OPTO 22 SNAP UP1 ADS
READ NETWORK SETTINGS
USING LINUX

OPTO22

Opto22 and its industrial hardware is known for its openness in protocols, documentation and bug reports. The haven't a complete Linux based suite to work with their equipment, however they produced some specific linux apps!.

SNAP UP1 ADS

This device is a combination of a controller (CPU) and I/O. This is not an Opto22's last generation equipment, and is not recommended for new designs, however still in production and supported. Due to its great flexibility to program diverse communication protocols, it's very attractive for developing Internet Of Things (IoT) gateways

The SNAP UP1 ADS have an on-board Ethernet. Its primary use is for general setup, download of control strategies and remote monitoring HMI. When the controller is in an Ethernet network, the manufacturer provides a Windows app named PAC MANAGER (part of a PAC PROJECT free suite), used to detect devices plugged into the network.

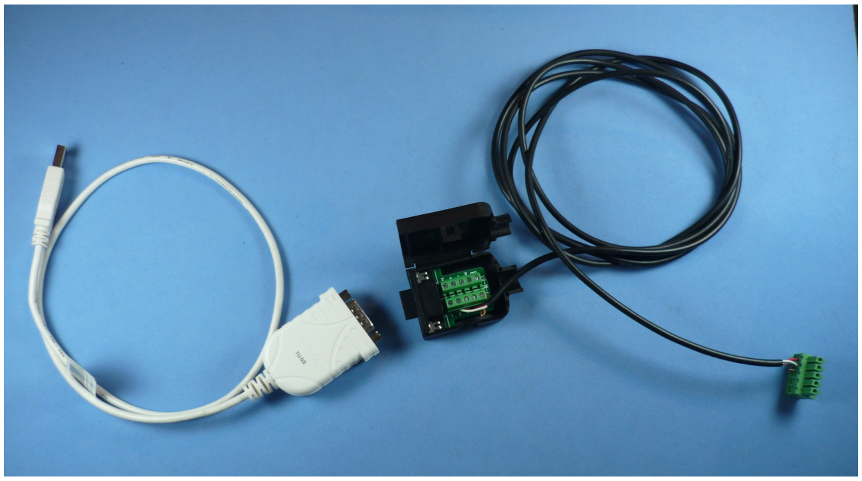
The main problem with the application, is that the device must have similar network settings to the computer running the app! (same network segment settings).If not the device isn't detected. If the network parameters of the device are unknown, there are only 2 alternatives:

- Reset device to default settings to set a new known network parameters (erasing control strategy and other configurations)

- Use another application via serial port to get the parameters

The application is OptoFlash-ENET, used to read and modify network parameters, get hardware info and update firmware for compatible devices: SNAP Ethernet brains, SNAP Ultimate and M4SENET-100 card

To communicate the application with the device a serial cable must be built. If there is no available serial port in the computer an USB to RS232 cable could be used, and if no solder is desired a convenient way is to use a DB-9 breakout connector!



FRAME ANALYSIS

OptoFlash-ENET uses an ASCII protocol to establish communications with the hardware. ASCII protocols are easy to analyse and the app provides a debugging window where all the communication frames can be seen. The app does 3 things:

1. 1. Send "Power Up Clear" command

2. Send Hardware info query

3. Send Network Settings info query

Opto22 literature says "power up clear" command clears a flag that is set every time a controller is turned on or rebooted. With this flag the monitoring app knows that something happened. If the acknowledgement of power up isn't sent, the controller refuses to execute any other command, so the application sends power up clear before any other query.

"power up clear" frame:

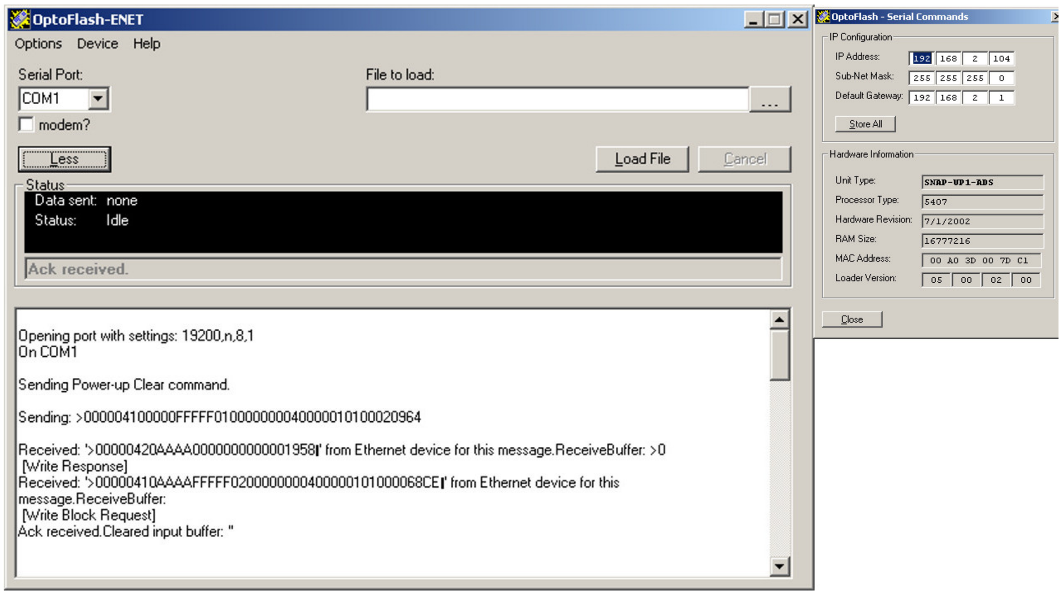
>000004100000FFFFF0100000000040000010100020964

Last four digits are CRC-16

Network settings query frame

>000004100001FFFFF0100000000400000101000CCD24

Again last four digits are CRC-16.



Serial Port: COM1
modem? ☐

Status
Data sent: none
Status: Idle
Ack received.

```
Opening port with settings: 19200,n,8,1
On COM1

Sending Power-up Clear command.

Sending: >000004100000FFFFF0100000000400000101000020964

Received: '>00000420AAAA0000000000001958' from Ethernet device for this message.ReceiveBuffer: >0
 [Write Response]
Received: '>00000410AAAAFFFFF020000000004000001010000068CE' from Ethernet device for this
message.ReceiveBuffer:
 [Write Block Request]
Ack received.Cleared input buffer: ''
```

IP Configuration
IP Address: 192 168 2 104
Sub-Net Mask: 255 255 255 0
Default Gateway: 192 168 2 1

Hardware Information
Unit Type: SNAP-UP1-ADS
Processor Type: 5407
Hardware Revision: 7/1/2002
RAM Size: 16777216
MAC Address: 00 A0 3D 00 7D C1
Loader Version: 05 00 02 00

COMMUNICATION USING LINUX

In the debugging window of OptoFlash-ENET app is noted that the serial communication parameters are 19200,8,N,1. Pretty common setup in industrial hardware!

Due to the communication frames are ASCII representation of HEX numbers, a serial port terminal program in Linux able to send characters is enough to establish communication.In this case CUTECOM was used

First "power up clear" command must be sent

>000004100000FFFFF0100000000400000101000020964

The device's answer similar to this:

>00000420AAAA0000000000001958
>00000410AAAAFFFFF020000000004000001010000068CE.

Next, network settings info query is sent:

>000004100001FFFFF0100000000400000101000CCD24

The device's answer similar to this:

>00010420AAAA000000000000E55C
>00010410AAAAFFFFF0200000004A0000010100000000000105000200400000 8F070107D20
100000
0C0A80268FFFFFF00C0A8020100000000000000000000000000000000FFFFFFFF008900170
000000
00079FFFF00A03D007DC15952

The windows app showed the actual network parameters, so there is known strings to search in the received frames, the most obvious is:

>00010420AAAA000000000000E55C
>00010410AAAAFFFFF0200000004A0000010100000000000105000200400000 8F070107D20
100000
0C0A80268FFFFFF00C0A8020100000000000000000000000000000000FFFFFFFF008900170
000000
00079FFFF**00A03D007DC1**5952

Which is the MAC shown in the Windows app

As the frame is the ASCII representation of HEX numbers, IP address and other network parameters must be encoded in that way!.

OptoFlash-ENET ip address displayed as 192.168.2.104 encoded in HEX: C0A80268. Looking for this string in the received frame:

>00010420AAAA000000000000E55C
>00010410AAAAFFFFF0200000004A0000010100000000000105000200400000 8F070107D20
100000
0
C0A80268FFFFFF00C0A8020100000000000000000000000000000000FFFFFFFF0089001700
00000
00079FFFF00A03D007DC15952

Looking closely the following characters FFFFFF00C0A80201 are HEX representation of 255.255.255.0 and 192.168.2.1 so the blocks that match actual network parameters settings were found!. The remaining characters in the frame probably are another info not shown in OptoFlash-ENET or are placeholders to add more information in the future.

This information is the first step to develop a Linux app that automatizes de process of sending, receiving and frame analysis

More test with different hardware models are required to see if the frame structure and size is the same, or if is different for other type of hardware