Embedded System Design Techniques™

Rapid Prototyping Embedded Systems using MicroPython

Session 5: Python Scripting for Testing and Debug

May 6th, 2016

Jacob Beningo, CSDP

DesignNews

Presented by:

CEC CONTINUING EDUCATION CENTER

Digi-Key ELECTRONICS

Course Overview

- Introduction to MicroPython
- Libraries and Peripheral Control
- Rapid Prototyping
- Building and Customizing Micro Python
- **Python Scripting for Testing and Debug**

Session Overview

- A few more configuration thoughts
- Writing reusable code
- Python with External Test Tools
- Debugging scripts
- Where to go from here?

A few more configuration thoughts

MicroPython NETDUINO_PLUS_2 configuration:

```
beningo@ubuntu:~/MicroPython/micropython/stmhal/boards/NETDUINO_PLUS_2$ ls
board_init.c  mpconfigboard.h  mpconfigboard.mk  pins.csv  stm32f4xx_hal_conf.h
beningo@ubuntu:~/MicroPython/micropython/stmhal/boards/NETDUINO_PLUS_2$
```

File	Description
board_init.c	Specialized board initialization code. Ex. HDR_PWR
mpconfigboard.h	Enable/Disable uPython board features. ie. SD, RTC
mpconfigboard.mk	Make file for the board
pins.csv	List of pin assignments and their default function
stm32f4xx_hal_conf.h	Hardware Abstraction Layer for STM32

A few more configuration thoughts

```
GNU nano 2.5.3                                  File: mpconfigboard.h

#define MICROPY_HW_BOARD_NAME       "NetduinoPlus2"
#define MICROPY_HW_MCU_NAME         "STM32F405RG"

#define MICROPY_HW_HAS_SWITCH       (1)

#define MICROPY_HW_HAS_FLASH        (1)
// On the netuino, the sdcard appears to be wired up as a 1-bit
// SPI, so the driver needs to be converted to support that before
// we can turn this on.
#define MICROPY_HW_HAS_SDCARD       (0)
#define MICROPY_HW_HAS_MMA7660      (0)
#define MICROPY_HW_HAS_LIS3DSH      (0)
#define MICROPY_HW_HAS_LCD          (0)
#define MICROPY_HW_ENABLE_RNG       (1)
#define MICROPY_HW_ENABLE_RTC       (0)
#define MICROPY_HW_ENABLE_TIMER     (1)
#define MICROPY_HW_ENABLE_SERVO     (1)
#define MICROPY_HW_ENABLE_DAC       (0)
#define MICROPY_HW_ENABLE_CAN       (0)

void NETDUINO_PLUS_2_board_early_init(void);
#define MICROPY_BOARD_EARLY_INIT    NETDUINO_PLUS_2_board_early_init
```

A few more configuration thoughts

```
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

```
# 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)
```

```
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
```

Writing Reusable Code

Reusable Code is ….

1) is modular
2) is loosely coupled
3) has high cohesion
4) has a clean interface
5) has a Hardware Abstraction Layer (HAL)
6) is readable and maintainable
7) is simple
8) uses encapsulation and abstract data types
9) is well documented

Writing Reusable Code

- Defining a class in Python

```
class I2C_Class():

    def __init__(self):
        ##
        # Defines the handle to the I2C device such as the aardvark connection id
        ##
        self.handle = 0
```

- Add methods to the class

```
def Open(self, port, bitrate):

    # Open the port
    self.handle = aa_open(port)
```

Writing Reusable Code

- Creating a new module

main.py

```
import pyb
import tasks

while True:
    tasks.Task_Led()
    pyb.Delay(250)
```

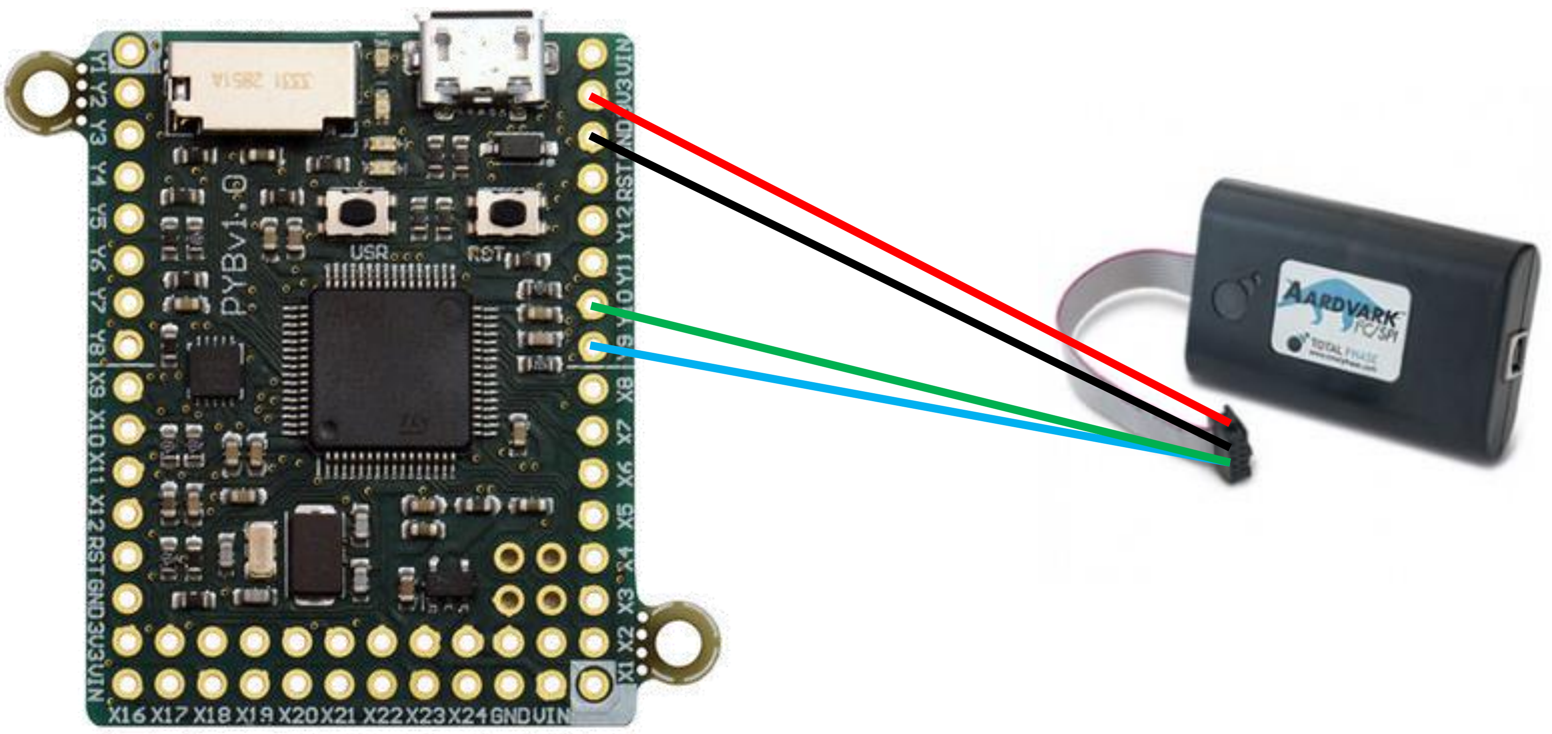
tasks.py

```
import pyb

def Task_Led():
    pyb.LED(1).toggle()

    return
```

Python with External Test Tools



Python with External Test Tools

Example interfacing to bus tool:

```
import sys
from aardvark_py import *

# Open the I2C port with the input port and bitrate
i2c.Open(0, 100000)

 # Prepare a data packet
data_out = array('B', [0xA, 0x01, 0x00, 0x01, 0x01, 0x01, 0x01, 0x01, 0x66, 0x10])

# Dump the data to the screen
print “Writing device data  "
print ''.join('{:02x} '.format(x) for x in data_out)

# Write the address and data
i2c.Write(address, data_out)
```

Debugging Python Scripts

- Four statements to catch run-time errors
  - try/except
    - Catch and recover from exceptions
  - try/finally
    - Perform cleanup actions whether an exception occurs or not
  - raise
    - Manually trigger an exception in code
  - assert
    - Conditionally trigger an exception

Default error handler prints error and exits the application!

Debugging Python Scripts

```
3898
3898
3898
Traceback (most recent call last):
  File "main.py", line 62, in <module>
  File "scheduler.py", line 61, in Run
  File "tasks.py", line 176, in PressureSample
OSError: 5
Micro Python v1.3.8 on 2014-12-29; PYBv1.0 with STM32F405RG
Type "help()" for more information.
>>>
```

Sensor Failed!

Execution halted

```
# tasks.py
def PressureSample():
    # Read the previous conversion
    RxData = bytearray(4)

    try:
        # Read the last conversion from the sensor
        RxData = I2C2.mem_read(2, int(BMP180_Address[0]), 0xF6)

        # Start next conversion
        I2C2.mem_write(0xE0, int(BMP180_Address[0]),0xF4)
    except OSError as er:
        print("Received Exception OSError: " + str(er))
```

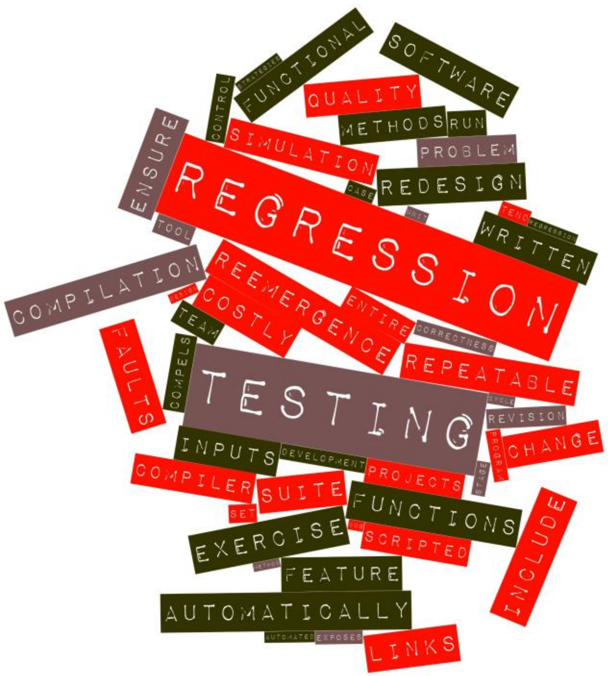
Where to go from here?

A few ideas of how you can use Python:

1) Regression Testing

2) Experimentation

3) Production



CEC CONTINUING EDUCATION CENTER
Presented by: Digi-Key ELECTRONICS

Where to go from here?

What can be done with MicroPython?

- Create a PyBoard Arduino shield break-out
- Play with the remaining peripherals, ADC, PWM, DAC, CAN, etc
- Build a robot, wifi connected weather station, drone, sensor node, etc
- Build a custom board to run MicroPython
- Modify and configure MicriPython to run on a custom board
- Write simple, reusable scripts to control microcontroller hardware
- Learn more about the Python programming language
  - Learn the language
  - Explore the design patterns and libraries available online

Course Concept Review

Additional Resources

- Download Course Material for
  - Updated C Doxygen Templates (Sept 2015)
  - Example source code
  - Templates
- Microcontroller API Standard
- EDN Embedded Basics Articles
- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

From www.beningo.com under

- Blog > CEC Rapid Prototyping with MicroPython

The Lecturer – Jacob Beningo

Jacob Beningo
Principal Consultant

Social Media / Contact

- E : jacob@beningo.com
- T : 248-719-6850
- : Jacob_Beningo
- : Beningo Engineering
- : JacobBeningo
- EDN : Embedded Basics

CONSULTING

- Secure Bootloaders
- Code Reviews
- Architecture Design
- Real-time Software
- Expert Firmware Analysis

EMBEDDED TRAINING

DOULOS CERTIFIED TRAINING PARTNER

BENINGO EMBEDDED GROUP

www.beningo.com

Presented by:

CEC CONTINUING EDUCATION CENTER

Digi-Key ELECTRONICS