OmmpNET Python OptoMMP Software

Copyright:

Description:

This software is a Python3.4 implementation of the OptoMMP protocol as described in the document, **OPTOMMP PROTOCOL GUIDE, Form 1465-180319—March 2018,** available at https://www.opto22.com/support/resources-tools/documents/1465-optommp-protocol-guide.

At present, code is limited in functionality and confined to the file OptoNET.py.

Testing has been limited to a small number of digital commands to verify the communications packet structures. However, the commands OrderedDict is relatively complete and most functions to implement the remaining commands would tend to be 1 or 2 lines of code per command.

Note: The write/read quadlet/block functions contain debugging prints which may be a bit verbose. Eventually they will be changed to use logging. Consider commenting them out once you are comfortable with the communication packet verification.

OmmpNET.py:

Commands:

Commands is an OrderedDict containing entries such as:

('Status R/W Operation Code',CmdFmt(prng(0xf0380000),4,'UI','I')),

- The key is a command name text string
- The value is a CmdFmt namedtuple

CmdFmt:

CmdFmt is defined as:

CmdFmt = namedtuple('CmdFmt',['ofs','len','typ','pak'])

The fields are used as follows:

- ofs – a prng which can be used to compute the lower part of the memory map offset
- len – the OptoMMP command data length in bytes
- typ – the OptoMMP data type as defined in the users guide section, **A: Opto 22 Hardware Memory Map**
- pak – this is a conversion from the OptoMMP type to the closest Python pack/unpack format type

CmdFmt.ofs:

- The function prng is used to create a Python range by setting up the beginning, ending, and range step. The beginning range is specified in the user's guide column.

- Many of the memory parameters exist as an array of values which are located on specified boundaries.
    - As an example, older racks could support up to 16 digital 4 point modules. So there could be digital 64 points.
    - Each digital point has a number of parameters that can be read and/or written.
    - The manual section DIGITAL POINT WRITE—READ/WRITE indicates that “only the first three points are shown in the table. Each successive point starts on an even 40 hex boundary and follows the same pattern.
    - The dictionary entry as: ('Digital Point State',CmdFmt(prng(0xF0800000,64,64),4,'B','I')) is used to create a range of 64 Digital Point State' offset values on 64 byte boundaries. This means the digital state of the first point is found at 0xF0800000, and the remaining 63 point states are addressed thereafter every 64 bytes. This is done as: range(ofs,cnt*siz+ofs,siz).
    - Memory mapped values that are not an array element have a default range(ofs,ofs+1) which means the range only contain the CmdFmt.ofs[0] element.

Index argument:

- The index argument is used with the prng value CmdFmt.ofs[index] to select an array element when necessary.
- If the memory mapped value to be read or written is not an array, then prng only contains the beginning offset.

Reading and writing memory mapped values:

Functions are provided for reading and writing values in the memory map. They are:

- def write_mmap_value(self,aa,cmd,value,index=0)
- def read_mmap_value(self,aa,cmd,index=0).

These functions call the appropriate block or quadlet read or write depending on the length returned by calcsize(CmdFmt.pak). At present, there is no function provided for reading a block of values, but doing so would be relatively easy by concatenating the contiguous memory map CmdFmt.pak values into a pack/unpack format string and then calling calcsize on the resulting string.

Example routine to generate a 10Hz squarewave using timer events:

```
def generate_10_hz_sqw_on_point_0(self,aa):
    """
    cfg p0.0 as a digital output
    deactivate p0.0
    set timer 0 delay to 50ms
    set timer 0 start event on activation of P0.0
    set timer 0 reaction event to turn off P0.0
    set timer 1 delay to 50ms
    set timer 1 start event to P0.0 clear
    set timer 1 reaction event to set P0.0
    activate p0.0 to start squarewave
    """
    # send a power up clear
    print('send a power up clear',self.power_up_clear(aa))
    # Set first point to an Output
    print('make digital point 0 an output',self.set_point_type(aa,0,0x180))
    # turn Off point P0.0
    print('turn off digital point 0',self.deactivate_digital_point(aa,0))
    # Set up a 50ms timer
    print('set t0 delay',self.set_timer_delay(aa,0,50))
    print('get t0 delay',self.get_timer_delay(aa,0))
    # start it when P0.0 Sets
    print('set t0 trigger on mask',self.set_timer_trigger_on_mask(aa,0,1))
    print('get t0 trigger on mask',self.get_timer_trigger_on_mask(aa,0))
    # use Event to clear P0.0
    print('set t0 reaction off mask',self.set_timer_reaction_off_mask(aa,0,1))
    print('get t0 reaction off mask',self.get_timer_reaction_off_mask(aa,0))
    # Set up a 50ms timer
    print('set t1 delay',self.set_timer_delay(aa,1,50))
    print('get t1 delay',self.get_timer_delay(aa,1))
    # start it when P0.0 clears
    print('set t1 trigger off mask',self.set_timer_trigger_off_mask(aa,1,1))
    print('get t1 trigger off mask',self.get_timer_trigger_off_mask(aa,1))
    # use Event to Set P0.0
    print('set t1 reaction on mask',self.set_timer_reaction_on_mask(aa,1,1))
    print('get t1 reaction on mask',self.get_timer_reaction_on_mask(aa,1))
    # turn On P0.0 to start timer 0
    print('turn on digital point 0 to start sqw',self.activate_digital_point(aa,0))
```

Debugging output for the above:

```
>>> om.generate_10_hz_sqw_on_point_0(aa)
WriteQuadletRequest(tl=0, tcode=0, dofs_hi=65535, dofs_lo=4030201856, value=1)
WriteQuadletRequest b'000000000000ffffff038000000000001'
WriteResponse b'000000200000000000000000'
send a power up clear RspFmt(rcode=0, data='No Error')
WriteQuadletRequest(tl=4, tcode=0, dofs_hi=65535, dofs_lo=4039114756, value=384)
WriteQuadletRequest b'000004000000fffff0c0000400000180'
WriteResponse b'000004200000000000000000'
make digital point 0 an output RspFmt(rcode=0, data='No Error')
WriteQuadletRequest(tl=8, tcode=0, dofs_hi=65535, dofs_lo=4035969028, value=1)
WriteQuadletRequest b'000008000000fffff090000400000001'
WriteResponse b'000008200000000000000000'
turn off digital point 0 RspFmt(rcode=0, data='No Error')
WriteQuadletRequest(tl=12, tcode=0, dofs_hi=65535, dofs_lo=4040425536, value=50)
WriteQuadletRequest b'00000c000000fffff0d4004000000032'
WriteResponse b'00000c200000000000000000'
set t0 delay RspFmt(rcode=0, data='No Error')
ReadQuadletRequest(tl=16, tcode=64, dofs_hi=65535, dofs_lo=4040425536)
ReadQuadletRequest b'000010400000fffff0d40040'
ReadQuadletResponse b'000010600000000000000000000000032'
get t0 delay RspFmt(rcode=0, data=(50,))
WriteBlockRequest(tl=20, tcode=16, dofs_hi=65535, dofs_lo=4040425472, length=8, value=1)
WriteBlockRequest b'000014100000fffff0d400000080000000000000000001'
WriteResponse b'000014200000000000000000'
set t0 trigger on mask RspFmt(rcode=0, data='No Error')
ReadBlockRequest(tl=24, tcode=80, dofs_hi=65535, dofs_lo=4040425472, length=8)
ReadBlockRequest b'000018500000fffff0d4000000080000'
ReadBlockResponse b'000018700000000000000000000080032000000000000001'
get t0 trigger on mask RspFmt(rcode=0, data=(1,))
WriteBlockRequest(tl=28, tcode=16, dofs_hi=65535, dofs_lo=4040425512, length=8, value=1)
WriteBlockRequest b'00001c100000fffff0d400280008000000000000000001'
WriteResponse b'00001c200000000000000000'
set t0 reaction off mask RspFmt(rcode=0, data='No Error')
ReadBlockRequest(tl=32, tcode=80, dofs_hi=65535, dofs_lo=4040425512, length=8)
ReadBlockRequest b'000020500000fffff0d4002800080000'
ReadBlockResponse b'000020700000000000000000000080032000000000000001'
get t0 reaction off mask RspFmt(rcode=0, data=(1,))
WriteQuadletRequest(tl=36, tcode=0, dofs_hi=65535, dofs_lo=4040425664, value=50)
WriteQuadletRequest b'000024000000fffff0d400c000000032'
WriteResponse b'000024200000000000000000'
set t1 delay RspFmt(rcode=0, data='No Error')
ReadQuadletRequest(tl=40, tcode=64, dofs_hi=65535, dofs_lo=4040425664)
ReadQuadletRequest b'000028400000fffff0d400c0'
ReadQuadletResponse b'000028600000000000000000000000032'
get t1 delay RspFmt(rcode=0, data=(50,))
WriteBlockRequest(tl=44, tcode=16, dofs_hi=65535, dofs_lo=4040425608, length=8, value=1)
WriteBlockRequest b'00002c100000fffff0d400880008000000000000000001'
```

WriteResponse b'00002c200000000000000000'
set t1 trigger off mask RspFmt(rcode=0, data='No Error')
ReadBlockRequest(tl=48, tcode=80, dofs_hi=65535, dofs_lo=4040425608, length=8)
ReadBlockRequest b'000030500000fffff0d4008800080000'
ReadBlockResponse b'000030700000000000000000000080032000000000000000001'
get t1 trigger off mask RspFmt(rcode=0, data=(1,))
WriteBlockRequest(tl=52, tcode=16, dofs_hi=65535, dofs_lo=4040425632, length=8, value=1)
WriteBlockRequest b'000034100000fffff0d400a000080000000000000000000001'
WriteResponse b'00003420000000000000000'
set t1 reaction on mask RspFmt(rcode=0, data='No Error')
ReadBlockRequest(tl=56, tcode=80, dofs_hi=65535, dofs_lo=4040425632, length=8)
ReadBlockRequest b'000038500000fffff0d400a000080000'
ReadBlockResponse b'000038700000000000000000000080032000000000000000001'
get t1 reaction on mask RspFmt(rcode=0, data=(1,))
WriteQuadletRequest(tl=60, tcode=0, dofs_hi=65535, dofs_lo=4035969024, value=1)
WriteQuadletRequest b'00003c000000fffff090000000000001'
WriteResponse b'00003c200000000000000000'
turn on digital point 0 to start sqw RspFmt(rcode=0, data='No Error')
>>>