


Pinger.1 Profinet Guide

True Loan

Exported on 04/10/2023

Table of Contents

1	Host Setup.....	4
1.1	CODESYS - Windows Host Machine	4
2	Profinet Slave Setup - building the app yourself.....	5
2.1	Get the MCU+ SDK and CCS IDE.....	5
2.2	Build the SD SBL.....	6
2.3	Build the Profinet Example.....	6
2.4	Verify Pinger First Using Linux.....	6
2.5	Setup Profinet SD Card	7
3	Profinet Controller Setup - phyBOARD-Electra	9
4	Configure the Profinet Network - CODESYS.....	10

 The MCU+ SDK hasn't been fully ported for use with the phyCORE-AM64x (PCM-072.A1) and Pinger Carrier board (PL7030.1). Things are very much preliminary and not without bugs/limitations.

You need access to this repo: https://github.com/tloanPhytec/mcu_plus_sdk_am64x_08_01_00_36.git
Ask True if you need access. This repo will eventually be moved to a location we all have default internal access to.

1 Host Setup

1.1 CODESYS - Windows Host Machine

The Profinet Controller is configured using the CODESYS development system on a windows machine. Create an account and install it:

<https://us.store.codesys.com/codesys.html>

This will install the CODESYS Development System (this can be thought of as your primary IDE for working with the industrial network) to the windows machine but some of the default dependencies it installs will need to be upgraded for use with ARM64, specifically the CODESYS Gateway (this is the server the CODESYS IDE will use to communicate to the controller, you wont interact with it directly)

Download this to your windows machine:

<https://us.store.codesys.com/codesys-edge-gateway-for-linux.html>

Open the CODESYS development system and create a new standard project.

navigate to Tools > CODESYS Installer

Install the new gateway package yRight click the project and select Build Configurations → Set Active → Release ou downloaded.

Download this to your windows machine too (64bit version):

<https://us.store.codesys.com/codesys-control-for-linux-arm-sl-bundle.html>

And install it using the CODESYS Installer

2 Profinet Slave Setup - building the app yourself

2.1 Get the MCU+ SDK and CCS IDE

You will need git on your windows machine to clone the mcu+ sdk to your local system.

<https://gitforwindows.org/>

Start git Bash and navigate to the C:/ disk

```
cd C:/
```

Create a directory to install

```
mkdir ti
cd ti
```

Its recommended to install these things to the location specified in this guide. You can deviate but problems arise when the Path contains spaces so be mindful (I think spaces and other characters that are ok in Windows paths are not handled well by the Clang compile used here in the Windows environment.

clone the mcu+ sdk:

```
git clone https://github.com/tloanPhytec/mcu_plus_sdk_am64x_08_01_00_36.git
```

Install the host dependencies outlined here:

https://software-dl.ti.com/mcu-plus-sdk/esd/AM64X/08_02_00_31/exports/docs/api_guide_am64x/SDK_DOWNLOAD_PAGE.html

Now install CCS to the same C:/ti directory

https://www.googleadservices.com/pagead/aclk?sa=L&ai=DChcSEwjPhJl15j-AhVymFsKHRTiAVkYABAAGgJ5bQ&ohost=www.google.com&cid=CAESa-D2OBeHb76MFN_GSrKQu2wU83u5iJh3cu8Aw3uhzCHTFpJp5vNX7O9I1iqiSLGWtiRBP5B8aybwWcfD_6OTRoF5c6D3RWkbJPHEvOQ04Co7WoRC9Vskf8xzKAXhj0FiKTfaus61Etc0a8SS&sig=AOD64_1bMPrwuevxuw_aDpcTemtADrc_iA&q&adurl&ved=2ahUKEwjX4IzI15j-AhXMkYkEHWoLCE0Q0Qx6BAgHEAE

https://www.googleadservices.com/pagead/aclk?sa=L&ai=DChcSEwjPhJl15j-AhVymFsKHRTiAVkYABAAGgJ5bQ&ohost=www.google.com&cid=CAESa-D2OBeHb76MFN_GSrKQu2wU83u5iJh3cu8Aw3uhzCHTFpJp5vNX7O9I1iqiSLGWtiRBP5B8aybwWcfD_6OTRoF5c6D3RWkbJPHEvOQ04Co7WoRC9Vskf8xzKAXhj0FiKTfaus61Etc0a8SS&sig=AOD64_1bMPrwuevxuw_aDpcTemtADrc_iA&q&adurl&ved=2ahUKEwjX4IzI15j-AhXMkYkEHWoLCE0Q0Qx6BAgHEAE

https://www.googleadservices.com/pagead/aclk?sa=L&ai=DChcSEwjPhJl15j-AhVymFsKHRTiAVkYABAAGgJ5bQ&ohost=www.google.com&cid=CAESa-D2OBeHb76MFN_GSrKQu2wU83u5iJh3cu8Aw3uhzCHTFpJp5vNX7O9I1iqiSLGWtiRBP5B8aybwWcfD_6OTRoF5c6D3RWkbJPHEvOQ04Co7WoRC9Vskf8xzKAXhj0FiKTfaus61Etc0a8SS&sig=AOD64_1bMPrwuevxuw_aDpcTemtADrc_iA&q&adurl&ved=2ahUKEwjX4IzI15j-AhXMkYkEHWoLCE0Q0Qx6BAgHEAE

<https://www.ti.com/tool/CCSTUDIO>

Setup your workspace in C:\ti\workspace (otherwise, be mindful of spaces in your path)

And then configure the CCS "Preferences" per the following to point to the SDK and other dependencies installed in the steps above:

https://software-dl.ti.com/mcu-plus-sdk/esd/AM64X/08_02_00_31/exports/docs/api_guide_am64x/CCS_SETUP_PAGE.html#CCS_PACKAGE_CHECK

2.2 Build the SD SBL

Navigate to View → Project Explorer to show the Project explorer, if not already up by default.

Right click the Project Explorer and select Import → CCs Project

Navigate to C:\ti\mcu_plus_sdk_am64x_08_01_00_36\examples\drivers\boot\sbl_sd\am64x-evm (this SBL SD example has the PHYCORE-AM64x DDR timings already provisioned in it).

Select the SD SBL Example that appears here and hit Finish.

Right click the project and select Build Configurations → Set Active → Release

Your deployed binaries will be in C:\ti\workspace\sbl_sd_am64x-evm_r5fss0-0_nortos_ti-arm-clang\Release

The specific SBL you want is deployed as **tiboot3.bin**

2.3 Build the Profinet Example

Navigate to View → Project Explorer to show the Project explorer, if not already up by default.

Right click the Project Explorer and select Import → CCs Project

Navigate to C:

\ti\mcu_plus_sdk_am64x_08_01_00_36\examples\industrial_comms\profinet_device_demo\irt_rgmii\phycore-am64x and hit "Select Folder"

Select the Profinet Example that appears here and hit Finish.

Right click the project and select Build Configurations → Set Active → Release (setting the Build config to "Release" is necessary to boot the application from SD Card, otherwise the app image is too large for the default SD SBL provided in this SDK to load it).

Your deployed binaries will be in C:\ti\workspace\profinet_device_irt_rgmii_demo_phycore-am64x_r5fss0-0_freertos_ti-arm-clang\Release

The specific app image you want is **profinet_device_irt_rgmii_demo_phycore-am64x_r5fss0-0_freertos_ti-arm-clang.appimage**

2.4 Verify Pinger First Using Linux

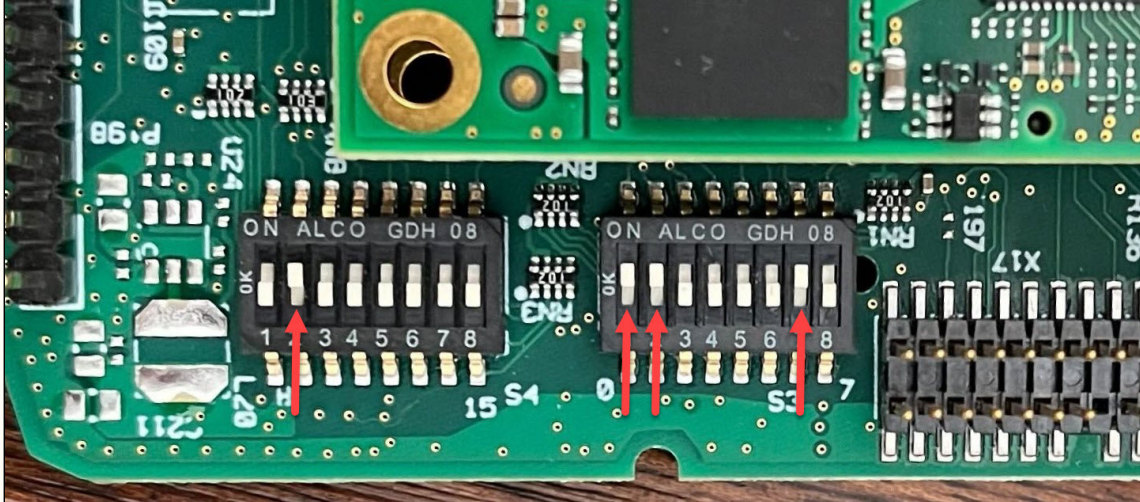
First ensure your Pinger 7030.1 carrier board has the following rework

- Must populate the zero-Ohm resistor at R223. This is necessary for the DP83869 ethernet phy to power-up properly.
- Must populate DH83869 PHYs if not present

And then use the following image to verify ethernet PHYs are working as expected using Linux: (ie flash this to an SD Card and boot the pinger with it)

P:\Development\PBA-C-28 phyGATE-AM64x\Software\ALPHA2-BSP\tisdk-default-image-phyboard-PINGER-am64xx-1.wic.xz

⚠ The Boot switch settings are the same as the Electra, but the switches are laid out differently.



With Ethernet verified on both ports, power off the pinger.

2.5 Setup Profinet SD Card

Using the same Pinger-Linux SD Card (we are going to re-use the FAT partition set up in the Linux image), delete all files in the Boot partition of the SD Card.

Copy tiboot3.bin to the Boot partition.

Copy profinet_device_irt_rgmii_demo_phycore-am64x_r5fs0-0_freertos_ti-arm-clang.appimage to the Boot partition of the SD Card and **rename it to "app"**.

Connect either PRU-Ethernet Port on the Pinger to the Electra's Eth1 interface.

Now boot your Profinet SD Card on the Pinger. You should see the following on the UART0 serial console

```
[BOOTLOADER PROFILE] SYSFW Load           :      18304us
[BOOTLOADER PROFILE] System_init           :      13581us
[BOOTLOADER PROFILE] Drivers_open          :       3352us
[BOOTLOADER PROFILE] Board_driversOpen     :           0us
[BOOTLOADER PROFILE] File read from SD card :      33868us
[BOOTLOADER PROFILE] CPU load              :      13351us
[BOOTLOADER_PROFILE] SBL Total Time Taken   :      83156us

Image loading done, switching to application ...
[APP] INFO: Configuring and starting PROFINET stack...
[APP] INFO: Initializing PRU instance ...

Did Map 0x30080000 len 0x2000 to 0x30080000
(dram0)
Did Map 0x30082000 len 0x2000 to 0x30082000 (dram1)
Did Map 0x300b4000 len 0x4000 to 0x300b4000 (iram0)
```

```

Did Map 0x300b8000 len 0x4000 to 0x300b8000 (iram1)
Did Map 0x30090000 len 0x10000 to 0x30090000 (shdram)
Did Map 0x300a2000 len 0x400 to 0x300a2000 (control0)
Did Map 0x300a4000 len 0x400 to 0x300a4000 (control1)
Did Map 0x300a0000 len 0x2000 to 0x300a0000 (intc)
Did Map 0x300a6000 len 0x2000 to 0x300a6000 (cfg)
Did Map 0x300a8000 len 0x2000 to 0x300a8000 (uart0)
Did Map 0x300ae000 len 0x2000 to 0x300ae000 (iep)
Did Map 0x300b0000 len 0x2000 to 0x300b0000 (ecap0)
Did Map 0x300b2000 len 0x400 to 0x300b2000 (mii_rt)
Did Map 0x300b2000 len 0x1c00 to 0x300b2000 (mdio)
PRU_PHY_detect:181 Phy 1 alive
PRU_PHY_detect:181 Phy 2 alive
Phy 1 : Disable GBit ANEG
Phy 2 : Disable GBit ANEG
Phy 1 : RGMII enable low latency
Phy 2 : RGMII enable low latency
Phy 1 : RGMII set TX Half/Full Threshold: 1
Phy 2 : RGMII set TX Half/Full Threshold: 1
Phy 1 : RGMII set RX Half/Full Threshold: 1
Phy 2 : RGMII set RX Half/Full Threshold: 1


[APP] INFO: Done!
[APP] INFO: Initializing permanent data...
[APP] INFO: Done!
[APP] INFO: Initializing physical device...
Creating thread <osal_task> at Prio 7
Did create thread <osal_task> at Prio 7 with id 0x700c81a0
Creating thread <osal_task> at Prio 6
Did create thread <osal_task> at Prio 6 with id 0x700c91a0
Creating thread <osal_task> at Prio 4
Did create thread <osal_task> at Prio 4 with id 0x700c89a0
[APP] INFO: Done!
[APP] INFO: Configuring physical device...
[APP] INFO: Done!
[APP] INFO: Configuring Ethernet interface...
[APP] INFO: Done!
[APP] INFO: Configuring SNMP interface...
[APP] INFO: Done!
[APP] INFO: Configuring logical device and equipment...
[APP] INFO: Done!
[APP] INFO: Starting PROFINET stack...
[APP] INFO: Done!
[APP] INFO: Starting PRU...
[APP] INFO: Done!
[APP] INFO: Stack configured successfully! Stack runs.

```


3 Profinet Controller Setup - phyBOARD-Electra

Boot the AM64 into linux using a standard ALPHA2 Linux tisdk-default-image.

Use ETH2 to connect your Electra to the network, ensure you have a valid network connection with a DHCP assigned address.

 You need your host machine on the same subnet as the Electra. For PHYTEC employees located at the Bainbridge Island office. This requires moving your host's ethernet connection on your Fortinet switch from the "computer" port to a "hardware" port.

According to Alec our routing rules should allow these to talk but for whatever reason, maybe CODESYS does some special networking thing, I havent been able to make progress on working with IT to get the communication to work, without first manually moving your host's network connection.

On your host machine, using 7zip, unzip the "CODESYS Control for Linux ARM64 SL *.package" from the host setup steps above (yes its weird, this same package was installed into CODESYS development system in the above steps, but this package is also a compressed package containing the target dependencies so we need to unpack it too)

Transfer the following files to the phyCORE-AM64x booted into Linux (scp, thumb drive, copy it to the sd card, etc):

- "CODESYS Control for Linux ARM64 SL *"\Delivery\codesyscontrol_linuxarm64_*_arm64.deb
- "CODESYS Control for Linux ARM64 SL *"\Dependency\codemeter-lite_*_arm64.deb

install them with the following commands in this order:

Target (Linux)

```
cd /<path to the .deb file>
opkg -V2 install --nodeps --offline-root / --add-arch arm64:13 ./codemeter-
lite_*_arm64.deb
opkg -V2 install --offline-root / --add-arch arm64:13 ./
codesyscontrol_linuxarm64_*_arm64.deb
```

You can now start CODESYSControl on the target manually like this:

```
/opt/codesys/bin/codesyscontrol.bin /etc/CODESYSControl.cfg
```

This has to be started on everyboot, unless you write a systemd service. Request this from True if needed, it is not necessary to run the example.

Now your Profinet Controller is all setup, from here on out you pretty much only interact with the controller/master using the CODESYS IDE.

4 Configure the Profinet Network - CODESYS

Using the CODESYS Development System on the host:

Create a new Standard Project.

Select the "CODESYS Control for Linux ARM64 SL" for Device and use Structured Test for PLC_PRG when prompted.

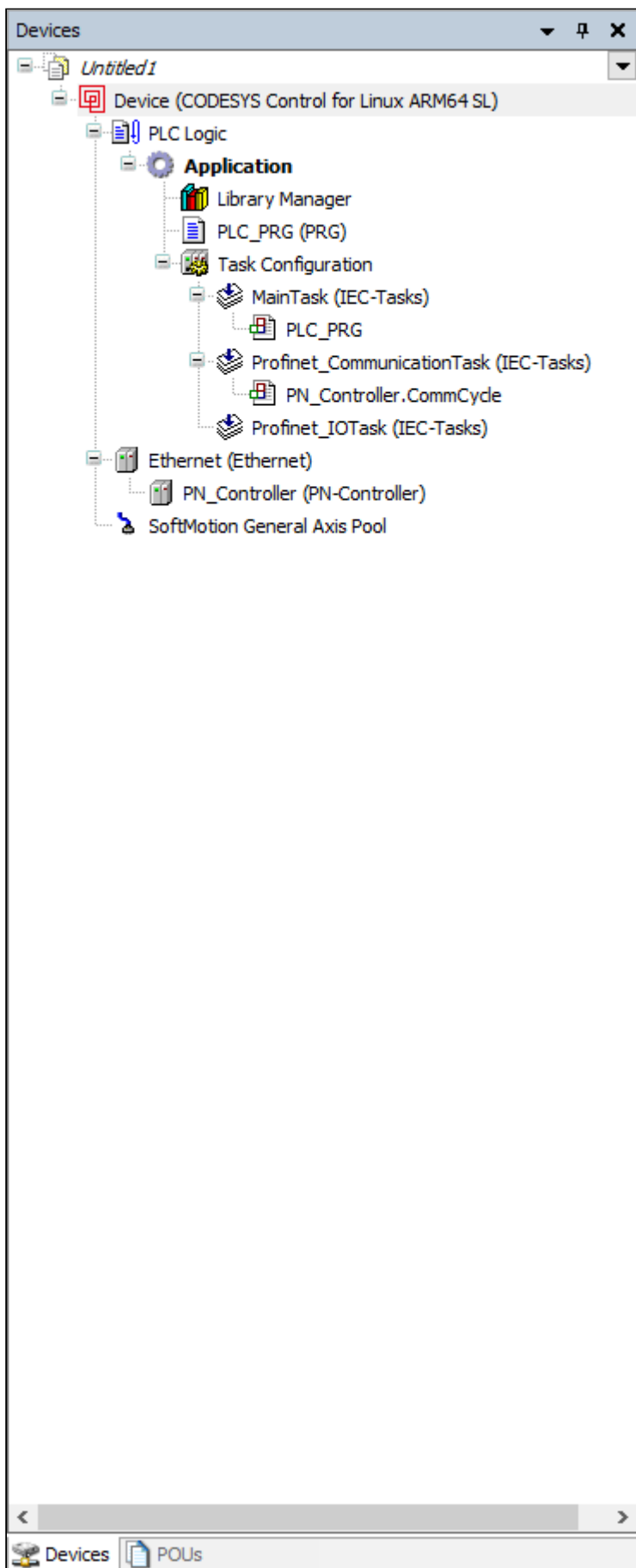
In the Devices window, Right click the "Device (CODESYS Control for Linux ARM64 SL)" and select "Add Device"

Add Fieldbuses → Ethernet Adapter → Ethernet to the "Device (CODESYS Control for Linux ARM64 SL)" to add Ethernet to the project.

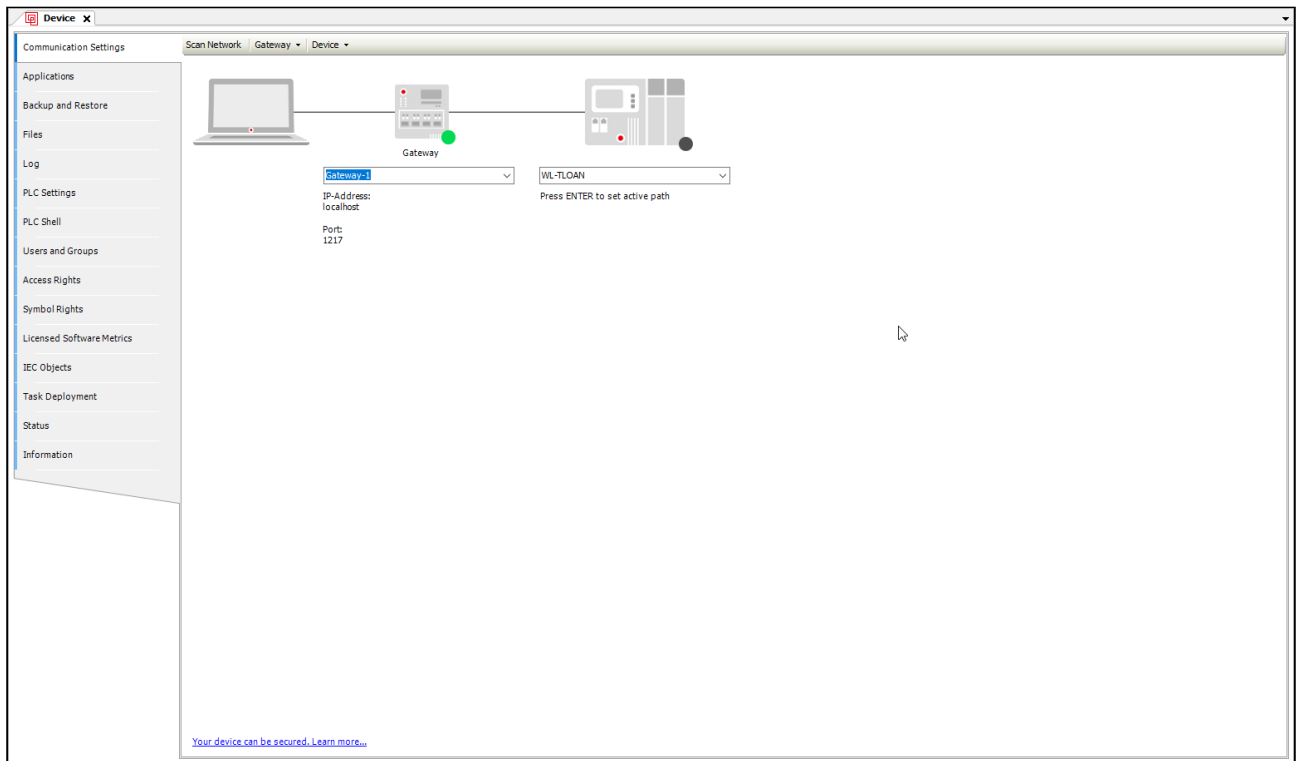
In the Devices window, Right click the new Ethernet Device and select "Add Device"

Add Fieldbuses → PROFINET IO → PROFINET IO Master → PN-Controller

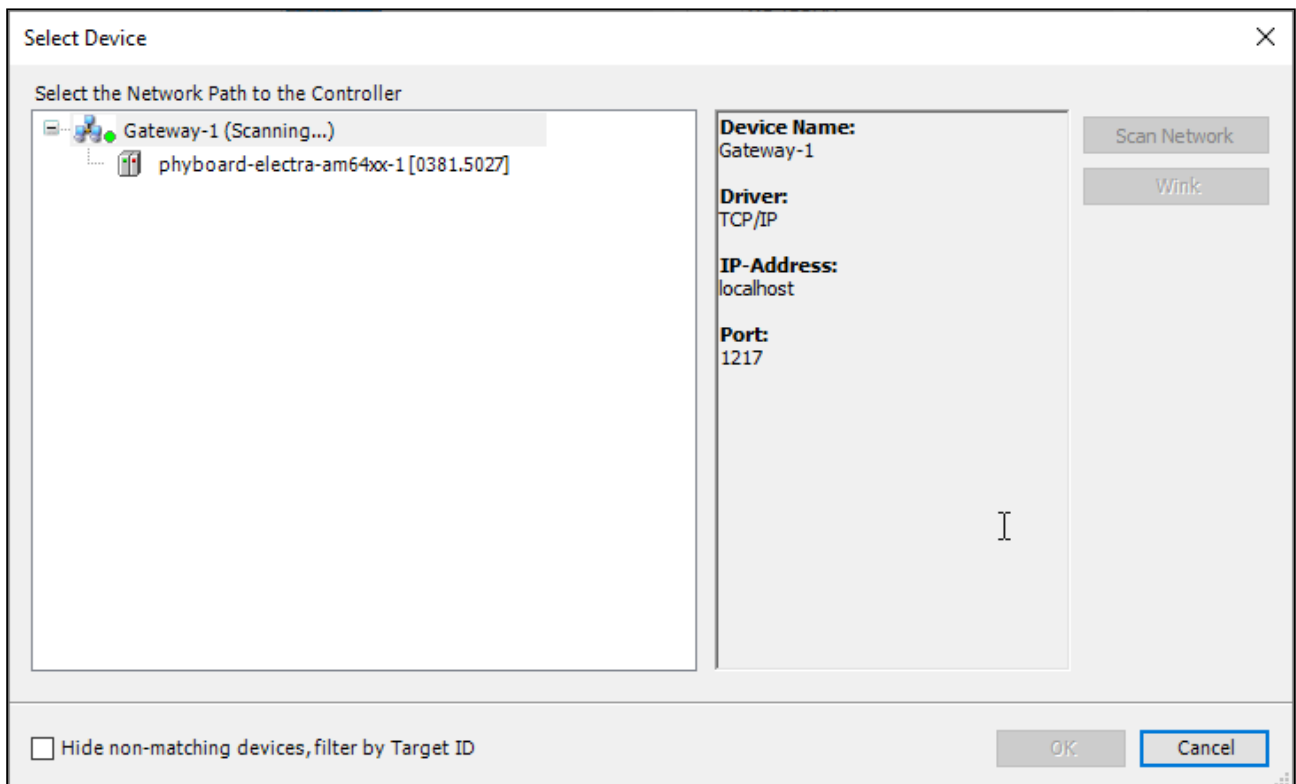
You should now have the following in your Devices window:



Double Click the "Device (CODESYS Control for Linux ARM64 SL)" to get the following window:



Select "Scan Network" (you must have first started the CODESYS runtime on the Electra in Linux)

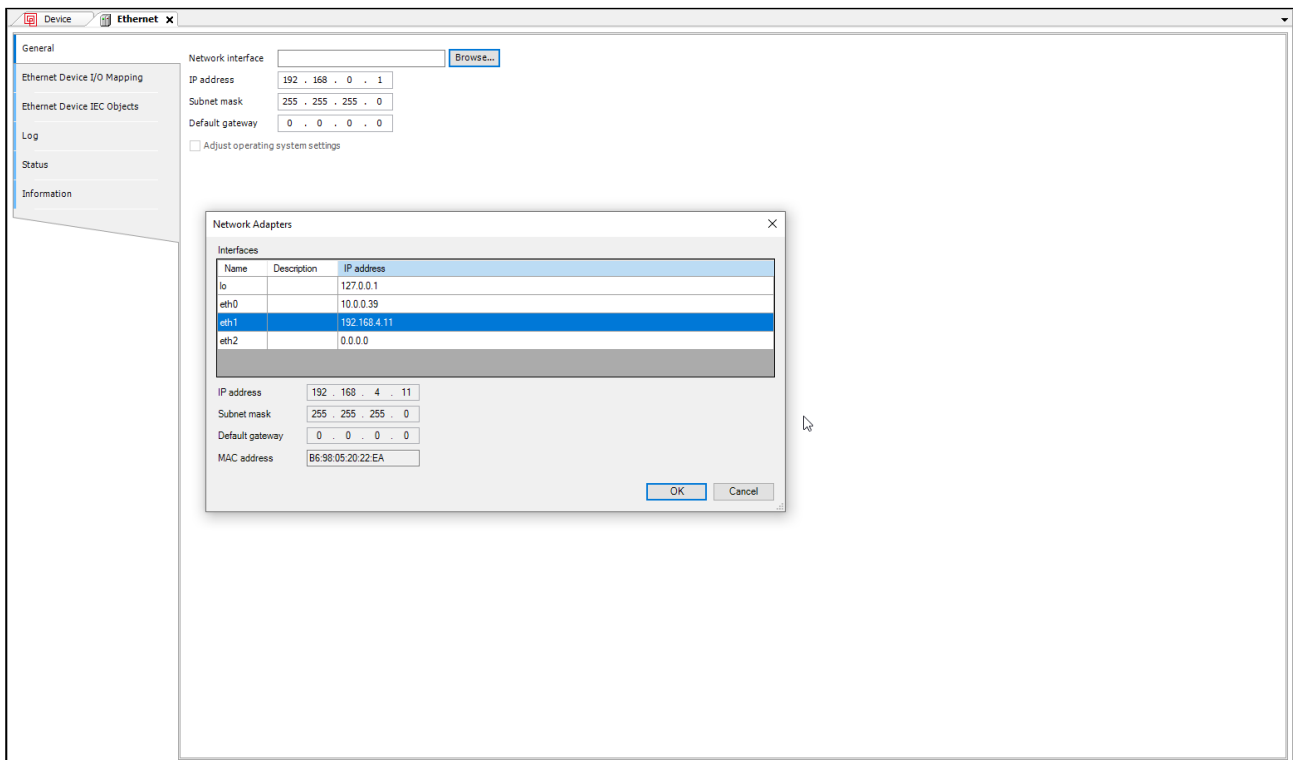


Select the electra and hit ok.

- ⚠** If the electra does not come up in a scan, I have discovered that the following could be the cause:
- Electra and host are on different subnets. See note above in the Profinet Controller setup steps
 - Your VPN is enabled within the office.

On first setup, you will be prompted to setup a codesys user on the electra. specify "root" for user AND "root" for password. (I had trouble leaving password empty.),

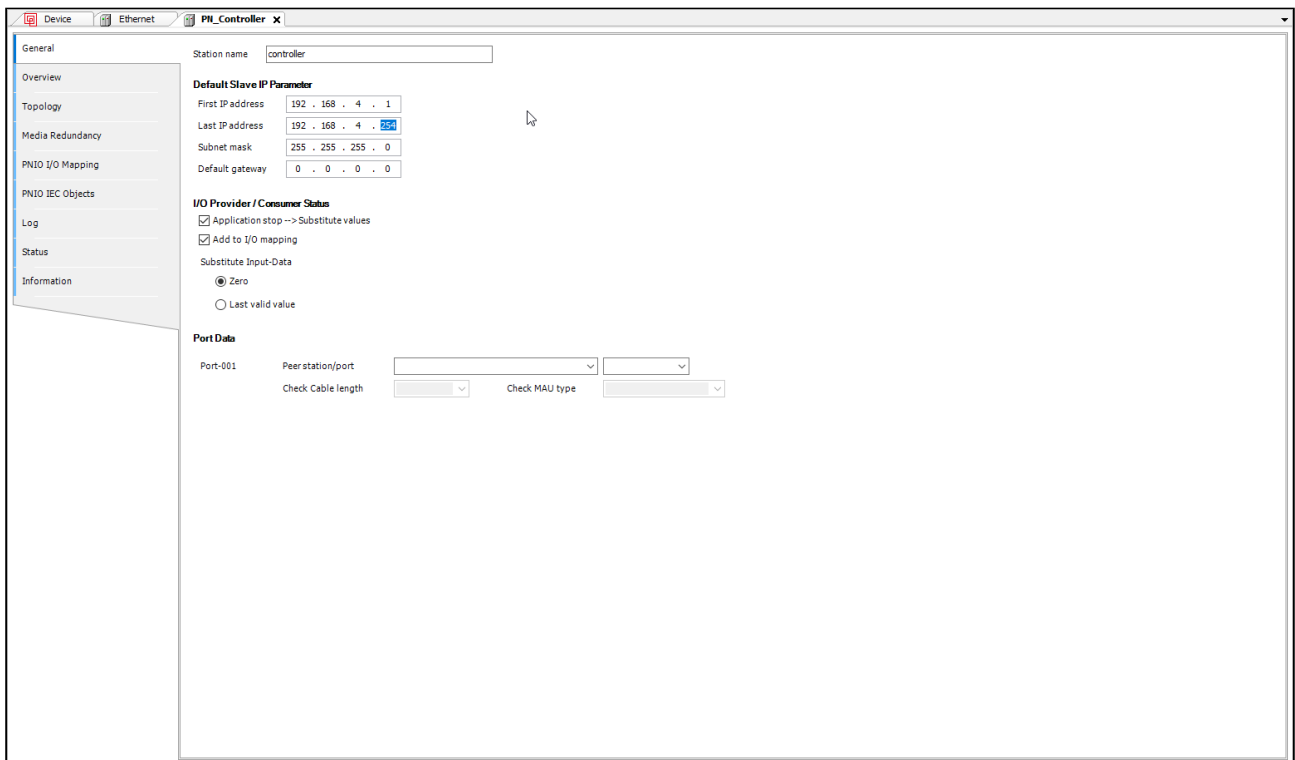
Double Click the Ethernet Device. In the General tab, hit the "Browse button" to scan available network interfaces on the controller.



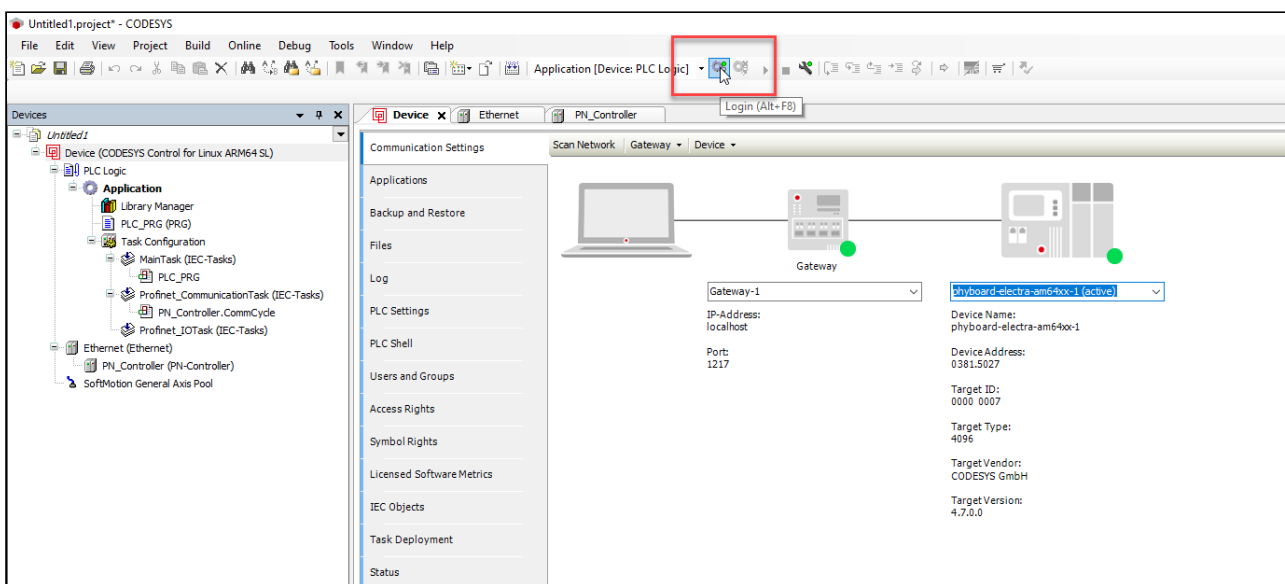
Select ETH1. This interface is the one used to connect to Profinet slaves and should already have a 192.168.4.11 static IP address by default.

Double click the PN_Controller device in the Devices window.

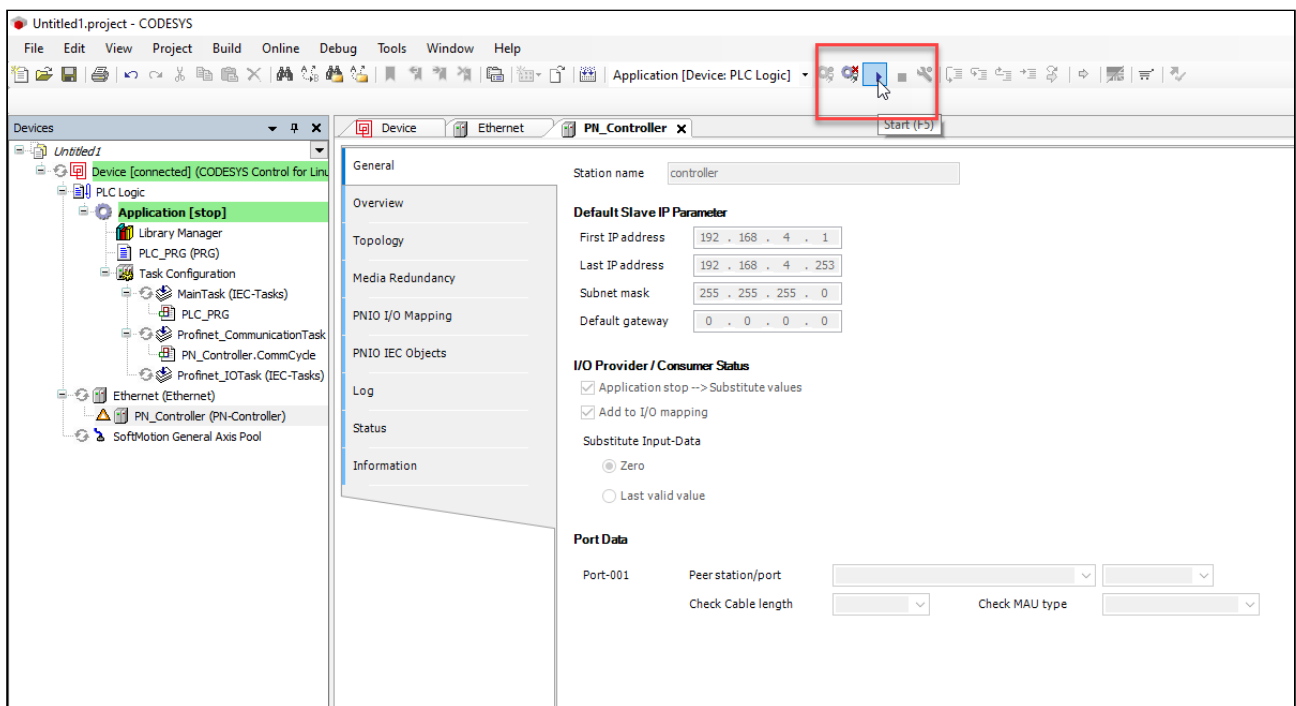
In the General tab, configure valid first and last IP addresses for Profinet Slaves (you could limit the range here if you wanted). these should be in the same subnet as the static IP address:



Now log into the Profinet Controller:



Once logged in, the CODESYS application will automatically be built and copied to the Electra. The application will initially be in a stopped state. Hit Start.

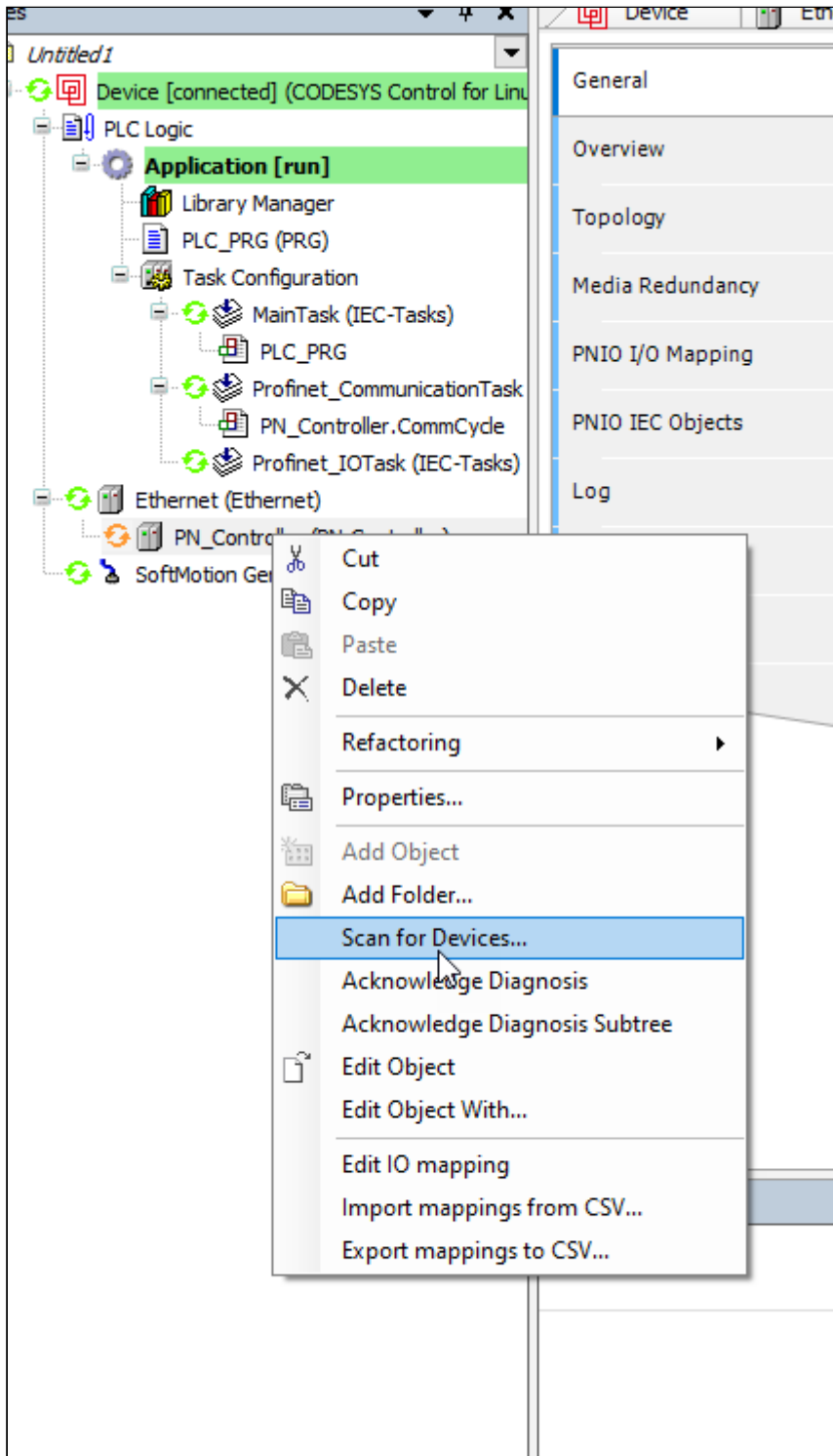


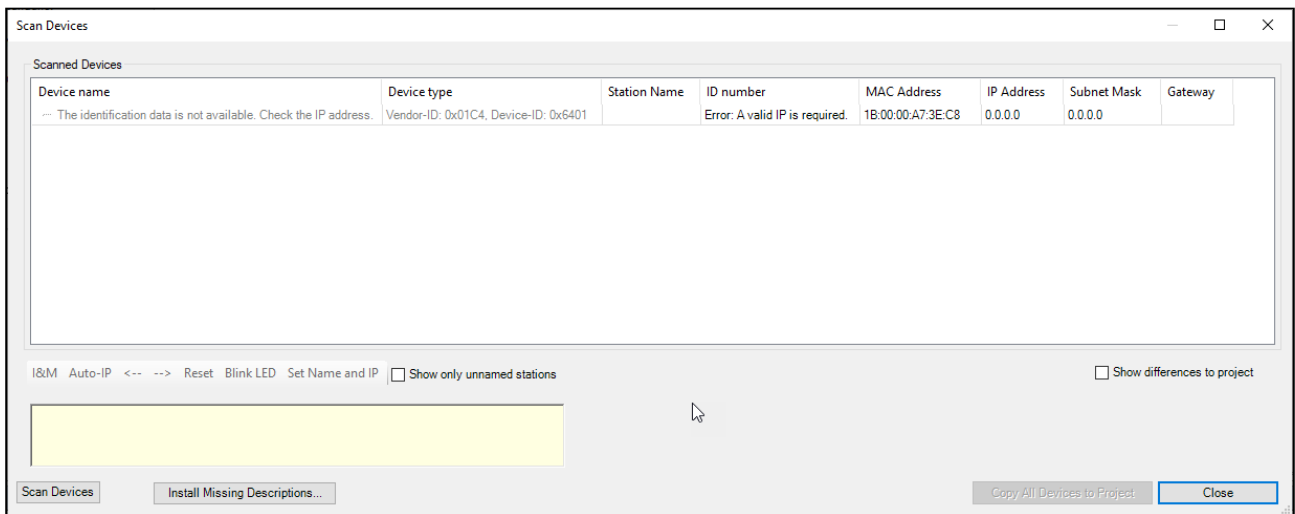
If everything worked so far, you should see the various parts of your CODESYS application start:



The green spinners indicate things are working. Orange spinners indicate things are working with an evaluation license.

Right click the PN_Controller and select "Scan for Devices..."

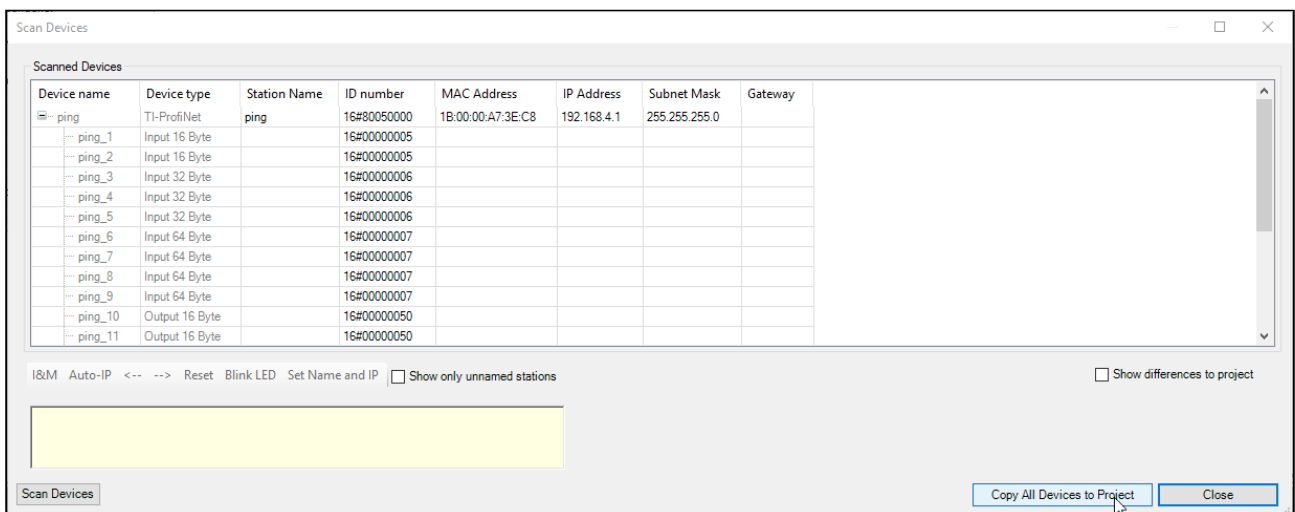




You should see the Profinet Slave come up in the scan.

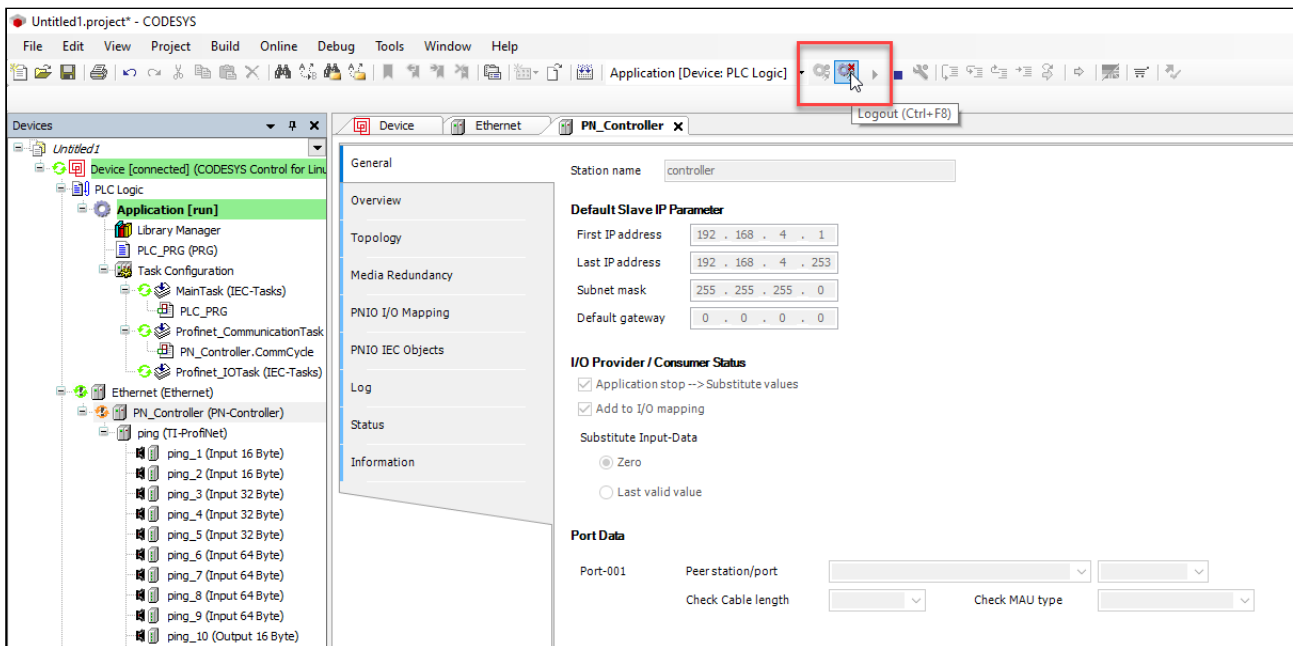
Select the "Station Name" Cell and type in a name (doesn't matter what you call it, I used "pinger1").

Select the device and Hit "Auto-IP". You should see Process Data enumerated out in the "Scan Devices" window. Hit "Copy All Devices to Project"



The Profinet Slave and its process data should be listed in the Devices Window now.

Log out



Double click the PLC_PRG in the Devices window. this file contains the main app code in Structured Text.

There are two windows in the Structured Text PLC_PRG. The upper is for variable declaration, the lower for logic.

Paste the following into the upper window:

```
PROGRAM PLC_PRG
VAR
    i2c_leds_1 AT %QB0 : BYTE :=0;

    i2c_led1 : BOOL :=1;
    i2c_led2 : BOOL :=0;
    i2c_led3 : BOOL :=1;
    i2c_led4 : BOOL :=0;
    i2c_led5 : BOOL :=0;
    i2c_led6 : BOOL :=0;
    i2c_led7 : BOOL :=0;
    i2c_led8 : BOOL :=0;

    iCounter      : UDINT;
    i              : USINT;

    tCycle        : TIME := T#1000MS;

    bToggleOn     : BOOL := TRUE;
    bToggleOff    : BOOL;

    fbTOn         : TON;
    fbTOff        : TON;

    usiOutput     : USINT;
    usiInput      : USINT;
```

```
END_VAR
```

And paste the following into the lower window:

```
fbTOn(IN := i2c_led1, PT := tCycle);

IF fbTOn.Q THEN
    i2c_led1 := 0;
    i2c_led2 := 1;
    i2c_led3 := 0;
    i2c_led4 := 1;
    i2c_leds_1 := SHL(BOOL_TO_BYTE(i2c_led8),7) + SHL(BOOL_TO_BYTE(i2c_led7),6) +
    SHL(BOOL_TO_BYTE(i2c_led6),5) + SHL(BOOL_TO_BYTE(i2c_led5),4) +
    SHL(BOOL_TO_BYTE(i2c_led4),3) + SHL(BOOL_TO_BYTE(i2c_led3),2) +
    SHL(BOOL_TO_BYTE(i2c_led2),1) + SHL(BOOL_TO_BYTE(i2c_led1),0);
END_IF

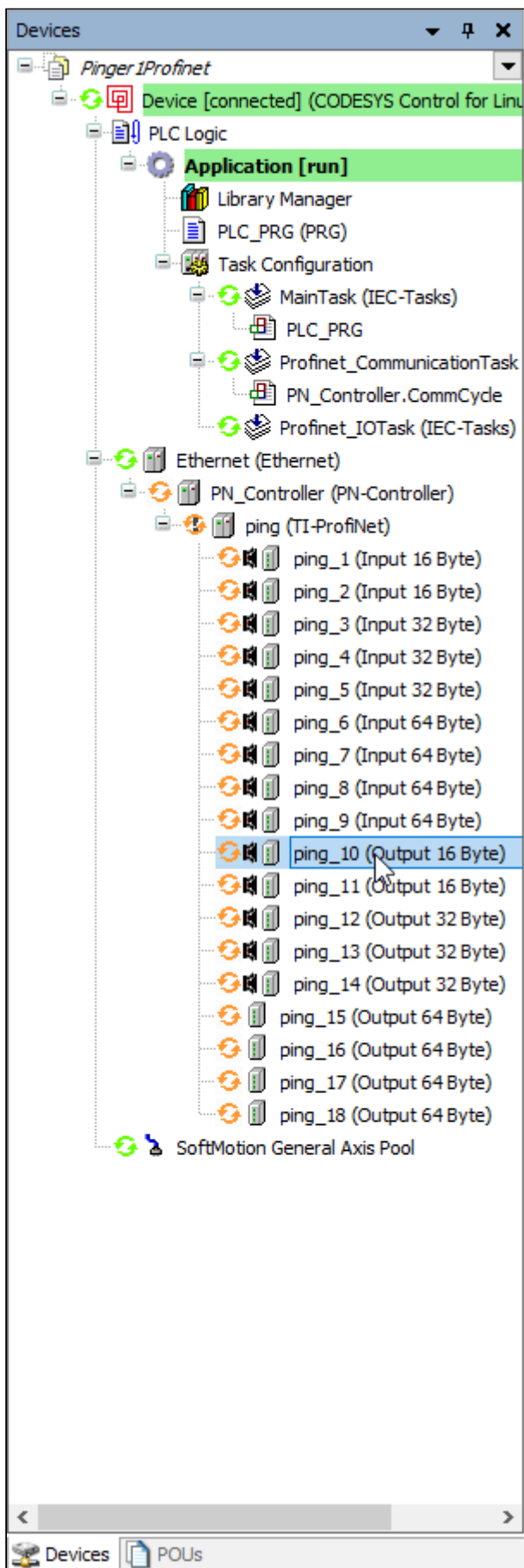
fbTOff(IN := i2c_led2, PT := tCycle);

IF fbTOff.Q THEN
    i2c_led1 := 1;
    i2c_led2 := 0;
    i2c_led3 := 1;
    i2c_led4 := 0;
    i2c_leds_1 := SHL(BOOL_TO_BYTE(i2c_led8),7) + SHL(BOOL_TO_BYTE(i2c_led7),6) +
    SHL(BOOL_TO_BYTE(i2c_led6),5) + SHL(BOOL_TO_BYTE(i2c_led5),4) +
    SHL(BOOL_TO_BYTE(i2c_led4),3) + SHL(BOOL_TO_BYTE(i2c_led3),2) +
    SHL(BOOL_TO_BYTE(i2c_led2),1) + SHL(BOOL_TO_BYTE(i2c_led1),0);

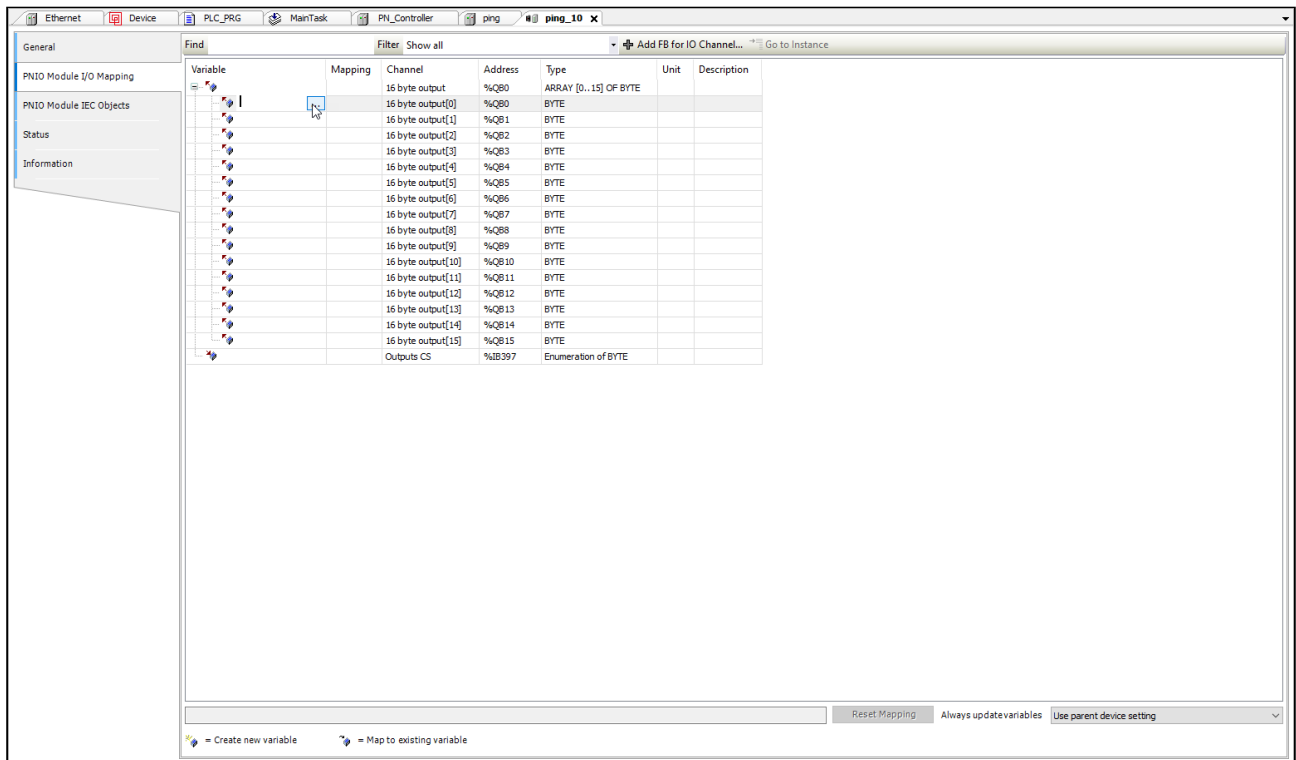
    iCounter := iCounter + 1;
END_IF
```

This is basically a blinky program that flips bits in the process data, which are mapped to the LED states on the slave. We now need to connect our `i2c_leds_1` variable to the process data its intended for.

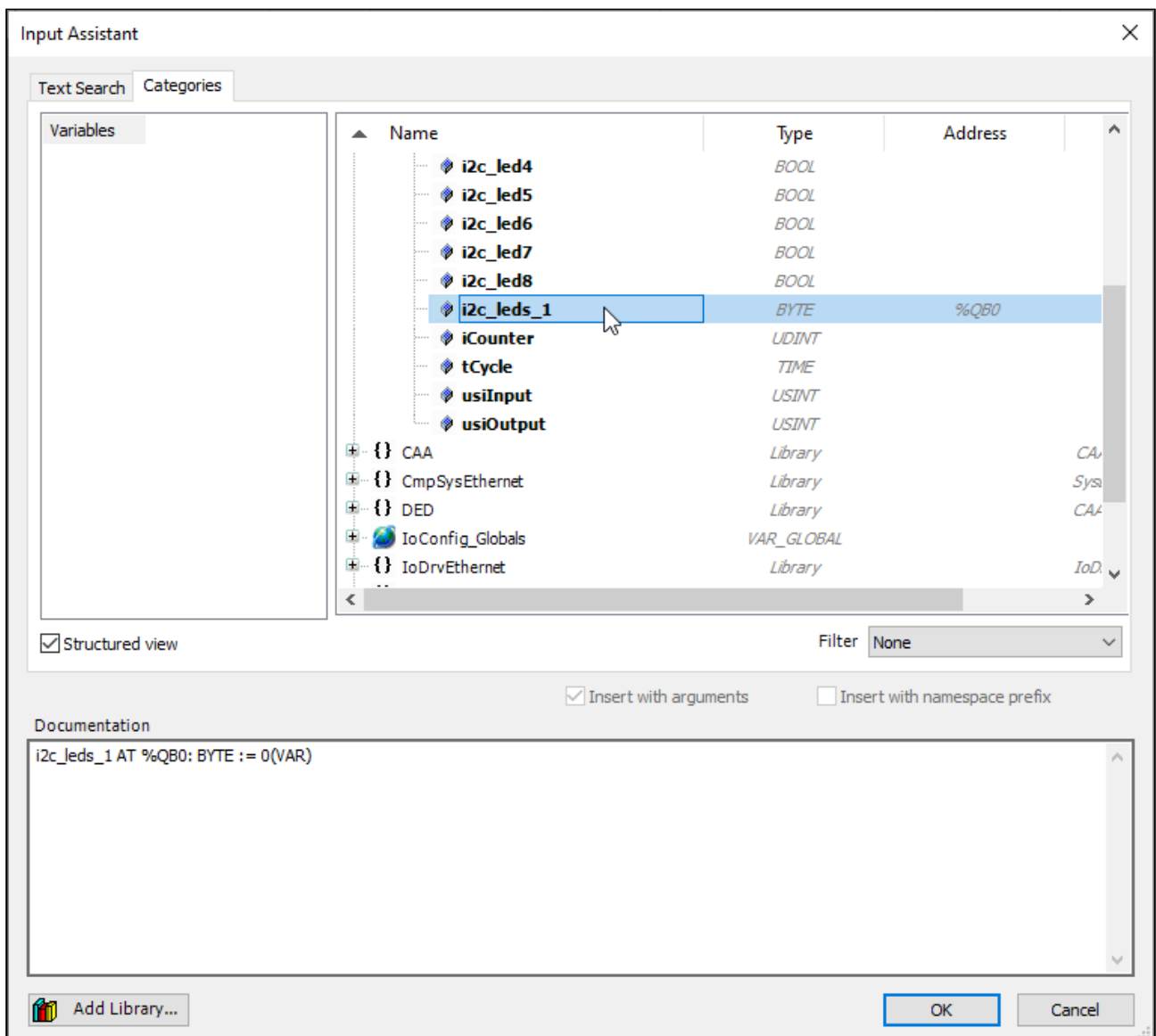
Double click the first Output data channel:



In the "PNIO Module I/O Mapping" tab, expand the 16 byte output channel and double click the left most cell to reveal ellipses:

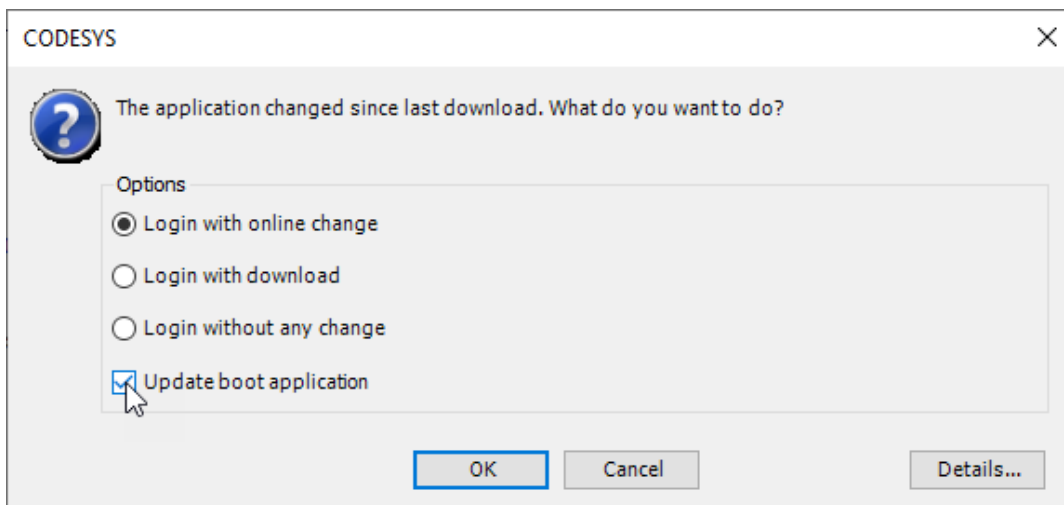


Click the revealed ellipses and double click "i2c_leds_1"



Now log back in.

If prompted (might only be on subsequent application updates), Update the boot application in the following pop-up window:



If the application starts in a stopped state, hit Start.

You should see your pinger's confetti boards flash some LEDs at different colors on 1 second intervals.

1 Blue (2 off)

then

1 Red 1 Green (1 off)

You can play with the PLC logic to get different colors. each bit is going to be different color. each RGB LED is mapped to 3 bits.