```
#!/usr/bin/python3

'''
Program to calibrate offset and gain of Hantek 6022BE/BL
1.) Measure offset at low and high speed for the four gain steps x10, x5, x2, x1
2.) Measure gain for the four gain steps x10, x5, x2, x1
3.) Write offset values into eeprom and config file

Configure with command line arguments:

    usage: calibrate.py [-h] [-c] [-g] [--no_eeprom]

    optional arguments:
        -h, --help           show this help message and exit
        -c, --create_config  create config file
        -e, --eeprom         store calibration values in eeprom
        -g, --measure_gain   interactively measure gain (as well as offset)
'''


from PyHT6022.LibUsbScope import Oscilloscope

import sys
import time
import binascii

import argparse



# average over 100ms @ 100kS/s -> 5 cycles @ 50 Hz or 6 cycles @ 60 Hz to cancel AC
hum
def read_avg( voltage_range, sample_rate=110, repeat = 1, samples = 12 * 1024 ):
    scope.set_sample_rate( sample_rate )
    scope.set_ch1_voltage_range(voltage_range)
    if ( 30 == sample_rate ):
        scope.set_num_channels( 1 )
    else:
        scope.set_num_channels( 2 )
        scope.set_ch2_voltage_range(voltage_range)

    time.sleep( 0.1 )

    sum1 = 0
    sum2 = 0
    count1 = 0
    count2 = 0

    for rep in range( repeat ): # repeat measurement
        ch1_data, ch2_data = scope.read_data( samples, raw=True, timeout=0)
```

```
        # skip first samples and keep 10000
        skip = samples - 10000

        # print( len( ch1_data), len( ch2_data ) )

        for sample in ch1_data[skip:]:
            sum1 += sample
            count1 += 1
        if ( 30 != sample_rate ):
            for sample in ch2_data[skip:]:
                sum2 += sample
                count2 += 1

    # measured values are 0x80 binary offset -> 0V = 0x80
    avg1 = sum1 / count1 - 0x80
    if ( count2 ):
        avg2 = sum2 / count2 - 0x80
    else:
        avg2 = 0
    return ( avg1, avg2 )


# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument( "-c", "--create_config", action="store_true",
    help="create a config file" )
ap.add_argument( "-e", "--eeprom", action="store_true",
    help="store calibration values in eeprom" )
ap.add_argument( "-g", "--measure_gain", action="store_true",
    help="interactively measure gain (as well as offset)" )

args = vars(ap.parse_args())
create_config = args["create_config"]
eeprom = args["eeprom"]
measure_gain = args["measure_gain"]

# print("create_config: {}, eeprom: {}, measure_gain: {}".format(
args["create_config"], args["eeprom"], args["measure_gain"] ) )



print("Setting up scope...")

scope = Oscilloscope()
scope.setup()
scope.open_handle()

if (not scope.is_device_firmware_present):
    scope.flash_firmware()
```

```
scope.supports_single_channel = True

# select two channels
scope.set_num_channels( 2 )
# set coupling of both channels to DC
scope.set_ch1_ch2_ac_dc( scope.DC_DC )


# get calibration values stored in the scope (all values are 0x80 binary offset,
0x80 = 0)
# use these values as default if new values are not plausible
# structure and representation of these values due to historical reason
# 32 byte integer offset CH1, CH2,... for 8 input ranges (20mV, 50mV,..., 5V) for
low and high speed
# these are also the factory calibration settings (0x80 = no offset)
# next 16 bytes are the gain correction values CH1, CH2, ... same voltage range,
only one speed
# 0x80 = 1.0, 0x80-125 = 0.75, 0x80+125 = 1.25
# next 32 bytes are the fractional error (range -0.5 ... +0.5) of the offset values
above,
# 0x80 = 0, 0x80-125 = -0.5, 0x80+125 = +0.5
#
ee_calibration = bytearray( scope.get_calibration_values( 32 + 16 + 32 ) )


# mV/div ranges
V_div = ( 20, 50, 100, 200, 500, 1000, 2000, 5000 )
# corresponding amplifier gain settings
gains = ( 10, 10,  10,   5,   2,    1,    1,    1 )
# available amplifier gains
gainSteps = ( 10, 5, 2, 1 )

# measure offset
# apply 0 V and measure the ADC values
print( "\nCalculate zero adjustment" )
input( "Apply 0 V to both channels and press <ENTER> " )

offset1 = {} # double offset at low speed
offset2 = {}
offlo1 = {}  # int offset at low speed
offlo2 = {}
offlo_1 = {} # difference between double and int value
offlo_2 = {}
offhi1 = {}  # only ch1 for high speed
offhi_1 = {} # only ch1 for high speed


for gain in gainSteps:
    # average 10 times over 100 ms (cancel 50 Hz / 60 Hz)
```

```
    print( "Measure offset at low speed for gain ", gain )
    # keep the values for later gain correction
    offset1[ gain ], offset2[ gain ] = read_avg( gain, 110, 10 )
    #print("debug low speed offset1/2:", offset1[ gain ], offset2[ gain ] )
    raw1 =  int( round( offset1[ gain ] ) )
    fine1 = int( round( (offset1[ gain ] - raw1) * 250.0 ) )
    raw2 =  int( round( offset2[ gain ] ) )
    fine2 = int( round( (offset2[ gain ] - raw2) * 250.0 ) )
    #print("debug low speed raw/fine:", raw1, fine1, raw2, fine2 )
    offlo1[ gain ] = raw1
    offlo_1[ gain ] = fine1
    offlo2[ gain ] = raw2
    offlo_2[ gain ] = fine2

    print( "Measure offset at high speed for gain ", gain )
    off1, off2 = read_avg( gain, 30, 10 )
    #print("debug high speed off", off1, off2)
    raw1 =  int( round( off1 ) )
    fine1 = int( round( (off1 - raw1) * 250.0 ) )
    #print("debug high speed raw/fine:", raw1, fine1, raw2, fine2 )
    offhi1[ gain ] = raw1
    offhi_1[ gain ] = fine1


if ( create_config ):
    # prepare config file
    configfile = "modelDSO6022.conf"
    config = open( configfile, "w" )
    config.write( ";OpenHantek calibration file for DSO6022\n;Created by tool
'calibrate.py'\n\n" )
    config.write( "[offset]\n" )


for index, gainID in enumerate( gains ):
    # print( gains[index], offlo1[gainID], offlo2[gainID], offhi1[gainID],
offhi2[gainID],  )

    if ( create_config ):
        # write integer offset for low speed sampling into config file
        voltID = V_div[ index ]
        if ( abs( offlo1[ gainID ] ) <= 25 ):   # offset too high -> skip
            config.write( "ch0\\%dmV=%d\n" % ( voltID, -offlo1[ gainID ] ) )
            #ji config.write( "ch0\\%dmV=%d\n" % ( voltID, 0x80 + offlo1[ gainID ] )
)
        if ( abs( offlo2[ gainID ] ) <= 25 ):   # offset too high -> skip
            config.write( "ch1\\%dmV=%d\n" % ( voltID, -offlo2[ gainID ] ) )
            #ji config.write( "ch1\\%dmV=%d\n" % ( voltID, 0x80 + offlo2[ gainID ] )
)
```

```
    # prepare eeprom content
    if ( abs( offlo1[ gainID ] ) <= 25 ):                           # offset too
high -> skip
        ee_calibration[ 2 * index ] = 0x80 + offlo1[ gainID ]       # CH1 offset
integer part
    if ( abs( offlo_1[ gainID ] ) <= 125 ):                         # frac part not
plausible
        ee_calibration[ 2 * index + 48 ] = 0x80 + offlo_1[ gainID ] # CH1 offset
fractional part
    if ( abs( offlo2[ gainID ] ) <= 25 ):
        ee_calibration[ 2 * index + 1 ] = 0x80 + offlo2[ gainID ]   # CH2 offset
integer part
    if ( abs( offlo_2[ gainID ] ) <= 125 ):
        ee_calibration[ 2 * index + 49 ] = 0x80 + offlo_2[ gainID ] # CH2 offset
fractional part
    if ( abs( offhi1[ gainID ] ) <= 25 ):
        ee_calibration[ 2 * index + 16 ] = 0x80 + offhi1[ gainID ]  # same for CH2
    if ( abs( offhi_1[ gainID ] ) <= 125 ):
        ee_calibration[ 2 * index + 64 ] = 0x80 + offhi_1[ gainID ] #


if ( measure_gain ):
    # measure gain interactively
    # apply a defined voltage, measure raw, correct offset and calculate gain
    print( "\nCalculate gain adjustment"  )
    print( "Apply the requested voltage (as exactly as possible) to both channels
and press <ENTER>" )
    print( "You can also apply a slightly lower or higher stable voltage and type in
this value\n" )


    # theoretical gain error of 6022 front end due to nominal resistor values (e.g.
5.1 kOhm instead 5.0)
    # these values are considered also in OpenHantek6022
    error = ( 1.00, 1.01, 0.99, 0.99 ) # gainSteps 10x, 5x, 2x, 1x

    gain1 = {}
    gain2 = {}

    index = 0 # index for gain error due to nominal resistor values
    for gain in gainSteps:
        voltage = 4 / gain # max input is slightly lower than 5V / gain, so use 4V
to be on the safe side
        setpoint = input( "Apply %4.2f V to both channels and press <ENTER> " %
voltage )
        try:
            setpoint = float( setpoint ) # did the user supply an own voltage
setting?
        except ValueError:
            setpoint = voltage # else assume the proposed value 'voltage'
```

```
        # we expect value 'target'
        target = error[ index ] * 100 * setpoint / voltage
        index += 1
        # get offset error for gain setting & channel
        off1 = offset1[ gain ]
        off2 = offset2[ gain ]
        # read raw values, average over 10 times 100 ms
        raw1, raw2 = read_avg( gain, 110, 10 ) # read @ 100kS/s
        # correct offset error
        value1 = raw1 - offset1[ gain ]
        value2 = raw2 - offset2[ gain ]
        # print( raw1, value1, raw2, value2 ) # expected: 80..120
        #
        if raw1 > 250 or value1 < 80 or value1 > 120: # overdriven or out of range
            gain1[ gain ] = None # ignore setting, no correction
        else:
            gain1[ gain ] = target / value1 # corrective gain factor
        if raw2 > 250 or value2 < 80 or value2 > 120: # same as for 1st channel
            gain2[ gain ] = None
        else:
            gain2[ gain ] = target / value2

            # print( gain1[ gain ], gain2[ gain ] )


    if ( create_config ):
        config.write( "\n[gain]\n" )

    for index, gainID in enumerate( gains ):
        voltID = V_div[ index ]
        g1 = gain1[ gainID ]
        g2 = gain2[ gainID ]
        if ( g1 ):
            # convert double 0.75 ... 1.25 -> byte 0x80-125 ... 0x80+125
            ee_calibration[ 2 * index + 32] = int( round( ( g1 - 1 ) * 500 + 0x80 )
)
        if ( g2 ):
            # convert double 0.75 ... 1.25 -> byte 0x80-125 ... 0x80+125
            ee_calibration[ 2 * index + 33 ] = int( round( ( g2 - 1 ) * 500 + 0x80 )
)
        if ( create_config ):
            if ( g1 ):
                config.write( "ch0\\%dmV=%6.4f\n" % ( voltID, g1 ) )
            if ( g2 ):
                config.write( "ch1\\%dmV=%6.4f\n" % ( voltID, g2 ) )

# endif ( measure_gain )


if ( create_config ):
```

```
    config.close()


if ( eeprom ):
    print( "eeprom content [  8 .. 39 ]: ", binascii.hexlify( ee_calibration[  0:32
] ) )
    print( "eeprom content [ 40 .. 55 ]: ", binascii.hexlify( ee_calibration[ 32:48
] ) )
    print( "eeprom content [ 56 .. 87 ]: ", binascii.hexlify( ee_calibration[ 48:80
] ) )
    # finally store the calibration values into eeprom
    scope.set_calibration_values( ee_calibration )


scope.close_handle()
```