Final Master's Project

University Master's Degree in Industrial Engineering

Hart slave emulator based on MicroPython

ANNEX

Author: Pau Ferrer Almirall
Supervisor: Manuel Moreno Eguílaz

Date: June 2020

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Summary

1. Emulator files

1.1. HartSlaveEmulator.py

```
# Pau Ferrer Almirall
# UPC BarcelonaTech
# Juny 2020
# Last modification: 06/06/2020

# Multiple HART sensors are connected in multi-drop mode
# Threading is used to emulate live sensors

# RTS = 0 means reception
# RTS = 1 means transmission

import time
import serial
import taskrepetitivethreads as task
import HartSlave
import HartUnits
import HartStatusErrors
import json
import glob


class Emulator:
    def __init__(self):
        self.state = 0              'Process state counter'
        self.delimiter = 0          'Delimiter byte'
        self.preamble_counter = 0   'Number of preamble bytes acknowledged'
        self.address_counter = 0    'Number of address bytes processed'
        self.data_counter = 0       'Number of data bytes processed'
        self.dlc = 0                'Data counter byte'
        self.command = 0            'Command byte'
        self.checksum = 0           'Checksum byte'
        self.address = [0,0,0,0,0]  'Address bytes'
        self.polladdress = 0        'Polling address byte'
        self.deviceID = [0,0,0]     'Device ID bytes'
        self.expdevtype = [0,0]     'Expanded Device Type bytes'
        self.burst = 0              'Burst flag bit'
        self.master = 0             'Master address bit'
        self.data = []              'Data bytes'
        self.status0 = 0            'First status byte'
        self.status1 = 0            'Second status byte'
        self.slavesdef = []         'Array with original slave definition'
        self.slaves=[]              'Array with live slave objects'
        self.ser = serial.Serial(   #Serial definition as user requires.
        port = 'COM2',
```

```
        baudrate = 1200,
        parity = serial.PARITY_ODD,
        stopbits = serial.STOPBITS_ONE,
        bytesize = serial.EIGHTBITS
        )
        self.ser.setRTS(False) #Reception
        print("Starting Slave Hart Simulator...")

        #Create Hart Slaves:
        #Option 1: from a single description file.
        #with open('HartDD.txt', 'r') as file:
        #    self.slavesdef = json.load(file)

        #Option 2: from multiple description files.
        path = 'C:/Users/Pachuli/Desktop/TFM/HartEmulator/*.txt'
        files=glob.glob(path)
        for file in files:
            with open(file , 'r') as DDfile:
                self.slavesdef.append(json.load(DDfile))
        for i in self.slavesdef:
            self.slaves.append(HartSlave.Slave(i))

        #Initialize UART Reception Thread
        self.threadRX= task.threadloopstartstop(0.001,False,self.newdata)
        self.threadRX.start()

    #New data aquisition method:
    def newdata(self):
        if self.ser.inWaiting() > 0:
            data = ord(self.ser.read(1))
            self.process(data)

    def process(self,data):
        if self.state==0: #preamble
            if data==0xFF:
                self.preamble_counter = self.preamble_counter + 1
                if self.preamble_counter >= 5:
                    self.state = 1
            else:
                self.preamble_counter = 0
        elif self.state==1: #Address aquisition.
            self.delimiter = data
            if data==0x82:  #Master question long frame.
                self.checksum = data
                self.state = 2
            elif data==0x02:    #Master question short frame.
                self.checksum = data
                self.state = 7
            elif data==0xFF: #If more than 5 preambles are included.
                self.preamble_counter = self.preamble_counter + 1
                self.state = 1
            else:
                self.state = 0
```

```
        elif self.state==2: #address long frame.
            self.checksum = self.checksum ^ data
            self.address[self.address_counter] = data
            self.address_counter = self.address_counter + 1
            if self.address_counter == 5:
                self.address_counter = 0
                self.deviceID = self.address[2:]
                self.expdevtype = self.address[:2]
                self.master = self.address[0] >> 7
                self.burst = (self.address[0] >> 6)^(self.master << 1)
                self.expdevtype[0] = self.expdevtype[0] ^ (self.burst << 6)
^ (self.master << 7)
                self.state = 3
        elif self.state==3: #STX command recognition
            self.command = data
            self.checksum = self.checksum ^ data
            if data== 0: #Get unique ID
                self.state = 4
            elif data == 1: #Read primary variable command
                self.state = 4
            elif data == 2: #Read Loop Current And Percent Of Range
                self.state = 4
            elif data == 3: #Read Dynamic Variables And Percent Of Range
(Demo for 1 variable)
                self.state = 4
            elif data == 6: #write polling address
                self.state = 4
            elif data == 11: #Get unique ID by tag
                self.state = 4
            elif data == 14: #Get primary variable transducer information
                self.state = 4
            elif data == 33: #Read device variables
                self.state = 4
            elif data == 48: #Read Additional Device Status
                self.state = 4
            elif data == 140 and self.expdevtype[0] == 0x0D and
self.expdevtype[1] == 0x28: #Device specific command: status information
                self.state = 4
            else:
                self.state=0
                self.preamble_counter = 0
                print('Command not supported by device.')

        elif self.state==4: #Data counter byte
            self.checksum = self.checksum ^ data
            self.dlc = data
            if data==0:
                self.state = 6
            else:
                self.state = 5

        elif self.state==5: #STX data bytes
            self.checksum = self.checksum ^ data
```

```
            self.data.append(data)
            self.data_counter = self.data_counter + 1
            if self.data_counter == self.dlc:
                self.data_counter = 0
                self.state = 6

        elif self.state==6: #Checksum validation + ACK data request and
reception
            if self.checksum == data:
                wrong=1
                if self.delimiter >= 0x80 :
                    if self.address == [0,0,0,0,0]:
                        for i in range (len(self.slaves)):
                            if self.data == self.slaves[i].slave['Tag']:

self.sendframe(self.slaves[i].getanswer(self.command,self.data))
                                wrong=0
                        if wrong:
                            print('No tag match.')
                    elif 1:
                        for i in range (len(self.slaves)):
                            if self.deviceID ==
self.slaves[i].slave['DeviceID'] and (self.expdevtype[0] ^
self.slaves[i].slave['ExpDevType'][0] == 0 or self.expdevtype[0] ^
self.slaves[i].slave['ExpDevType'][0] >= 0x40) and (self.expdevtype[1] ==
self.slaves[i].slave['ExpDevType'][1]):

self.sendframe(self.slaves[i].getanswer(self.command,self.data))
                                wrong=0
                        if wrong:
                            print('No address match.')
                else:
                    for i in range (len(self.slaves)):
                        if self.polladdress ==
self.slaves[i].slave['PollAddress']:

self.sendframe(self.slaves[i].getanswer(self.command,self.data))
                                wrong=0
                    if wrong:
                        print('No polling address match.')

            else:
                print("Wrong Checksum")
            self.state = 0
            self.preamble_counter = 0
            self.address_counter = 0
            self.data_counter = 0
            self.data=[]
            self.address=[0,0,0,0,0]

        elif self.state==7:
            self.checksum = self.checksum ^ data
            self.address = [data]
```

```
            self.master = self.address[0] >> 7
            self.burst = ( self.address[0] >> 6 ) ^ (self.burst ^
(self.master << 1))
            self.polladdress = self.address[0] ^ (self.burst << 6) ^
(self.master << 7)
            #print (self.polladdress)
            self.state = 3


    def sendframe(self,data): #ACK framing
        self.checksum = 0
        step = 0.001
        self.ser.setRTS(True) #TX mode (Hart modem)
        time.sleep(0.2) #Wait for a while until sending a new byte
        ack=bytearray()
        for i in range(5): #5 to 20, but generally 5
            ack.append(0xFF)
        ack.append(self.delimiter ^ 0x04)
        self.checksum = self.checksum ^ (self.delimiter ^ 0x04)
        for i in self.address:
            ack.append(i)
            self.checksum = self.checksum ^ i
        ack.append(self.command)
        self.checksum = self.checksum ^ self.command
        ack.append(len(data))
        self.checksum = self.checksum ^ len(data)
        for i in data:
            ack.append(i)
            self.checksum = self.checksum ^ i
        ack.append(self.checksum)
        self.ser.write(ack)
        self.ser.setRTS(False) #RX mode
        print("send a new response to master")



if __name__ == "__main__":
    import time

    Emu = Emulator()

    try:
        while True:
            pass
    except KeyboardInterrupt:
        print('Game Over')
```

1.2. HartSlave.py

```
# PID class for Hart Slave
# Pau Ferrer Almirall
# UPC BarcelonaTech
# Juny 2020
# Last modification: 06/06/2020

import taskrepetitivethreads as task
import struct
import HartUnits
import HartStatusErrors
import HartDeviceVariableCodes as HDVC
import time
import math

class Slave:
    def __init__(self, data):
        self.slave = data       'Data contained in the STX frame.'
        self.PrimaryVar = 0     'Auxiliary variable to store new PV values.'
        self.Status = [0,0]     'Device status to be sent in the ACK frame.'

        self.threadPV= task.threadloopstartstop(1,False,self.PVupdate)
        self.threadPV.start()

    def getanswer(self,command,data):
        if command == 0 or command == 11: #Get Long Address.
            self.Status = self.getstatus()
            return [self.Status[0], self.Status[1], 0xFE, self.slave['ExpDevType'][0],
self.slave['ExpDevType'][1], 0x05, 0x07, 0x00, 0x00, 0x00, 0x01, self.slave['DeviceID'][0],
self.slave['DeviceID'][1], self.slave['DeviceID'][2], 0x05, 0x01, 0x00, 0x00,
self.slave['ExpDevStatus'], self.slave['ManufID'][0], self.slave['ManufID'][1], 0x00, 0x00,
0x01]

        elif command == 1: #Get Primary Variable.
            ans = self.getstatus()
            ans.append(self.slave['Units'])
            b = struct.pack('f', self.PrimaryVar)
            for i in reversed(b): ans.append(i)
            return ans

        elif command == 2: #Read Loop Current And Percent Of Range.
            ans = self.getstatus()
            b=struct.pack('f', self.slave['PrimaryVariableLoopCurrent'])
            for i in reversed(b): ans.append(i)
            b=struct.pack('f', (self.slave['PrimaryVariableLoopCurrent']-4)/16)
            for i in reversed(b): ans.append(i)
            return ans

        elif command == 3: #Read Dynamic Variables And Percent Of Range (Demo for 1
variable).
            ans = self.getstatus()
            b=struct.pack('f', self.slave['PrimaryVariableLoopCurrent'])
```

```
        for i in reversed(b): ans.append(i)
        ans.append(self.slave['Units'])
        b=struct.pack('f', self.PrimaryVar)
        for i in reversed(b): ans.append(i)
        return ans

    elif command == 6: #Write polling address.
        ans = self.getstatus()
        self.slave['PollAddress'] = data[0]
        ans.append(self.slave['PollAddress'])
        ans.append(0x00)
        return ans

    elif command == 14: #Get Primary Variable.
        ans = self.getstatus()
        for i in self.slave['DeviceID']:
            ans.append(i)
        ans.append(self.slave['Units'])
        b = struct.pack('f', 0.1)
        for i in reversed(b): ans.append(i)
        b = struct.pack('f', 50.2)
        for i in reversed(b): ans.append(i)
        b = struct.pack('f', 10.0)
        for i in reversed(b): ans.append(i)
        return ans

    elif command == 33: #Read device variables.
        ans = self.getstatus()
        num = len(data)
        ans.append(HDVC.PrimaryVariable)
        ans.append(self.slave['Units'])
        b = struct.pack('f', self.PrimaryVar)
        value = [0x7F, 0xA0, 0x00, 0x00]
        for i in reversed(b): ans.append(i)
        if num > 1:
            ans.append(HDVC.SecondaryVariable)
            ans.append(0xFA)
            for i in reversed(value): ans.append(i)
            if num > 2:
                ans.append(HDVC.TertiaryVariable)
                ans.append(0xFA)
                for i in reversed(value): ans.append(i)
                if num > 3:
                    ans.append(HDVC.QuaternaryVariable)
                    ans.append(0xFA)
                    for i in reversed(value): ans.append(i)
                    return ans
                else:
                    return ans
            else:
                return ans
        else:
            return ans
```

```
        elif command == 48: #Read Additional Device Status.
            ans = self.getstatus()
            for i in range (6):
                ans.append(0x00)
            ans.append(self.slave['ExpDevStatus'])
            ans.append (0x01)
            for i in range (3):
                ans.append(0x00)
            for i in range (3):
                ans.append(0x00)
            for i in range (6):
                ans.append(0x00)
            return ans

        elif command == 140: #Device specific command: status information.
            ans = self.getstatus()
            for i in range (17):
                ans.append(0x00)
            return ans

    def PVupdate(self): #Method run in thread. Live update of PV value.

self.PrimaryVar=self.slave['PrimaryVariable']*(1+0.1*math.sin(math.pi*time.process_time()/5))

    def getstatus(self): #Status bytes.
        Status = [0,0]
        if self.slave['CommsStatus'] == 0:
            Status[0] = self.slave['ResponseCode']
            return Status
        else:
            Status[0] = self.slave['CommsStatus']
            Status[1] = self.slave['FieldDevStatus']
            return Status


if __name__ == "__main__":
    import time
    with open('HartDD.txt', 'r') as file:
            slaves = json.load(file)
    hart1 = Slave(slaves[0])
    hart2 = Slave(slaves[1])
    print(hart1.getanswer(0,[0]))
    print(hart2.getanswer(0,[0]))
    print(hart1.getanswer(1,[0]))
    print(hart2.getanswer(1,[0]))
    print(hart1.getanswer(6,[0x06,0x00]))
    print(hart2.getanswer(6,[0x04,0x00]))
```

1.3. HartDeviceVariableCodes.py

```
#Hart Device Variable Codes

BatteryLife = 0xF3 #Float in days
PercentRange = 0xF4
LoopCurrent = 0xF5
PrimaryVariable = 0xF6
SecondaryVariable = 0xF7
TertiaryVariable = 0xF8
QuaternaryVariable = 0xF9

code = {
    243: 'Battery Life',
    244: 'Percent Range',
    245: 'Loop Current',
    246: 'Primary Variable',
    247: 'Secondary variable',
    248: 'Tertiary Variable',
    249: 'Quaternary Variable'
}
```

1.4. HartStatusErrors.py

```
#Hart Status & Error Codes
#Pau Ferrer
#Juny 2020. This implementations does not support multiple
communication errors. To indicate multiple comunication errors,
each bit should be treated individually.


#Field Device Status

StatusOK                =   0x00
DeviceMalfunction       =   0x80
ConfigurationChanged    =   0x40
ColdStart               =   0x20
MoreStatusAvalaible     =   0x10
LoopCurrentFixed        =   0x08
LoopCurrentSaturated    =   0x04
NonPVOutOfLimits        =   0x02
PVOutOfLimits           =   0x01

#Comunication Status

NonErrors               =   0x00
CommunicationError      =   0x80 #bit always set to indicate a
communication error
VirticalParityError     =   0xC0
OverrunError            =   0xA0
FramingError            =   0x90
```

```
LongitudinalParityError =   0x88
Reserved1               =   0x84
BufferOverflow          =   0x82
Reserved0               =   0x81

#Response Codes

Notification            =   0x00
warning                 =   0x02 #Command executed with some kind
of deviation that should be described in the response. Response
Data Bytes are returned.

#Extended Device Status Codes
OK = 0x00
MaintenanceRequired = 0x01
DeviceVariableAlert = 0x02
CriticalPowerFailure = 0x04
```

1.5. HartUnits.py

```
#Hart Units (some of them)

#Pressure
bars =          7
millibars =     8
pascals =       11
kilopascals =   12
Megapascals =   237

#Temperature
Celsius =       32
Fahrenheit =    33
Rankin =        34
Kelvin =        35

NotUsed =       250

units = {
    7: 'Bars',
    8: 'Millibars',
    11: 'Pascals',
    12: 'Kilopascals',
    237: 'Megapascals',
    32: 'Celcius',
    33: 'Fahrenheit',
    34: 'Rankin',
    35: 'Kelvin',
    250: 'Not Used'
}
```

1.6. taskrepetitivethreads.py

```
import time
import threading

class threadloopstartstop(threading.Thread):
    def __init__ (self, interval, default, function, args=[], kwargs={}):
        threading.Thread.__init__(self)
        self.interval = interval
        self.function = function
        self.args = args
        self.kwargs = kwargs
        self.finished = threading.Event()
        self.flagstop = default

    def shutdown(self):
        self.finished.set()
        self.join()

    def stop(self):
        self.flagstop = True

    def restart(self):
        self.flagstop = False

    def isstopped(self):
        return self.flagstop

    def run(self):
        while not self.finished.isSet():
            self.finished.wait(self.interval)
            if not self.flagstop:
                self.function(*self.args, **self.kwargs)

if __name__ == "__main__":
    def my_function (a, b, c):
        print("Here I am:", a, b, c)
    print("Calling my_function() in a thread every 1/10th of second for two
seconds.")
    t = threadloopstartstop(0.5, False, my_function, (1,0,-1))
    t.start()
    print('start at:',time.asctime())
    time.sleep(2)
    print('stop at:',time.asctime())
    t.stop()
    print('restart at:',time.asctime())
    t.restart()
    time.sleep(2)
    print('stop at:',time.asctime())
    t.stop()
    print('shutdown at:',time.asctime())
    t.shutdown()
    print("Done!")
```

2. MicroPython implementation

2.1. main.py

```
# main.py -- put your code here!
# Pau Ferrer Almirall
# UPC BarcelonaTech
# Juny 2020
# Last modification: 15/06/2020
# MicroPython implementation for HartSlaveSimulator.py

import time
##import serial
##import taskrepetitivethreads as task
import HartSlave
import HartUnits
import HartStatusErrors
import json
import os
import _thread
import pyb


class Emulator:
    def __init__(self):
        self.state = 0
        self.delimiter = 0
        self.preamble_counter = 0
        self.address_counter = 0
        self.data_counter = 0
        self.dlc = 0
        self.command = 0
        self.checksum = 0
        self.address = [0,0,0,0,0]
        self.polladdress = 0
        self.deviceID = [0,0,0]
        self.expdevtype = [0,0]
        self.burst = 0
        self.master = 0
        self.data = []
        self.status0 = 0
        self.status1 = 0
        self.slavesdef = []
        self.USB = pyb.USB_VCP()
##        self.uart = pyb.UART(1, 9600)
##        self.uart.init(9600, bits=8, parity=None, stop=1)
##        self.ser = serial.Serial(
##        port = 'COM3',
##        baudrate = 1200,
##        parity = serial.PARITY_ODD,
```

```
##        stopbits = serial.STOPBITS_ONE,
##        bytesize = serial.EIGHTBITS
##        )
##        self.ser.setRTS(False) #Reception

        #print('Starting Slave Hart Simulator...')

        #Create Hart Slaves
        #with open('HartDD.txt', 'r') as file:
        #    self.slavesdef = json.load(file)


        files=os.listdir()
        for file in files:
            if file.startswith('HartDD') and file.endswith('.txt'):
                print(file)
                with open(file , 'r') as DDfile:
                    self.slavesdef.append(json.load(DDfile))

        self.slaves=[]
        for i in self.slavesdef:
            self.slaves.append(HartSlave.Slave(i))

        #Initialize UART Reception Thread
        #_thread.start_new_thread(self.newdata, ())
##        self.threadRX= task.threadloopstartstop(0.001,False,self.newdata)
##        self.threadRX.start()
        #print ('init done.')

##    def newdata(self):
##        if self.USB.any():
##            data = ord(self.USB.recv(1, timeout=5000))
##            self.process(data)

        while (True):
            if self.USB.any():
                data = ord(self.USB.recv(1, timeout=5000))
                self.process(data)

    def process(self,data):
        if self.state==0: #preamble
            if data==0xFF:
                self.preamble_counter = self.preamble_counter + 1
                if self.preamble_counter >= 5:
                    self.state = 1
            else:
                self.preamble_counter = 0
        elif self.state==1:
            self.delimiter = data
            if data==0x82:  #Master question long frame
                self.checksum = data
                self.state = 2
            elif data==0x02:    #Master question short frame
```

```
                self.checksum = data
                self.state = 7
            elif data==0xFF:
                self.preamble_counter = self.preamble_counter + 1
                self.state = 1
            else:
                self.state = 0
        elif self.state==2: #address long frame
            self.checksum = self.checksum ^ data
            self.address[self.address_counter] = data
            self.address_counter = self.address_counter + 1
            if self.address_counter == 5:
                self.address_counter = 0
                self.deviceID = self.address[2:]
                self.expdevtype = self.address[:2]
                self.master = self.address[0] >> 7
                self.burst = (self.address[0] >> 6)^(self.master << 1)
                self.expdevtype[0] = self.expdevtype[0] ^ (self.burst << 6) ^
(self.master << 7)
                self.state = 3
        elif self.state==3: #command
            self.command = data
            self.checksum = self.checksum ^ data
            if data== 0: #Get unique ID
                self.state = 4
            elif data == 1: #Read primary variable command
                self.state = 4
            elif data == 2: #Read Loop Current And Percent Of Range
                self.state = 4
##              elif data == 3: #Read Dynamic Variables And Percent Of Range (Demo
for 1 variable)
##                  self.state = 4
            elif data == 6: #write polling address
                self.state = 4
            elif data == 11: #Get unique ID by tag
                self.state = 4
            elif data == 14: #Get primary variable transducer information
                self.state = 4
            elif data == 33: #Read device variables
                self.state = 4
            elif data == 48: #Read Additional Device Status
                self.state = 4
            elif data == 140 and self.expdevtype[0] == 0x0D and
self.expdevtype[1] == 0x28: #Device specific command: status information
                self.state = 4
            else:
                self.state = 0
                self.preamble_counter = 0
                self.address_counter = 0
                self.data_counter = 0
                self.data=[]
                self.address=[0,0,0,0,0]
                #print('Command not supported by device.')
```

```
        elif self.state==4: #dlc
            self.checksum = self.checksum ^ data
            self.dlc = data
            if data==0:
                self.state = 6
            else:
                self.state = 5

        elif self.state==5: #data
            self.checksum = self.checksum ^ data
            self.data.append(data)
            self.data_counter = self.data_counter + 1
            if self.data_counter == self.dlc:
                self.data_counter = 0
                self.state = 6

        elif self.state==6: #checksum
            if self.checksum == data:
                wrong=1
                if self.delimiter >= 0x80 :
                    if self.address == [0,0,0,0,0]:
                        for i in range (len(self.slaves)):
                            if self.data == self.slaves[i].slave['Tag']:

self.sendframe(self.slaves[i].getanswer(self.command,self.data))
                                wrong=0
                        if wrong:
                            self.state = 0
                            self.preamble_counter = 0
                            self.address_counter = 0
                            self.data_counter = 0
                            self.data=[]
                            self.address=[0,0,0,0,0]
                            #print('No tag match.')
                    elif 1:
                        for i in range (len(self.slaves)):
                            if self.deviceID == self.slaves[i].slave['DeviceID']
and (self.expdevtype[0] ^ self.slaves[i].slave['ExpDevType'][0] == 0 or
self.expdevtype[0] ^ self.slaves[i].slave['ExpDevType'][0] >= 0x40) and
(self.expdevtype[1] == self.slaves[i].slave['ExpDevType'][1]):

self.sendframe(self.slaves[i].getanswer(self.command,self.data))
                                wrong=0
                        if wrong:
                            self.state = 0
                            self.preamble_counter = 0
                            self.address_counter = 0
                            self.data_counter = 0
                            self.data=[]
                            self.address=[0,0,0,0,0]
                            #print('No address match.')
                else:
```

```
                    for i in range (len(self.slaves)):
                        if self.polladdress ==
self.slaves[i].slave['PollAddress']:

self.sendframe(self.slaves[i].getanswer(self.command,self.data))
                            wrong=0
                    if wrong:
                        self.state = 0
                        self.preamble_counter = 0
                        self.address_counter = 0
                        self.data_counter = 0
                        self.data=[]
                        self.address=[0,0,0,0,0]
                        #print('No polling address match.')

            else:
                print("Wrong Checksum")
            self.state = 0
            self.preamble_counter = 0
            self.address_counter = 0
            self.data_counter = 0
            self.data=[]
            self.address=[0,0,0,0,0]

        elif self.state==7:
            self.checksum = self.checksum ^ data
            self.address = [data]
            self.master = self.address[0] >> 7
            self.burst = ( self.address[0] >> 6 ) ^ (self.burst ^ (self.master <<
1))
            self.polladdress = self.address[0] ^ (self.burst << 6) ^ (self.master
<< 7)
            #print (self.polladdress)
            self.state = 3


    def sendframe(self,data): #answer from slave to the HART master
        self.checksum = 0
        step = 0.001
##        self.ser.setRTS(True) #TX mode (Hart modem)
        time.sleep(0.2) #Wait for a while until sending a new byte
        ack=bytearray()
        for i in range(5): #5 or 4
            ack.append(0xFF)
        ack.append(self.delimiter ^ 0x04)
        self.checksum = self.checksum ^ (self.delimiter ^ 0x04)
        for i in self.address:
            ack.append(i)
            self.checksum = self.checksum ^ i
        ack.append(self.command)
        self.checksum = self.checksum ^ self.command
        ack.append(len(data))
        self.checksum = self.checksum ^ len(data)
```

```
        for i in data:
            ack.append(i)
            self.checksum = self.checksum ^ i
        ack.append(self.checksum)
        #print(ack)
        self.USB.send(ack)
##        self.ser.setRTS(False) #RX mode
        #print("send a new response to master")



if __name__ == "__main__":
    import time

    Emu = Emulator()

    try:
        while True:
            pass
    except:
        pass
```

2.2. HartSlave.py

```
# PID class for Hart Slave. MicroPython implementation.
# Pau Ferrer Almirall
# UPC BarcelonaTech
# Juny 2020
# Last modification: 15/06/2020

##import taskrepetitivethreads as task
import struct
import HartUnits
import HartStatusErrors
import HartDeviceVariableCodes as HDVC
import time
import math
import _thread

class Slave:
    """Common base class for all Slaves"""
    def __init__(self, data):
        self.slave = data
        self.PrimaryVar = 0
        self.Status = [0,0]

        _thread.start_new_thread(self.PVupdate, ())
##        self.threadPV= task.threadloopstartstop(1,False,self.PVupdate)
##        self.threadPV.start()

    def getanswer(self,command,data):
```

```
        if command == 0 or command == 11: #Get Long Address
            self.Status = self.getstatus()
            return [self.Status[0], self.Status[1], 0xFE,
self.slave['ExpDevType'][0], self.slave['ExpDevType'][1], 0x05, 0x07, 0x00,
0x00, 0x00, 0x01, self.slave['DeviceID'][0], self.slave['DeviceID'][1],
self.slave['DeviceID'][2], 0x05, 0x01, 0x00, 0x00, self.slave['ExpDevStatus'],
self.slave['ManufID'][0], self.slave['ManufID'][1], 0x00, 0x00, 0x01]

        elif command == 1: #Get Primary Variable
            ans = self.getstatus()
            ans.append(self.slave['Units'])
            b = struct.pack('f', self.PrimaryVar)
            for i in reversed(b): ans.append(i)
            return ans

        elif command == 2: #Read Loop Current And Percent Of Range
            ans = self.getstatus()
            b=struct.pack('f', self.slave['PrimaryVariableLoopCurrent'])
            for i in reversed(b): ans.append(i)
            b=struct.pack('f', (self.slave['PrimaryVariableLoopCurrent']-4)/16)
            for i in reversed(b): ans.append(i)
            return ans

        elif command == 3: #Read Dynamic Variables And Percent Of Range (Demo
for 1 variable)
            ans = self.getstatus()
            b=struct.pack('f', self.slave['PrimaryVariableLoopCurrent'])
            for i in reversed(b): ans.append(i)
            ans.append(self.slave['Units'])
            b=struct.pack('f', self.PrimaryVar)
            for i in reversed(b): ans.append(i)
            return ans

        elif command == 6: #write polling address
            ans = self.getstatus()
            self.slave['PollAddress'] = data[0]
            ans.append(self.slave['PollAddress'])
            ans.append(0x00)
            return ans

        elif command == 14: #Get Primary Variable
            ans = self.getstatus()
            for i in self.slave['DeviceID']:
                ans.append(i)
            ans.append(self.slave['Units'])
            b = struct.pack('f', 0.1)
            for i in reversed(b): ans.append(i)
            b = struct.pack('f', 50.2)
            for i in reversed(b): ans.append(i)
            b = struct.pack('f', 10.0)
            for i in reversed(b): ans.append(i)
            return ans
```

```
        elif command == 33: #Read device variables
            ans = self.getstatus()
            num = len(data)
            ans.append(HDVC.PrimaryVariable)
            ans.append(self.slave['Units'])
            b = struct.pack('f', self.PrimaryVar)
            value = [0x7F, 0xA0, 0x00, 0x00]
            for i in reversed(b): ans.append(i)
            if num > 1:
                ans.append(HDVC.SecondaryVariable)
                ans.append(0xFA)
                for i in reversed(value): ans.append(i)
                if num > 2:
                    ans.append(HDVC.TertiaryVariable)
                    ans.append(0xFA)
                    for i in reversed(value): ans.append(i)
                    if num > 3:
                        ans.append(HDVC.QuaternaryVariable)
                        ans.append(0xFA)
                        for i in reversed(value): ans.append(i)
                        return ans
                    else:
                        return ans
                else:
                    return ans
            else:
                return ans

        elif command == 48: #Read Additional Device Status
            ans = self.getstatus()
            for i in range (6):
                ans.append(0x00)
            ans.append(self.slave['ExpDevStatus'])
            ans.append (0x01)
            for i in range (3):
                ans.append(0x00)
            for i in range (3):
                ans.append(0x00)
            for i in range (6):
                ans.append(0x00)
            return ans

        elif command == 140: #Device specific command: status information
            ans = self.getstatus()
            for i in range (17):
                ans.append(0x00)
            return ans

    def PVupdate(self):
        self.PrimaryVar=self.slave['PrimaryVariable']*(1+
0.1*math.sin(math.pi*time.time()/5))

    def getstatus(self):
```

```
        Status = [0,0]
        if self.slave['CommsStatus'] == 0:
            Status[0] = self.slave['ResponseCode']
            return Status
        else:
            Status[0] = self.slave['CommsStatus']
            Status[1] = self.slave['FieldDevStatus']
            return Status


if __name__ == "__main__":
    import time
    with open('HartDD.txt', 'r') as file:
            slaves = json.load(file)
    hart1 = Slave(slaves[0])
    hart2 = Slave(slaves[1])
    print(hart1.getanswer(0,[0]))
    print(hart2.getanswer(0,[0]))
    print(hart1.getanswer(1,[0]))
    print(hart2.getanswer(1,[0]))
    print(hart1.getanswer(6,[0x06,0x00]))
    print(hart2.getanswer(6,[0x04,0x00]))
```

3. Other improvised tolos

3.1. HartMaster.py

```
"""
Pau Ferrer Almirall
UPC BarcelonaTech
Juny 2020
Last modification: 06/06/2020

Improvised master simulator.
Enter com#a%, with # the command number to send and % the addres number in the
simulated slaves list.

"""

import serial
import time
import struct
import HartUnits
import HartDeviceVariableCodes as HDVC

def xor(l):
    r = 0x00
    for v in l: r = r ^ v
    return r

def read1():
        if ser.read() == b'\xff':
            a=ser.read()
            while a == b'\xff':
                a=ser.read()
            a=list(a)
            checksum = a[0]
            address = bytearray()
            for i in range(5):
                a = ser.read()
                a = list(a)
                address.append(a[0])
                checksum = checksum ^ a[0]

            a=ser.read()
            a=list(a)
            if a[0] == 1:
                checksum = checksum ^ a[0]
                b=ser.read()
                b=list(b)
                checksum = checksum ^ b[0]
                status = bytearray()
                units = bytearray()
```

```
                data = bytearray()
                a = ser.read(2)
                a=list(a)
                checksum = checksum ^ a[0] ^ a[1]
                status.append(a[0])
                status.append(a[1])
                a = ser.read()
                a=list(a)
                checksum = checksum ^ a[0]
                units.append(a[0])
                for i in range(b[0]-3):
                    a = ser.read()
                    a=list(a)
                    checksum = checksum ^ a[0]
                    data.append(a[0])
                a = ser.read()
                a=list(a)
                if a[0] == checksum:
                    pvar = struct.unpack('>f',data)
                else:
                    print('Wrong checksum')
                    pass
            else:
                print('Wrong command read')
                pass
            print ('Address:' )
            print(bytes(address))
            print ('Status:')
            print(bytes(status))
            print ('Units:')
            print(HartUnits.units[units[0]])
            print ('PrimaryVariable:')
            print(pvar)
        else:
            pass

ser = serial.Serial()
ser.baudrate = 9600
ser.port = 'COM3'
#parity=serial.PARITY_ODD,
#stopbits=serial.STOPBITS_TWO,
#bytesize=serial.SEVENBITS

ser.open()

print ('Enter your commands below.\r\nInsert "exit" to leave the application.')

raw_input=1
while 1 :
    # get keyboard input
    raw_input = input(">> ")
    if raw_input=='test':
        check=0x82 ^ 0x83 ^ 0x04 ^ 0x01 ^ 0x01 ^ 0x01 ^ 0x01 ^ 0x00
```

```
        for i in range(10):
            stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,0x82,0x83,0x04,0x01,0x01,0x01,
0x01,0x00,check])
            ser.write(stx)
            time.sleep(1)
            read1()

    elif raw_input == 'exit':
        ser.close()
        exit()
    elif raw_input == 'com0a1':
        check=0x82 ^ 0x83 ^ 0x04 ^ 0x01 ^ 0x01 ^ 0x01 ^ 0x00 ^ 0x00
        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,0x82,0x83,0x04,0x01,
0x01,0x01,0x00,0x00,check])
        ser.write(stx)
    elif raw_input == 'com0a2':
        check=0x82 ^ 0xA0 ^ 0xED ^ 0x02 ^ 0x02 ^ 0x02 ^ 0x00 ^ 0x00
        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,0x82,0xA0,0xED,0x02,
0x02,0x02,0x00,0x00,check])
        ser.write(stx)
    elif raw_input == 'com1a1':
        check=0x82 ^ 0x83 ^ 0x04 ^ 0x01 ^ 0x01 ^ 0x01 ^ 0x01 ^ 0x00
        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,0x82,0x83,0x04,0x01,
0x01,0x01,0x01,0x00,check])
        ser.write(stx)
    elif raw_input == 'com1a2':
        check=0x82 ^ 0xA0 ^ 0xED ^ 0x02 ^ 0x02 ^ 0x02 ^ 0x01 ^ 0x00
        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,0x82,0xA0,0xED,0x02,
0x02,0x02,0x01,0x00,check])
        ser.write(stx)
    elif raw_input == 'com6a1':
        print('Enter new polling address for device (0-15):')
        newpoll = int(input(">> "))
        if newpoll < 16 and newpoll >= 0:
            check=0x82 ^ 0x83 ^ 0x04 ^ 0x01 ^ 0x01 ^ 0x01 ^ 0x06 ^ 0x01 ^ newpoll
            stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,0x82,0x83,0x04,
0x01,0x01,0x01,0x06,0x01,newpoll,check])
            ser.write(stx)
        else:
            print('Wrong data input.')
    elif raw_input == 'com6a2':
        print('Enter new polling address for device (0-15):')
        newpoll = int(input(">> "))
        if newpoll < 16 and newpoll >= 0:
            check=0x82 ^ 0xA0 ^ 0xED ^ 0x02 ^ 0x02 ^ 0x02 ^ 0x06 ^ 0x01 ^ newpoll
            stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,0x82,0xA0,
0xED,0x02,0x02,0x02,0x06,0x01,newpoll,check])
            ser.write(stx)
        else:
            print('Wrong data input.')
    elif raw_input == 'com11a1':
        check=0x82 ^ 0x83 ^ 0x04 ^ 0x01 ^ 0x01 ^ 0x01 ^ 0x0b ^ 0x06 ^ 0x01 ^ 0x02 ^
0x03 ^ 0x04 ^ 0x05 ^ 0x06
```

```
        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,0x82,0x83,0x04,0x01,0x01,0x01,
0x0b,0x06,0x01,0x02,0x03,0x04,0x05,0x06,check])
        ser.write(stx)
    elif raw_input == 'com11a2':
        check=0x82 ^ 0xA0 ^ 0xED ^ 0x02 ^ 0x02 ^ 0x02 ^ 0x0b ^ 0x06 ^ 0x07 ^ 0x02 ^
0x03 ^ 0x04 ^ 0x05 ^ 0x06
        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,0x82,0xA0,0xED,0x02,
0x02,0x02,0x0b,0x06,0x02,0x03,0x04,0x05,0x06,0x07,check])
        ser.write(stx)
    elif raw_input == 'com14a1':
        check=0x82 ^ 0x83 ^ 0x04 ^ 0x01 ^ 0x01 ^ 0x01 ^ 0x0e ^ 0x00
        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,0x82,0x83,0x04,0x01,
0x01,0x01,0x0e,0x00,check])
        ser.write(stx)
    elif raw_input == 'com14a2':
        check=0x82 ^ 0xA0 ^ 0xED ^ 0x02 ^ 0x02 ^ 0x02 ^ 0x0e ^ 0x00
        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,0x82,0xA0,0xED,
0x02,0x02,0x02,0x0e,0x00,check])
        ser.write(stx)
    elif raw_input == 'com33a1':
        print('How many Device Variables should be consulted(0-4):')
        num = int(input(">> "))
        if num <= 4 and num >= 1:
            check=0x82 ^ 0x83 ^ 0x04 ^ 0x01 ^ 0x01 ^ 0x01 ^ 0x21
            if num >= 1:
                check = check ^ HDVC.PrimaryVariable
                if num >= 2:
                    check = check ^ HDVC.SecondaryVariable
                    if num >= 3:
                        check = check ^ HDVC.TertiaryVariable
                        if num >= 4:
                            check = check ^ HDVC.QuaternaryVariable ^ 4
                            stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,
0x82,0x83,0x04,0x01,0x01,0x01,0x21,0x04, HDVC.PrimaryVariable,
HDVC.SecondaryVariable,HDVC.TertiaryVariable,HDVC.QuaternaryVariable, check])
                        else:
                            check = check ^ 3
                            stx = bytearray([0xFF,0xFF,0xFF,0xFF,
0xFF,0x82,0x83,0x04,0x01,0x01,0x01,0x21,0x03,HDVC.PrimaryVariable,
HDVC.SecondaryVariable, HDVC.TertiaryVariable,check])
                    else:
                        check = check ^ 2
                        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,
0x82,0x83,0x04,0x01,0x01,0x01,0x21,0x02,HDVC.PrimaryVariable,
HDVC.SecondaryVariable,check])
                else:
                    check = check ^ 1
                    stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,
0x82,0x83,0x04,0x01,0x01,0x01,0x21,0x01,HDVC.PrimaryVariable,check])
            ser.write(stx)
        else:
            print('Unsuported number of variables requested.')
            pass
    elif raw_input == 'com33a2':
```

```
        print('How many Device Variables should be consulted(0-4):')
        num = int(input(">> "))
        if num <= 4 and num >= 1:
            check=0x82 ^ 0xA0 ^ 0xED ^ 0x02 ^ 0x02 ^ 0x02 ^ 0x21
            if num >= 1:
                check = check ^ HDVC.PrimaryVariable
                if num >= 2:
                    check = check ^ HDVC.SecondaryVariable
                    if num >= 3:
                        check = check ^ HDVC.TertiaryVariable
                        if num >= 4:
                            check = check ^ HDVC.QuaternaryVariable ^ 4
                            stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,
0x82,0xA0,0xED,0x02,0x02,0x02,0x21,0x04,HDVC.PrimaryVariable,
HDVC.SecondaryVariable,HDVC.TertiaryVariable,HDVC.QuaternaryVariable, check])
                        else:
                            check = check ^ 3
                            stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,
0x82,0xA0,0xED,0x02,0x02,0x02,0x21,0x03,HDVC.PrimaryVariable,
HDVC.SecondaryVariable,HDVC.TertiaryVariable,check])
                    else:
                        check = check ^ 2
                        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,
0x82,0xA0,0xED,0x02,0x02,0x02,0x21,0x02,HDVC.PrimaryVariable,
HDVC.SecondaryVariable,check])
                else:
                    check = check ^ 1
                    stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,
0x82,0xA0,0xED,0x02,0x02,0x02,0x21,0x01,HDVC.PrimaryVariable,check])
            ser.write(stx)
        else:
            print('Unsuported number of variables requested.')
            pass
    elif raw_input == 'com48a1':
        check=0x82 ^ 0x83 ^ 0x04 ^ 0x01 ^ 0x01 ^ 0x01 ^ 0x30 ^ 0x00
        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,
0x82,0x83,0x04,0x01,0x01,0x01,0x30,0x00,check])
        ser.write(stx)
    elif raw_input == 'com48a2':
                    else:
        check=0x82 ^ 0xA0 ^ 0xED ^ 0x02 ^ 0x02 ^ 0x02 ^ 0x30 ^ 0x00
        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,
0x82,0xA0,0xED,0x02,0x02,0x02,0x30,0x00,check])
        ser.write(stx)
    elif raw_input == 'com140a1':
        check=0x82 ^ 0x83 ^ 0x04 ^ 0x01 ^ 0x01 ^ 0x01 ^ 0x8C ^ 0x00
        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,
0x82,0x83,0x04,0x01,0x01,0x01,0x8C,0x00,check])
        ser.write(stx)
    elif raw_input == 'com140a3':
        check=0x82 ^ 0x8D ^ 0x28 ^ 0x03 ^ 0x03 ^ 0x03 ^ 0x8C ^ 0x00
        stx = bytearray([0xFF,0xFF,0xFF,0xFF,0xFF,
0x82,0x8D,0x28,0x03,0x03,0x03,0x8C,0x00,check])
```

```
        ser.write(stx)

    elif raw_input == 'read9':
        a = ser.read(96)
        print(a)

    elif raw_input == 'read':
        a = ser.read(1)
        print(a)

    elif raw_input == 'read0' or raw_input == 'read11':
        a=[]
        if ser.read() == b'\xff':
            b=ser.read()
            while b == b'\xff':
                b=ser.read()
        a.append(b)
        b=list(b)
        checksum = b[0]
        for i in range(31):
            b=ser.read()
            a.append(b)
            b=list(b)
            checksum = checksum ^ b[0]
        b=ser.read()
        a.append(b)
        b=list(b)
        if checksum == b[0]:
            print(a)
        else:
            print('Wrong checksum')

    elif raw_input == 'read1':
        if ser.read() == b'\xff':
            a=ser.read()
            while a == b'\xff':
                a=ser.read()
            a=list(a)
            checksum = a[0]
            address = bytearray()
            for i in range(5):
                a = ser.read()
                a = list(a)
                address.append(a[0])
                checksum = checksum ^ a[0]

            a=ser.read()
            a=list(a)
            if a[0] == 1:
                checksum = checksum ^ a[0]
                b=ser.read()
                b=list(b)
                checksum = checksum ^ b[0]
```

```
                status = bytearray()
                units = bytearray()
                data = bytearray()
                a = ser.read(2)
                a=list(a)
                checksum = checksum ^ a[0] ^ a[1]
                status.append(a[0])
                status.append(a[1])
                a = ser.read()
                a=list(a)
                checksum = checksum ^ a[0]
                units.append(a[0])
                for i in range(b[0]-3):
                    a = ser.read()
                    a=list(a)
                    checksum = checksum ^ a[0]
                    data.append(a[0])
                a = ser.read()
                a=list(a)
                if a[0] == checksum:
                    pvar = struct.unpack('>f',data)
                else:
                    print('Wrong checksum')
                    pass
            else:
                print('Wrong command read')
                pass
            print ('Address:' )
            print(bytes(address))
            print ('Status:')
            print(bytes(status))
            print ('Units:')
            print(HartUnits.units[units[0]])
            print ('PrimaryVariable:')
            print(pvar)
        else:
            pass

    elif raw_input == 'read6':
        if ser.read() == b'\xff':
            a=ser.read()
            while a == b'\xff':
                a=ser.read()
            a=list(a)
            checksum = a[0]
            address = bytearray()
            for i in range(5):
                a = ser.read()
                a = list(a)
                address.append(a[0])
                checksum = checksum ^ a[0]
            a=ser.read()
            a=list(a)
```

```
            if a[0] == 6:
                checksum = checksum ^ a[0]
                b=ser.read()
                b=list(b)
                checksum = checksum ^ b[0]
                status = bytearray()
                poll = bytearray()
                loop = bytearray()
                a = ser.read(2)
                a=list(a)
                checksum = checksum ^ a[0] ^ a[1]
                status.append(a[0])
                status.append(a[1])
                a = ser.read()
                a=list(a)
                checksum = checksum ^ a[0]
                poll.append(a[0])
                a = ser.read()
                a=list(a)
                checksum = checksum ^ a[0]
                loop.append(a[0])
                a = ser.read()
                a=list(a)
                if a[0] == checksum:
                    print ('Full device address:' )
                    print(bytes(address))
                    print ('Status:')
                    print(bytes(status))
                    print ('New device polling address:')
                    print(bytes(poll))
                    print ('PrimaryVariable:')
                    print(bytes(loop))
                else:
                    print('Wrong checksum')
                    pass
            else:
                print('Wrong command read')
                pass

        else:
            pass

    elif raw_input == 'read14':
        if ser.read() == b'\xff':
            a=ser.read()
            while a == b'\xff':
                a=ser.read()
            a=list(a)
            checksum = a[0]
            address = bytearray()
            for i in range(5):
                a = ser.read()
                a = list(a)
```

```
            address.append(a[0])
            checksum = checksum ^ a[0]

        a=ser.read()
        a=list(a)
        if a[0] == 14:
            checksum = checksum ^ a[0]
            b=ser.read()
            b=list(b)
            checksum = checksum ^ b[0]
            status = bytearray()
            sensorID = bytearray()
            units = bytearray()
            data1 = bytearray()
            data2 = bytearray()
            data3 = bytearray()
            a = ser.read(2)
            a=list(a)
            checksum = checksum ^ a[0] ^ a[1]
            status.append(a[0])
            status.append(a[1])
            a = ser.read(3)
            a=list(a)
            checksum = checksum ^ a[0] ^ a[1] ^ a[2]
            for i in a:
                sensorID.append(i)
            a = ser.read()
            a=list(a)
            checksum = checksum ^ a[0]
            units.append(a[0])
            a = ser.read(4)
            a=list(a)
            checksum = checksum ^ a[0] ^ a[1] ^ a[2] ^a[3]
            for i in a:
                data1.append(i)
            a = ser.read(4)
            a=list(a)
            checksum = checksum ^ a[0] ^ a[1] ^ a[2] ^a[3]
            for i in a:
                data2.append(i)
            a = ser.read(4)
            a=list(a)
            checksum = checksum ^ a[0] ^ a[1] ^ a[2] ^a[3]
            for i in a:
                data3.append(i)
            a = ser.read()
            a=list(a)
            if a[0] == checksum:
                UpperTransducerLimit = struct.unpack('f',data1)
                LowerTransducerLimit = struct.unpack('f',data2)
                MinimumSpan = struct.unpack('f',data3)
            else:
                print('Wrong checksum')
```

```
                pass
            else:
                print('Wrong command read')
                pass
            print ('Address:' )
            print(bytes(address))
            print ('Status:')
            print(bytes(status))
            print ('Primary Variable SensorID:')
            print(bytes(sensorID))
            print ('Units:')
            print(HartUnits.units[units[0]])
            print ('Upper Transducer Limit:')
            print(UpperTransducerLimit)
            print ('Lower Transducer Limit:')
            print(LowerTransducerLimit)
            print ('Minimum Span:')
            print(MinimumSpan)
        else:
            pass

    elif raw_input == 'read33':
        if ser.read() == b'\xff':
            a=ser.read()
            while a == b'\xff':
                a=ser.read()
            a=list(a)
            checksum = a[0]
            address = bytearray()
            for i in range(5):
                a = ser.read()
                a = list(a)
                address.append(a[0])
                checksum = checksum ^ a[0]
            a=ser.read()
            a=list(a)
            if a[0] == 33:
                checksum = checksum ^ a[0]
                b=ser.read()
                b=list(b)
                checksum = checksum ^ b[0]
                status = bytearray()
                var1 = bytearray()
                units1 = bytearray()
                data1 = bytearray()
                a = ser.read(2)
                a=list(a)
                checksum = checksum ^ a[0] ^ a[1]
                status.append(a[0])
                status.append(a[1])
                a = ser.read()
                a=list(a)
                checksum = checksum ^ a[0]
```

```
                var1.append(a[0])
                a = ser.read()
                a=list(a)
                checksum = checksum ^ a[0]
                units1.append(a[0])
                a = ser.read(4)
                a = list(a)
                checksum = checksum ^ a[0] ^ a[1] ^ a[2] ^ a[3]
                print (a)
                for i in reversed(a):
                    data1.append(i)
                if num > 1:
                    var2 = bytearray()
                    units2 = bytearray()
                    data2 = bytearray()
                    a = ser.read()
                    a=list(a)
                    checksum = checksum ^ a[0]
                    var2.append(a[0])
                    a = ser.read()
                    a=list(a)
                    checksum = checksum ^ a[0]
                    units2.append(a[0])
                    a = ser.read(4)
                    a=list(a)
                    checksum = checksum ^ a[0] ^ a[1] ^ a[2] ^a[3]
                    print (a)
                    for i in reversed(a):
                        data2.append(i)
                    if num > 2:
                        var3 = bytearray()
                        units3 = bytearray()
                        data3 = bytearray()
                        a = ser.read()
                        a=list(a)
                        checksum = checksum ^ a[0]
                        var3.append(a[0])
                        a = ser.read()
                        a=list(a)
                        checksum = checksum ^ a[0]
                        units3.append(a[0])
                        a = ser.read(4)
                        a=list(a)
                        checksum = checksum ^ a[0] ^ a[1] ^ a[2] ^a[3]
                        print (a)
                        for i in reversed(a):
                            data3.append(i)
                        if num > 3:
                            var4 = bytearray()
                            units4 = bytearray()
                            data4 = bytearray()
                            a = ser.read()
                            a = list(a)
```

```
                checksum = checksum ^ a[0]
                var4.append(a[0])
                a = ser.read()
                a=list(a)
                checksum = checksum ^ a[0]
                units4.append(a[0])
                a = ser.read(4)
                a = list(a)
                checksum = checksum ^ a[0] ^ a[1] ^ a[2] ^ a[3]
                print (a)
                for i in reversed(a):
                    data4.append(i)
                a = ser.read()
                a=list(a)
                if a[0] == checksum:
                    value1 = struct.unpack('f',data1)
                    value2 = struct.unpack('f',data2)
                    value3 = struct.unpack('f',data3)
                    value4 = struct.unpack('f',data4)
                    print ('Address:' )
                    print (bytes(address))
                    print ('Status:')
                    print (bytes(status))
                    print ('First Variable:' )
                    print (HDVC.code[var1[0]])
                    print ('Units:')
                    print (HartUnits.units[units1[0]])
                    print ('Value:')
                    print (value1)
                    print ('Second Variable:' )
                    print (HDVC.code[var2[0]])
                    print ('Units:')
                    print (HartUnits.units[units2[0]])
                    print ('Value:')
                    print (value2)
                    print ('Third Variable:' )
                    print (HDVC.code[var3[0]])
                    print ('Units:')
                    print (HartUnits.units[units3[0]])
                    print ('Value:')
                    print (value3)
                    print ('Forth Variable:' )
                    print (HDVC.code[var4[0]])
                    print ('Units:')
                    print (HartUnits.units[units4[0]])
                    print ('Value:')
                    print (value4)
                else:
                    print(str(checksum))
                    print('Wrong checksum')
                    pass
            else:
                a = ser.read()
```

```
                    a=list(a)
                    if a[0] == checksum:
                        value1 = struct.unpack('f',data1)
                        value2 = struct.unpack('f',data2)
                        value3 = struct.unpack('f',data3)
                        print ('Address:' )
                        print (bytes(address))
                        print ('Status:')
                        print (bytes(status))
                        print ('First Variable:' )
                        print (HDVC.code[var1[0]])
                        print ('Units:')
                        print (HartUnits.units[units1[0]])
                        print ('Value:')
                        print (value1)
                        print ('Second Variable:' )
                        print (HDVC.code[var2[0]])
                        print ('Units:')
                        print (HartUnits.units[units2[0]])
                        print ('Value:')
                        print (value2)
                        print ('Third Variable:' )
                        print (HDVC.code[var3[0]])
                        print ('Units:')
                        print (HartUnits.units[units3[0]])
                        print ('Value:')
                        print (value3)
                    else:
                        print(str(checksum))
                        print('Wrong checksum')
                        pass
            else:
                a = ser.read()
                a=list(a)
                if a[0] == checksum:
                    value1 = struct.unpack('f',data1)
                    value2 = struct.unpack('f',data2)
                    print ('Address:' )
                    print (bytes(address))
                    print ('Status:')
                    print (bytes(status))
                    print ('First Variable:' )
                    print (HDVC.code[var1[0]])
                    print ('Units:')
                    print (HartUnits.units[units1[0]])
                    print ('Value:')
                    print (value1)
                    print ('Second Variable:' )
                    print (HDVC.code[var2[0]])
                    print ('Units:')
                    print (HartUnits.units[units2[0]])
                    print ('Value:')
                    print (value2)
```

```
                    else:
                        print(str(checksum))
                        print('Wrong checksum')
                        pass
            else:
                a = ser.read()
                a=list(a)
                if a[0] == checksum:
                    value1 = struct.unpack('f',data1)
                    print ('Address:' )
                    print (bytes(address))
                    print ('Status:')
                    print (bytes(status))
                    print ('First Variable:' )
                    print (HDVC.code[var1[0]])
                    print ('Units:')
                    print (HartUnits.units[units1[0]])
                    print ('Value:')
                    print (value1)
                else:
                    print(str(checksum))
                    print('Wrong checksum')
                    pass
        else:
            print('Wrong command read')
            pass
    else:
        pass

elif raw_input == 'read48':
    a=[]
    if ser.read() == b'\xff':
        b=ser.read()
        while b == b'\xff':
            b=ser.read()
    a.append(b)
    b=list(b)
    checksum = b[0]
    for i in range(29):
        b=ser.read()
        a.append(b)
        b=list(b)
        checksum = checksum ^ b[0]
    b=ser.read()
    a.append(b)
    b=list(b)
    if checksum == b[0]:
        print(a)
    else:
        print('Wrong checksum')

elif raw_input == 'read140':
    a=[]
```

```
        if ser.read() == b'\xff':
            b=ser.read()
            while b == b'\xff':
                b=ser.read()
        a.append(b)
        b=list(b)
        checksum = b[0]
        for i in range(26):
            b=ser.read()
            a.append(b)
            b=list(b)
            checksum = checksum ^ b[0]
        b=ser.read()
        a.append(b)
        b=list(b)
        if checksum == b[0]:
            print(a)
        else:
            print('Wrong checksum')

    else:
        write = 'D'+ str(len(raw_input)) +'!'+ raw_input
        write = write.encode()
        ser.write(write)
```

3.2. Jsonwriter.py

```
"""
Pau Ferrer Almirall
UPC BarcelonaTech
Juny 2020

Improvised Json writer to create DDFiles.
If only one slave is uncommented there will be 1 file per slave,
uncomment the rest of the slaves to inlcude them all in the same file.
"""

import json
import HartUnits
import HartStatusErrors

slaves=[]
slaves.append({
    'PrimaryVariable': 0x19,
    'Units': HartUnits.Celsius,
    'CommsStatus': HartStatusErrors.NonErrors,
    'ResponseCode': HartStatusErrors.Notification,
    'FiedDevStatus': HartStatusErrors.StatusOK,
    'ExpDevStatus': HartStatusErrors.OK,
    'PollAddress': 0x03,
```

```
    'ExpDevType': [0x03, 0x04],
    'DeviceID': [0x01, 0x01, 0x01],
    'ManufID': [0x00, 0x03],
    'Tag': [0x01,0x02,0x03,0x04,0x05,0x06],
    'PrimaryVariableLoopCurrent': 11.5
})
##slaves.append({
##    'PrimaryVariable': 0x0A,
##    'Units': HartUnits.bars,
##    'CommsStatus': HartStatusErrors.NonErrors,
##    'ResponseCode': HartStatusErrors.Notification,
##    'FiedDevStatus': HartStatusErrors.StatusOK,
##    'ExpDevStatus': HartStatusErrors.OK,
##    'PollAddress': 0x01,
##    'ExpDevType': [0x60, 0xED],
##    'DeviceID': [0x02, 0x02, 0x02],
##    'ManufID': [0x00, 0x60],
##    'Tag': [0x02,0x03,0x04,0x05,0x06,0x07],
##    'PrimaryVariableLoopCurrent': 5.6
##})
##slaves.append({
##    'PrimaryVariable': 0x0A,
##    'Units': HartUnits.bars,
##    'CommsStatus': HartStatusErrors.NonErrors,
##    'ResponseCode': HartStatusErrors.Notification,
##    'FiedDevStatus': HartStatusErrors.StatusOK,
##    'ExpDevStatus': HartStatusErrors.OK,
##    'PollAddress': 0x02,
##    'ExpDevType': [0x0D, 0x28],
##    'DeviceID': [0x03, 0x03, 0x03],
##    'ManufID': [0x00, 0x0D],
##    'Tag': [0x03,0x04,0x05,0x06,0x07,0x08],
##    'PrimaryVariableLoopCurrent': 11.3
##})

with open('HartDD.txt', 'w') as outfile: #Modify file name for every DDFile
    json.dump(slaves, outfile, indent=1)
```