I've written some code that imports a dll file and creates python objects to interact with it. I've done this by translating the features provided in an instruction manual to methods of the object.

I’m going to send you two examples in which I first provide you with the manual entry and then I provide you with the python code I have written. After that, I’m going to send you a new manual entry and ask that you write python wrapper code for me in the same manner that I have written the examples.

Example 1:

Manual Entry -

6.8 SuperK EXTREME System (S4x2) and SuperK Fianium Main module

This section describes specific registers for SuperK EXTREME with product numbers starting with S4x2.

Module type (60h) and module address

The module type number is 60h. The standard module address is 15 (0Fh).

8-bit unsigned integer.

System type (6Bh)

The system type is a newer implementation used to differentiate systems with minor differences. Older versions might not respond to this register. In that case the system type should be interpreted as 0 = SuperK Extreme. 8-bit unsigned integer.

0 = SuperK Extreme

1 = SuperK Fianium

Inlet temperature (11h)

The inlet temperature readout 1/10 °C.16-bit signed integer.

Emission (30h)

Emission on/off. The value 0 turns emission off, and the value 3 turns it on (if interlock circuit has been reset). 8-bit unsigned integer.

Reading the register returns the current emission state of the SuperK Extreme. Normally, it returns the same value as the one that was written. However, while switching emission on or off, the SuperK EXTREME will go through a small number of different states. So briefly, the returned value will differ from the one that was written.

Setup (31h)

With the Setup register, the operation mode of the SuperK EXTREME System can be controlled. The possible values are listed below; however, in some systems, not all modes are available.16-bit unsigned integer.

0: Constant current mode

1: Constant power mode

2: Externally modulated current mode

3: Externally modulated power mode

4: External feedback mode (Power Lock)

Interlock (32h)

If the door interlock is in place, the key switch on the front plate is in On position and the External bus is terminated with e.g. a bus defeater then the Interlock circuit can be reset via the Interbus interface by sending a value greater than 0 to the Interlock register.

Additionally, the opposite function (switching interlock relays off) can be done by sending the value 0 to the interlock register.

Reading the interlock register returns the current interlock status, which consists of two unsigned bytes. The first byte (LSB) tells if the interlock circuit is open or closed. The second byte (MSB) tells where the interlock circuit is open, if relevant.

MSB LSB Description

-

0

Interlock off (interlock circuit open)

0

1

Waiting for interlock reset

0

2

Interlock is OK

1

0

Front panel interlock / key switch off

2

0

Door switch open

3

0

External module interlock

4

0

Application interlock

5

0

Internal module interlock

6

0

Interlock power failure

7

0

Interlock disabled by light source

255

-

Interlock circuit failure

Pulse picker ratio (34h)

For SuperK EXTREME Systems featuring the pulse picker option, the divide ratio for the pulse picker can be controlled with the pulse picker ratio register.

Note: When reading the pulse picker value, an 8-bit unsigned integer will be returned if the ratio is lower than 256, and a 16-bit unsigned integer otherwise. This is for historical reasons.

Watchdog interval (36h)

The system can be set to make an automatic shut-off (laser emission only – not electrical power) in case of lost communication. The value in the watchdog interval register determines how many seconds with no communication the system will tolerate. If the value is 0, the feature is disabled. 8-bit unsigned integer.

Power level (37h)

Power level setpoint in tenths of percent (permille, ‰). 16-bit unsigned integer.

Current level (38h)

Current level setpoint in tenths of percent (permille, ‰). 16-bit unsigned integer.

NIM delay (39h)

On systems with NIM trigger output, the delay of this trigger signal can be adjusted with the NIM delay register. The input for this register should be an unsigned 16-bit value from 0 to 1023. The range is 0 – 9.2 ns. The average step size is 9 ps.

Status bits (66h)

This register returns the mainboard status bits. 16-bit integer.

Bit 0: Emission on

Bit 1: Interlock relays off

Bit 2: Interlock supply voltage low (possible short circuit)

Bit 3: Interlock loop open

Bit 4: Output Control signal low

Bit 5: Supply voltage low

Bit 6: Inlet temperature out of range

Bit 7: Clock battery low voltage

...

Bit 13: CRC error on startup (possible module address conflict)

Bit 14: Log error code present

Bit 15: System error code present

Python code -

"""

Python module to control NKT Extreme and Fianium Lasers.

Note: I chose to use specific setter methods over properties so that the user

explicitly has to adjust potentially dangerous conditions

(i.e. laser.set\_emission(True)). I felt this style leaves less ambiguity that

the user is actively turning the laser on (vs. laser.emission = True). Less

dangerous properties follow the dedicated setter method format for consistency.

"""

import nkt\_tools.NKTP\_DLL as nkt

class Extreme:

status\_messages = {

0: 'Emission on',

1: 'Interlock relays off',

2: 'Interlock supply voltage low (possible short circuit)',

3: 'Interlock loop open',

4: 'Output Control signal low',

5: 'Supply voltage low',

6: 'Inlet temperature out of range',

7: 'Clock battery low voltage',

8: '-',

9: '-',

10: '-',

11: '-',

12: '-',

13: 'CRC error on startup (possible module address conflict)',

14: 'Log error code present',

15: 'System error code present'

}

"""

dict : system status translation bits > string

========= ========================================================

Bit Index Status

========= ========================================================

Bit 0: Emission on

Bit 1: Interlock relays off

Bit 2: Interlock supply voltage low (possible short circuit)

Bit 3: Interlock loop open

Bit 4: Output Control signal low

Bit 5: Supply voltage low

Bit 6: Inlet temperature out of range

Bit 7: Clock battery low voltage

...

Bit 13: CRC error on startup (possible module address conflict)

Bit 14: Log error code present

Bit 15: System error code present

========= ========================================================

"""

setup\_options = {

0: 'Constant current mode',

1: 'Constant power mode',

2: 'Externally modulated current mode',

3: 'Externally modulated power mode',

4: 'External feedback mode (Power Lock)'

}

def \_\_init\_\_(self, portname=None):

"""

Searches for connected NKT lasers and defines instrument parameters.

Make sure devices are not connected via another program already.

If multiple Extreme/Fianium lasers are connected to the same computer,

specificy the port of the desired laser upon instantiation.

Parameters

----------

portname : str, optional

Enter if portname for laser is known/multiple lasers are connected.

If not supplied, system searches for laser. None by default.

Raises

------

RuntimeError

Throws error if multiple NKT lasers are found on one computer.

Supply portname for desired laser if multiple present.

"""

print('Searching for connected NKT Laser...')

self.\_portname = None # COM port for laser. Auto found if not given.

self.\_module\_address = 15 # module address = 15 for Extreme/Fianium

self.\_device\_type = None # Should be 0x60 for Extreme/Fianium

self.\_system\_type = None # 0 == Extreme; 1 == Fianium

self.\_inlet\_temperature = None

self.\_emission\_state = None

self.\_setup\_status = None

self.\_interlock\_status = None

self.\_pulse\_picker\_ratio = None

self.\_watchdog\_interval = None

self.\_power\_level = None

self.\_current\_level = None

self.\_nim\_delay = None

if portname: # Allow user to init specific NKT Laser on portname

self.\_portname = portname

self.\_device\_type = 96 # Could put check here

else: # Search for connection w/ laser

# Open available ports

nkt.openPorts(nkt.getAllPorts(), 1, 1)

# Get open NKT ports

portlist = nkt.getOpenPorts().split(',')

extreme\_found = False

for portName in portlist: # Sweep open nkt ports

# get binary devList of connected nkt devices

comm\_result, devList = nkt.deviceGetAllTypes(portName)

# Get byte at location 15 (device address for extreme/fianium)

device\_type = devList[self.module\_address]

# Double check device\_type matches extreme/fianium laser

if hex(device\_type) == '0x60': # 96 == 0x60 in hex

if extreme\_found: # If extreme found on other port, error

err\_msg = ('''Multiple NKT Lasers found on computer.

COM port 1 = %s

COM port 2 = %s

Please initialize Extreme class with designated \

portname to avoid conflict'''

% (self.portname, portName))

raise RuntimeError(err\_msg)

else: # If this is first laser found,

extreme\_found = True

self.\_portname = portName

self.\_device\_type = device\_type

# Close all ports

closeResult = nkt.closePorts('')

if extreme\_found:

print('NKT Extreme/Fianium Found:')

print('Comport: ', self.portname, 'Device type: ', "0x%0.2X"

% self.device\_type, 'at address:', self.module\_address)

print('System Type = ', self.system\_type)

print('Inlet Temperature = %3.1f C' % self.inlet\_temperature)

else:

print('No Extreme/Fianium Laser Found')

module\_address = property(lambda self: self.\_module\_address)

"""int, read-only: Module address = 15 for Extreme/Fianium."""

device\_type = property(lambda self: self.\_device\_type)

"""int, read-only: Should be 96 (0x60) for Extreme/Fianium.

Assigned and checked during object init."""

portname = property(lambda self: self.\_portname)

"""str, read-only: COM port for laser.

Autofound during init if not given. User can supply when creating object.

"""

@property

def system\_type(self):

"""

str, read-only:

Access register 0x6B to determine Extreme/Fianium

From Manual:

The system type is a newer implementation used to differentiate systems

with minor differences. Older versions might not respond to this

register. In that case the system type should be interpreted as

0 = SuperK Extreme. 8-bit unsigned integer.

0 = SuperK Extreme

1 = SuperK Fianium

"""

# Determine whether system is Extreme or Fianium

register\_address = 0x6B

\_, type\_index = nkt.registerReadU8(self.portname,

self.module\_address,

register\_address, -1)

if not type\_index: # Errors should default to Extreme

type\_index = 0

type\_list = ["SuperK Extreme", "SuperK Fianium"]

self.\_system\_type = type\_list[type\_index]

return self.\_system\_type

@property

def inlet\_temperature(self):

"""

float, read-only:

Accesses register 0x11 to return inlet temperature w/ 0.1 C precision.

Updates the value of non-public attr when called.

Return

------

float

Inlet temperature w/ 0.1 C precision.

"""

register\_address = 0x11

comm\_result, value = nkt.registerReadS16(self.portname,

self.module\_address,

register\_address, -1)

self.\_inlet\_temperature = value / 10

return self.\_inlet\_temperature

@property

def emission\_state(self):

"""

Accesses register 0x30 to return emission state of laser.

Updates the value of non-public attr when called.

Return

------

bool

True = emission off; False = emission on

"""

register\_address = 0x30

comm\_result, value = nkt.registerReadU8(self.portname,

self.module\_address,

register\_address, -1)

if value == 3:

self.\_emission\_state = True

elif value == 0:

self.\_emission\_state = False

else:

self.\_emission\_state = 'Unknown'

print('Unknown Emissions State Detected')

return self.\_emission\_state

@property

def setup\_status(self):

"""

Reads value of register 0x16 and returns corresponding status message.

See Extreme.setup\_options for possible outcomes. Use Extreme.set\_mode()

to change value.

Returns

-------

str

Current setup status of laser based on manual values.

"""

register\_address = 0x16

comm\_result, setup\_key = nkt.registerReadU8(self.portname,

self.module\_address,

register\_address, -1)

self.\_setup\_status = Extreme.setup\_options[setup\_key]

return self.\_setup\_status

@property

def interlock\_status(self):

"""

Print interlock status to terminal.

Reads register 0x32 and converts bytes into strings based on manual.

Manual:

Reading the interlock register returns the current interlock status,

which consists of two unsigned bytes. The first byte (LSB) tells if the

interlock circuit is open or closed. The second byte (MSB) tells where

the interlock circuit is open, if relevant.

=== === =======================================

MSB LSB Description

=== === =======================================

- 0 Interlock off (interlock circuit open)

0 1 Waiting for interlock reset

0 2 Interlock is OK

1 0 Front panel interlock / key switch off

2 0 Door switch open

3 0 External module interlock

4 0 Application interlock

5 0 Internal module interlock

6 0 Interlock power failure

7 0 Interlock disabled by light source

255 - Interlock circuit failure

=== === =======================================

Return

------

tuple(int, str)

(LSB, Desription) returns result according to table in manual.

"""

register\_address = 0x32

comm\_result, reading = nkt.registerRead(self.portname,

self.module\_address,

register\_address, -1)

LSB = reading[0] # What manual calls first byte

MSB = reading[1] # What manual calls second byte

# Interlock status message based on manual

output\_options = ['Interlock off (interlock circuit open)',

'Front panel interlock/key switch off',

'Door switch open',

'External module interlock',

'Application interlock',

'Internal module interlock',

'Interlock power failure',

'Interlock disabled by light source']

if LSB == 0:

reason = output\_options[MSB]

self.\_interlock\_status = (LSB, 'Interlocked: %s' % reason)

elif LSB == 1:

self.\_interlock\_status = (LSB, 'Waiting for interlock reset')

elif LSB == 2:

self.\_interlock\_status = (LSB, 'Interlock is OK')

return self.\_interlock\_status

@property

def pulse\_picker\_ratio(self):

"""

Get pulse picker ratio by reading register 0x34.

Manual:

For SuperK EXTREME Systems featuring the pulse picker option, the

divide ratio for the pulse picker can be controlled with the pulse

picker ratio register. Note: When reading the pulse picker value, an

8-bit unsigned integer will be returned if the ratio is lower than 256,

and a 16-bit unsigned integer otherwise.This is for historical reasons.

Return

------

ratio : int

Pulse picker divide ratio

"""

register\_address = 0x34

comm\_result, ratio = nkt.registerReadU16(self.portname,

self.module\_address,

register\_address, -1)

self.\_pulse\_picker\_ratio = ratio

return self.\_pulse\_picker\_ratio

@property

def watchdog\_interval(self):

"""

Get the watchdog interval by reading register 0x36.

Manual:

The system can be set to make an automatic shut-off (laser emission

only – not electrical power) in case of lost communication. The value

in the watchdog interval register determines how many seconds with no

communication the system will tolerate. If the value is 0, the

feature is disabled. 8-bit unsigned integer.

Return

------

ratio : int

Pulse picker divide ratio

"""

register\_address = 0x36

comm\_result, interval = nkt.registerReadU8(self.portname,

self.module\_address,

register\_address, -1)

self.\_watchdog\_interval = interval

return self.\_watchdog\_interval

@property

def power\_level(self):

"""

Get power level setpoint with 0.1% precision.

Read register 0x37 and converts from permille to percent.

Return

------

power\_level : float

Power level setpoint in percent w/ 0.1% precision.

"""

register\_address = 0x37

comm\_result, power = nkt.registerReadU16(self.portname,

self.module\_address,

register\_address, -1)

self.\_power\_level = power / 10

return self.\_power\_level

@property

def current\_level(self):

"""

Get current level setpoint with 0.1% precision.

Read register 0x38 and converts from permille to percent.

Return

------

current\_level : float

Current level setpoint in percent w/ 0.1% precision.

"""

register\_address = 0x38

comm\_result, current = nkt.registerReadU16(self.portname,

self.module\_address,

register\_address, -1)

self.\_current\_level = current / 10

return self.\_current\_level

@property

def nim\_delay(self):

"""

Get NIM trigger delay time.

Reads register 0x38 and converts from int value to delay in seconds.

Manual:

On systems with NIM trigger output, the delay of this trigger signal

can be adjusted with the NIM delay register. The input for this

register should be an unsigned 16-bit value from 0 to 1023. The range

is 0 – 9.2 ns. The average step size is 9 ps.

Return

------

nim\_delay : float

Delay time given in seconds.

"""

register\_address = 0x38

step = 9e-12 # Step size for delay is 9 ps

comm\_result, delay = nkt.registerReadU16(self.portname,

self.module\_address,

register\_address, -1)

self.\_nim\_delay = delay \* step

return self.\_nim\_delay

def set\_emission(self, state):

"""

Change emission state of laser to on/off

Uses nktp\_dll to write to register 0x30.

Parameters

----------

state : bool

True turns laser on, false turns emission off

"""

register\_address = 0x30

if state is True:

nkt.registerWriteU8(self.portname, self.module\_address,

register\_address, 0x03, -1)

elif state is False:

nkt.registerWriteU8(self.portname, self.module\_address,

register\_address, 0x00, -1)

def set\_mode(self, setup\_key):

"""

Sets the "setup" of the laser according to options in manual.

Checks value provided is withing Extreme.setup\_options.keys(),

then writes to register 0x16. Get current status w/

Extreme.setup\_status

Manual:

With the Setup register, the operation mode of the SuperK EXTREME

System can be controlled. The possible values are listed below;

however, in some systems, not all modes are available.16-bit unsigned

integer.

0: Constant current mode

1: Constant power mode

2: Externally modulated current mode

3: Externally modulated power mode

4: External feedback mode (Power Lock)

Parameters

----------

setup\_key : int

Interger corresponding to a key inside Extreme.setup\_options

"""

register\_address = 0x16

if setup\_key in Extreme.setup\_options.keys():

nkt.registerWriteU8(self.portname, self.module\_address,

register\_address, setup\_key, -1)

print('Mode set to: ', self.setup\_status)

else:

print('Warning: Invalid Key Provided')

print('Mode remains as: ', self.setup\_status)

def set\_interlock(self, value):

"""

Reset or trip interlock with >0 or 0, respectively.

Manual:

If the door interlock is in place, the key switch on the front plate is

in On position and the External bus is terminated with e.g. a bus

defeater then the Interlock circuit can be reset via the Interbus

interface by sending a value greater than 0 to the Interlock register.

Additionally, the opposite function (switching interlock relays off)

can be done by sending the value 0 to the interlock register.

Parameters

----------

value: int

0 trips interlock. >0 resets interlock.

"""

register\_address = 0x32

if value > 0:

value = 1

else:

value = 0

nkt.registerWriteU8(self.portname, self.module\_address,

register\_address, value, -1)

def set\_pulse\_picker\_ratio(self, ratio):

"""

Sets pulse picker ratio by writing register 0x34.

Manual:

For SuperK EXTREME Systems featuring the pulse picker option, the

divide ratio for the pulse picker can be controlled with the pulse

picker ratio register.

Parameters

----------

ratio : int

Interger corresponding to a key inside Extreme.setup\_options

"""

register\_address = 0x34

if type(ratio) is int:

nkt.registerWriteU16(self.portname, self.module\_address,

register\_address, ratio, -1)

else:

raise ValueError('ratios needs to be int')

def set\_watchdog\_interval(self, timeout):

"""

Set the watchdog interval by calling registerWriteU8 on 0x36.

Manual:

The system can be set to make an automatic shut-off (laser emission

only – not electrical power) in case of lost communication. The value

in the watchdog interval register determines how many seconds with no

communication the system will tolerate. If the value is 0, the

feature is disabled. 8-bit unsigned integer.

Parameters

----------

timeout : int

time (seconds) the system will toleratre for communication loss.

"""

register\_address = 0x36

if type(timeout) is int:

nkt.registerWriteU8(self.portname, self.module\_address,

register\_address, timeout, -1)

else:

raise ValueError('timeout needs to be int')

def set\_power(self, power):

"""

Set power level setpoint with 0.1% precision.

Converts from percent to permille and write register 0x37.

Parameters

----------

power : float

Power level setpoint in percent w/ 0.1% precision. (0 <= P <= 100)

"""

register\_address = 0x37

setpoint = power \* 10

if (power >= 0) and (power <= 100):

nkt.registerWriteU16(self.portname, self.module\_address,

register\_address, setpoint, -1)

else:

self.set\_emission(False)

self.set\_power(0)

raise ValueError("Power must be between 0 and 100%\n"

"Setting output to 0.")

def set\_current(self, current):

"""

Set current level setpoint with 0.1% precision.

converts from percent to permille and write register 0x38.

Parameters

----------

current : float

Current level setpoint in percent w/ 0.1% precision (0 <= I <= 100)

"""

register\_address = 0x38

setpoint = current\*10

if (current >= 0) and (current <= 100):

nkt.registerWriteU16(self.portname, self.module\_address,

register\_address, setpoint, -1)

else:

self.set\_emission(False)

self.set\_current(0)

raise ValueError("Current must be between 0 and 100%\n"

"Setting output to 0.")

def set\_nim\_delay(self, nim\_delay):

"""

Set NIM trigger delay time.

Writes register 0x38 and converts from delay time in seconds to

corresponding int value using setpoint = int(nim\_delay/9e-12)

Manual:

On systems with NIM trigger output, the delay of this trigger signal

can be adjusted with the NIM delay register. The input for this

register should be an unsigned 16-bit value from 0 to 1023. The range

is 0 – 9.2 ns. The average step size is 9 ps.

Parameters

----------

nim\_delay : float

Delay time given in seconds. (0 <= nim\_delay <= 9.207e-9)

"""

register\_address = 0x38

step = 9e-12 # Step size for delay is 9 ps

int\_delay = int(nim\_delay/step)

if (int\_delay >= 0) and (int\_delay <= 1023):

nkt.registerWriteU16(self.portname, self.module\_address,

register\_address, int\_delay, -1)

else:

print('NIM Delay Value Out of Range (0 <= Delay <= 9.207e-9)')

def print\_status(self):

"""

Read system status in bytes, translate to str, print.

Reads system status using registerReadU16 on register 0x66.

Translates binary into str for of equipment status through

Extreme.status\_messages.

Returns

-------

str : bits

binary results of register read in string format.

"""

register\_address = 0x66

result, byte = nkt.registerReadU16(self.portname, self.module\_address,

register\_address, -1)

print(nkt.RegisterResultTypes(result))

bits = bin(byte)

for index, bit in enumerate(reversed(bits)):

if bit == 'b':

break

elif bit == '1':

print(Extreme.status\_messages[index])

return (bits)

def test\_read\_funcs(self):

outputs = (self.system\_type,

self.inlet\_temperature,

self.emission\_state,

self.setup\_status,

str(self.interlock\_status),

self.pulse\_picker\_ratio,

self.watchdog\_interval,

self.power\_level,

self.current\_level,

self.nim\_delay)

output\_msg = ("""

System type = %s

Inlet Temperature = %s

Emission state = %s

Setup status = %s

Interlock Status = %s

Pulse picker ratio = %s

Watchdog interval = %s

Power level = %s

Current level = %s

NIM delay = %s

""" % outputs)

print(output\_msg)

Example 2:

Manual entry –

SuperK VARIA (A301)

This section describes specific registers for SuperK VARIA (A301) accessory.

Module type and module address

The SuperK VARIA module type is 68h. The address depends on the tiny rotary switch (marked 0..9 and A..F) located close to the connectors. The module address is determined by adding 16 (10h) to the hexadecimal setting of the switch.

Monitor input (13h)

Current output power in tenths of percent (permille, ‰). Requires the optional monitor to be attached for this register content to be valid. 16-bit unsigned integer.

ND setpoint (32h)

The output level of the SuperK VARIA is controlled with an unsigned 16-bit integer value sent to the neutral density filter setpoint register. The value in this register is in tenths of percent (permille, ‰).

SWP setpoint (33h)

Short wave pass filter setpoint register in 1/10 nm. 16-bit unsigned integer.

LWP setpoint (34h)

Long wave pass filter setpoint register in 1/10 nm. 16-bit unsigned integer.

Status bits (66h)

This register returns the status bits. 16-bit integer.

Bit 0: -

Bit 1: Interlock off

Bit 2: Interlock loop in

Bit 3: Interlock loop out

Bit 4: -

Bit 5: Supply voltage low

Bit 6: -

Bit 7: -

Bit 8: Shutter sensor 1

Bit 9: Shutter sensor 2

Bit 10: -

Bit 11: -

Bit 12: Filter 1 moving

Bit 13: Filter 2 moving

Bit 14: Filter 3 moving

Bit 15: Error code present

Python code –

"""Python module to control NKT Varia."""

import nkt\_tools.NKTP\_DLL as nkt

class Varia:

status\_messages = {

0: '-',

1: 'Interlock off',

2: 'Interlock loop in',

3: 'Interlock loop out',

4: '-',

5: 'Supply voltage low',

6: '-',

7: '-',

8: 'Shutter sensor 1',

9: 'Shutter sensor 2',

10: '-',

11: '-',

12: 'Filter 1 moving',

13: 'Filter 2 moving',

14: 'Filter 3 moving',

15: 'Error code present'

}

"""

dict : system status translation bits > string.

========= ===================

Bit Index Status

========= ===================

Bit 0: -

Bit 1: Interlock off

Bit 2: Interlock loop in

Bit 3: Interlock loop out

Bit 4: -

Bit 5: Supply voltage low

Bit 6: -

Bit 7: -

Bit 8: Shutter sensor 1

Bit 9: Shutter sensor 2

Bit 10: -

Bit 11: -

Bit 12: Filter 1 moving

Bit 13: Filter 2 moving

Bit 14: Filter 3 moving

Bit 15: Error code present

========= ===================

"""

def \_\_init\_\_(self, portname=None):

"""

Searches for connected NKT Varia and defines instrument parameters.

Make sure devices are not connected via another program already.

If multiple Varias are connected to the same computer,

specificy the port of the desired Varia upon instantiation.

Parameters

----------

portname : str, optional

Enter if portname for Varia is known/multiple lasers are connected.

If not supplied, system searches for Varia. None by default.

Raises

------

RuntimeError

Throws error if multiple NKT Varias are found on one computer.

Supply portname for desired Varia if multiple present.

"""

print('Searching for connected NKT Varia...')

self.\_portname = None # COM port name. Autosearches if not provided.

self.\_module\_address = None # 16-25 for Varia. Auto searches in init.

self.\_device\_type = None # This should update to 0x68 if init is right

if portname: # Allow user to init specific NKT Laser on portname

self.\_portname = portname

else: # User didn't specify port

# Open available ports

nkt.openPorts(nkt.getAllPorts(), 1, 1)

# Get open NKT ports

portlist = nkt.getOpenPorts().split(',')

varia\_found = False

for portName in portlist:

if portName == '':

continue

# get binary devList of connected nkt devices

comm\_result, devList = nkt.deviceGetAllTypes(portName)

# Address for Varia depends on specific hardware.

# Search possible addresses (16-25) searching for connection.

for n in range(9):

trial\_address = 16 + n # Address=16+Rotary switch #(0..9)

try:

device\_type = devList[trial\_address]

except (IndexError):

print('No Varia on port: ', portName)

break

# Check if device\_type matches varia (0x68)

if hex(device\_type) == '0x68': # Check for varia in hex

if varia\_found: # If varia found on other port, error

err\_msg = ('''Multiple Varias found on computer.

COM port 1 = %s

COM port 2 = %s

Please initialize Varia class with designated \

portname to avoid conflict'''

% (self.portname, portName))

raise RuntimeError(err\_msg)

else: # If first laser found,

varia\_found = True

self.\_portname = portName

self.\_module\_address = trial\_address

self.\_device\_type = device\_type

break

# Close all ports

closeResult = nkt.closePorts('')

if varia\_found:

print('NKT Varia Found:')

print('Comport: ', self.portname, 'Device type: ', "0x%0.2X"

% self.device\_type, 'at address:', self.module\_address)

else:

print('No Varia Found')

portname = property(lambda self: self.\_portname)

"""str, read-only: COM port for laser.

Autofound during init if not given. User can supply when creating object.

"""

module\_address = property(lambda self: self.\_module\_address)

"""int, read-only: # 16-25 for Varia. Auto searches in init."""

device\_type = property(lambda self: self.\_device\_type)

"""int, read-only: This should update to 104 (0x68) if init is right.

Assigned and checked during object init."""

@property

def monitor\_input(self):

"""

Uses optional monitor to get laser power in percent.

Calls registerREad16U on register 0x13. Converts reading from in to

float. Requires the optional monitor to be attached for this register

content to be valid.

Returns

-------

float

Output power given in percent with 0.1% precision.

"""

register\_address = 0x13

comm\_result, reading = nkt.registerReadU16(self.portname,

self.module\_address,

register\_address, -1)

print(reading)

output\_power = reading / 10

return output\_power

@property

def nd\_setpoint(self):

"""

Unclear what this parameter actually controls.

Writes to register 0x32.

Manual:

The output level of the SuperK VARIA is controlled with an unsigned

16-bit integer value sent to the neutral density filter setpoint

register. The value in this register is in tenths of percent

(permille, ‰).

Parameters

----------

value : float

Setpoint for neutral density filter given in % with 0.1% precision.

"""

register\_address = 0x32

comm\_result, reading = nkt.registerReadU16(self.portname,

self.module\_address,

register\_address, -1)

return reading/10

@nd\_setpoint.setter

def nd\_setpoint(self, value):

register\_address = 0x32

value = int(value \* 10) # convert percent to permille

nkt.registerWriteU16(self.portname, self.module\_address,

register\_address, value, -1)

@property

def long\_setpoint(self, wavelength):

"""

Sets the short wave pass value with 0.1 nm precision.

Converts wavelength value [nm] to int [1/10 nm] then writes to register

0x33.

Parameters

----------

wavelength : float

Lower bandpass value given in nanometers w/ 0.1 nm precision.

"""

register\_address = 0x33

comm\_result, reading = nkt.registerReadU16(self.portname,

self.module\_address,

register\_address, -1)

return reading/10

@long\_setpoint.setter

def long\_setpoint(self, wavelength):

register\_address = 0x33

value = int(wavelength \* 10) # convert nm to 1/10 nm

nkt.registerWriteU16(self.portname, self.module\_address,

register\_address, value, -1)

@property

def short\_setpoint(self):

"""

Sets the long wave pass value with 0.1 nm precision.

Converts wavelength value [nm] to int [1/10 nm] then writes to register

0x34.

Parameters

----------

wavelength : float

Upper bandpass value given in nanometers w/ 0.1 nm precision.

"""

register\_address = 0x34

comm\_result, reading = nkt.registerReadU16(self.portname,

self.module\_address,

register\_address, -1)

return reading/10

@short\_setpoint.setter

def short\_setpoint(self, wavelength):

register\_address = 0x34

value = int(wavelength \* 10) # convert nm to 1/10 nm

nkt.registerWriteU16(self.portname, self.module\_address,

register\_address, value, -1)

def print\_status(self):

"""

Read system status in bytes, translate to str, print.

Reads system status using registerReadU16 on register 0x66.

Translates binary into str for of equipment status through

Varia.status\_messages.

Returns

-------

str : bits

binary results of register read in string format.

"""

register\_address = 0x66

result, byte = nkt.registerReadU16(self.portname, self.module\_address,

register\_address, -1)

print(nkt.RegisterResultTypes(result))

bits = bin(byte)

for index, bit in enumerate(reversed(bits)):

if bit == 'b':

break

elif bit == '1':

print(Varia.status\_messages[index])

return (bits)

def read\_all\_properties(self):

print('Input Power = ', self.monitor\_input)

print('ND Setpoint = ', self.nd\_setpoint)

print('Long Setpoint = ', self.long\_setpoint)

print('Short Setpoint = ', self.short\_setpoint)

def demo\_nkt\_registerReads(self):

"""

Tests various registerRead functions supplied by NKTPDLL.

Makes read request to register 0x66 (system status) using various

registerRead commands as an example of the data type returned by each.

"""

register\_address = 0x66

print('registerRead: ',

nkt.registerRead(self.portname, self.module\_address,

register\_address, -1))

print('registerReadU8: ',

nkt.registerReadU8(self.portname, self.module\_address,

register\_address, -1))

print('registerReadS8: ',

nkt.registerReadS8(self.portname, self.module\_address,

register\_address, -1))

print('registerReadU16: ',

nkt.registerReadU16(self.portname, self.module\_address,

register\_address, -1))

print('registerReadU32: ',

nkt.registerReadU32(self.portname, self.module\_address,

register\_address, -1))

print('registerReadF32: ',

nkt.registerReadF32(self.portname, self.module\_address,

register\_address, -1))

print('registerReadAscii: ',

nkt.registerReadAscii(self.portname, self.module\_address,

register\_address, -1))

Can you please create a similar python wrapper for the following manual entry:

6.10 SuperK SELECT (A203)

This section describes specific registers for a SuperK SELECT.

Module type and module address

The SuperK SELECT module type is 67h. The address depends on the tiny rotary switch (marked 0..9 and A..F) located close to the connectors. The module address is determined by adding 16 (10h) to the hexadecimal setting of the switch.

Monitor 1 readout (10h)

Readout from optional optical power monitor no. 1 in tenths of percent (permille, ‰). 16-bit unsigned integer.

Monitor 2 readout (11h)

Readout from optional optical power monitor no. 2 in tenths of percent (permille, ‰). 16-bit unsigned integer.

Monitor 1 gain (32h)

Gain setting for optional optical power monitor no.1. There are eight gain levels, numbered 0..7, with 0 being thet lowest gain level. Each level increase doubles the sensitivity. 8-bit unsigned integer.

Note: Monitor gain settings should not be altered when the SuperK light source is running in external feedback mode (Power Lock).

Monitor 2 gain (33h)

Gain setting for optional optical power monitor no.2. There are eight gain levels, numbered 0..7, with 0 being thet lowest gain level. Each level increase doubles the sensitivity. 8-bit unsigned integer.

Note: Monitor gain settings should not be altered when the SuperK light source is running in external feedback mode (Power Lock).

RF switch (34h)

The SuperK SELECT is equipped with two RF input connectors, and one or two AOTF crystals. When switching from one crystal to the other, the operator may either unplug the RF cable and plug it into the other connector, or use the internal RF switch. When activated, the RF switch swaps the two RF connections. Set the register value to 0 for normal operation, or 1 to swap the crystal connections. 8-bit unsigned integer.

Please note, that RF power must be off before the RF switch position is changed.

Monitor switch (35h)

The SuperK SELECT can have up to two optical power monitors, but has only one monitor output connector. Use the Monitor switch register to select which power detector should be connected to the output connector. Set the register value to 0 to get the power from the first crystal, or 1 to get the power from the second crystal. 8-bit unsigned integer.

Crystal 1 minimum wavelength (0x90)

Returns the minimum usable wavelength in crystal 1. The value is an unsigned 32-bit integer, and the resolution is pm.

Crystal 1 maximum wavelength (0x91)

Returns the maximum usable wavelength in crystal 1. The value is an unsigned 32-bit integer, and the resolution is pm.

Crystal 2 minimum wavelength (0xA0)

Returns the minimum usable wavelength in crystal 2. The value is an unsigned 32-bit integer, and the resolution is pm.

Crystal 2 maximum wavelength (0xA1)

Returns the maximum usable wavelength in crystal 2. The value is an unsigned 32-bit integer, and the resolution is pm.