pyBox0 Documentation

Release 0.3.0

Kuldeep Singh Dhaka

January 14, 2017

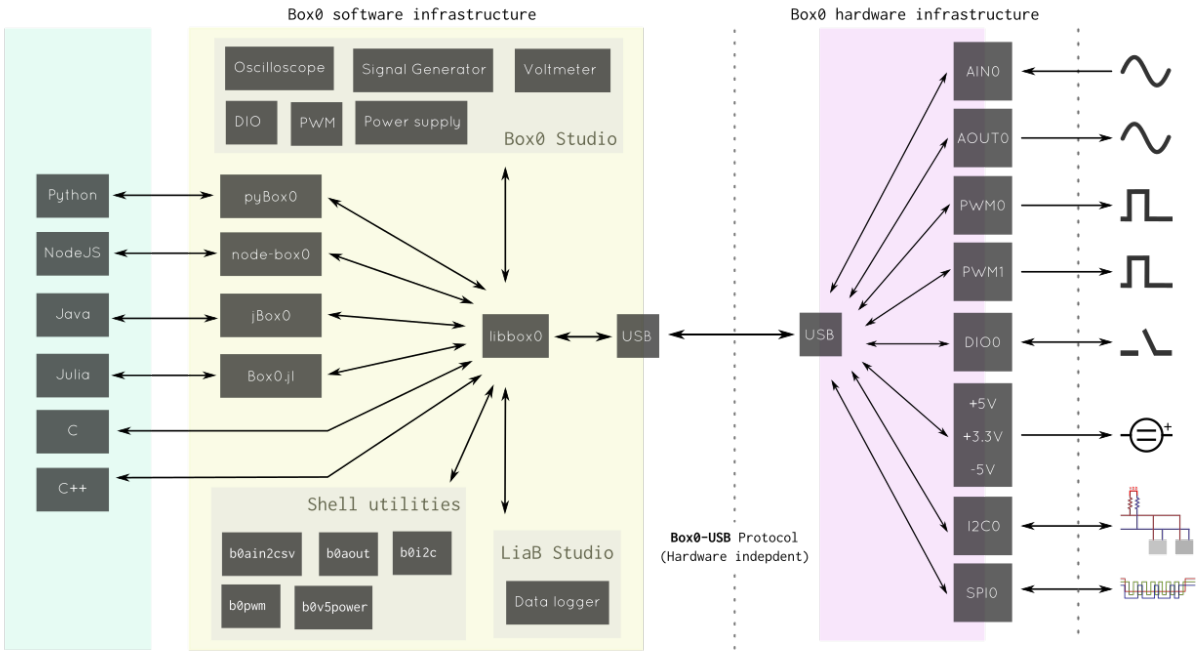
Contents

Contents:

CHAPTER 1

Introduction to pyBox0

pyBox0 is a Python binding of libbox0. libbox0 is the C library that does the communication with physical devices.



1.1 Importing

```
import box0
```

1.2 What is a Device

A device is an interface to the physical device you have.

A device can be acquired via multiple method, at the moment USB only.

1.3 Opening a device

```
import box0

dev = box0.usb.open_supported()
# ... do something with "dev"
```

In the above code, `box0.usb.open_supported` try to open any USB device connected that can be used as Box0. quick and easy!

You can do something with "dev" like

```
import box0

dev = box0.usb.open_supported()
print(dev.name) # Print device name (provided by device - brand)
print(dev.serial) # Print serial number (provided by device)
print(dev.manuf) # Print manufacturer name (provided by device)
```

1.4 What is a Module

A module is a portion of device that perform a dedicated function. A device can contain multiple modules.

Example of module with their uses.

Name of module	Short name	Use
Analog In	AIN	Read analog signal
Analog Out	AOUT	Generate analog signal
Digital Input/Output	DIO	Generate, Read digital signal
Serial Peripherial Interface	SPI	Communicate with SPI slaves
Inter Integerated Communication	I2C	Communicate with I2C slaves
Pulse Width Modulation	PWM	Generate Pulse Width Modulation

1.5 Opening a module

From the above, we know how to open a device. Now, we will open a module from device.

```
import box0

dev = box0.usb.open_supported()

my_ain = dev.ain() # Open Analog In (with index=0) from device

# .... do something with "my_ain"
```

The above pattern can be applied for all type of module.

Short name	<method>
AIN	`box0.Device.ain()`
AOUT	`box0.Device.aout()`
DIO	`box0.Device.dio()`
SPI	`box0.Device.spi()`
I2C	`box0.Device.i2c()`
PWM	`box0.Device.pwm()`

You can use `my_module = dev.<method>()`.

1.6 Exception and failure

libbox0. functions return a negative integer value (actually enum) to tell that some kind of error has occured.

pyBox0 convert these negative values to Exception with the help of a exception class `box0.ResultException`

```
import box0

try:
        dev = box0.usb.open_supported()
except ResultException, e:
        print("failed! (%s)" % e)
        # name of the exception: e.name()
        # explaination of exception: e.explain()
```

1.7 Resource management

Device, resource and driver are resources which are taken for a time and returned back when it is no more required.

A device, module and driver after closing cannot be used. Doing so will result in undefined behaviour. You can use `close()` method for closing, the `del` keyword leads to `close()` too.

You can also use `with` keyword for automatic disposal when execution of a block finishes. Device, module and driver support `with` statement.

CHAPTER 2

Demo code

2.1 Reading data from AIN

```
import box0
import numpy

# find thing to work with
dev = box0.usb.open_supported()
ain0 = dev.ain()
ain0.snapshot_prepare() # prepare for snapshot mode (ie snapshot of signal)

# do the work
values = numpy.empty(100, dtype=numpy.float32) # count=100, can vary though
ain0.snapshot_start(values) # blocking method (till data not readed)
print(values)

# dispose resources
ain0.close()
dev.close()
```

2.2 Toggle pins using DIO

```
import box0
import time

dev = box0.usb.open_supported()
dio0 = dev.dio()
dio0.basic_prepare()

#note: connect LED on "0" pin of "DIO0"
pin0 = dio0.pin(0)
pin0.output()
pin0.high()
pin0.enable()

dio0.basic_start()

while True:
        try:
                pin0.toggle()
```

```
                time.sleep(0.1)
        except KeyboardInterrupt:
                break

dio0.basic_stop()

dio0.close()
dev.close()
```

2.3 Generate Constant voltage

```
import box0
import numpy

CONSTANT_VOLTAGE = 1.5

dev = box0.usb.open_supported()
aout0 = dev.aout()

aout0.snapshot_prepare()
values = numpy.array([CONSTANT_VOLTAGE], dtype=numpy.float32)
aout0.snapshot_start(values) # non-blocking, return right after operation

input("Press Enter to exit")

aout0.snapshot_stop()
aout0.close()
dev.close()
```

2.4 Controlling LED strip connected to DIO

```
import box0
import time

dev = box0.usb.open_supported()
dio0 = dev.dio()

dio0.basic_prepare()

def wait():
        time.sleep(.05)

for i in range(8):
        pin = dio0.pin(i)
        pin.output()
        pin.hiz = False

dio0.basic_start()

try:
        while True:
                for i in range(8):
                        dio0.pin(i).value = True
```

```
                        wait()

                for i in range(8):
                        dio0.pin(i).value = False
                        wait()

                for i in range(8):
                        dio0.pin(7 - i).value = True
                        wait()

                for i in range(8):
                        dio0.pin(7 - i).value = False
                        wait()

except KeyboardInterrupt:
        pass

dio0.basic_stop()
dio0.close()
dev.close()
```

2.5 Reading using ADS1220 ADC

```
import box0
import time
import sys
from box0.driver import Ads1220

gain = Ads1220.GAIN_1

# get the gain
if len(sys.argv) > 1:
        gain = Ads1220.__dict__['GAIN_' + sys.argv[1]]
        print("chosen gain: %s" % sys.argv[1])

dev = box0.usb.open_supported()
spi0 = dev.spi(0)
spi0.master_prepare()
ads1220 = box0.driver.Ads1220(spi0, 0)

ads1220.gain_set(gain)

try:
        print("Values:")
        while True:
                ads1220.start()
                time.sleep(0.05)
                print(ads1220.read())
except KeyboardInterrupt:
        pass

ads1220.close()
spi0.close()
dev.close()
```

2.6 AIN -> AOUT streaming

```
import time
import box0
import numpy as np

dev = box0.usb.open_supported()
ain = dev.ain(0)
aout = dev.aout(0)

speed = 10000
bitsize = 12

ain.stream_prepare()
aout.stream_prepare()

ain.bitsize_speed_set(bitsize, speed)
aout.bitsize_speed_set(bitsize, speed)

ain.stream_start()
aout.stream_start()

try:
        count = speed / 10
        data = np.empty(count)
        while(True):
                ain.stream_read(data)
                aout.stream_write(data)
except:
        # no problem
        pass

ain.stream_stop()
aout.stream_stop()

ain.close()
aout.close()
dev.close()
```

2.7 Plotting AIN snapshot data with PyPlot

```
import box0
import time
import numpy as np
from pylab import *

SAMPLE_SPEED = 100000
SAMPLE_COUNT = 500
BITSIZE = 12

dev = box0.usb.open_supported()
ain0 = dev.ain(0)
ain0.snapshot_prepare()

xlabel('time (s)')
```

```
ylabel('voltage (V)')
title('About as simple as it gets, folks')
grid(True)

ain0.bitsize_speed_set(BITSIZE, SAMPLE_SPEED)
s = np.empty(int(SAMPLE_COUNT), dtype=np.float64)
ain0.snapshot_start(s)

t = arange(0.0, SAMPLE_COUNT / float(SAMPLE_SPEED), 1 / float(SAMPLE_SPEED))
clf()
grid(True)

print("s is" + str(s))
print("t is" + str(t))

plot(t, s, 'r.-')

savefig("test.png")

ain0.close()
dev.close()

show()
```

2.8 Plotting AIN snapshot data with PyQtGraph

```
from pyqtgraph.Qt import QtGui, QtCore
import numpy as np
import pyqtgraph as pg
import box0
from scipy import signal

## derived from pyqtgraph demo "PlotWidget"

app = QtGui.QApplication([])
mw = QtGui.QMainWindow()
mw.setWindowTitle('AIN Demo')
mw.resize(800,800)

pw = pg.PlotWidget(name='AIN0')  ## giving the plots names allows us to link their axes together
mw.setCentralWidget(pw)
mw.show()

## Create an empty plot curve to be filled later, set its pen
p1 = pw.plot()
p1.setPen((200,200,100))

dev = box0.usb.open_supported()
ain0 = dev.ain(0)

ain0.snapshot_prepare()

bs = 12
speed = 600000

ain0.bitsize_set(bs, speed)
```

```
byte_per_sample = (bs + 7) / 8
count = ain0.buffer_size / byte_per_sample

pw.setLabel('left', 'Value', units='V')
pw.setLabel('bottom', 'Time', units='s')
pw.setXRange(0, (1.0 * count) / speed)
pw.setYRange(0, 3)

def updateData():
        ##filtering
        ## http://stackoverflow.com/a/13740532
        #~ niqFreq = sv.speed / 2.0
        #~ cutoff = 100.0 # Hz
        #~ Wn = cutoff / niqFreq
        #~ order = 3
        #~ print("niqFreq:", niqFreq)
        #~ print("cutoff:", cutoff)
        #~ print("order:", order)
        #~ print("Wn:", Wn)
        #~ b, a = signal.butter(order, Wn, 'low')
        #~ y = signal.filtfilt(b, a, y)

        global speed, count
        y = np.empty(count)
        ain0.snapshot_start(y)

        x = np.linspace(0.0, count, count) / speed
        p1.setData(y=y, x=x)

t = QtCore.QTimer()
t.timeout.connect(updateData)
t.start(50)

## Start Qt event loop unless running in interactive mode or using pyside.
if __name__ == '__main__':
    import sys
    if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_VERSION'):
        QtGui.QApplication.instance().exec_()

t.stop()

ain0.close()
dev.close()
```

2.9 Plotting AIN stream data with PyQtGraph

```
from pyqtgraph.Qt import QtGui, QtCore
import numpy as np
import pyqtgraph as pg
import box0

## derived from pyqtgraph demo "PlotWidget"

app = QtGui.QApplication([])
mw = QtGui.QMainWindow()
```

```
mw.setWindowTitle('AIN Demo')
mw.resize(800,800)

pw = pg.PlotWidget(name='AIN0')  ## giving the plots names allows us to link their axes together
mw.setCentralWidget(pw)
mw.show()

## Create an empty plot curve to be filled later, set its pen
p1 = pw.plot()
p1.setPen((200,200,100))

pw.setLabel('left', 'Value', units='V')
pw.setLabel('bottom', 'Time', units='s')
pw.setXRange(0, 1)
pw.setYRange(0, 3)

dev = box0.usb.open_supported()
ain0 = dev.ain(0)

SPEED = 10000
BITSIZE = 12

ain0.stream_prepare()

ain0.bitsize_speed_set(BITSIZE, SPEED)

class Poller(QtCore.QThread):
        feed = QtCore.pyqtSignal(np.ndarray, np.ndarray, name = 'feed')

        def __init__(self):
                QtCore.QThread.__init__(self)

        def start(self, mod, size):
                self.interruption_requested = False
                self.module = mod
                self.count = size
                QtCore.QThread.start(self)

        def stop(self):
                self.interruption_requested = True

        def run(self):
                global np
                while not self.interruption_requested:
                        data = np.empty(self.count)
                        self.module.stream_read(data)
                        x = np.linspace(0.0, 1.0, sps)
                        self.feed.emit(x, data)

ain0.stream_start()

def update(x, y):
        global p1
        p1.setData(y=y, x=x)

poller = Poller()
sps = SPEED ## 1second
poller.feed.connect(update)
```

```
poller.start(ain0, sps)

## Start Qt event loop unless running in interactive mode or using pyside.
if __name__ == '__main__':
    import sys
    if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_VERSION'):
        QtGui.QApplication.instance().exec_()

poller.stop()
poller.wait()

ain0.stream_stop()
ain0.close()
dev.close()
```

2.10 Small Power supply controlling program (via pyUSB)

```
#/bin/python

#
# box-v5 Power Supply
# Author: Kuldeep Singh Dhaka <kuldeep@madresistor.com>
# Licence: GPLv3 or later
#

import usb.core
import usb.util
from usb.util import CTRL_IN, CTRL_OUT, CTRL_RECIPIENT_DEVICE, CTRL_TYPE_VENDOR

BOX0V5_PS_EN_GET = 201
BOX0V5_PS_EN_SET = 202

POWER_ANALOG = 0x01
POWER_DIGITAL = 0x02

# python2 raw_input and python3 input
try: input = raw_input
except: pass

# open device
dev = usb.core.find(idVendor=0x1d50, idProduct=0x8085)
if dev is None:
        raise ValueError("Device not found")

# assign 1st configuration
dev.set_configuration()

print("Welcome! please enter a command:")
print("  b - [both] digital supply enable and analog supply enable")
print("  a - [analog] analog supply enable and digital supply disable")
print("  d - [digital] digital supply enable and analog supply disable")
print("  n - [none] digital supply disable and analog supply disable")
print("  s - [status] both suppy status")
print("  e - [exit] exit the program")

def power_supply_set(dev, analog, digital):
```

```
        """
        Activate/deactive power supply
        dev: USB Device
        analog: activate/deactivate Analog supply
        digital: activate/deactivate Digital supply
        """
        bmReqType = CTRL_OUT | CTRL_RECIPIENT_DEVICE | CTRL_TYPE_VENDOR
        mask = POWER_ANALOG | POWER_DIGITAL
        value = 0x00
        if analog: value |= POWER_ANALOG
        if digital: value |= POWER_DIGITAL
        wValue = (mask << 8) | value
        dev.ctrl_transfer(bmReqType, BOX0V5_PS_EN_SET, wValue)

def power_supply_get(dev):
        """
        Read power supply status
        dev: USB Device
        return a tuple (<analog-supply>, <digital-supply>)
        """
        bmReqType = CTRL_IN | CTRL_RECIPIENT_DEVICE | CTRL_TYPE_VENDOR
        data = dev.ctrl_transfer(bmReqType, BOX0V5_PS_EN_GET, 0, 0, 1)
        analog = (data[0] & POWER_ANALOG) != 0x00
        digital = (data[0] & POWER_DIGITAL) != 0x00
        return analog, digital

#turn both supply off
power_supply_set(dev, False, False)

try:
        while True:
                c = input("> ")
                if c == "b":
                        power_supply_set(dev, True, True)
                elif c == "a":
                        power_supply_set(dev, True, False)
                elif c == "d":
                        power_supply_set(dev, False, True)
                elif c == "n":
                        power_supply_set(dev, False, False)
                elif c == "s":
                        analog, digital = power_supply_get(dev)
                        print("Analog: " + ("Enabled" if analog else "Disabled"))
                        print("Digital: " + ("Enabled" if digital else "Disabled"))
                elif c == "e":
                        break;
                else:
                        print("unknown command: " + c)
except KeyboardInterrupt: pass

#turn all supply off
power_supply_set(dev, False, False)

#close device
del dev
```

2.11 PWM basic example

```
import box0
import time

dev = box0.usb.open_supported()
pwm0 = dev.pwm()

pwm0.speed_set(1000)
pwm0.period_set(250)
pwm0.width_set(0, 100)
# same as 4Hz  40% duty cycle

pwm0.output_start()

try:
        while True:
                time.sleep(0.1)
except KeyboardInterrupt:
        pass

pwm0.output_stop()
pwm0.close()
dev.close()
```

2.12 Time varying PWM output

```
#/bin/python

import box0
import time

dev = box0.usb.open_supported()
pwm1 = dev.pwm(1)
pwm1.speed_set(1000)
pwm1.period_set(100)
pwm1.width_set(0, 0)
pwm1.output_start()

try:
        while True:
                for i in range(1, 50, 5):
                        pwm1.width_set(0, i)
                        time.sleep(.1)

                for i in range(50, 1, -5):
                        pwm1.width_set(0, i)
                        time.sleep(.1)
except:
        pass

pwm1.output_stop()

pwm1.close()
dev.close()
```

CHAPTER 3

The pyBox0 API Reference

3.1 The “box0” module

3.1.1 Device

3.1.2 ResultException

3.1.3 Version

3.2 The “module” module

3.2.1 Module

3.2.2 Analog In

3.2.3 Analog Out

3.2.4 Digital Input/Output

3.2.5 Pulse Width Modulation

3.2.6 Inter Integerated Communication

3.2.7 Serial Peripherial Interface

3.3 The “usb” backend

3.4 The “box0v5” (box0-v5)

3.5 The “driver” module

CHAPTER 4

Indices and tables

- genindex
- modindex
- search