```
import xbee
from machine import I2C
from machine import Pin
import binascii

# import time

#   debug controls terminal output
debug = False
dictionary = True

print("Waiting for data...\n")
# addressing is 16-bit network
dac_addr = 71
ppe_addr = 32
ppe_shadow = 0x00
act_on = 0
act_off = 1
act_delay = 8
err = False
ppemask = 0x00

# RMTx = XBee addr, payload ID, active
if dictionary:
    RMT0 = {
        "net_id": 0xC00E,
        "pl_id": 77,
        "active": True
    }
    RMT1 = {
        "net_id": 0xEAE3,
        "pl_id": 182,
        "active": True
    }
    RMT2 = {
        "net_id": 0xC00E,
        "pl_id": 2,
        "active": False
    }
    RMT3 = {
        "net_id": 0xBA09,
        "pl_id": 4,
        "active": False
    }
else:

    RMT0 = [0xC00E, 77, True]
    RMT1 = [0xEAE3, 182, True]
    RMT2 = [0xC00E, 2, False]
    RMT3 = [0xBA09, 4, False]

RMT = [RMT0, RMT1, RMT2, RMT3]

# remotes = HA83 joystick number assigned to each of four wireless remotes
remotes = [0, 1, 2, 3]

xbaddr = [None] * 4
pl_id = [None] * 4
rmt_active = [None] * 4
# rmt_highp=[None]*4

for x in remotes:
    if dictionary:
        xbaddr[x] = RMT[x]["net_id"]
        pl_id[x] = RMT[x]["pl_id"]
        rmt_active[x] = RMT[x]["active"]
    else:
        xbaddr[x] = RMT[x][0]
```

```
        pl_id[x] = RMT[x][1]
        rmt_active[x] = RMT[x][2]

# JSx = [LED0, LED1, SW0, SW1, x, y]
# this is determined by circuit wiring and can't be changed in software
if dictionary:
    JS0 = {
        "led0": Pin.board.D8,
        "led1": Pin.board.D4,
        "sw0": Pin.board.D12,
        "sw1": Pin.board.D10,
        "jsx": 7,
        "jsy": 5
    }
    JS1 = {
        "led0": Pin.board.D9,
        "led1": Pin.board.D5,
        "sw0": Pin.board.D2,
        "sw1": Pin.board.D0,
        "jsx": 6,
        "jsy": 4
    }
    JS2 = {
        "led0": 0x01,
        "led1": 0x02,
        "sw0": 0x04,
        "sw1": 0x08,
        "jsx": 2,
        "jsy": 0
    }
    JS3 = {
        "led0": 0x20,
        "led1": 0x10,
        "sw0": 0x80,
        "sw1": 0x40,
        "jsx": 3,
        "jsy": 1
    }
else:
    JS0 = [Pin.board.D8, Pin.board.D4, Pin.board.D12, Pin.board.D10, 7, 5]
    JS1 = [Pin.board.D9, Pin.board.D5, Pin.board.D2, Pin.board.D0, 6, 4]
    JS2 = [0x01, 0x02, 0x04, 0x08, 2, 0]
    JS3 = [0x20, 0x10, 0x80, 0x40, 3, 1]

JS = [JS0, JS1, JS2, JS3]

LED0 = [None] * 4
LED1 = [None] * 4
SW0 = [None] * 4
SW1 = [None] * 4
xch = [None] * 4
ych = [None] * 4

for x in remotes:
    if dictionary:
        LED0[x] = JS[x]["led0"]
        LED1[x] = JS[x]["led1"]
        SW0[x] = JS[x]["sw0"]
        SW1[x] = JS[x]["sw1"]
        xch[x] = JS[x]["jsx"]
        ych[x] = JS[x]["jsy"]
    else:
        LED0[x] = JS[x][0]
        LED1[x] = JS[x][1]
        SW0[x] = JS[x][2]
        SW1[x] = JS[x][3]
        xch[x] = JS[x][4]
        ych[x] = JS[x][5]
```

```
for x in SW0:
    if isinstance(x, Pin):
        Pin(x, Pin.OUT, value=1)
for x in SW1:
    if isinstance(x, Pin):
        Pin(x, Pin.OUT, value=1)
for x in LED0:
    if isinstance(x, Pin):
        Pin(x, Pin.IN, Pin.PULL_UP)
    else:
        ppemask = ppemask + x
for x in LED1:
    if isinstance(x, Pin):
        Pin(x, Pin.IN, Pin.PULL_UP)
    else:
        ppemask = ppemask + x

ACT = Pin(Pin.board.D3, Pin.OUT)

i2c = I2C(1, freq=400000)

ba = bytearray(4)
barray = bytearray(8)
oldbap = bytearray(8)
oldbas = bytearray(8)
oldba3 = bytearray(8)
oldba4 = bytearray(8)
i2cbuf = bytearray(8)
buffers = [oldbap, oldbas, oldba3, oldba4]

#   bytearray structure
#   0 - payload id, 77 for primary, 182 for secondary
#   1 - joystick "X" position (0-255)
#   2 - joystick "Y" position (0-255)
#   3 - pushbutton 1 (green) status (0/1)
#   4 - pushbutton 2 (red) status (0/1)
#   5 - unused
#   6 - led 1 (green) control (0=on, 1=off)
#   7 - led 2 (red) control (0=on, 1=off)

ACT.value(act_off)


def dacval(pwmch, adcval):
    i2cbuf[0] = 0x30 + pwmch
    i2cbuf[1] = adcval
    i2cbuf[2] = 0
    if debug:
        print("DAC: ", pwmch, adcval, binascii.hexlify(i2cbuf, "/"))
    try:
        i2c.writeto(dac_addr, i2cbuf[0:3])
    except Exception as e:
        print("Error writing I2C to DAC board: %s" % str(e))
    return


def ppeini():
    # address 00: configures pins for input (1) or output (0)
    # address 06: configures input pins for pullup (1)
    # address 09: configures starting port value
    i2cbuf[0:8] = b'\x00\x00\x06\x00\x09\x00\x00\x00'
    i2cbuf[1] = ppemask
    i2cbuf[3] = ppemask
    i2cbuf[5] = 255 - ppemask
    if debug:
        print(hex(ppemask), binascii.hexlify(i2cbuf, "/"))
    try:
```

```
        i2c.writeto(ppe_addr, i2cbuf[0:2])
        i2c.writeto(ppe_addr, i2cbuf[2:4])
        i2c.writeto(ppe_addr, i2cbuf[4:6])
    except Exception as e:
        print("Unable to initialize I2C PPE port: %s" % str(e))
    i2cbuf[0:8] = b'\0\0\0\0\0\0\0\0'
    return 0


def ppeval(mask, value, shadow):
    newvalue = shadow & (0xff - mask)
    if value != 0:
        newvalue = newvalue | mask
    return newvalue


def ppeout(value):
    i2cbuf[0] = 0x09
    i2cbuf[1] = value
    try:
        i2c.writeto(ppe_addr, i2cbuf[0:2])
    except Exception as e:
        print("Error writing to PPE: %s" % str(e))
    return


def ppeinp():
    try:
        i2cbuf[0:1] = i2c.readfrom_mem(ppe_addr, 9, 1)
    except Exception as e:
        print("Error reading from PPE: %s" % str(e))
    return i2cbuf[0]


def transmit(destaddr, desterr):
    if debug:
        print("T", hex(destaddr), binascii.hexlify(barray, "/"))
    try:
        xbee.transmit(destaddr, barray)
    except Exception as e:
        desterr = desterr + 1
        print(desterr, "Transmit failure: %s" % str(e))
    return desterr


# Initialize PWM outputs for primary and secondary X-Y
# Initialize PPE inputs
ppeini()

for x in xch:
    dacval(x, 128)
for y in ych:
    dacval(y, 128)

send_ct = [0, 0, 0, 0]
act_timer = 0
ppeinold = 0

while True:
    # Check for changes to LED outputs from HA-8-3
    # primary channel (Player 1)
    # turn off activity indicator after delay
    if act_timer > 0:
        act_timer = act_timer - 1
        if act_timer == 0:
            ACT.value(act_off)
    for x in remotes:
        if rmt_active[x]:
```

```
            barray[0] = pl_id[x]
            if isinstance(LED0[x], Pin):
                barray[6] = LED0[x].value()
                barray[7] = LED1[x].value()
            else:
                if x == 2:
                    ppeinput = ppeinp()
                    barray[6] = ppeinput & LED0[x]
                    barray[7] = ppeinput & LED1[x]
            # send_ct is used to turn off transmit attempts after ten consecutive errors
            if (barray != buffers[x]) & (send_ct[x] <= 10):
                send_ct[x] = transmit(xbaddr[x], send_ct[x])
                buffers[x][0:8] = barray[0:8]

    #   Now check for incoming messages from wireless controller(s)
    received_msg = xbee.receive()
    if received_msg:
        # turn on activity indicator
        ACT.value(act_on)
        act_timer = act_delay
        # Get the sender's 16-bit address and payload from the received message.
        payload = received_msg['payload']
        sender = received_msg['sender_nwk']
        pl_type = payload[0]

        if debug:
            print("R", hex(sender), binascii.hexlify(payload, "/"))
            #   process incoming message based on primary/secondary indicator in byte 0 of
message

        for x in remotes:
            if sender == xbaddr[x] and pl_type == pl_id[x]:
                rmt_active[x] = True
                send_ct[x] = 0
                if debug:
                    print(x,payload[1],payload[2],255-payload[2])
                dacval(xch[x], payload[1])
                dacval(ych[x], 255 - payload[2])
                if isinstance(SW0[x], Pin):
                    SW0[x].value(payload[3])
                    SW1[x].value(payload[4])
                else:
                    ppe_shadow = ppeval(SW0[x], payload[3], ppe_shadow)
                    ppe_shadow = ppeval(SW1[x], payload[4], ppe_shadow)
                    ppeout(ppe_shadow)
```