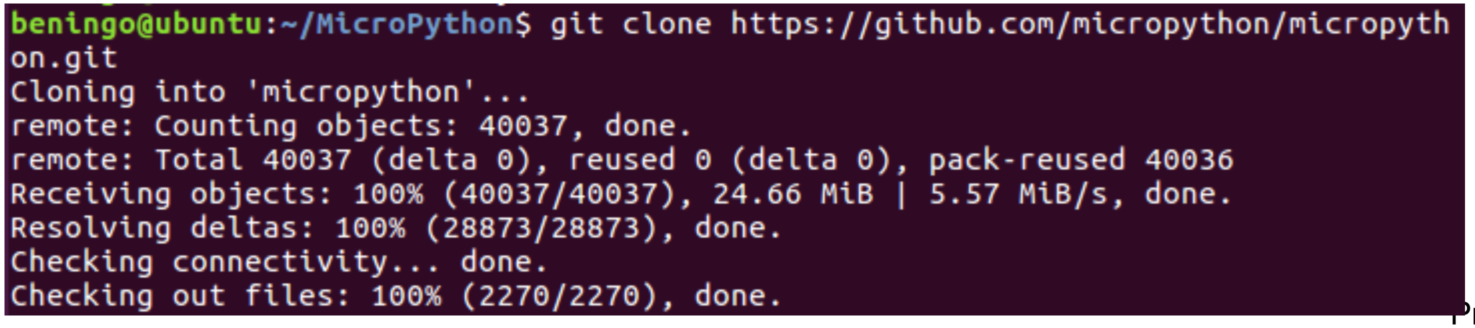
```
sudo apt-get install gcc-arm-none-eabi
```

3) Install git

```
sudo apt-get install git
```

4) Install MicroPython

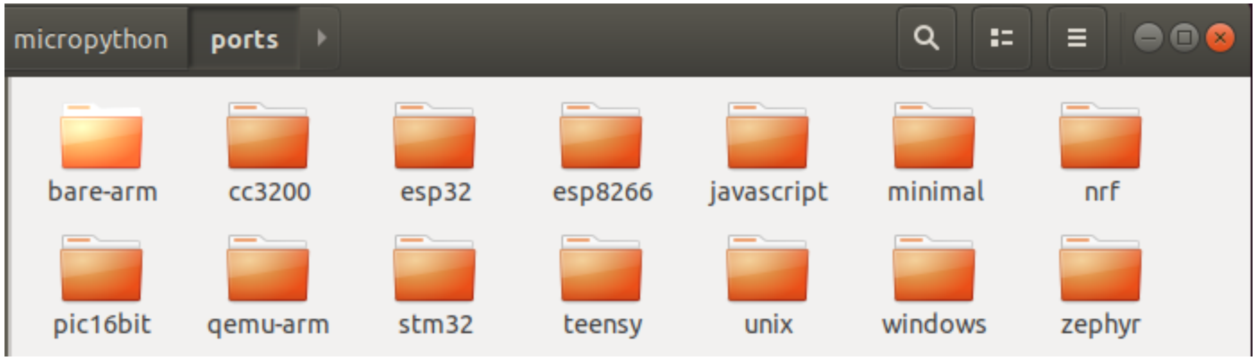
```
git clone https://github.com/micropython/micropython.git
```



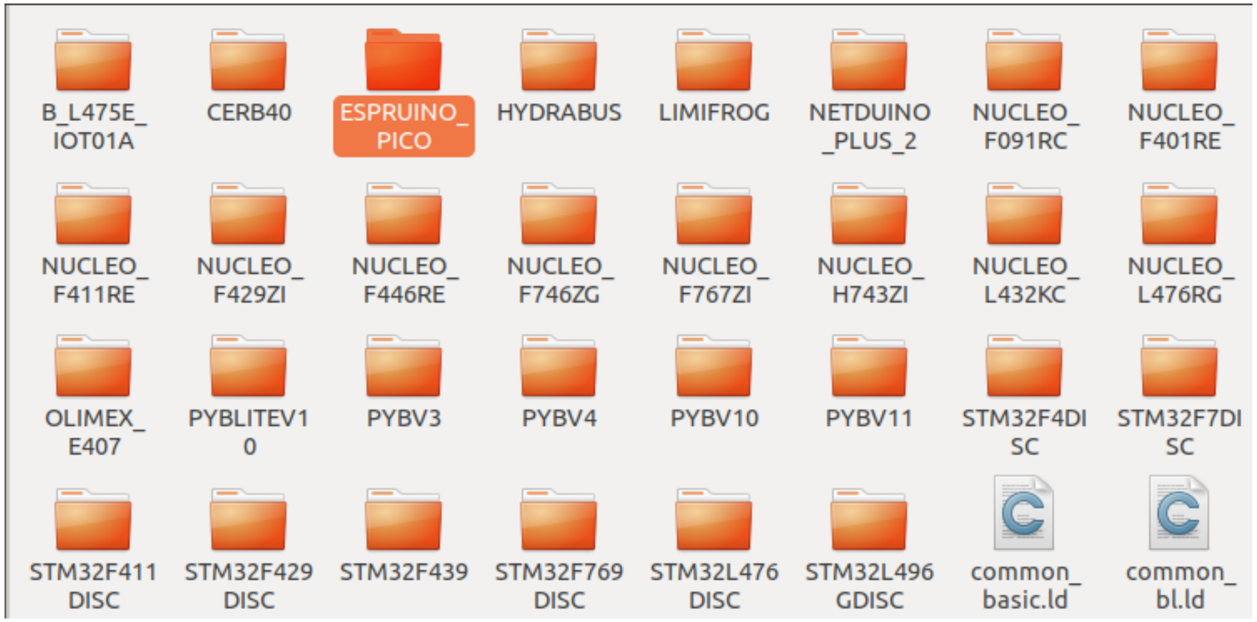
```
beningo@ubuntu:~/MicroPython$ git clone https://github.com/micropython/micropyth
on.git
Cloning into 'micropython'...
remote: Counting objects: 40037, done.
remote: Total 40037 (delta 0), reused 0 (delta 0), pack-reused 40036
Receiving objects: 100% (40037/40037), 24.66 MiB | 5.57 MiB/s, done.
Resolving deltas: 100% (28873/28873), done.
Checking connectivity... done.
Checking out files: 100% (2270/2270), done.
```

Presented by:

Available Ports



bare-arm, cc3200, esp32, esp8266, javascript, minimal, nrf, pic16bit, qemu-arm, stm32, teensy, unix, windows, zephyr

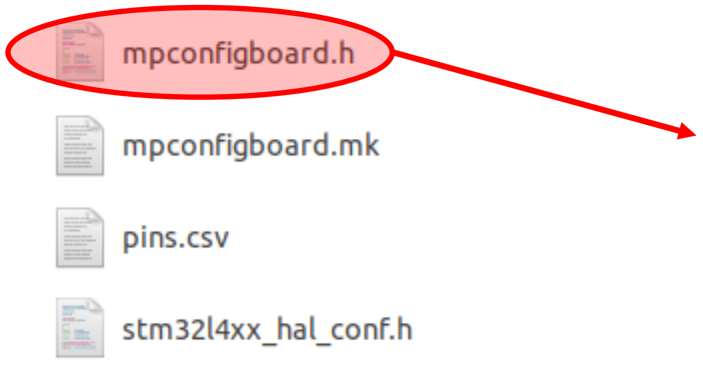


B_L475E_IOT01A, CERB40, ESPRUINO_PICO, HYDRABUS, LIMIFROG, NETDUINO_PLUS_2, NUCLEO_F091RC, NUCLEO_F401RE, NUCLEO_F411RE, NUCLEO_F429ZI, NUCLEO_F446RE, NUCLEO_F746ZG, NUCLEO_F767ZI, NUCLEO_H743ZI, NUCLEO_L432KC, NUCLEO_L476RG, OLIMEX_E407, PYBLITEV10, PYBV3, PYBV4, PYBV10, PYBV11, STM32F4DISC, STM32F7DISC, STM32F411DISC, STM32F429DISC, STM32F439, STM32F769DISC, STM32L476DISC, STM32L496GDISC, common_basic.ld, common_bl.ld

What do we want to Customize?

- Heap and Linker file
- Default pin initialization
- Enable / Disable MicroPython features
- Customize pin access names
- Adjust memory initialization
- Add additional error recovery code
- Add new features to the kernel (Bluetooth, etc)

Customize the Features



mpconfigboard.h

mpconfigboard.mk

pins.csv

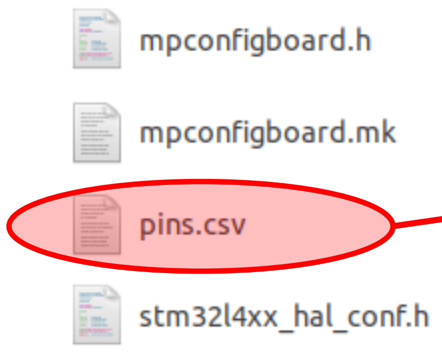
stm32l4xx_hal_conf.h

```
#define MICROPY_HW_BOARD_NAME       "B-L475E-IOT01A"
#define MICROPY_HW_MCU_NAME         "STM32L475"

#define MICROPY_HW_HAS_SWITCH       (1)
#define MICROPY_HW_ENABLE_RNG       (1)
#define MICROPY_HW_ENABLE_RTC       (1)
#define MICROPY_HW_ENABLE_USB       (1)


// LEDs
#define MICROPY_HW_LED1             (pin_A5) // green
#define MICROPY_HW_LED2             (pin_B14) // green
#define MICROPY_HW_LED_ON(pin)      (mp_hal_pin_high(pin))
#define MICROPY_HW_LED_OFF(pin)     (mp_hal_pin_low(pin))
```

Customize the Make File

- MCU series
- CMSIS target definition
- Alternate function mapping file for the board
- Linker file to be used
- Memory definitions for flash
- Debug probe configuration file

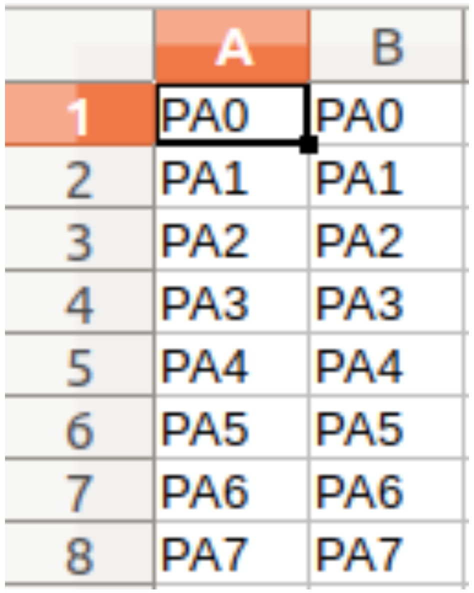
Customize the Pins


mpconfigboard.h

mpconfigboard.mk

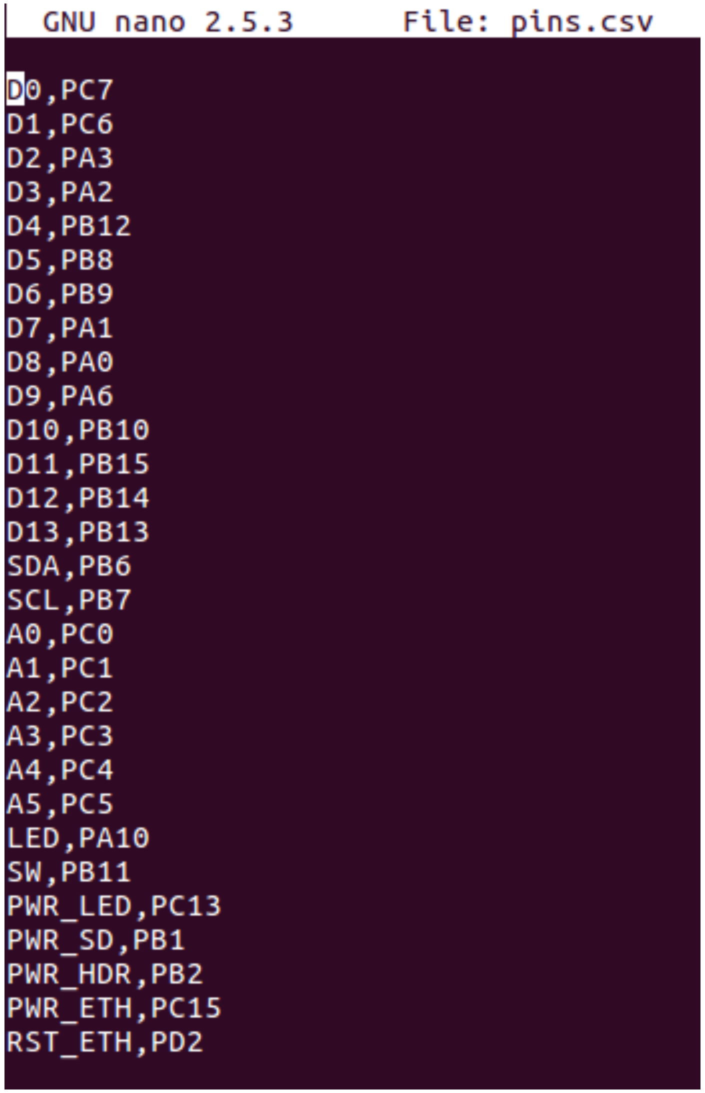
pins.csv

stm32l4xx_hal_conf.h



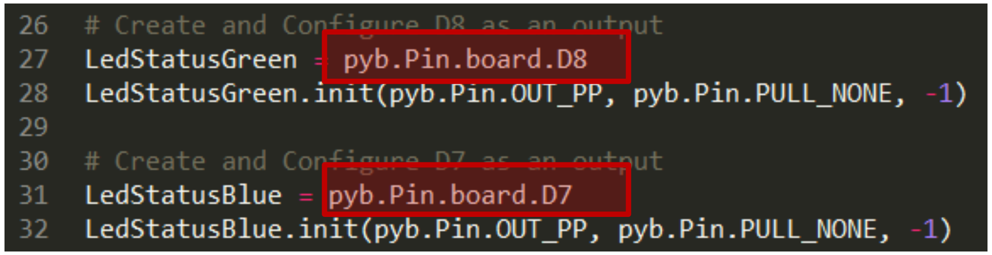
	A	B
1	PA0	PA0
2	PA1	PA1
3	PA2	PA2
4	PA3	PA3
5	PA4	PA4
6	PA5	PA5
7	PA6	PA6
8	PA7	PA7

Customize the Pins

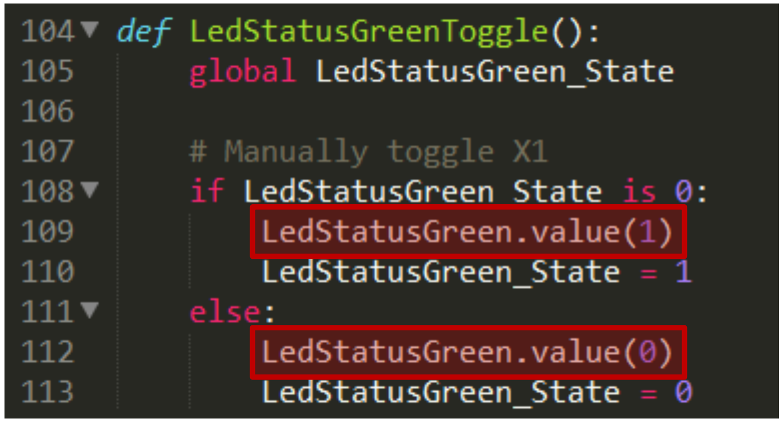


```
  GNU nano 2.5.3        File: pins.csv

D0,PC7
D1,PC6
D2,PA3
D3,PA2
D4,PB12
D5,PB8
D6,PB9
D7,PA1
D8,PA0
D9,PA6
D10,PB10
D11,PB15
D12,PB14
D13,PB13
SDA,PB6
SCL,PB7
A0,PC0
A1,PC1
A2,PC2
A3,PC3
A4,PC4
A5,PC5
LED,PA10
SW,PB11
PWR_LED,PC13
PWR_SD,PB1
PWR_HDR,PB2
PWR_ETH,PC15
RST_ETH,PD2
```

Init D7, D8 to control Status LEDs

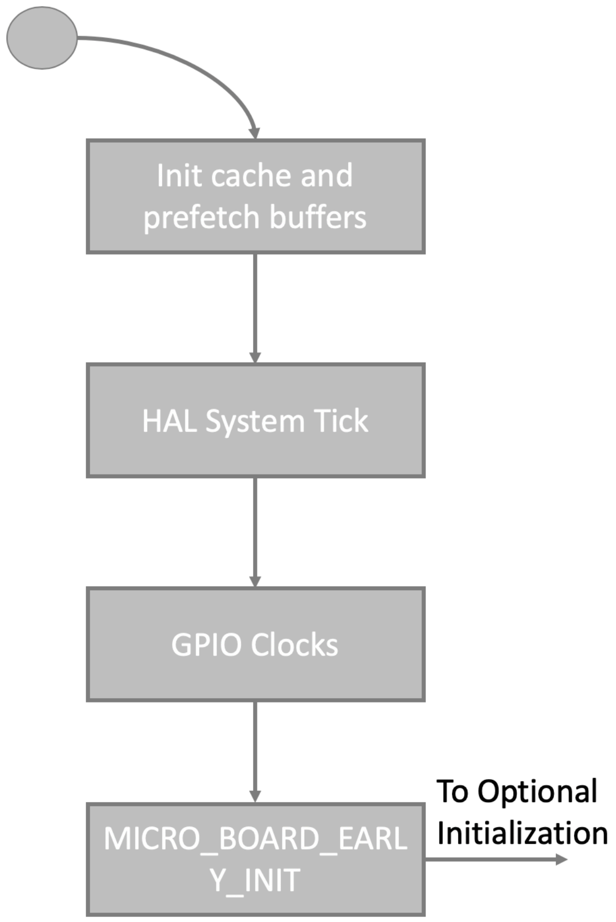


```python
# Create and Configure D8 as an output
LedStatusGreen = pyb.Pin.board.D8
LedStatusGreen.init(pyb.Pin.OUT_PP, pyb.Pin.PULL_NONE, -1)

# Create and Configure D7 as an output
LedStatusBlue = pyb.Pin.board.D7
LedStatusBlue.init(pyb.Pin.OUT_PP, pyb.Pin.PULL_NONE, -1)
```



```python
def LedStatusGreenToggle():
    global LedStatusGreen_State

    # Manually toggle X1
    if LedStatusGreen_State is 0:
        LedStatusGreen.value(1)
        LedStatusGreen_State = 1
    else:
        LedStatusGreen.value(0)
        LedStatusGreen_State = 0
```

MicroPython Start-up



Steps for Customizing the Start-up

- Update the board mpconfigboard.h module with the MICROPY_BOARD_EARLY_INIT definition along with the function name that will be called.

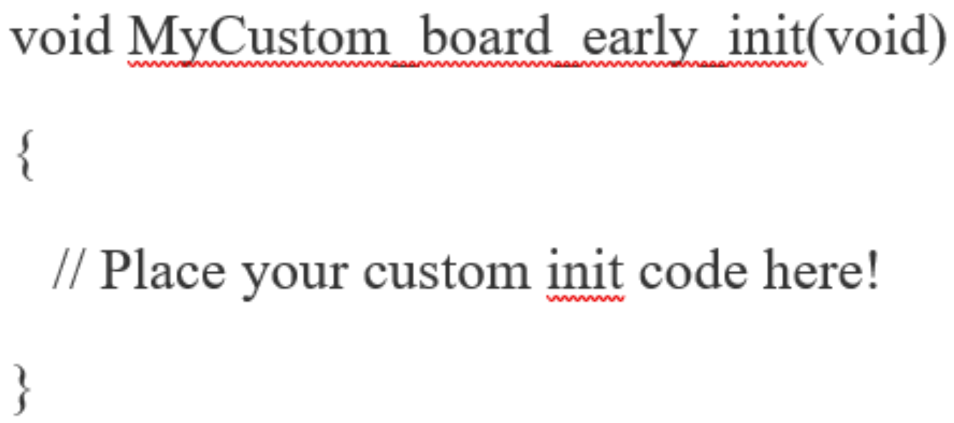
```
void MyCustom_board_early_init(void);

#define MICROPY_BOARD_EARLY_INIT    MyCustom_board_early_init
```

Steps for Customizing the Start-up

- Create a module to contain the code

Steps for Customizing the Start-up

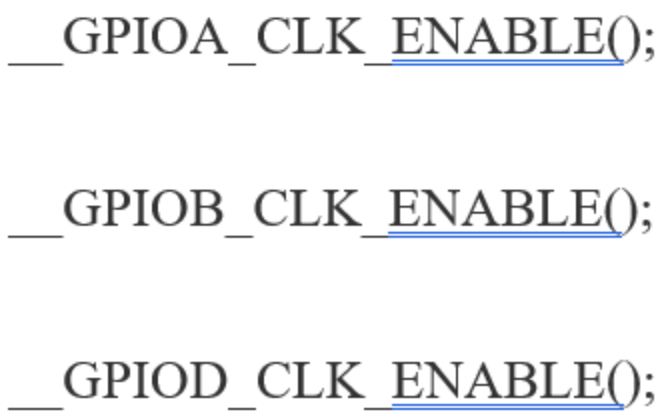
- Define the function that will be executed



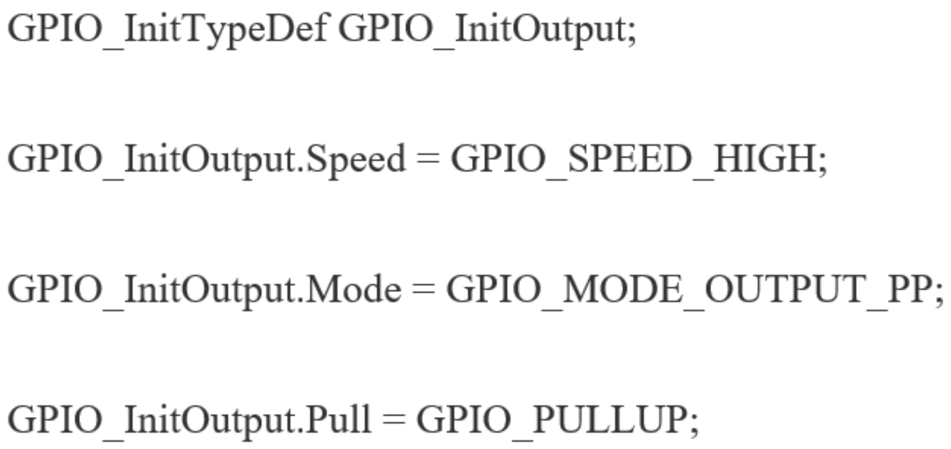
```
void MyCustom_board_early_init(void)
{
   // Place your custom init code here!
}
```

Steps for Customizing the Start-up

- Add the custom start-up code

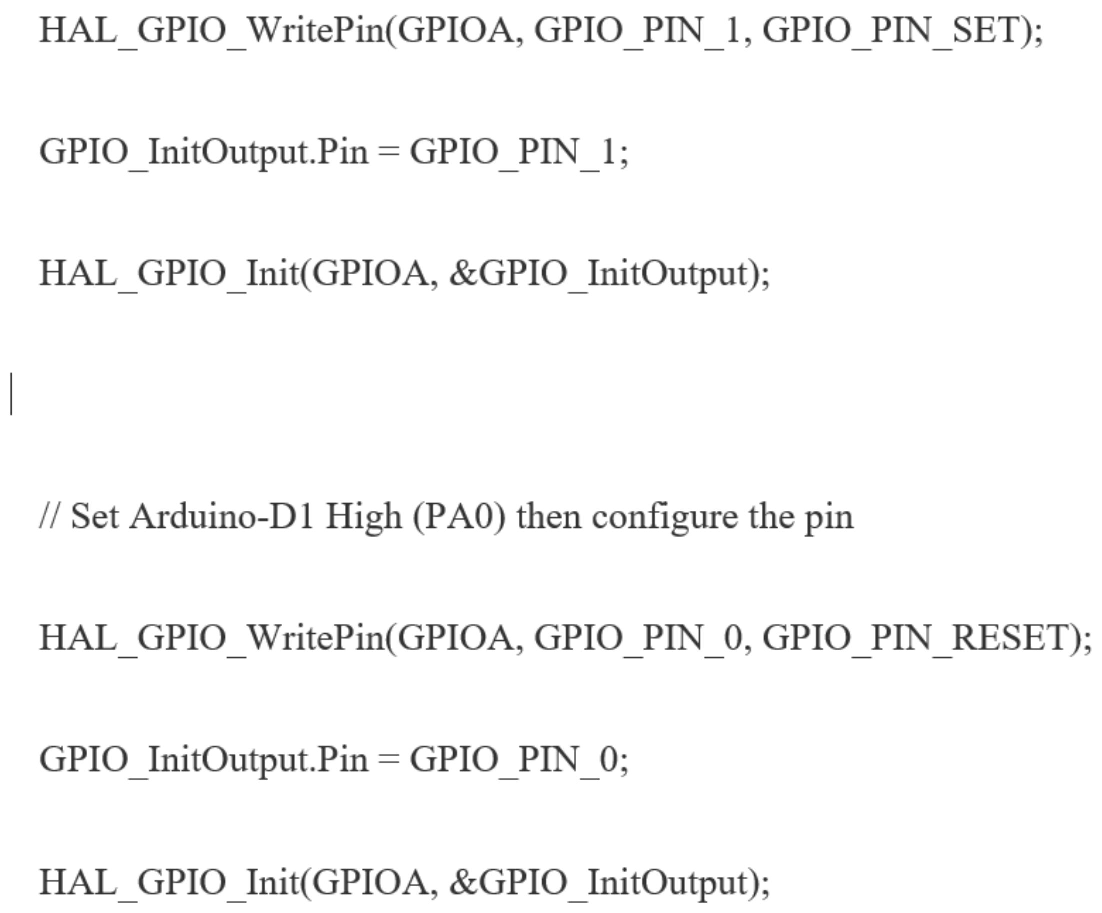


```
__GPIOA_CLK_ENABLE();

__GPIOB_CLK_ENABLE();

__GPIOD_CLK_ENABLE();
```



```
GPIO_InitTypeDef GPIO_InitOutput;

GPIO_InitOutput.Speed = GPIO_SPEED_HIGH;

GPIO_InitOutput.Mode = GPIO_MODE_OUTPUT_PP;

GPIO_InitOutput.Pull = GPIO_PULLUP;
```

Steps for Customizing the Start-up



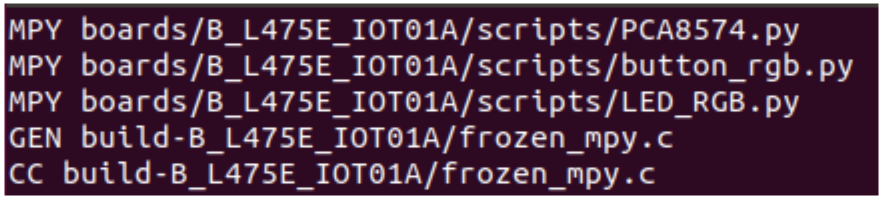
```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);

GPIO_InitOutput.Pin = GPIO_PIN_1;

HAL_GPIO_Init(GPIOA, &GPIO_InitOutput);


// Set Arduino-D1 High (PA0) then configure the pin

HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);

GPIO_InitOutput.Pin = GPIO_PIN_0;

HAL_GPIO_Init(GPIOA, &GPIO_InitOutput);
```

Advantages to using MPY

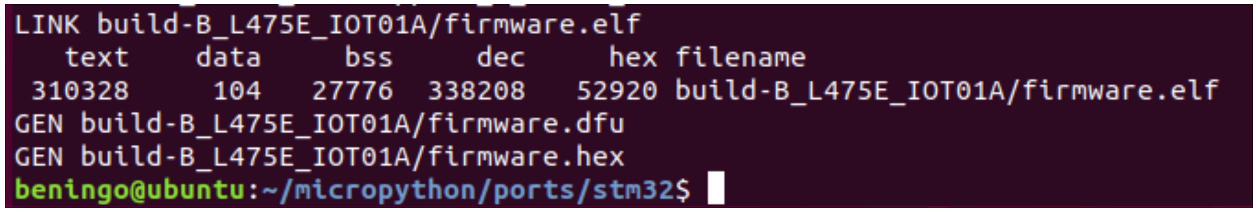
- The Python module cannot be modified without flashing the kernel
- The module is compiled into byte code which keeps the source code away from prying eyes
- Updating the application scripts is faster because there are fewer modules to update
- If something goes wrong with the file system and it gets set back to default, the compiled modules will still be present and can be called as part of the default script to get the system into a safe state
- You can put the compiled module into zero wait RAM if it has speed critical functionality that will ensure it executes as efficiently as possible.
- The compiled module can now also be stored and executed from flash which will free up RAM for the Python compiler and scripts that are stored on the file system.

Compiling the Kernel

- make BOARD= B_L475E_IOT01A FROZEN_MPY_DIR=boards/ B_L475E_IOT01A /scripts

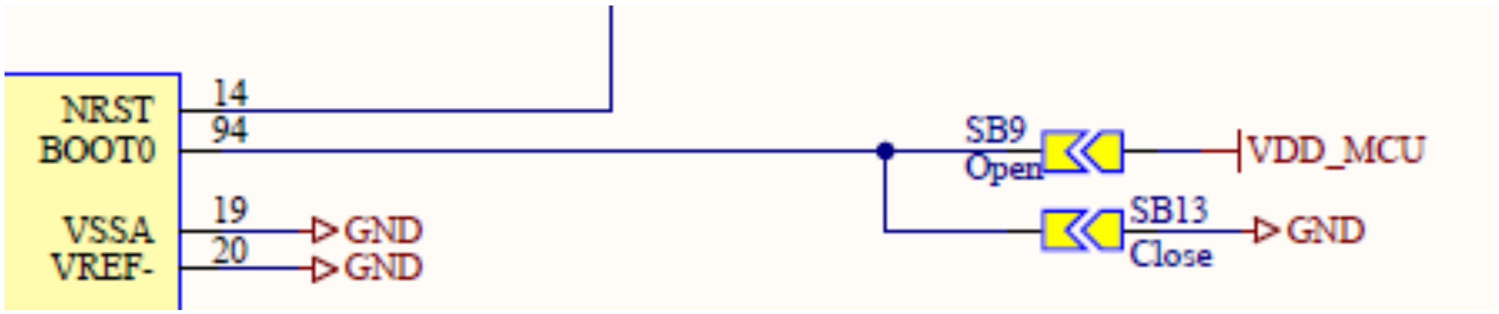


```
MPY boards/B_L475E_IOT01A/scripts/PCA8574.py
MPY boards/B_L475E_IOT01A/scripts/button_rgb.py
MPY boards/B_L475E_IOT01A/scripts/LED_RGB.py
GEN build-B_L475E_IOT01A/frozen_mpy.c
CC build-B_L475E_IOT01A/frozen_mpy.c
```



```
LINK build-B_L475E_IOT01A/firmware.elf
   text    data     bss     dec     hex filename
 310328     104   27776  338208   52920 build-B_L475E_IOT01A/firmware.elf
GEN build-B_L475E_IOT01A/firmware.dfu
GEN build-B_L475E_IOT01A/firmware.hex
beningo@ubuntu:~/micropython/ports/stm32$
```

Deploying the Kernel



```
dfu-util -a 0 0483:df11 -D build-B_L475E_IOT01A/firmware.dfu
```

Additional Resources

- Download Course Material for
  - http://bit.ly/MicroPythonProjects
  - Blog
  - YouTube Videos
- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

From www.beningo.com under

- Blog > CEC – Designing Embedded Systems using MicroPython