D:\Research\code\attocube\anc350v5.py

```python
from lantz.foreign import LibraryDriver
from lantz import Feat, DictFeat, Action, Q_

import time
from ctypes import c_uint, c_void_p, c_double, pointer, POINTER, c_int, c_bool, c_char_p, byref

"Author: Qian Lin"
"qian.lin@balliol.ox.ac.uk"
"10/19/2023"

class ANC350(LibraryDriver):

    LIBRARY_NAME = 'anc350v4.dll'
    LIBRARY_PREFIX = 'ANC_'

    RETURN_STATUS = {0:'ANC_Ok', -1:'ANC_Error', 1:"ANC_Timeout", 2:"ANC_NotConnected", 3:"ANC_DriverError",
                     7:"ANC_DeviceLocked", 8:"ANC_Unknown", 9:"ANC_NoDevice", 10:"ANC_NoAxis",
                     11:"ANC_OutOfRange", 12:"ANC_NotAvailable", 13:"ANC_FileError"}

    def __init__(self):
        super(ANC350, self).__init__()

        self.lib.configureAQuadBIn.errcheck = ANC350.checkError
        self.lib.configureAQuadBOut.errcheck = ANC350.checkError
        self.lib.configureDutyCycle.errcheck = ANC350.checkError
        self.lib.configureExtTrigger.errcheck = ANC350.checkError
        self.lib.configureNslTrigger.errcheck = ANC350.checkError
        self.lib.configureNslTriggerAxis.errcheck = ANC350.checkError
        self.lib.configureRngTrigger.errcheck = ANC350.checkError
        self.lib.configureRngTriggerEps.errcheck = ANC350.checkError
        self.lib.configureRngTriggerPol.errcheck = ANC350.checkError
        self.lib.connect.errcheck = ANC350.checkError
        self.lib.disconnect.errcheck = ANC350.checkError
        self.lib.discover.errcheck = ANC350.checkError
        self.lib.discoverRegistered.errcheck = ANC350.checkError
        self.lib.enableRefAutoReset.errcheck = ANC350.checkError
        self.lib.enableRefAutoUpdate.errcheck = ANC350.checkError
        self.lib.enableSensor.errcheck = ANC350.checkError
        self.lib.enableTrace.errcheck = ANC350.checkError
        self.lib.forceDisconnect.errcheck = ANC350.checkError
        self.lib.getActuatorName.errcheck = ANC350.checkError
        self.lib.getActuatorType.errcheck = ANC350.checkError
        self.lib.getAmplitude.errcheck = ANC350.checkError
        self.lib.getAxisStatus.errcheck = ANC350.checkError
        self.lib.getDcVoltage.errcheck = ANC350.checkError
        self.lib.getDeviceConfig.errcheck = ANC350.checkError
        self.lib.getDeviceInfo.errcheck = ANC350.checkError
        self.lib.getFirmwareVersion.errcheck = ANC350.checkError
        self.lib.getFrequency.errcheck = ANC350.checkError
        self.lib.getLutName.errcheck = ANC350.checkError
        self.lib.getLutUsage.errcheck = ANC350.checkError
        self.lib.getPosition.errcheck = ANC350.checkError
        self.lib.getRefPosition.errcheck = ANC350.checkError
        self.lib.getSensorVoltage.errcheck = ANC350.checkError
        self.lib.loadLutFile.errcheck = ANC350.checkError
```

```
56          self.lib.measureCapacitance.errcheck = ANC350.checkError
57          self.lib.moveReference.errcheck = ANC350.checkError
58          self.lib.registerExternalIp.errcheck = ANC350.checkError
59          self.lib.resetPosition.errcheck = ANC350.checkError
60          self.lib.saveParams.errcheck = ANC350.checkError
61          self.lib.selectActuator.errcheck = ANC350.checkError
62          self.lib.setAmplitude.errcheck = ANC350.checkError
63          self.lib.setAxisOutput.errcheck = ANC350.checkError
64          self.lib.setDcVoltage.errcheck = ANC350.checkError
65          self.lib.setFrequency.errcheck = ANC350.checkError
66          self.lib.setLutUsage.errcheck = ANC350.checkError
67          self.lib.setSensorVoltage.errcheck = ANC350.checkError
68          self.lib.setTargetGround.errcheck = ANC350.checkError
69          self.lib.setTargetPosition.errcheck = ANC350.checkError
70          self.lib.setTargetRange.errcheck = ANC350.checkError
71          self.lib.startAutoMove.errcheck = ANC350.checkError
72          self.lib.startContinousMove.errcheck = ANC350.checkError
73          self.lib.startMultiStep.errcheck = ANC350.checkError
74          self.lib.startSingleStep.errcheck = ANC350.checkError
75
76
77          #Discover systems
78          ifaces = c_uint(0x03) # USB interface
79          devices = c_uint()
80          self.lib.discover(ifaces, pointer(devices))
81          if not devices.value:
82              raise RuntimeError('No controller found. Check if controller is connected or
    if another application is using the connection')
83          self.dev_no = c_uint(devices.value - 1)
84          self.device = None
85
86
87          return
88
89      def initialize(self, devNo=None):
90
91          if not devNo is None: self.devNo = devNo
92          device = c_void_p()
93
94          self.lib.connect(self.dev_no, pointer(device))
95          self.device = device
96
97
98
99      def finalize(self):
100         self.lib.disconnect(self.device)
101         self.device = None
102
103     @staticmethod
104     def checkError(code, func, args):
105         if code != 0:
106             raise Exception("Driver Error {}: {} in {} with parameters: {}".format(code,
    ANC350.RETURN_STATUS[code],str(func.__name__),str(args)))
107         return
108
109
110
111     @DictFeat(units='V', keys=(0, 1, 2))
112     def DCvoltage(self, axis):
113         axis = int(axis)
```

```
114         ret_volt = c_double()
115         self.lib.getDcVoltage(self.device, c_uint(axis), byref(ret_volt))
116         print(ret_volt)
117         return ret_volt.value
118 
119     @DCvoltage.setter
120     def DCvoltage(self, axis, DCvoltage):
121         axis = int(axis)
122         self.lib.setDcVoltage(self.device, c_uint(axis), c_double(DCvoltage))
123 
124     @DictFeat(units='V', keys=(0, 1, 2))
125     def amplitude(self, axis):
126         axis = int(axis)
127         ret_amplitude = c_double()
128         self.lib.getAmplitude(self.device, c_uint(axis), pointer(ret_amplitude))
129         return ret_amplitude.value
130 
131     @amplitude.setter
132     def amplitude(self, axis, amplitude):
133         axis = int(axis)
134         self.lib.setAmplitude(self.device, c_uint(axis), c_double(amplitude))
135 
136     @DictFeat(units='V', keys=(0, 1, 2))
137     def sensorvoltage(self, axis):
138         axis = int(axis)
139         ret_volt = c_double()
140         self.lib.getSensorVoltage(self.device, c_uint(axis), pointer(ret_volt))
141         return ret_volt.value
142 
143     @sensorvoltage.setter
144     def sensorvoltage(self, axis, sensorvoltage):
145         axis = int(axis)
146         self.lib.setSensorVoltage(self.device, c_uint(axis), c_double(sensorvoltage))
147 
148 
149     @DictFeat(units='Hz',keys=(0,1,2))
150     def frequency(self, axis):
151         axis = int(axis)
152         ret_freq = c_double()
153         self.lib.getFrequency(self.device, c_uint(axis), pointer(ret_freq))
154         return ret_freq.value
155 
156     @frequency.setter
157     def frequency(self, axis, freq):
158         axis = int(axis)
159         self.lib.setFrequency(self.device, c_uint(axis), c_double(freq))
160 
161 
162     @DictFeat(units='m',keys=(0,1,2))
163     def position(self, axis):
164         axis = int(axis)
165         ret_pos = c_double()
166         self.lib.getPosition(self.device, c_uint(axis), pointer(ret_pos))
167         return ret_pos.value
168 
169     @position.setter
170     def position(self, axis, pos):
171         axis = int(axis)
172         self.lib.setTargetPosition(self.device, c_uint(axis), c_double(pos))
173         self.lib.startAutoMove(self.device, c_uint(axis), 1, 0)
```

```
174         return
175
176
177
178     @DictFeat(units='F',keys=(0,1,2))
179     def capacitance(self, axis):
180         axis = int(axis)
181         ret_c = c_double()
182         self.lib.measureCapacitance(self.device, c_uint(axis), pointer(ret_c))
183         return ret_c.value
184
185     @DictFeat(keys=(0,1,2))
186     def status(self, axis):
187         axis = int(axis)
188         status_names = [
189             'connected',
190             'enabled',
191             'moving',
192             'target',
193             'eot_fwd',
194             'eot_bwd',
195             'error',
196         ]
197         status_flags = [c_uint() for _ in range(7)]
198         status_flags_p = [pointer(flag) for flag in status_flags]
199         self.lib.getAxisStatus(self.device, c_uint(axis), *status_flags_p)
200
201         ret = dict()
202         for status_name, status_flag in zip(status_names, status_flags):
203             ret[status_name] = True if status_flag.value else False
204         return ret
205
206     # Untested
207     @Action()
208     def stop(self):
209         for axis in range(3):
210             self.lib.startContinousMove(self.device, c_uint(axis), 0, 1)
211
212
213     @Action()
214     def jog(self, axis, speed):
215         axis = int(axis)
216         backward = c_bool(speed < 0.0)
217         start = c_bool(speed != 0.0)
218         self.lib.startContinousMove(self.device, c_uint(axis), start, backward)
219         return
220
221     @Action()
222     def single_step(self, axis, direction):
223         axis = int(axis)
224         backward = c_bool(direction <= 0)
225         self.lib.startSingleStep(self.device, c_uint(axis), backward)
226         return
227
228     @Action()
229     def multi_step(self, axis, steps):
230         axis = int(axis)
231         backward = c_bool(steps <= 0)
232         self.lib.startMultiStep(self.device, c_uint(axis), backward, c_uint(max(1,
min(32767, abs(int(steps))))))
```

```
233         return
234
235     @Action(units=['', 'm'])
236     def move(self, axis, pos):
237         axis = int(axis)
238         self.lib.setTargetPosition(self.device, c_uint(axis), c_double(pos))
239         self.lib.startAutoMove(self.device, c_uint(axis), 1, 1)
240         time.sleep(20)
241         self.lib.startAutoMove(self.device, c_uint(axis), 0, 1)
242         return
243
244
245     @Action(units = ['','m'])
246     def set_target_range(self, axis, target_range):
247         axis = int(axis)
248         self.lib.setTargetRange(self.device, c_uint(axis), c_double(target_range))
249         return
250
251     @Action()
252     def register_externalIp(self,IP):
253         self.lib.registerExternalIp(c_char_p(IP))
254         return
255
256     @Action()
257     def discover_registered(self):
258         '''Discover only Preregistered Devices.
259         The function works similar to ANC_discover but it "discovers" only devices
    connected via ethernet that have been
260         preregistered by ANC_registerExternalIp .'''
261         devices = c_uint()
262         self.lib.discoverRegistered(pointer(devices))
263         return devices.value
264
265     @Action()
266     def force_disconnect(self):
267         self.lib.forceDisconnect(self.device)
268         self.device = None
269
270     @Action()
271     def get_actuator(self,axis):
272         axis = int(axis)
273         actuator = ActuatorType()
274         self.lib.getActuatorType(self.device, c_uint(axis), byref(actuator))
275         return ActuatorType.to_string(actuator.value)
276
277     @Action()
278     def configure_rng_trigger(self, axis, lower, upper):
279         lower = int(lower)
280         upper = int(upper)
281         axis = int(axis)
282         self.lib.configureRngTrigger(self.device, c_uint(axis), c_uint(lower),
    c_uint(upper))
283
284
285     @Action()
286     def configure_rng_trigger_pol(self, axis, polarity):
287         axis = int(axis)
288         polarity = int(polarity)
289         self.lib.configureRngTriggerPol(self.device, c_uint(axis), c_uint(polarity))
290
```

```
291
292     @Action()
293     def configure_rng_trigger_eps(self, axis, epsilon):
294         axis = int(axis)
295         epsilon = int(epsilon)
296         self.lib.configureRngTriggerEps(self.device, c_uint(axis), c_uint(epsilon))
297
298     # ----------------------------------------------
299
300 class ActuatorType(c_int):
301     ActLinear = 0
302     ActGonio = 1
303     ActRot = 2
304
305     _type_str_mapping = {
306         ActLinear: "Linear",
307         ActGonio: "Goniometric",
308         ActRot: "Rotational"
309     }
310
311     @classmethod
312     def to_string(cls, value):
313         return cls._type_str_mapping.get(value, "Unknown")
314
315
316 if __name__ == '__main__':
317     from lantz.log import log_to_screen, DEBUG
318     log_to_screen(DEBUG)
319     s = ANC350()
```