

# ICSS EIP Adapter Developer Guide

---

## Contents

---

### About this Guide

Recommended Reading

### Glossary of terms

### About EIP Adapter on PRU-ICSS

### Sources

### Features & Capabilities

Device Capabilities

### Design

Half Duplex

Switching

DLR

PTP/1588

QoS

### Molex Stack

### Memory Map

### Hardware Resource Usage

Interrupts

Timers

Instruction memory and Data Memory

### Porting your own EIP Stack

### Application Interface

### Initialization

Board and Hardware Initialization

Mapping the Interrupts

Allocating Memory for Structures

Registering the callbacks

Configuring Switch parameters

Common Settings

AM335x

AM437x

### Basic self debugging

EDS File Installation and Verification with RSLinx

Conformance Testing (ODVA CTT v12)

Molex EIP Tool

TCP/IP Object

Ethernet Link Object

Device Performance Testing

Communication between EtherNet/IP Adapter and Scanner devices

## About this Guide

### Recommended Reading

Before reading this developers should review the following wikis if they haven't already. Most of the material will be cross-referenced and will not be repeated.

1. ICSS EMAC LLD Design ([http://processors.wiki.ti.com/index.php/Processor\\_SDK\\_RTOS\\_ICSS-EMAC-Design](http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_ICSS-EMAC-Design)) : This discusses the firmware and driver architecture in detail. This wiki will reference it often.
2. Industrial SDK Getting Started Guide : Introduction to Industrial SDK and how to run the examples
3. PTP/1588 Developer Guide : PTP/1588 Developer guide

This wiki is provided to help developers use the EtherNet/IP adapter application, provide general information on the architecture and adapt it to their needs.

EtherNet/IP is a set of protocols that sit on top of regular Ethernet (802.3) TCP/IP stack and support standard TCP/IP traffic. These protocols are grouped under the umbrella term CIP (Common Industrial Protocol). They provide a means for developing Industrial communication solutions. The standard body for EtherNet/IP is ODVA (Open DeviceNet Vendor Association).

Since most of the EtherNet/IP protocol sits above the TCP/IP stack it can be thought of as a standard TCP/IP switch with a CIP stack layer on top of the TCP/IP layer. There are a few additional protocols like PTP-1588 and DLR which work at layer 2 and are integrated into the switch firmware and driver but they are not essential to a basic EtherNet/IP application, only the CIP + TCP/IP stack is essential. The CIP stack provided with the EIP application is from a third party named Molex (<http://www.molex.com/molex/industry/industryMenu.action>). The stack contained in the Industrial SDK is provided as a demo and comes with a 1 hour time limit. To use the application beyond the 1 hour time limit user must restart the application.

To use the Molex EtherNet/IP stack in your commercial project please contact Molex to obtain a commercial license. This guide also has material to enable developers to port their own EtherNet/IP stack and use it.

There is a [white paper](http://www.ti.com.cn/cn/lit/wp/spry249a/spry249a.pdf) (<http://www.ti.com.cn/cn/lit/wp/spry249a/spry249a.pdf>) which explicitly discusses the EtherNet/IP implementation on AM335x, it covers the theoretical aspects and discusses the implementation in general, but being a white paper does not dig deep into the implementation details.

This wiki will not cover the theoretical aspects of EtherNet/IP and CIP since this is covered in the white paper.

## Glossary of terms

*Glossary of EtherNet/IP terms*

Terms	Abbreviation
EIP	Ethernet IP
TC	Transparent clock
OC	Ordinary clock
TS	Timestamp
Slave	EIP application belonging to adapter class
EMAC	Two port EMAC
Switch	Either EtherNet/IP or HSR/PRP
RPI	Requested Packet Interval
DLR	Device Level Ring : A redundancy protocol
CIP	Common Industrial Protocol
ODVA	Open DeviceNet Vendor Association
BC/MC/UC	Broadcast/Multicast/Unicast
ACD	Address Collision Detection

## About EIP Adapter on PRU-ICSS

The EIP application is basically a standard 2 port Ethernet switch, more commonly known as a relay. The IEP application shares much of its design with the ICSS EMAC that is described in [here](#).

However there are a few differences from the EMAC design:

- Port to port forwarding : There are two types of forwarding that are supported: cut through and delayed cut through
- Collision on Port and Host queues : Since both ports can simultaneously write to the same Host queue, collisions can happen. A collision buffer is provided for this purpose. The EMAC does not contain this feature..
- Device Level Ring : This is a redundancy protocol that is used in EtherNet/IP devices. DLR is based on ring topology and requires forwarding hence it is exclusive to switches.
- PTP/1588 : IEEE Time Synchronization protocol. EtherNet/IP uses the Drives profile of PTP-1588. It is discussed in depth [here](#). PTP OC is not exclusive to switches. It can be implemented in a MAC. This is not implemented in the ICSS EMAC in TI's Industrial SDK presently.

More details on these in the design section.

These differences (particularly the difference between a switch and EMAC) are also captured in the EMAC Design guide [here](#).

The EIP implementation is made up of three parts

1. Firmware : Firmware implements the Switching layer, Statistics, DLR and PTP
2. Driver : Some portions of DLR and PTP lie here. Learning/FDB is also a part of this.
3. EtherNet/IP stack : From Molex

## Sources

The EtherNet IP specific files are located under **`$(IA_SDK_HOME)/protocols/ethernetip_adapter`**

## Features & Capabilities

The EIP Adapter application has been certified by ODVA in its lab and the declaration of conformity certificate is shown below for reference. The same is available from ODVA's website [here](https://marketplace.odva.org/products/883-tm3359-am3359-industrial-communications-engine) (<https://marketplace.odva.org/products/883-tm3359-am3359-industrial-communications-engine>)



## Declaration of Conformity to the EtherNet/IP™ Specification

ODVA hereby issues this Declaration of Conformity to *The EtherNet/IP™ Specification* for the product(s) described below. The Vendor listed below (the "Vendor") holds a valid Terms of Usage Agreement, which is incorporated herein by reference, for the EtherNet/IP Technology from ODVA, thereby agreeing that it is the Vendor's ultimate responsibility to assure that its EtherNet/IP Compliant Products conform to *The EtherNet/IP Specification* and that *The EtherNet/IP Specification* is provided by ODVA to the Vendor on an AS IS basis without warranty. NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ARE BEING PROVIDED BY ODVA.

In recognition of the below EtherNet/IP Compliant Product(s) having been EtherNet/IP Conformance Tested at ODVA-authorized Test Service Provider and having received a passing result from ODVA at the Composite Test Revision Level specified below, this Declaration of Conformity authorizes the Vendor to use the EtherNet/IP Certification Marks in conjunction with the specific EtherNet/IP Compliant Product(s) described below, for so long as the Vendor's Terms of Usage Agreement for the EtherNet/IP Technology remains valid.

**EtherNet/IP™**  
conformance tested

EtherNet/IP CONFORMANCE TESTED™

Certification Logo Mark

Certification Word Mark

This Declaration of Conformity is issued on October 13, 2014, on behalf of ODVA by:

Katherine Voss  
Executive Director

Vendor Information				
Vendor Name	Texas Instruments			
Test Information				
Test Date	May 15, 2014			
Composite Test Revision	CT11			
ODVA File Number	11290.01			
Product Information				
Identity Object Instance				
Vendor ID (Attribute 1)	806			
Device Type (Attribute 2)	0x0C			
Device Profile Name	Communications Adapter			
Products Covered under this Declaration of Conformity (Identity Object Instance)				
No.	Product Code (Attribute 3)	Product Name (Attribute 7)	Product Revision (Attribute 4)	SOC File Name
1	1026	TI/Molex EIP Adapter	1.001	TI_EIP_ICSS_Adapter.stc

### Device Capabilities

- Ethernet/IP Capabilities

- Explicit Messaging and Implicit Messaging
- UCMM
- Class 1 and Class 3 Connection

- Object classes supported

- Identity Object
- Message Router Object
- Assembly Object
- Connection Manager Object
- Ethernet Link Object
- TCP/IP Interface Object
- QOS Object
- Device Level Ring Object

- Supported Connections

- Exclusive Owner Connection
- Input Only Connection
- Listen Only Connection

- Total CIP connections : 9

- IO Messaging : 6
- Explicit Messaging : 3

- PHY Configuration

- Duplex : Half/Full/Auto
- Speed : 10/100/Auto

- Device Configuration : EDS File ([here](http://processors.wiki.ti.com/images/7/71/TI-AM335x_EIP_adapter_sample_EDS_ICSS.zip))

- QoS scheme : 3-bit VLAN PCP

- No of levels supported : 8
- Number of queues : 4. 2 QoS levels per queue

- Statistics

- Interface counters supported per port

- Conformance : CT 13 compliant

- Min RPI supported : 1ms

- Device Level Ring

- Beacon Based

- Min. beacon interval : 200 us
- Min. beacon timeout : 400 us
- Self configuring
- **PTP/1588**
  - Supports Drives Profile : E2E clock.
  - PTP over UDP
  - Transparent Clock supported
  - Ordinary Clock supported
  - Single and Two step clock supported
- **Learning/FDB** : Yes
  - 1024 entries per port
  - Learning table on DDR
- **Storm Prevention** : Yes. Configurable per port

## Design

As mentioned before EIP is based on the [EMAC LLD design](#) and shares most of the design elements from queues to interrupts to statistics. The key differences are listed below

### Half Duplex

Half Duplex support is provided in the firmware. To enable support, the flag `halfDuplexEnable` must be enabled in EMAC Config. Half Duplex support is not available on a per port basis so the switch does not support Half Duplex on one port and a Full Duplex connection on another. The same is true for 10Mbps mode.

### Switching

If a packet is not meant for consumption by the Host, it is forwarded to the other port subject to rules of forwarding. The rules are same for all packets except specific protocol frames which have their own set of rules. Before discussing rules of forwarding, we will discuss forwarding modes here.

### Forwarding Modes

1. **Cut-Through** Data is received on Rx FIFO and copied directly to Tx FIFO. There are no queues involved. It's the fastest mode of data transfer since no data copying is involved.
2. **Delayed Cut-Through** Data is first received into the port queues (Refer to EMAC Design guide) and then forwarded. This is a slower operation compared to cut-through but allows full inspection of the packet.

The key differences are listed below.

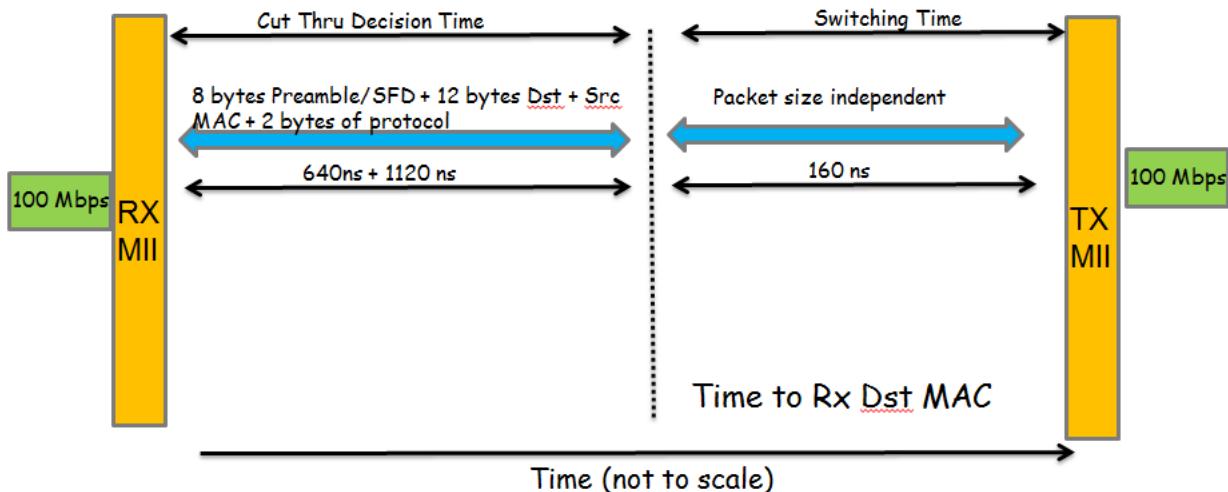
*Cut Through vs Delayed Cut Through*

Cut-Through	Delayed Cut-Through
Rx FIFO to Tx FIFO copy	Rx FIFO --> Port Queues --> Tx FIFO
Faster	Slower
Packet is not inspected completely before Tx	Packet is fully inspected before Tx
CRC check is not done before Tx	If CRC/FCS error is present packet is dropped
Depends on availability of FIFO. If Host is transmitting on same port then it does not work	Since it is queue based, this option is always available.

As always the switch needs to decide when to forward the packets and how, this is done inside the firmware by waiting for a certain number of bytes and then applying the forwarding rules. The number of bytes firmware waits for and the forwarding rules are different for each protocol example. For instance for EtherNet/IP this is 22 bytes because some DLR packet information is available only on the 22nd byte.

The number of bytes firmware waits for decides the resident time for a packet inside the switch. Since the PRU is a fully deterministic architecture this is a known number. The illustration below explains how this is calculated.

# Forwarding Latency with PRU-ICSS



**With destination MAC as the switching decision point, cut-thru latency of PRU-ICSS switch is approximately 1.28us regardless of incoming packet size**

- Cut-Through decision is being taken after receiving 14 bytes. 12 bytes of MAC Address and 2 bytes of protocol.
- It takes 1120 ns ( $14 * 80$  ns) to receive 14 bytes.
- There is 8 byte preamble/SFD which takes 640 ns
- It takes further 160 ns for PRU-ICSS firmware to decide and start the transmission of packet.
- $640 + 1120 + 160 = 1920$ ns or ~2us total latency in cut through mode

Using the calculation above, resident delay for a cut-through frame on EIP firmware is  $22 * 80 + 640 + 160 = 2560$ ns or 2.5 us. The resident delay for a delayed cut-through packet varies depending on scheduling delays. For a 66 byte PTP frame the delay is around 9-10 us based on experimental results.

## General Forwarding Rules

1. **Broadcast frames** Frames are first run through the Storm Prevention check and then get cut-through (subject to link availability on other port) as well as get forwarded to Host.
2. **Multicast frames** DLR and PTP frames bypass Storm Prevention check and have their own rules of forwarding which will be discussed later. Other frames are treated like Broadcast (forward to host and cut-through)
3. **Unicast frame to Host** UC frames matching interface MAC get forwarded to Host
4. **Unicast frames not to Host** UC frames not matching interface MAC get cut-through
5. **Cut-Through decision but other port is occupied** Frame gets delayed cut-through

## DLR

DLR stands for Device Level Ring. It is a Layer 2 redundancy protocol which is independent of CIP. It relies on circulation of periodic frames through the network called Beacons or Announce frames to monitor the health of the network, a variety of other frames are used to find out where the fault is and which devices participate in the network. A detailed discussion on DLR is beyond the scope of this guide so only a basic introduction is provided. A good introduction to the DLR protocols is provided in this youtube video (<https://www.youtube.com/watch?v=q8jkLGKFTK8>)

DLR network consists of

1. **DLR Supervisor** : A supervisor is responsible for sending out periodic frames called Beacons or Announce frames at regular intervals to monitor the network for faults. A controller or PLC is typically the DLR supervisor. The main job of a DLR supervisor is to detect transition from ring to linear topology and open one of its ports for communication which is usually closed in a ring configuration to prevent a loop.
2. **DLR Slave** : A DLR slave forwards beacon/announce frames from one port to other without any modification, it also checks for continuous reception of DLR frames and if any is missed it reports the status to supervisor. The EIP Adapter application is a DLR slave.

All DLR frames contain the protocol identifier 0x80E1 (comes immediately after destination MAC) which is how the firmware identifies the protocol. Once frame type is identified as DLR, the flow changes from normal frame to one for DLR processing.

## Frame Types

1. **Beacon** : Is the most common frame type and used to monitor ring health. It is sent out periodically on both the ports by the Supervisor based on the ring parameter *beaconInterval*. The lowest possible value for this is 100us. The beacons are transmitted simultaneously from both ports and reach the opposite ports of same device after traversing the entire network, this is how the supervisor confirms that ring is functional. Failure to observe a beacon within a specified interval called *beaconTimeout* triggers a state change to Fault. This type of frame is cut-through.
2. **Announce** : Analogous to Beacon but much slower, this is for legacy networks. EIP Adapter does not support this mode. All Announce frames get cut-through.
3. **Sign On** : This contains information about every node on the network. This is sent on one port by the Supervisor and gets trapped by every node on the network, the devices then add their own information and send it over the other port. These frames are sent to the Host over Host queues, the Host adds its own information and then transmits it over the other port.
4. **Locate Fault** : This is a frame sent by the Supervisor in response to a missed beacon or any other fault event. All devices are supposed to report their neighbor status on receiving this frame. EIP Adapter cuts-through this frame and generates Neighbor Check Requests for neighboring devices.

4. **Neighbor Check Request** : This is a frame that is generated by every node including the supervisor in response to a fault generating event on the network. A device is supposed to consume this frame and generate a neighbor check request. Our application forwards this frame to Host for a response. A device generates 3 such frames on each port at intervals of 100ms and if no response is received then a status is reported to Supervisor that corresponding port has an issue.
5. **Neighbor Check Response** : A device is supposed to consume this frame. If this frame isn't received after 3 requests then a neighbor status frame is sent indicating the port on which device is faulty. This frame is not forwarded.
6. **Link Status/Neighbor Status** : This is sent by a device to supervisor to indicate a link break or an issue with adjacent device. This frame is cut-through.

## Timers

DLR uses lots of timers because it has to keep track of when every beacon arrives and whether there has been a timeout, there's also a timeout related to neighbor check response. The lowest beacon interval according to the standard (and what the application supports) is 100us and with a minimum 2x timeout interval the minimum value for timeout comes to 200us. Thus the minimum resolution for a DLR Beacon timeout timer should be 200us. This is only possible with either watchdog timers (internal to PRUSS) or DMTimers (on the SoC).

The DMTimer ISR is configured to get triggered on expiry of the count (counts downwards). Every time a beacon is received in firmware the timer is triggered to restart, this prevents the ISR from getting hit.

On the older Industrial SDK based releases DLR uses DMTimer 4 and 5 as beacon timeout timers for Port 0 and 1 respectively but for Processor SDK based releases this has been moved to the watchdog timers IEP\_PD\_WD and IEP\_PDI\_WD. The watchdog clock divider is set to 10us in the PRU-ICSS register PRUSS\_IPI\_WD\_PREDIV so the granularity of timeout

timer is 10us. Thus A value of 405 us cannot be configured as timeout value, it can either be 400 us or 410 us.

For the neighbor check response timers we use software timers since the requirement is very lax (100 ms). SysBIOS provides API's for configuring software timers which are used here, if porting to another operating system this must be taken care of.

The timers and their ISR's are listed below

1. **beaconTimerPort0()**
  1. Beacon timeout timer for Port 0.
  2. Timer Handle is `dltimer_PORT0`
2. **beaconTimerPort1()**
  1. Beacon timeout timer for Port 1.
  2. Timer Handle is `dltimer_PORT1`
3. **neighborTimeoutISR0()**
  1. ISR for Neighbor Timeout Timer on Port0.
  2. Timer Handle name is `dlnightoutClock[0]`
4. **neighborTimeoutISR1()**
  1. ISR for Neighbor Timeout Timer on Port1.
  2. Timer Handle name is `dlnightoutClock[1]`

The timers with their default values are configured in the DLR initialization API `initDLR()`. The beacon timeout timers are configured with network parameters received from the master when the first beacon is received.

## Interrupts

There are four interrupt contexts to DLR

1. **Related to timers** : When timers expire they trigger an ISR. Total of four timer related ISR's are applicable to DLR. They are covered above.
2. **Related to link break** : Certain DLR events need to be processed when a link break occurs. EMAC LLD has one link ISR per port to handle this. A DLR callback is registered for the link ISR which is specific to this. There is one callback per port, these are identical and only differ in terms of which port they belong to.
  1. `EIP_DLR_Port0ProcessLinkBrk()` Link break callback for Port 0.
  2. `EIP_DLR_Port1ProcessLinkBrk()` Link break callback for Port 1.
3. **Related to state change processing** DLR mostly uses a fast ISR scheme for communicating with the host where firmware parses the fields, copies the data to preset locations and triggers an interrupt for the ARM. There is one ISR for each port. The acceptable latency of the ISR is determined by the beacon interval. If running at minimum level i.e. 100us, the ISR must be triggered and completed within that time.
  1. `EIP_DLR_Port0ISR()` : Fast ISR for Port 0
  2. `EIP_DLR_Port1ISR()` : Fast ISR for Port 1
4. Related to standard Rx interrupt which uses the Host queues : Some DLR packets need to be processed by the Host and as per the spec are sent to the highest priority queue. They are handled by a DLR specific callback. This callback is the function `EIP_DLR_processDLRFrame()`

## State Machine and DLR Actions

A DLR Ring has three states (two as per standard). The state information is stored in shared RAM at the location pointed by `DLR_STATE_MACHINE_OFFSET` (See memory map for details)

1. **No DLR** : This is the state right after boot up and in the absence of any DLR frames. It's not an official classification, at this stage the beacon timeout timers are configured with default values and all flags are in a reset state. Corresponds to `DLR_IDLE_STATE_VAL`
2. **Fault** : When the first beacon is received this is the state of state machine. A field in the DLR frame corresponds to this and is set by the Supervisor. When a supervisor receives two beacon frames that have traversed the entire network and reached opposing ports it changes this state to Normal. A fault state also occurs when a collision is detected or network is reset. When the first DLR beacon is received and the state machine is in reset state the DLR beacon timeout interval is copied and the DLR timeout timer is configured with the value in the Fast ISR. Corresponds to the value `DLR_FAULT_STATE_VAL`
3. **Normal** : Indicates that ring is up and functional. This corresponds to the value `DLR_NORMAL_STATE_VAL`

Besides the state machine two other variables are of importance.

1. Events common to both ports : `DLR_COMMON_EVENTS_OFFSET` is the memory location which corresponds to this. The values stored in this variable and the actions corresponding to it are
  1. `DLR_RING_NORMAL_TRANSITION` : Node has transitioned from Fault to Normal. Perform appropriate actions. Look for the flag `DLR_RING_NORMAL_TRANSITION_MASK` in the code.
  2. `DLR_RING_FAULT_TRANSITION` : Node has transitioned from Reset to Fault. Configure the timers. Look for the flag `DLR_RING_FAULT_TRANSITION_MASK`
  3. `DLR_STOP_BOTH_TIMERS` : Stop both beacon timeout timers.

4. DLR\_PORT0\_BEACON\_RCVD : First beacon received for Port 0
  5. DLR\_PORT1\_BEACON\_RCVD : First beacon received for Port 1
  6. DLR\_RESET\_EVENT\_OCCURRED : Reset the state machine
2. Events symmetrical but unique to each port : The memory location corresponding to this is **DLR\_PORT\_EVENTS\_OFFSET**. Common events are
1. DLR\_LOCFAULT\_RCVD : Locate Fault Packet received.
  2. DLR\_NCREQ\_RCVD : Neighbor Check request packet received
  3. DLR\_NCRES\_RCVD : Neighbor Check response packet received
  4. DLR\_START\_TIMER : Start the beacon timeout timer corresponding to the port
  5. DLR\_RING\_FAULT\_RCVD : Beacon received with ring status set to fault, transition to fault state
  6. DLR\_DROP\_PACKET : Internal to firmware implementation
  7. DLR\_UPDATE\_SUP\_CFG : Supervisor has changed, update params
  8. DLR\_HOST\_FWD : Internal to firmware implementation
  9. DLR\_TIMER\_RUNNING : When beacon timeout timer is started this is set
  10. IS\_A\_DLR\_FRAME : Internal to firmware implementation
  11. COMPARE\_SUP\_PRED : Internal to firmware implementation
  12. UPDATE\_NEW\_SUP\_CFG : Internal to firmware implementation
  13. DLR\_FLUSH\_TABLE\_RCVD : Flush table packet received. Flush learning table
  14. DLR\_SEND\_LEARNING\_UPDATE : Send learning update frame.

## **QoS**

All DLR frames when they are sent to Host go on the highest priority queue. A callback function **EIP\_DLR\_processDLRFrame()** in the EIP application for the highest priority queue specifically handles this.

## **Interface**

All DLR information relevant to the stack is stored in the structure *dlrStruct* which maps directly to the DLR object as defined in the standard, attribute ID's refer to the corresponding ID's for DLR Object. The macro **IS\_A\_DLR\_SUPERVISOR** has been kept for future use and cannot be enabled right now.

The structure is used as a global in *icss\_dlr.c*. A third party stack can declare this as an extern to access the member variables.

```
/** 
 * @brief DLR parent structure through which all other structures can be accessed
 */
typedef struct {
    /**Supervisor config. Attribute ID 4*/
    superConfig supConfig;
    /**Supervisor address. Attribute ID 10*/
    activeSuperAddr addr;
    /**State Machine variables. Attributes 1 through 3*/
    dlrStateMachineVar SMVariables;
    #ifdef IS_A_DLR_SUPERVISOR
    /**Attribute ID's 6 and 7*/
    lastActiveNode activeNode[2];
    #endif
    /**DLR Capabilities, flag
    The Map is as follows
    * 0 : Announce Based Ring Node
    * 1 : Beacon based Ring Node
    * 2-4 : Reserved
    * 5 : Supervisor capable
    * 6 : Redundant capable
    * 7 : Flush Table frame capable
    * 8-31 : Reserved
    */
    /**DLR Capabilities of the device. Attribute ID 12*/
    uint32_t dlrCapabilities;
    /**Active Supervisor precedence. Attribute ID 11*/
    uint8_t activeSuperPred;
    /**DLR Port 0 Interrupt number for ARM*/
    uint8_t port0IntNum;
    /**DLR Port 1 Interrupt number for ARM*/
    uint8_t port1IntNum;
    #ifdef IS_A_DLR_SUPERVISOR
    /**Number of Ring Faults since power up. Attribute ID 5*/
    uint32_t numRingFaultsPowerUp;
    /**Number of ring participants. Attribute ID 8*/
    uint16_t ringParticipantsCount;
    /**pointer to array of structures. Attribute ID 9*/
    protocolParticipants **ringNodes;
    #endif
} dlrStruct;
```

## **Initialization**

DLR is initialized using the API *initDLR()* and started with the API call *startDLR()*. A call to the API *stopDLR()* disables DLR feature in the firmware. Initialization needs to be done only on startup. DLR can be enabled and disabled at runtime.

The IP address for DLR node is configured through the API *addDLRModuleIPAddress(uint32\_t ipAddress)*. Developer needs to call this once an IP address has been acquired or else DLR frames will not carry an IP address.

## **PTP/1588**

EtherNet/IP uses End 2 End mode for measuring line delay and uses UDP messages over IPv4 (Annex D). All PTP/1588 information including implementation details is provided in the [PTP developer guide](#).

PTP/1588 in EtherNet/IP has been provided to enable CIP Sync feature in EtherNet/IP. This feature is unavailable at the moment because the stack does not support it however if someone wishes to implement the feature the PTP/1588 driver provides all relevant parameters.

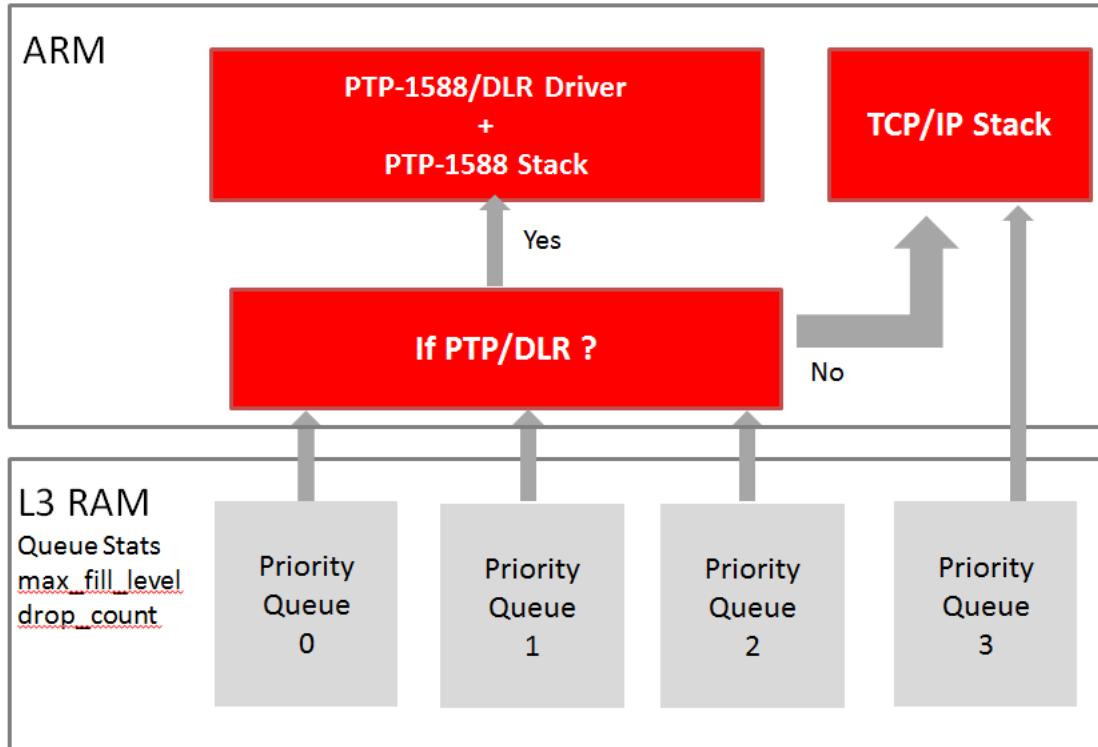
## QoS

Refer to this guide to understand how QoS works in EMAC-LLD. There are primarily two callbacks associated with the main Rx interrupt

1. Real Time (RT) Callback : rxRTCallBack For high priority protocol frames.
2. Non-Real Time Callback : rxCallBackFor other frames.

The decision to send which frame to which callback is completely configurable from *main.c*. The variable *ethPrioQueue* is set to the queue priority above which a frame goes to the TCP/IP stack callback. For EtherNet/IP this is set to *ICSS\_EMAC\_QUEUE4* which means Queue 4 and above (does not exist) goes to NDK (TCP/IP) stack. Any other queue goes to the RT callback. If the variable is configured to *ICSS\_EMAC\_QUEUE3* then Queue 3 and 4 will go to NDK.

The diagram below shows the mapping for EtherNet/IP



## Host Receive queues

- Queue 1 :
  - PTP and DLR frames are received on this queue.
  - This is called the Real Time (RT) queue and is linked to the RT callback rxRTCallBack
  - If a frame isn't DLR or PTP it goes to the TCP/IP stack
- Queue 2 : Same behavior as Queue 1
- Queue 3 : Same behavior as Queue 1 & 2
- Queue 4 : Mapped to NDK

## Transmit queues

- Queue 1
  - PTP and DLR frames are sent on this frame
- Queue 2 : Regular queue. No mapping
- Queue 3 : Same as Queue 2
- Queue 4 : Same as Queue 3

## Molex Stack

The application uses Molex EtherNet/IP stack ver. 3.3.3. Since the stack is only for evaluation purposes the application only runs for an hour. To get the full version for the stack please contact Molex.

## Memory Map

### DLR Memory Map

DLR uses 48 bytes in ICSS Shared RAM and 5 bytes each in PRU0 and PRU1 Data RAM. The data in shared RAM data sits right on top of PTP data. The map is provided below

<b>DLR</b>				
Shared Memory Offset	Value	PRU0 DRAM Offset	Value	PRU1 DRAM Offset
0x140 - 0x170	Used for storing common events and other variables	0x200 - 0x205	Used for storing Port events and Neighbor check request retries for Port 0	0x200 - 0x205
DLR_COMMON_EVENTS_OFFSET	0x140 - Stores events common to DLR. See above for more information.	DLR_PORT_EVENTS_OFFSET	0x200 - Stores Port events for Port 0	DLR_PORT_EVENTS_OFFSET
DLR_STATE_MACHINE_OFFSET	0x144 - Stores state machine values. See above	DLR_NCREQ_NUM_RETRIES_OFFSET	0x204 - Num times a neighbor check request has been resent	DLR_NCREQ_NUM_RETRIES_OF
DLR_BEACON_TIMEOUT_OFFSET	0x148 - Stores beacon timeout value			
DLR_BEACON_INTERVAL_OFFSET	0x14C - Stores beacon interval value			
DLR_VLAN_ID_SUPID_OFFSET	0x150 - Stores VLAN ID and Supervisor ID			
DLR_ACTIVE_SUP_MAC_0123	0x154 - MAC ID of Supervisor			
DLR_SIGNON_PKT_RCVD_OFFSET	0x15C - SignOn frame rcvd ack			
DLR_ENABLED_OFFSET	0x160 - Writing 1 here enables DLR			
DLR_ACTIVE_SUP_IP_OFFSET	0x164 - IP address of supervisor			
DLR_LOOP_DETECTED_OFFSET	0x168 - Loop detected in network (no DLR case)			

#### PTP/1588 Memory Map

EtherNet/IP supports E2E Transparent Clock and Ordinary Clock. The memory map is documented [here](#). Only the common and E2E specific mapping are applicable.

#### Media Counters Memory Map

The map is provided below

SQE\_Test\_Errors : SQE test errors are applicable to very old legacy devices where the transceiver is different from NIC, as such they are not supported in firmware or hardware

Carrier\_Sense\_Errors : Not supported in firmware, CSError can be read from PHY registers directly. On TLK110 for example the registers FCSCR (0x14) and PHYSTS (0x10) provide this status

#### Media Counters

DMEM0/1 Offset	Value
0x1F50	Late_Collisions
0x1F54	Single_Collisions
0x1F58	Multiple_Collisions
0x1F5C	Excessive_Collisions
0x1F60	Alignment_Errors
0x1F68	MAC_Receive_Errors
0x1F70	Deferred_Transmissions
0x1F74	MAC_Transmit_Errors
0x1F78	Frame_Too_Long
0x1F80	FCS_Errors

## Hardware Resource Usage

### Interrupts

EtherNet/IP uses

- Two identical interrupts from PRU-ICSS to ARM for DLR
  - The interrupts must be serviced in less than T microseconds assuming T is the min beacon timeout for DLR. TI's sysBios services them in less than 100us.
- Two interrupts from PRU-ICSS to ARM for PTP/1588
  - The interrupts must be serviced within the Sync interval for the network.
  - If Sync interrupt is processed with a lot of delay it has a direct bearing on the jitter.
- One interrupt from PRU-ICSS to ARM for Receive on all queues. This is common to all switch implementations.
  - Refer to the ICSS [EMAC Design guide](#).

The specific interrupt numbers are provided [below](#) and mapping can be understood by looking at the file X\_pruss\_intc\_mapping.h file

### Timers

As explained previously DLR uses the on board DMTimer (GPTimer under new notation) 4 & 5 for beacon timeout so another application cannot use them.

- DMTimers
  - Resolution required in microseconds
  - Min resolution : 200 us (minimum DLR beacon timeout value)
  - Two. 4 & 5
  - BIOS API's are used to configure, start and stop the timers.
- Generic timers
  - Resolution required in milliseconds
  - Min resolution : 100 ms (neighbor timeout)
  - Configured using BIOS API's

### Instruction memory and Data Memory

Please refer to the data sheet and memory map.

## Porting your own EIP Stack

Porting an EIP Stack generally requires porting of Socket implementation and providing APIs in application to inform about the application status and volatile storage information. Socket implementation adaptation is not in the scope of this document

The current Ethernet/IP application is specific to Molex Stack and need to be changed accordingly to use it along with other stacks. Developer needs to modify following files to make the application work with new EIP stack.

- hal.c - This file has functions which configure Ethernet Link,TCP/IP and QOS object properties. Features like Statistics, saving information to non volatile storage are also done here. Following table has the mapping of Ethernet/IP Object and the APIs used to get or set the attribute values

*Ethernet/IP Object details to API mapping*

Ethernet/IP Object	API to get/set attribute values
Identity (0x01)	EIPAPP_resetImplement
Connection Manager(0x06)	EIPHAL_getIncarnID EIPHAL_setIncarnID
QOS(0x48)	EIPHAL_setQoSValues EIPHAL_getQoSValues

TCP/IP(0xF5)	EIPHAL_getConfigurationControl EIPHAL_setConfigurationControl EIPHAL_getInterfaceConfiguration EIPHAL_setInterfaceConfiguration EIPHAL_setACDEnable EIPHAL_getACDEnable EIPHAL_getInterfaceConfigACDStatus EIPHAL_setLastConflictDetails EIPHAL_getLastConflictDetails EIPHAL_getMediaCounters
Ethernet Link(0xF6)	EIPHAL_getInterfaceSpeed EIPHAL_getInterfaceFlags EIPHAL_getInterfaceCounters EIPHAL_getPhysicalAddress EIPHAL_getInterfaceControl EIPHAL_setInterfaceControl EIPHAL_getInterfaceStarupType EIPHAL_getPhysicalAddress

- user\_acd\_ndk.c - This file has functions that interface between lower layers and ACD Stack . ACD stack needs to call following APIs to implement ACD mechanism
  - EIPACD\_sendArpFrame - function to send out ARP frames
  - EIPACD\_conflictDetected - function to save the Conflict parameters to non volatile memory
  - EIPACD\_event -function to be called by ACD stack to inform application of ACD events

## Application Interface

### Initialization

Initialization involves

- Board and Hardware Initialization
- Mapping the interrupts
- Allocating memory for the structures
- Registering the callbacks
- Configuring Switch parameters

### Board and Hardware Initialization

This part is application specific. In Processor SDK the main API to call is `Board_Init()` with predefined flags which initialize each specific HW module of choice. For example in EtherNet/IP adapter application the modules which are configured by this are

- Pin Mux
- Timers
- UART

The following code performs this task

```
Board_init(BOARD_INIT_PINMUX_CONFIG | BOARD_INIT_MODULE_CLOCK | BOARD_INIT_UART_STDIO);
```

Some modules like I2C's are not initialized by this. They are initialized with the call

```
Board_i2cLedInit();
```

For more about pinmux and hardware specific initialization please consult the [EMAC-LLD Porting Guide](#).

### Mapping the Interrupts

The EtherNet/IP adapter uses the following PRU-ICSS Interrupts

<i>Stack/application interrupts</i>		
Firmware Interrupt	Host Interrupt	Remarks
Rx Interrupt	PRU_ICSS_EVTOUT0	Notifies host when firmware has stored a frame in host receive queue
DLR Port 0 Interrupt	PRU_ICSS_EVTOUT1	Raised when there is a state change in DLR on Port 0
DLR Port 1 Interrupt	PRU_ICSS_EVTOUT2	Raised when there is a state change in DLR on Port 1

PTP/1588 Sync Interrupt	PRU_ICSS_EVTOUT3	Raised when PTP/1588 Sync frame is received
PTP/1588 Delay Response Interrupt	PRU_ICSS_EVTOUT4	When a delay request sent by device is acknowledged with a delay response.
LINK0 Interrupt	PRU_ICSS_EVTOUT6	Interrupt is raised when the Link on MII0 port comes up or goes down
LINK1 Interrupt	PRU_ICSS_EVTOUT6	Interrupt is raised when the Link on MII1 port comes up or goes down

The interrupt numbers are configured at the start of application, this along with the interrupt mapping contained in *tieip\_pruss\_intc\_mapping.h* is used to program the PRUSS Interrupt controller and ARM interrupt controller. The interrupt mapping is explained [here](#)

Sample interrupt initialization code from EtherNet/IP example provided below

```
/*Initialize PTP*/
ptpConfig.syncIntNum = 23;
ptpConfig.genericIntNum = 24;
/*This is unused if E2E mode is used*/
if(P2P == ptpConfig.type)
{
    ptpConfig.pDelayRespP1IntNum = 25;
}
/*Initialize DLR*/
dlrObj.port0IntNum = 21;
dlrObj.port1IntNum = 22;
/*Initialize Rx interrupt and Link interrupt*/
switchEmacCfg->rxIntNum = 20;
switchEmacCfg->linkIntNum = 26;
```

### Allocating Memory for Structures

Application explicitly allocates memory only for the ICSS EMAC Handle and the Switch configuration. This is done by the following code

```
/*Create Handle*/
handle = PRUICSS_create(pruss_config, PRUICSS_INSTANCE);
/*Allocate memory for Handle*/
emachandle = (ICSS_EmacHandle*)malloc(sizeof(ICSS_EmacConfig));
/*Switch configuration*/
ICSSEMAC_InitConfig *switchEmacCfg;
/*Allocate memory for config structure*/
switchEmacCfg = (ICSSEMAC_InitConfig *)malloc(sizeof(ICSSEMAC_InitConfig));
```

Memory for specific modules is allocated and handled by the respective module initialization API's. For example `initDLR()` and `ptpInit()` handle initialization for DLR and PTP respectively.

### Registering the callbacks

The standard callbacks which are common to all switch implementations are allocated by the API `ICSSEmacDRVInit()` with the following

```
/* Callback mallocs */
ICSS_EmacCallBackObject *callBackObj = (ICSS_EmacCallBackObject *)malloc(sizeof(ICSS_EmacCallBackObject));
/*Learning callback*/
callBackObj->learningExCallBack = (ICSS_EmacCallBackConfig *)malloc(sizeof(ICSS_EmacCallBackConfig));
/*Generic Rx and Tx callbacks for the NDK*/
callBackObj->rxCallBack=(ICSS_EmacCallBackConfig*)malloc(sizeof(ICSS_EmacCallBackConfig));
callBackObj->txCallBack=(ICSS_EmacCallBackConfig*)malloc(sizeof(ICSS_EmacCallBackConfig));
/*Default Real time callback*/
callBackObj->rxRTCallBack = (ICSS_EmacCallBackConfig *)malloc(sizeof(ICSS_EmacCallBackConfig));
/*Assign the callbacks*/
((ICSS_EmacObject *)handle->object)->callBackHandle = callBackObj;
```

- Learning callback is used by DLR to disable learning of Supervisor MAC ID. This is a protocol specific requirement. It is set to NULL by default and is mapped to the API `checkSupervisorException()` in the DLR Initialization API.
- Generic Tx and Rx callbacks are mapped to the default TCP/IP Transmit and Receive API's respectively inside the EMAC-LLD.
- RT callback is meant for the highest priority queue or which ever queue is configured exclusively for protocol specific frames. Please see the QoS section for how it works with EtherNet/IP.

For protocol specific callbacks a separate callback is registered which overwrites the default and assigns the Real Time callback to a protocol specific API. In EtherNet/IP this API is `processProtocolFrames` which is a combined callback for PTP and DLR frames. The callback `processProtocolFrames()` copies the frame to a temporary buffer and parses the protocol field to check if it's a PTP or DLR frame before passing it to the appropriate handler.

Callback assignment for EtherNet/IP in `eip_driver_init()`

```
/*Packet processing callback*/
(((ICSS_EmacObject *)
    icssEmacHandle->object)->callBackHandle)->rxRTCallBack)->callBack =
    (ICSS_EmacCallBack)processProtocolFrames;
(((ICSS_EmacObject *)
    icssEmacHandle->object)->callBackHandle)->rxRTCallBack)->userArg =
    icssEipHandle;
```

### Configuring Switch parameters

Switch/EMAC parameters are part of the structure `ICSSEMAC_InitConfig`. The members are explained in [EMAC LLD Design guide](#)

For EtherNet/IP configure the parameters as follows.

### Common Settings

- `portMask` : set to `ICSS_EMAC_MODE_SWITCH`
- `ethPrioQueue` : set to Queue 4. `ICSS_EMAC_QUEUE4`
- `halfDuplexEnable` : Enable half duplex by setting to 1
- `enableIntrPacing` : Enable pacing by setting to 1
- `intrPacingMode` : Use pacing mode 1. `INTR_PACING_MODE1`
- `pacingThreshold` : Set to 100
- `learningEn` : Enable by setting to 1

### AM335x

- `rxIntNum` : Set to 20 if default mapping is used
- `linkIntNum` : 26 if default mapping is used

### AM437x

- `rxIntNum` : Set to 52 if default mapping is used
- `linkIntNum` : 58 if default mapping is used

## Basic self debugging

This section is meant to be a self-help guide for users who are trying to evaluate TI's Ethernet/IP solution and gain some experience with the ISDK and EVM for Ethernet/IP.

This document mentions several tools and software packages. The following minimum revisions are required to perform a complete set of the tests described in this document.

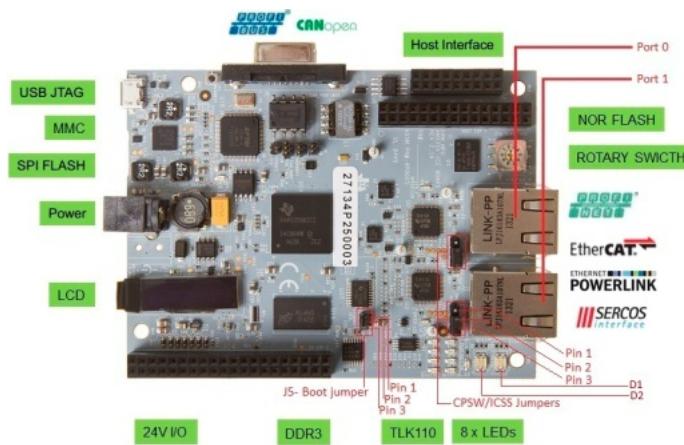
1. RSLinx (Rockwell Automation) version 2.57 or later
2. EDITT (Pyramid Solutions), version 1.18 or later <http://network.pyramidsolutions.com/software-#> Molex EtherNet/IP Tool v2.4 or later (EIPTool.exe)
3. CIP Tool – a CIP Explicit client that allows the user to build CIP explicit messages. A tool such as Pyramid Solutions' EIP Scan will suffice.
4. Molex Plugfest Performance Packet Generator v1.1 or later
5. NIST Industrial Ethernet Network Performance (IENETP) Test Tool v1.1.2 or later.
6. Hilscher netAnalyzer PCI-card (NANL-C500-RE) installed in a Microsoft Windows PC.

The test descriptions contained in this section will contain images captured from these tools for illustration. All rights are retained by the producers of these tools.

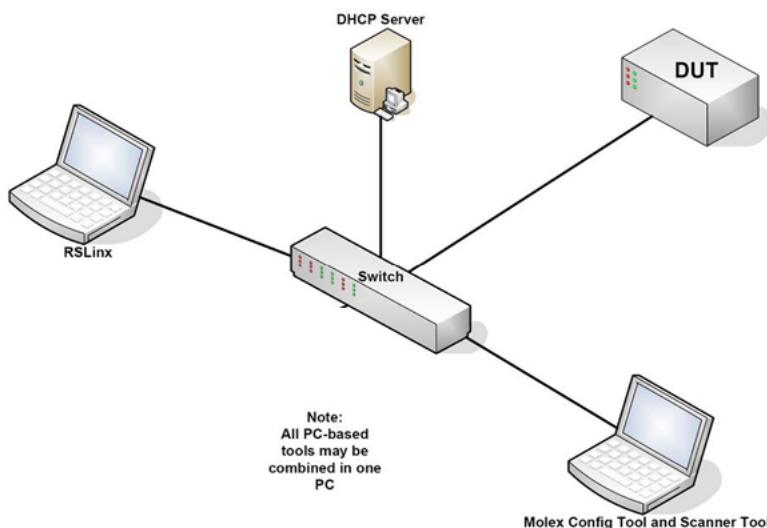
### Test Platform Description

TI EtherNet/IP adapter example is a generic EtherNet/IP adapter module which can be run against an EtherNet/IP Scanner device. The Application uses a Molex EtherNet/IP stack on top of TI's NDK TCP/IP stack. The example included in ISDK release package is a limited demo application where the user will be allowed to execute it for one hour at a single stretch.

EtherNet/IP adapter implementation is available on low latency ICSS cut-thru switch. On the ICE v2 EVM, the jumper settings must be set appropriately to select ICSS. Jumpers J18 and J19 (Please refer ICE V2 picture below) are used for this purpose. ICSS can be selected by connecting jumper pin between pins 2 & 3.



*ICEv2 EVM marked with ICSS mode jumpers*

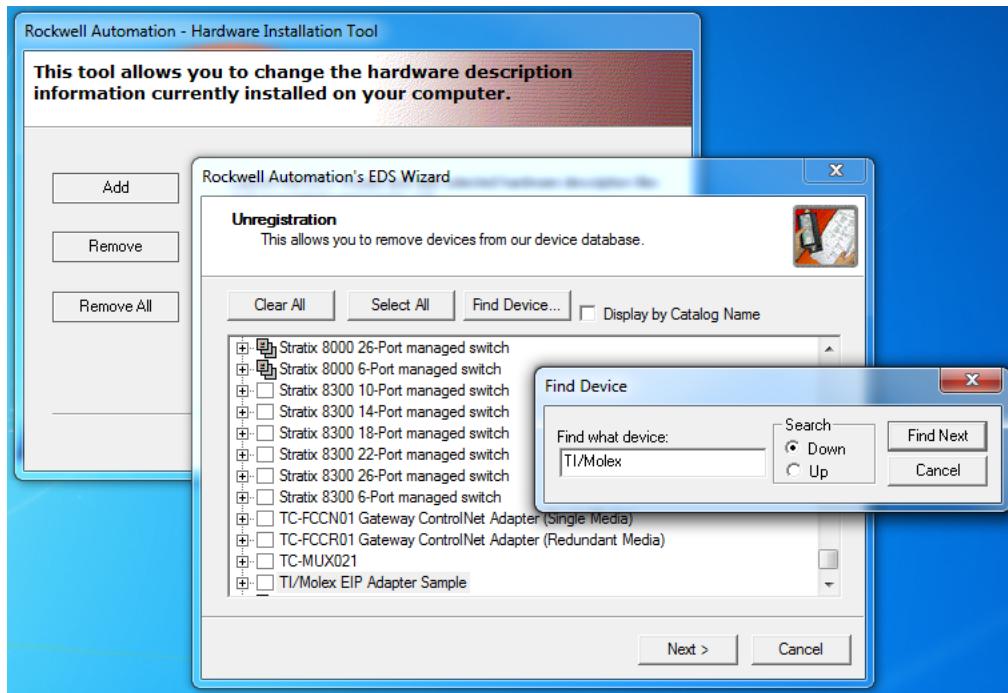
*Test setup***EDS File Installation and Verification with RSLinx**

Here we can verify the correctness of the EDS file parameters. The requirement is that the DUT should be properly detected and listed in RSLinx.

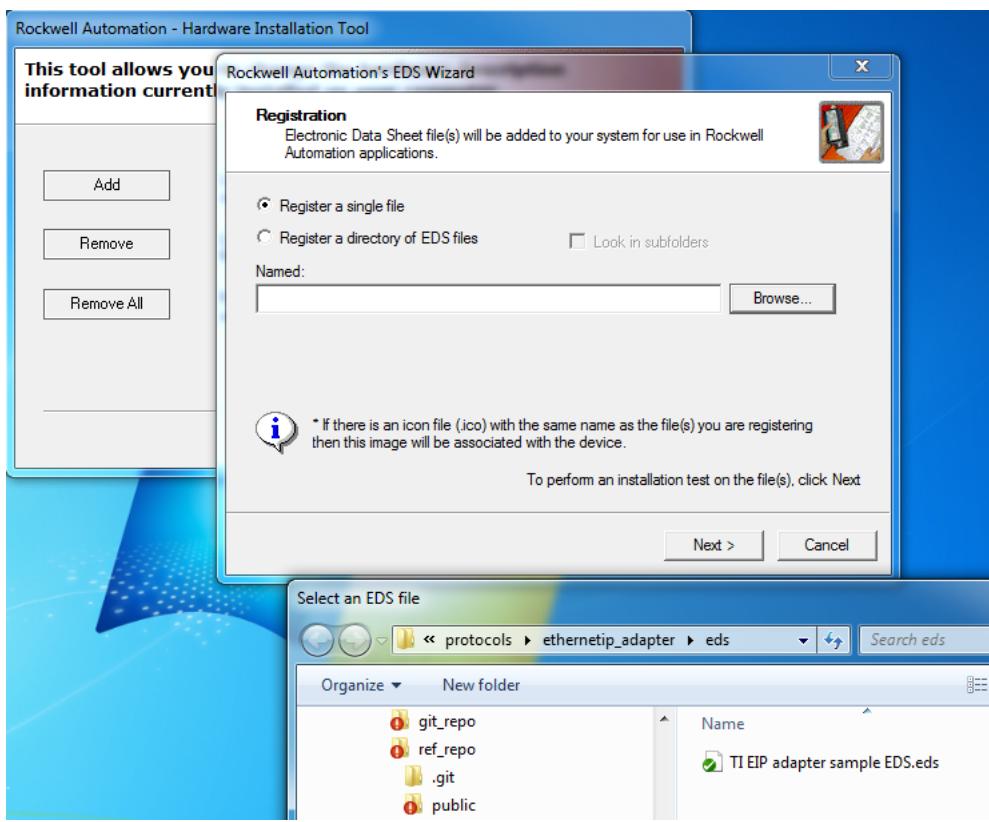
Before proceeding with RSLinx, we need to register the EDS file to the tool library using 'EDS Hardware Installation Tool'. This tool comes along with RSLinx and can be found at **(C:\Program Files (x86)\Rockwell Software\RSCommon\RSHWare.exe)**.

**NOTE:** The EDS file for TI/Molex EIP adapter can be found in ISDK package at **(ISDK\_HOME)\protocols\ethernetip\_adapter\eds**. The EDS files for standard devices can be downloaded from Rockwell Automation website [http://www.ab.com/networks/eds/cgi-bin/search.cgi?network\\_type=EtherNet/IP](http://www.ab.com/networks/eds/cgi-bin/search.cgi?network_type=EtherNet/IP) by searching for the device model name.

**Test Procedure:** 1. On startup the Hardware installation tool will provide the option for adding or removing the EDS file (Remove the already existing EDS file and always use the latest updated one). We can search the EDS file by name and remove the already registered one.

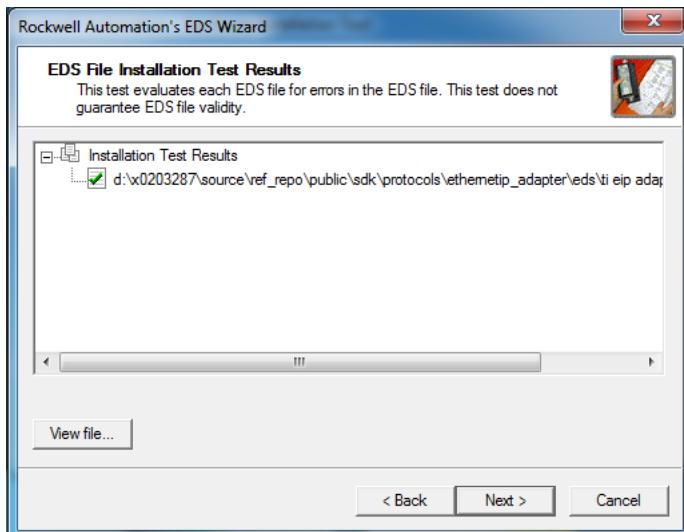
*Removing the already registered EDS file*

2. Now we can add the latest EDS file into the library using 'Add' option



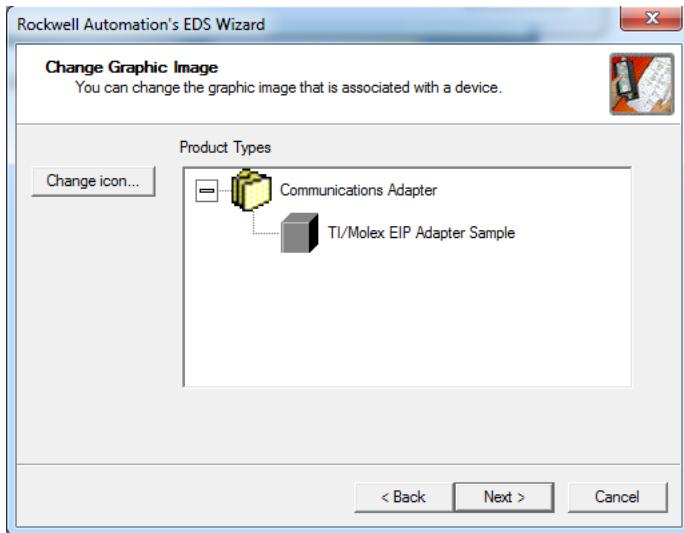
Register a new EDS file using 'Add' option in the installation tool.

3. If there are no syntax errors, it will be represented by a 'Green' Check mark near to the file name. We can view the file also by clicking on 'View File'. Click 'Next'



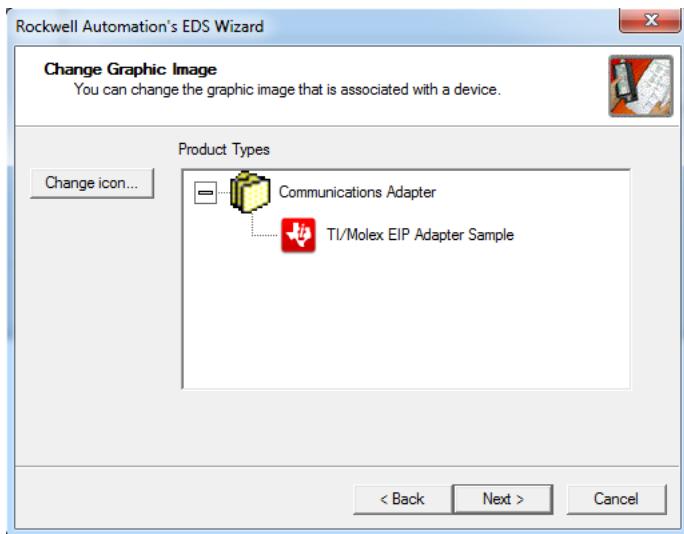
The EDS file installation

4. The Device is going to be added with a default icon. We can use a custom icon by clicking on 'Change Icon'



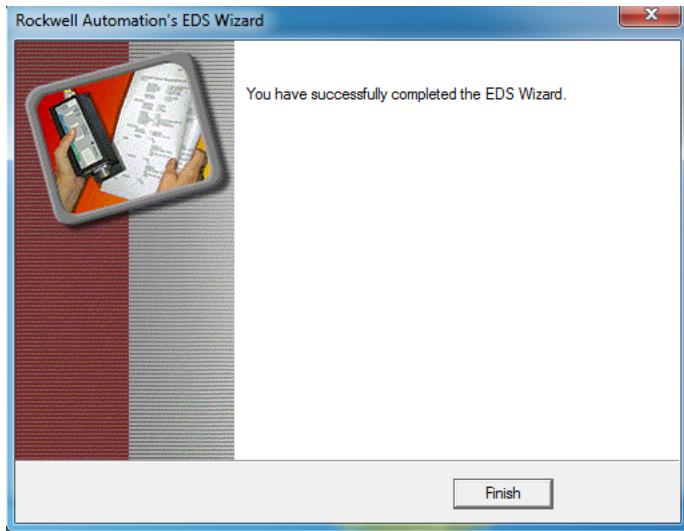
*Assigning an icon to the DUT*

5. If there is a vendor specific icon embedded in the [Device] section of the EDS file, the EDS wizard will show this icon. Click 'Next'



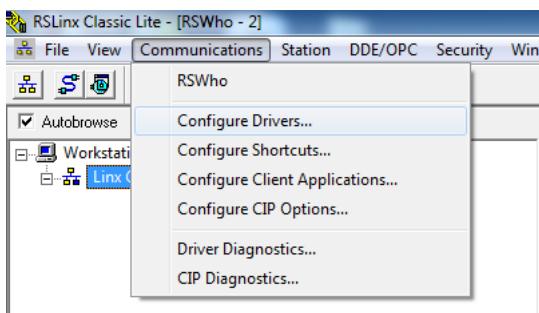
*EDS file with vendor specific Icon*

6. Click 'Next' and finish the EDS installation wizard.



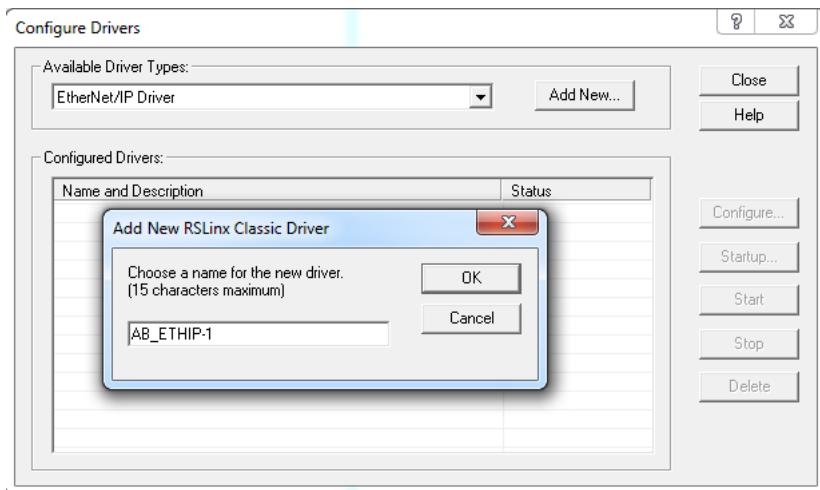
*EDS file added successfully*

7. Launch RSLinx Classic (C:\Program Files (x86)\Rockwell Software\RSLinx\RSLINX) and instantiate the proper communications driver. From the main menu select "Communications" then "Configure Drivers"; pick EtherNet/IP Driver from "Available Drivers" and select "Add New". You can keep the default name (AB\_ETHIP-x) or choose new name. Also, to avoid any confusion, make sure that the "Ethernet devices" driver is not listed in the "Configured Drivers" list. If it is, select the driver and then select "Stop" or "Delete".



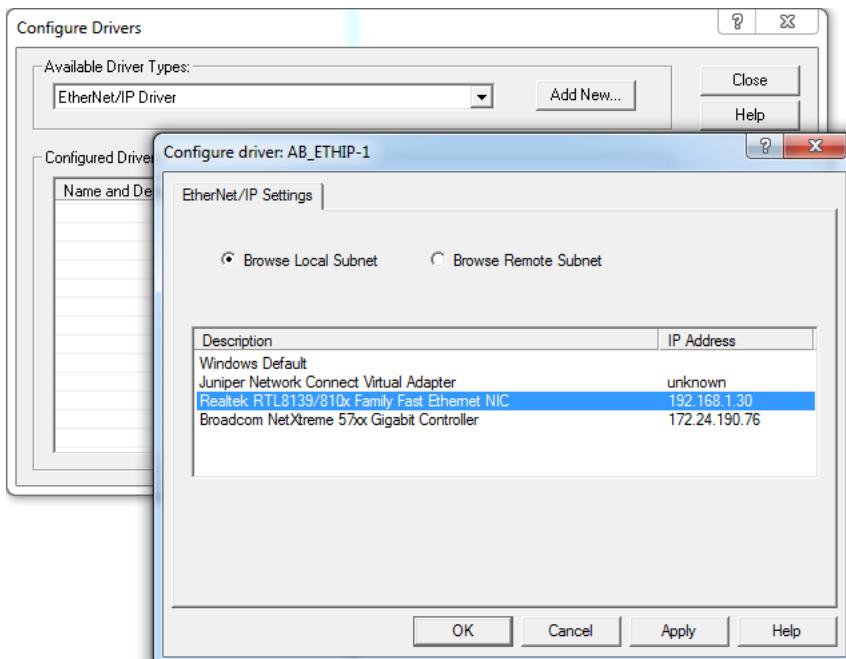
*Configuring proper communication driver*

8. Click On 'Add New' and select a name for the Driver. Click OK



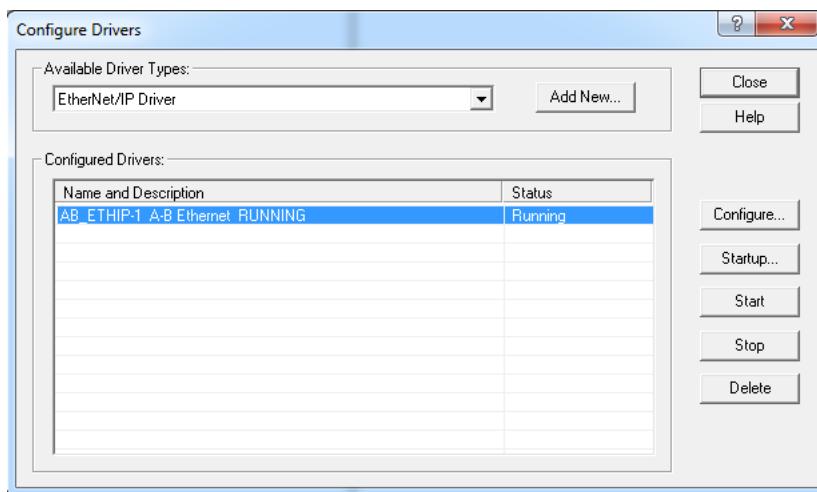
*Configuring driver name.*

9. Select the proper NIC which is configured in the same subnet as that of the DUT IP address. Click OK

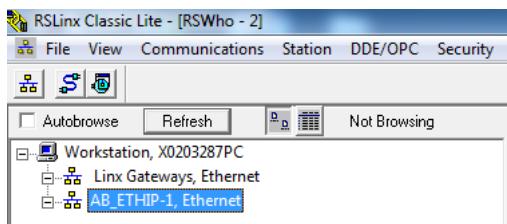


*Selecting the NIC*

10. Now the communication driver is configured properly. Driver status should show 'Running'. Click 'Close'.

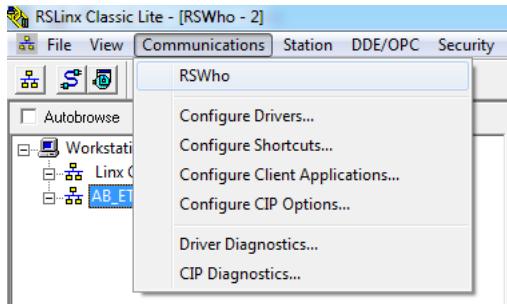


*Completing Driver configuration*



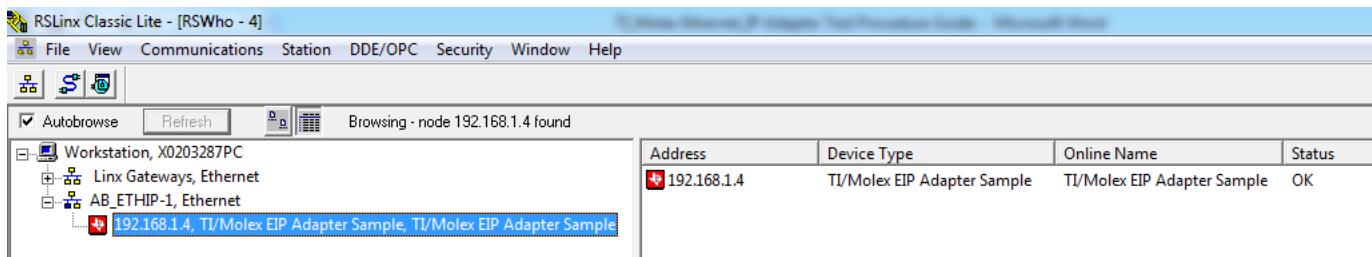
*Newly configured driver is listed in RSLinx*

11. Power up the DUT and run the RSWho utility in RSLinx. Open the RSWho window within RSLinx (Connections →RSWho) and browse the network with the DUT using the EtherNet/IP driver. Verify that the DUT is discovered by RSWho and is displayed correctly.



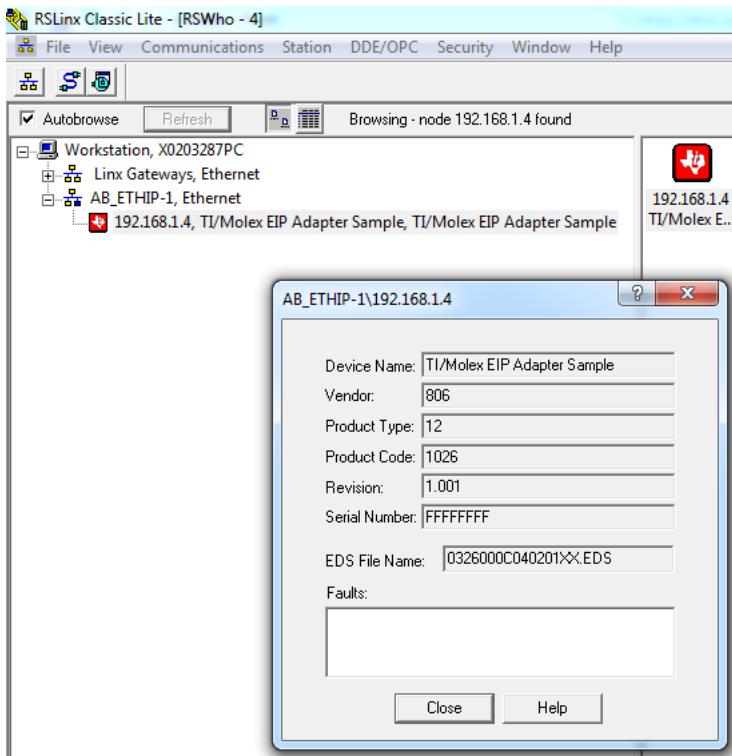
*Running RSWho*

12. When the 'Auto browse' option is enabled, the tool will automatically browse the network. If any valid DUT is found which is matching with the EDS file information, it will be listed immediately.



*The DUT is detected and displayed'*

13. Right click on detected device and select 'Device Properties'. Make sure that the device related information is matching with the EDS file contents.



#### DUT properties

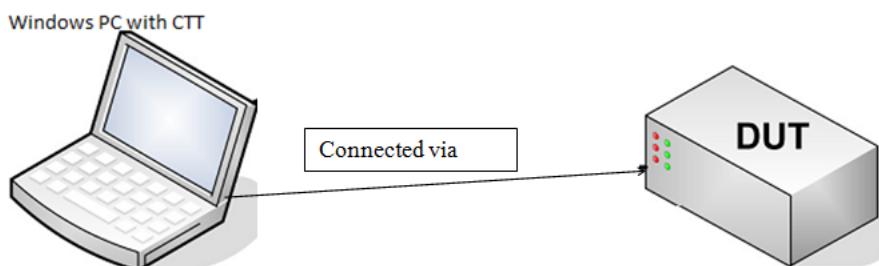
**NOTE 1:** Sometimes, the Antivirus running in the test machine (say Symantec) may hinder the discovery of DUT by RSLinx. If the DUT is not detected at all, try disabling antivirus and run RSWho once again.

**NOTE 2:** The requirement is that the device is displayed as a recognized device type based on the EDS; the device should not be displayed as a question mark. If the DUT does not show up with this driver but can be detected when the RSLinx Ethernet driver is used, then this either means there is something wrong with the Ethernet settings of the test PC (Firewall, VPN etc.) or the DUT does not properly support the List Identity request.

**NOTE 3:** Shut down RSLinx before proceeding to the next step.

#### Conformance Testing (ODVA CTT v12)

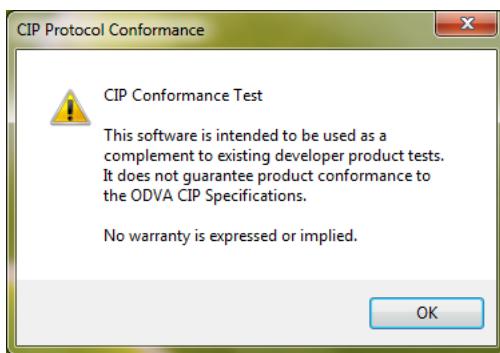
**Test Setup** Conformance Test Tool is installed in Windows Test Machine. The machine is devoid of Firewall and Antivirus software which may interfere with some of the tests. The DUT connected to the Test machine using an Ethernet cable.



#### Test Setup for Conformance Testing

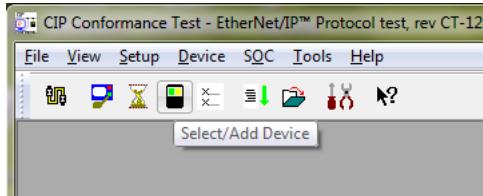
##### Test Procedure

- When the tool is launched, it will show a confirmation message. Click OK to proceed

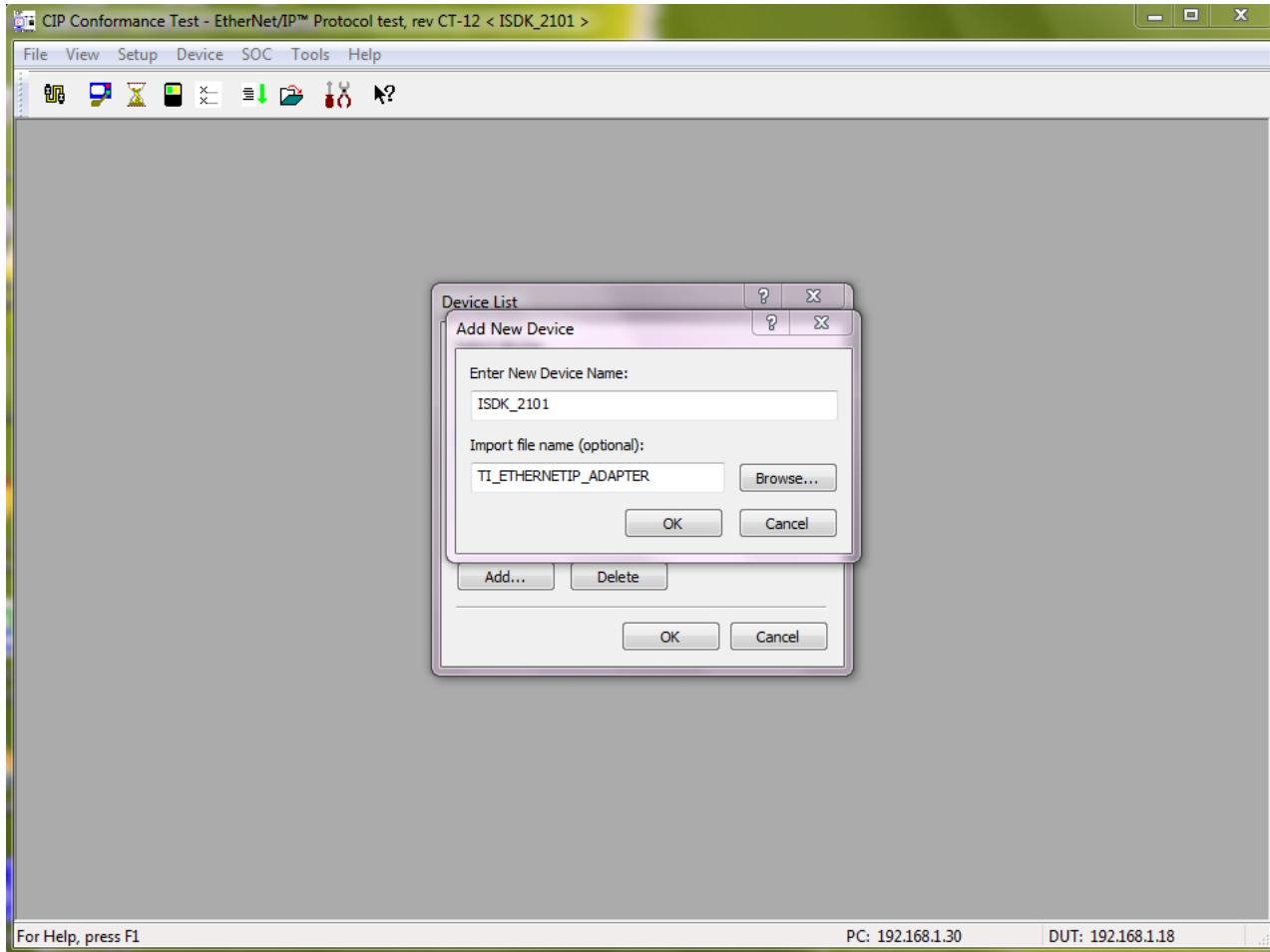


#### ODVA CTT Startup Message

(ii) Select Add Device Option. Browse and select the STC file ({IA\_SDK\_HOME}\examples\ethernetip\_adapter\stc) provided along with the release package. Provide a name for the device and press 'OK'

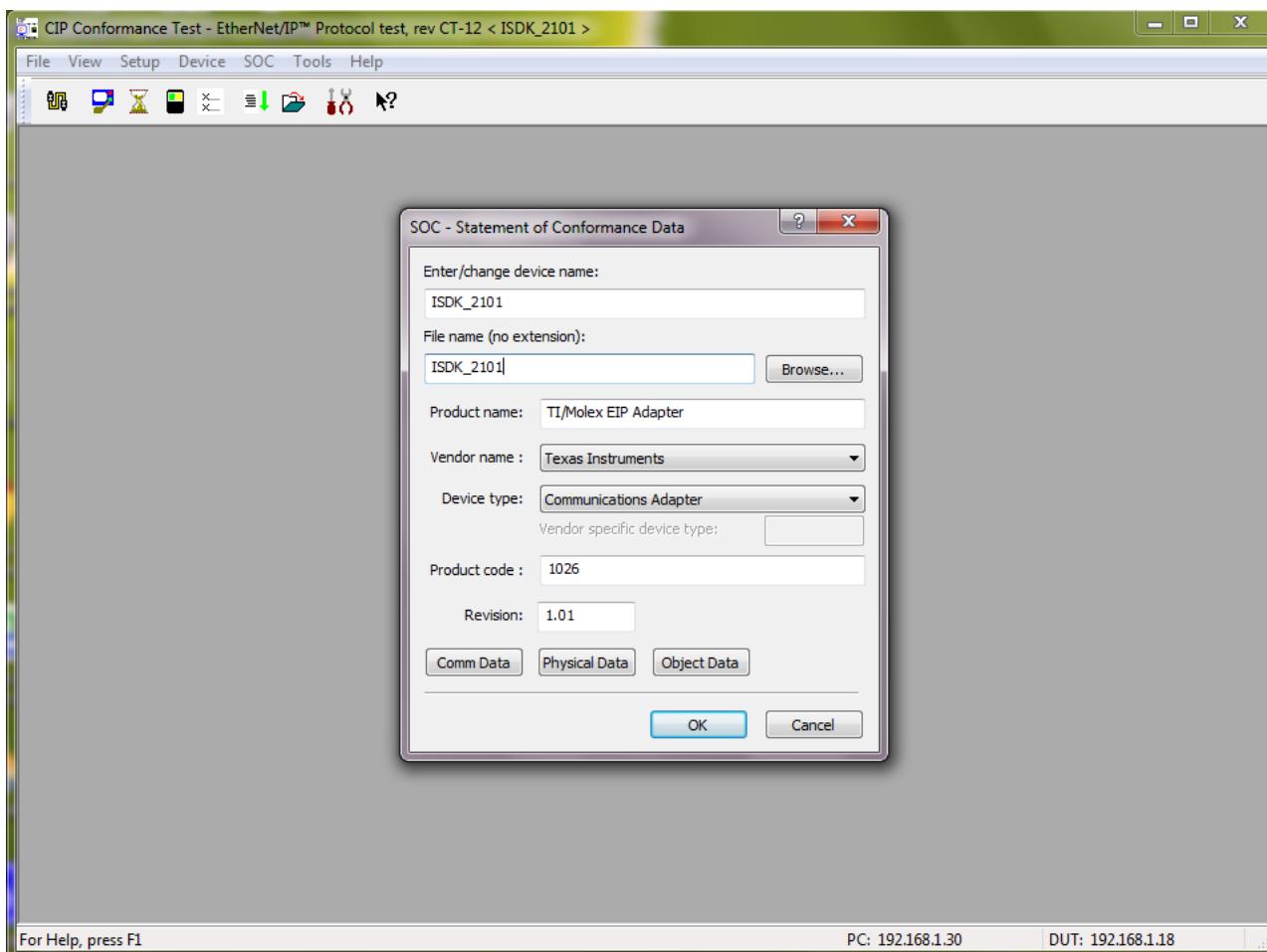


#### Select/Add a new Device



#### Adding a new Device

(iii) Make sure that the Adapter related information (Vendor Name, Device Type, Product code etc...) are read and listed properly by the CTT. Give a log file name so that the CTT log is saved in this name.

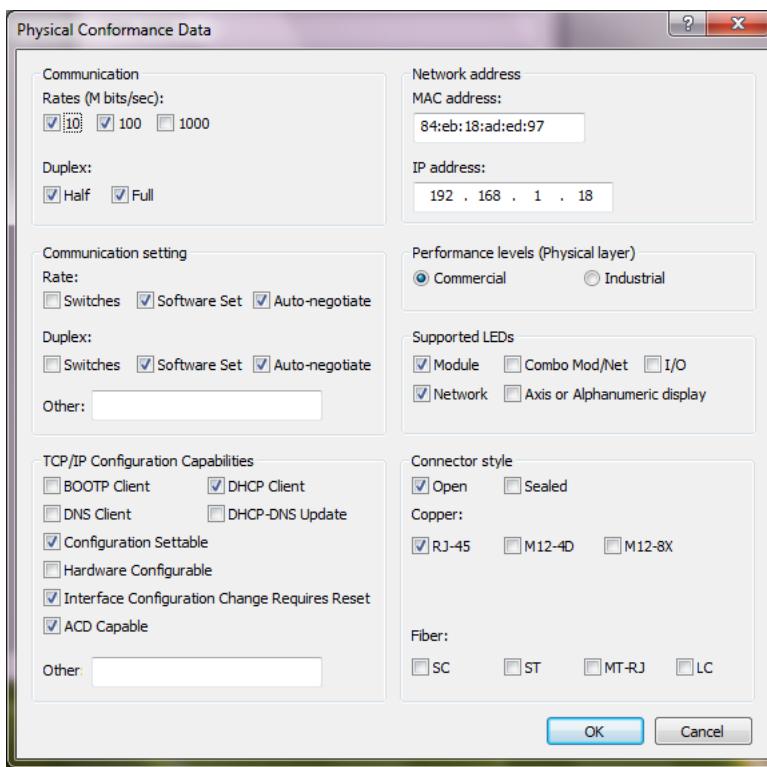


#### Configuring Device name and CTT log file name

**NOTE:** In-case 'Texas Instruments' is not listed as Vendor Name in the tool, the below steps need to be followed.

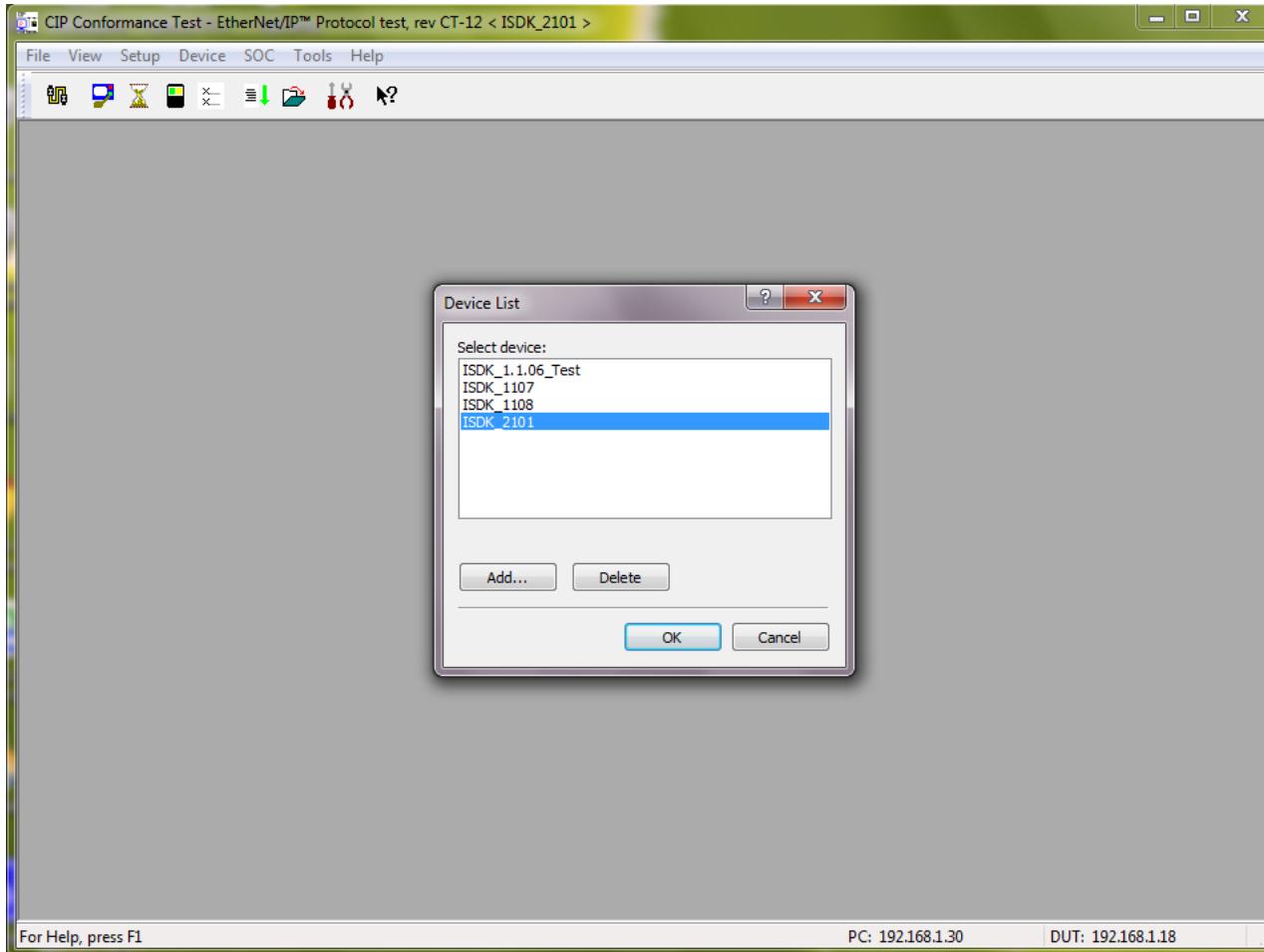
1. Open the 'VID.dat' file from the Conformance Tool installation directory.
2. Go to the vendor ID 806 in the file and remove the negative sign (-) in the ENet Column.
3. Restart the CTT.

(iv) Select Physical data and change the IP Address and MAC address as that of DUT.



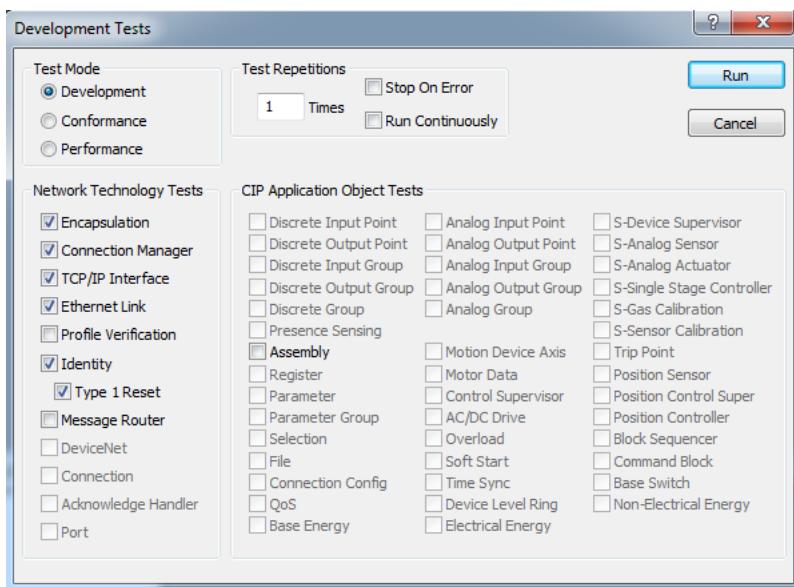
Configuring Physical Data to match DUT parameters

(v) Now the DUT has been added to the CTT device list. We can edit the parameters as and when required. When more than one device is listed in the device list, select the appropriate one to start the tests.

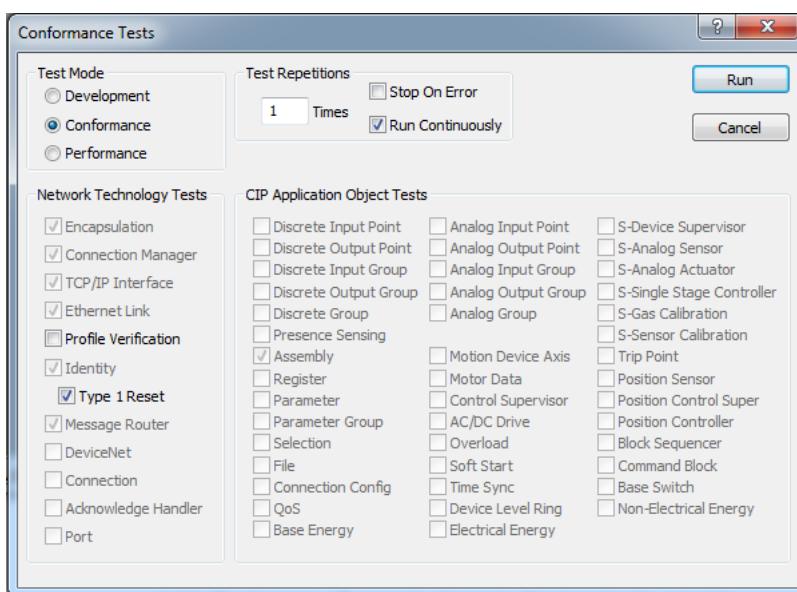


The newly added device is listed in Device List

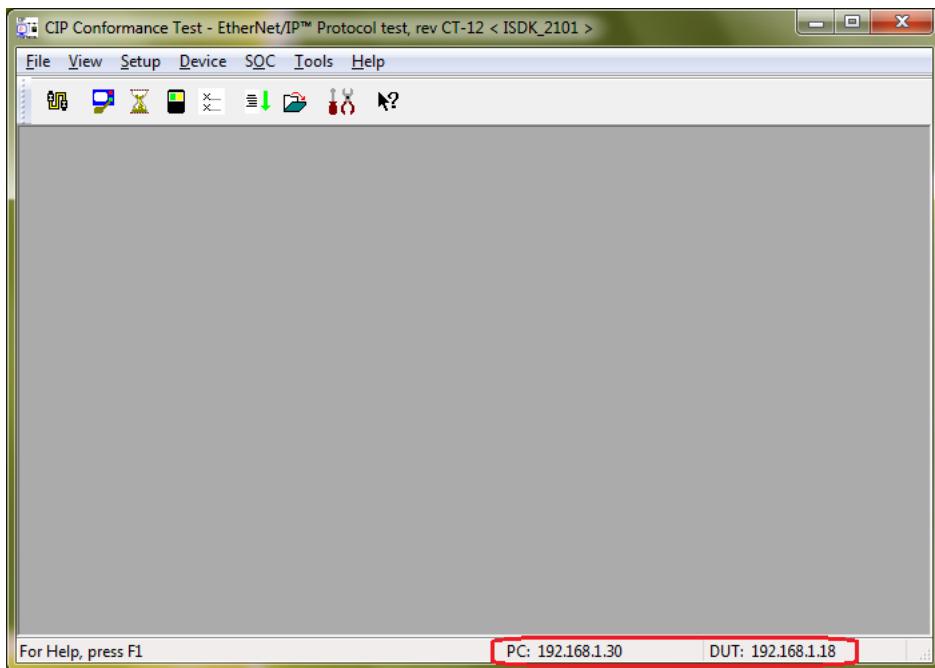
(vi) Click on the icon (or Tools -> Run Tests). In the test window, we can select individual development tests under test mode 'Development'.

*CTT Development Test Window*

(vii) In Conformance Test mode, the tool will automatically select the supported test cases as per the STC file. (Tick ‘Run continuously’ option when continuous iterations are required).

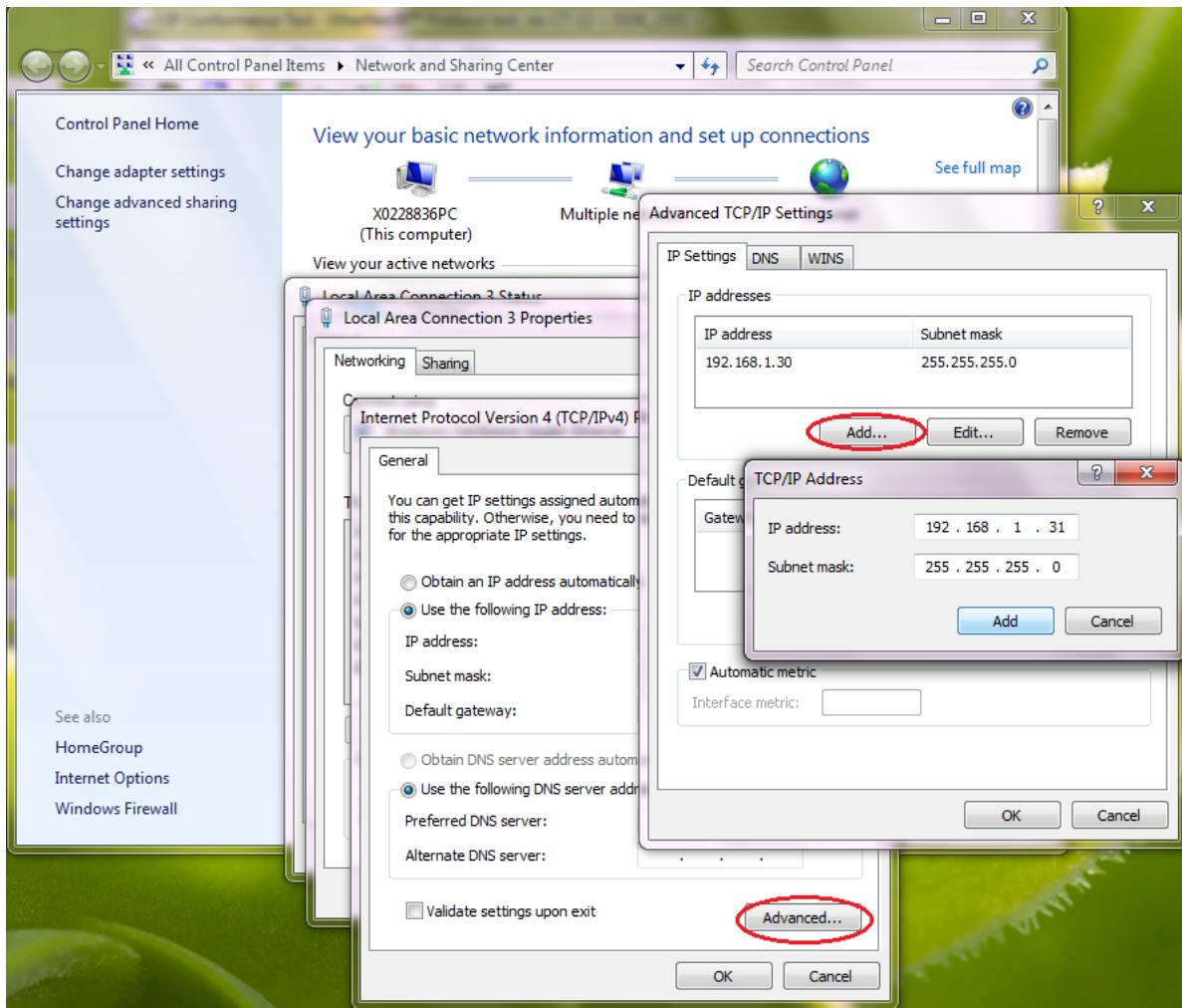
*CTT Conformance Test Window*

**IMPORTANT:** Make sure that the NIC of the PC in which the CTT is running is configured properly. Ideally the DUT IP and the IP address of the test machine should belong to the same subnet as shown below-



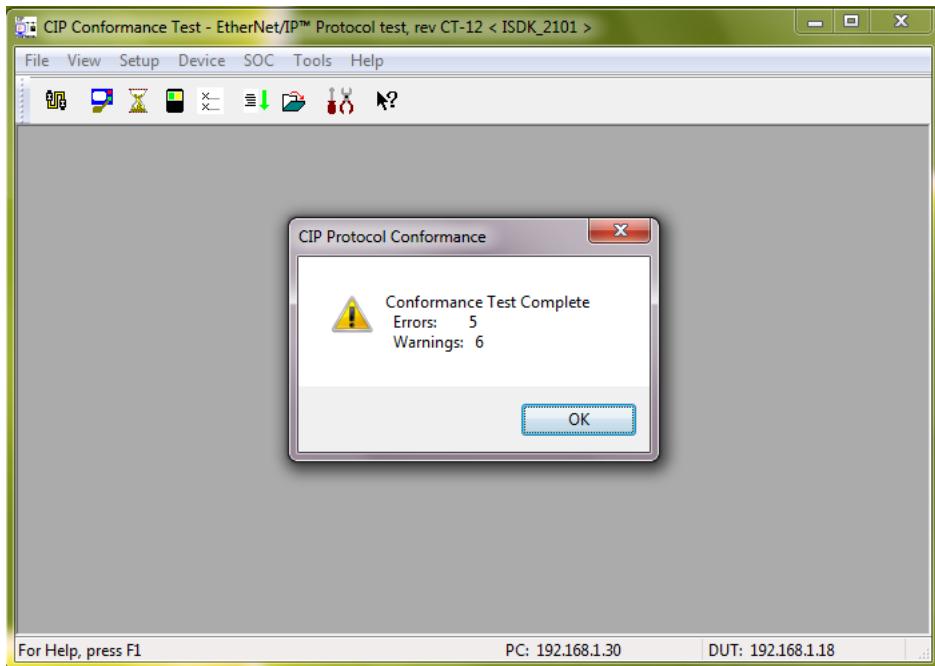
Test PC should be configured in the same subnet as that of DUT

**NOTE:** Additional IP needs to be configured to the test machine before running the CTT tests as follows:



#### Configuring additional IP

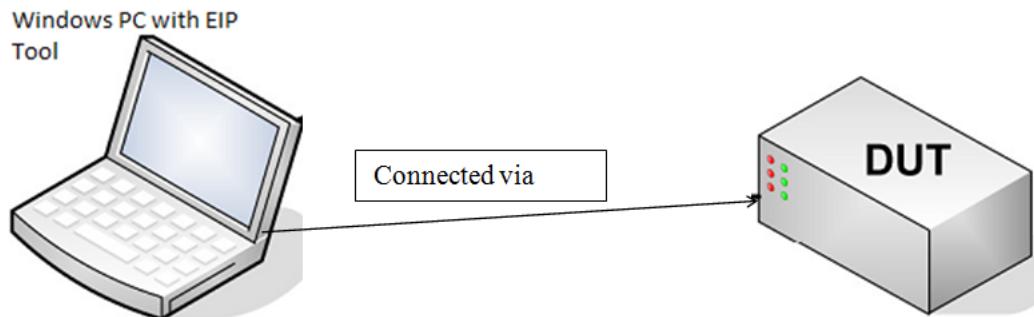
- When the testing is completed, CTT will show the results which will reveal the number of errors and warnings.



Final Test Results Window (The number of errors may vary)

#### Molex EIP Tool

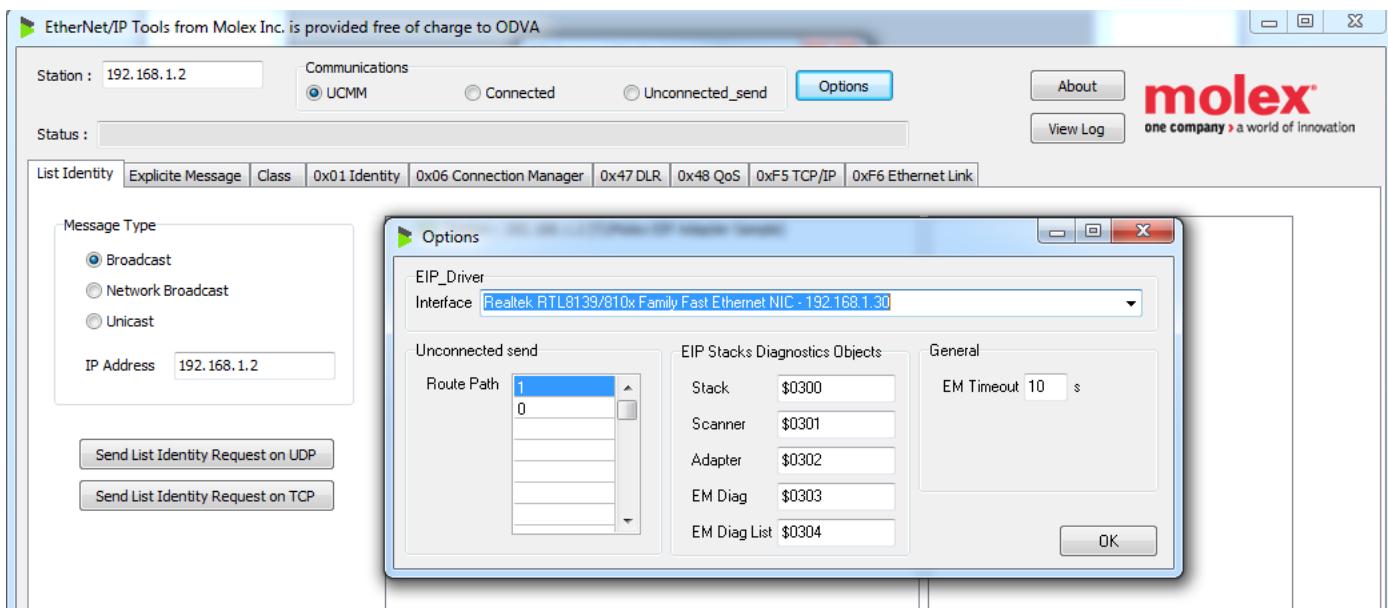
- Test Setup**
1. The Molex EIP tool can be downloaded from Molex Website and shall be extracted to Windows PC (Installation not required).
  2. Launch EIP\_Tools.exe
  3. The DUT connected to the Test machine over LAN.



Test setup for EIP tool test

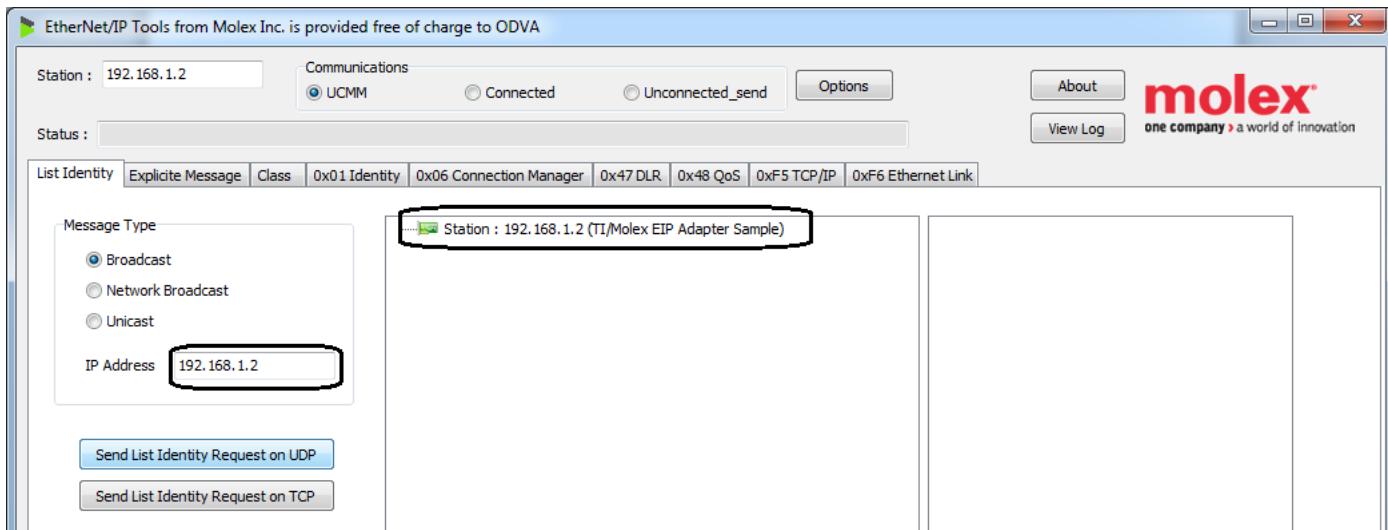
#### Test Procedure

**NOTE:** Before proceeding with communicating with DUT, make sure that the test machine NIC is configured in the same subnet as that of DUT.



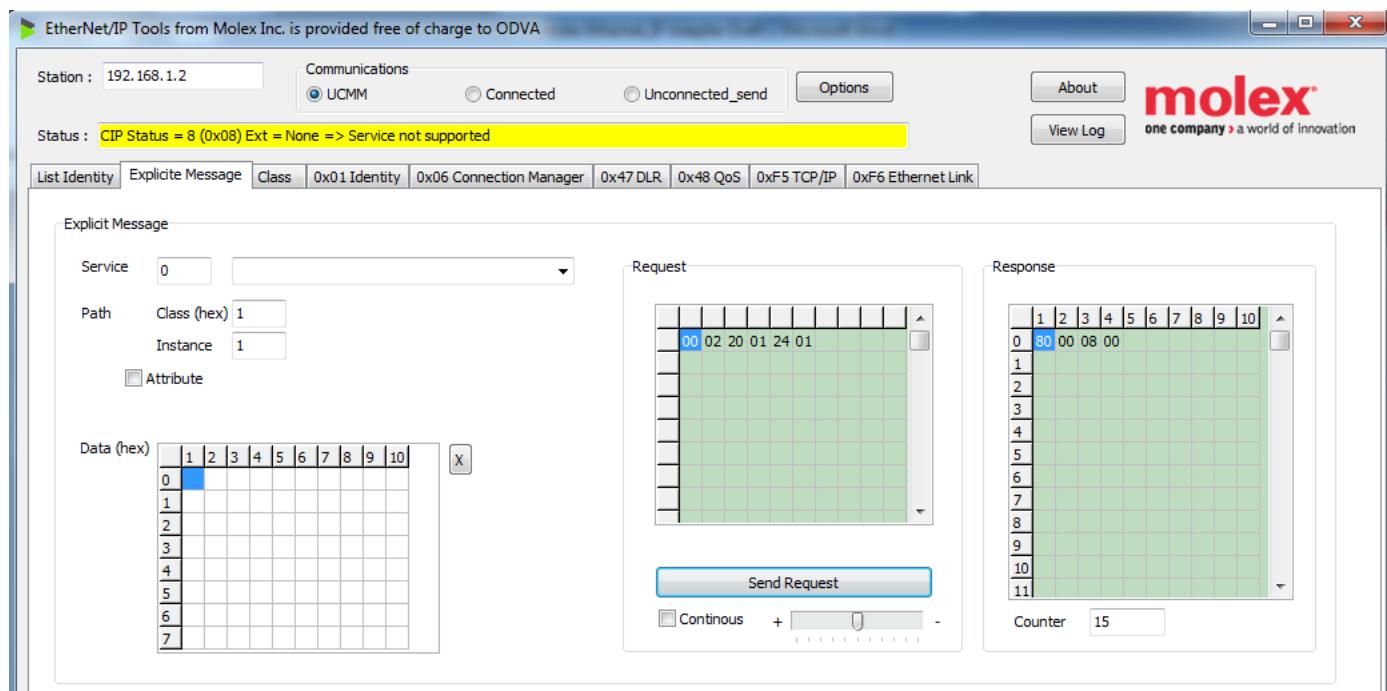
Selecting the proper NIC

(ii) Configure the DUT IP address and issue a 'List Identity Request' to the DUT. If the DUT is detected, it will be listed properly as TI/Molex EIP Adapter Sample.

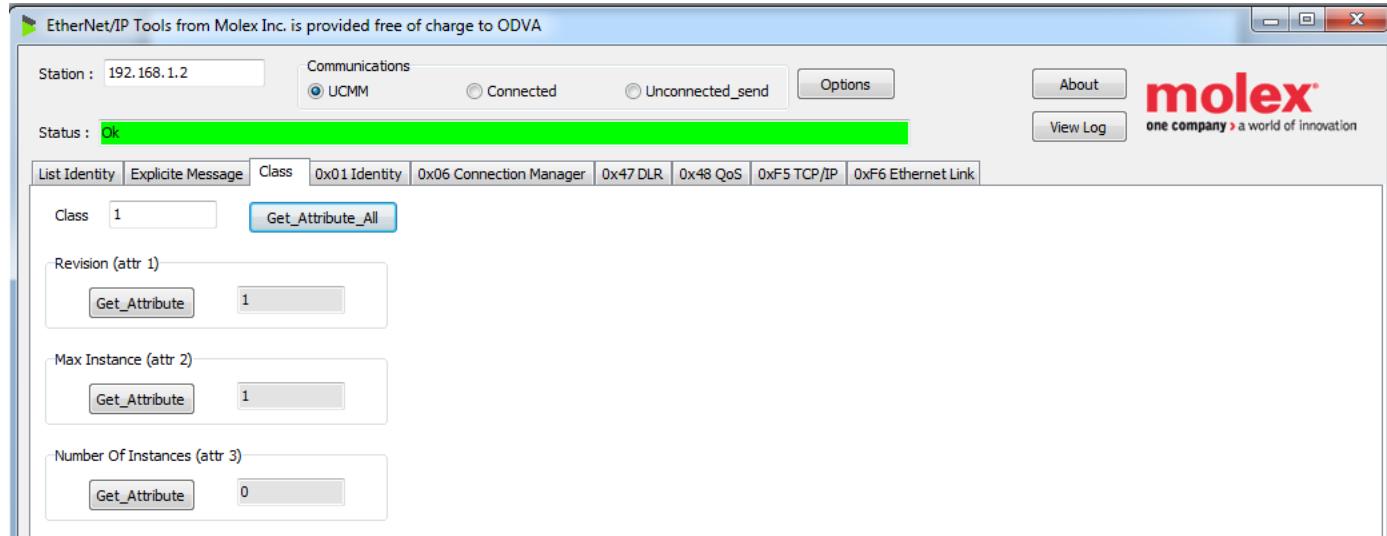


DUT is detected properly.

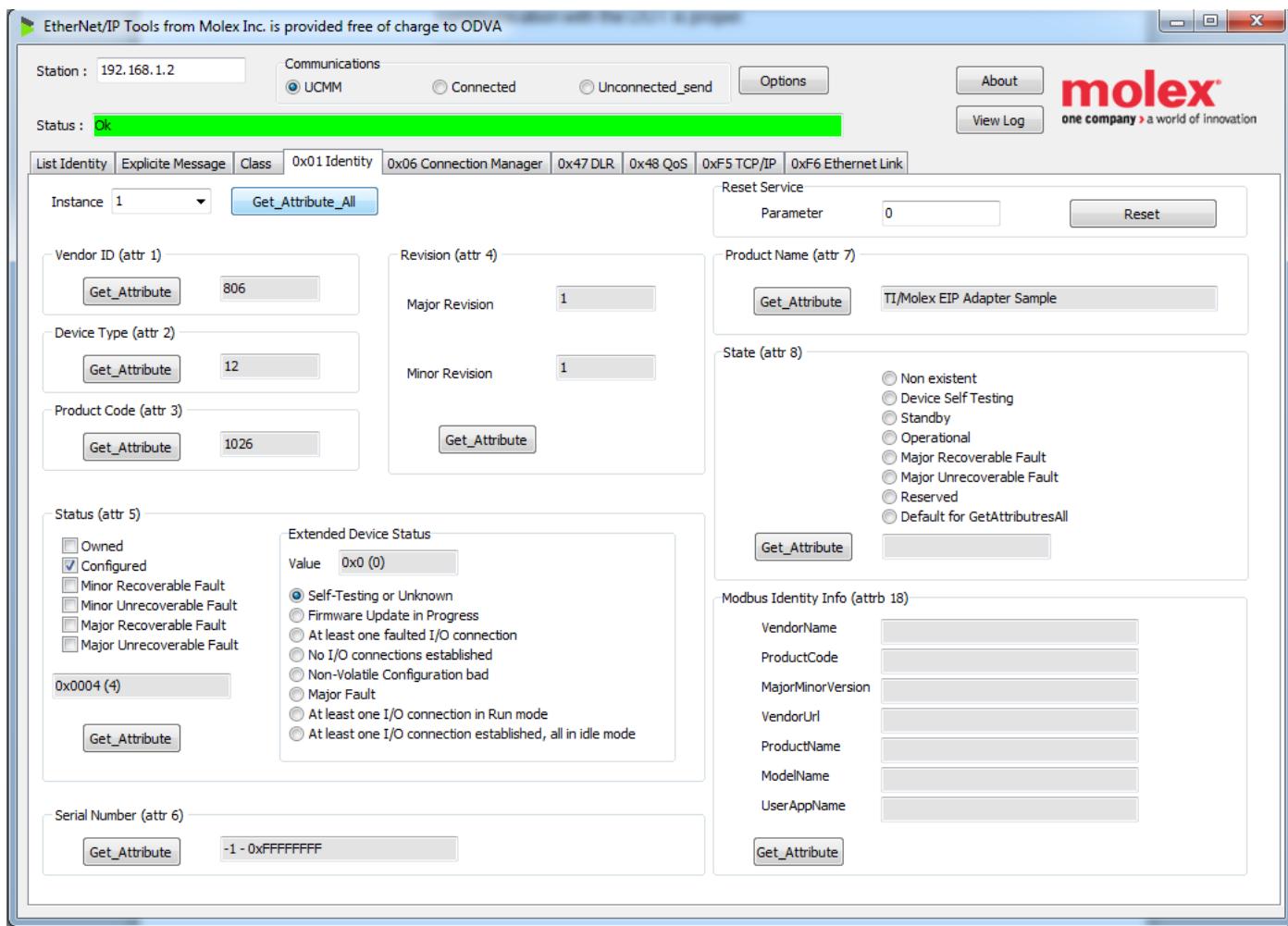
(iii) Now we can communicate with the DUT using EtherNet/IP Request-Response from different tabs in this tool. Explicite Message – This is used for verifying explicit Request-Response communication between this tool and the DUT. Status will be highlighted with yellow indicating there is/are unsupported services.

*Explicit Message Window*

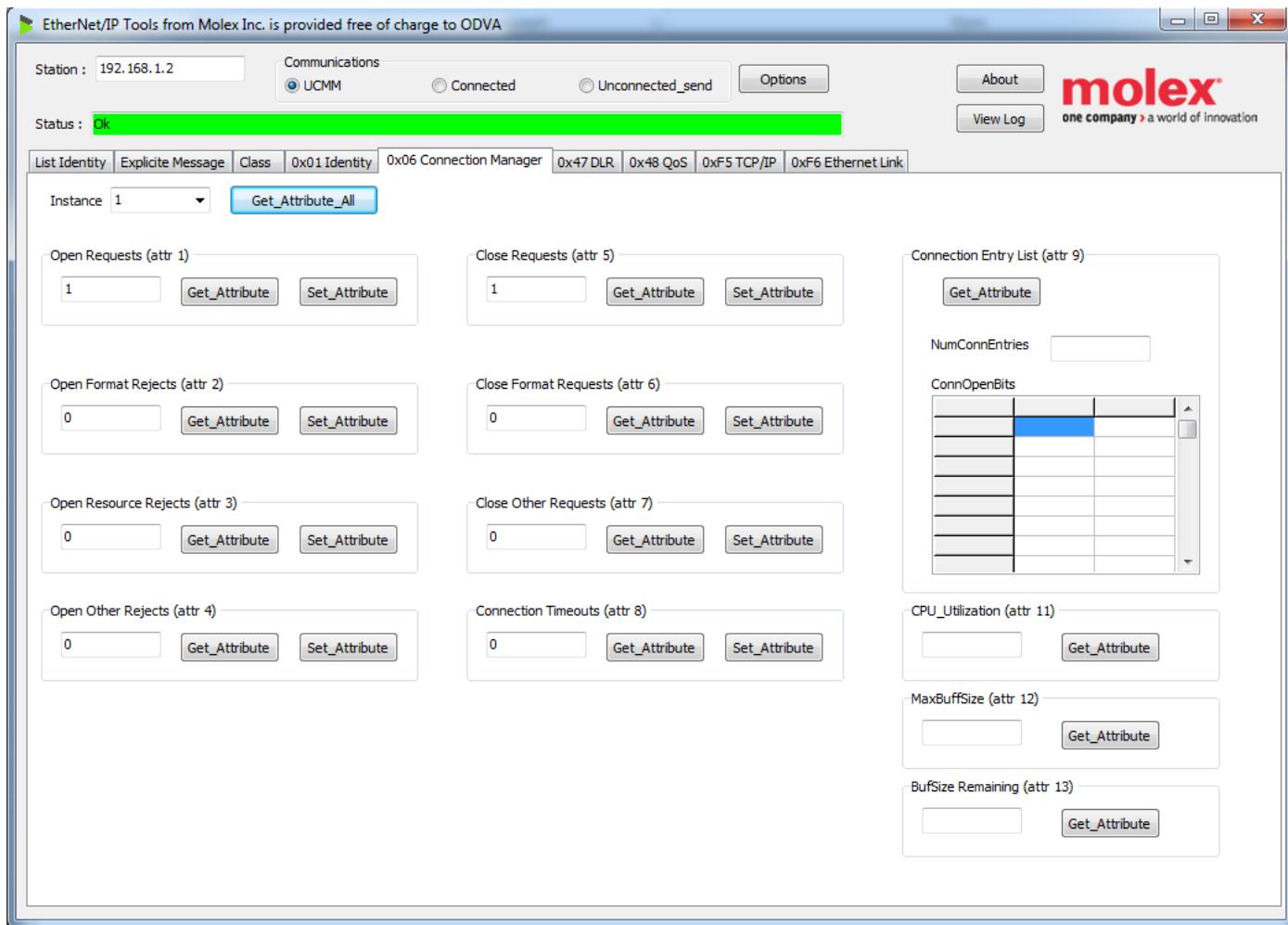
Class – We can read the Revision, Maximum instances and number of instances using 'Get\_Attribute\_All'. Use Get\_Attribute to read individual parameters. Status will be 'OK' highlighted with Green when the communication with the DUT is proper.

*Class Object Window*

0x01 Identity – Use Get\_Attribute\_All to read all the Identity related information from the EtherNet/IP Adapter. All the supported parameters will be shown in the tool. Get\_Attribute will read individual parameters also. Unsupported fields will be read and filled as blank.

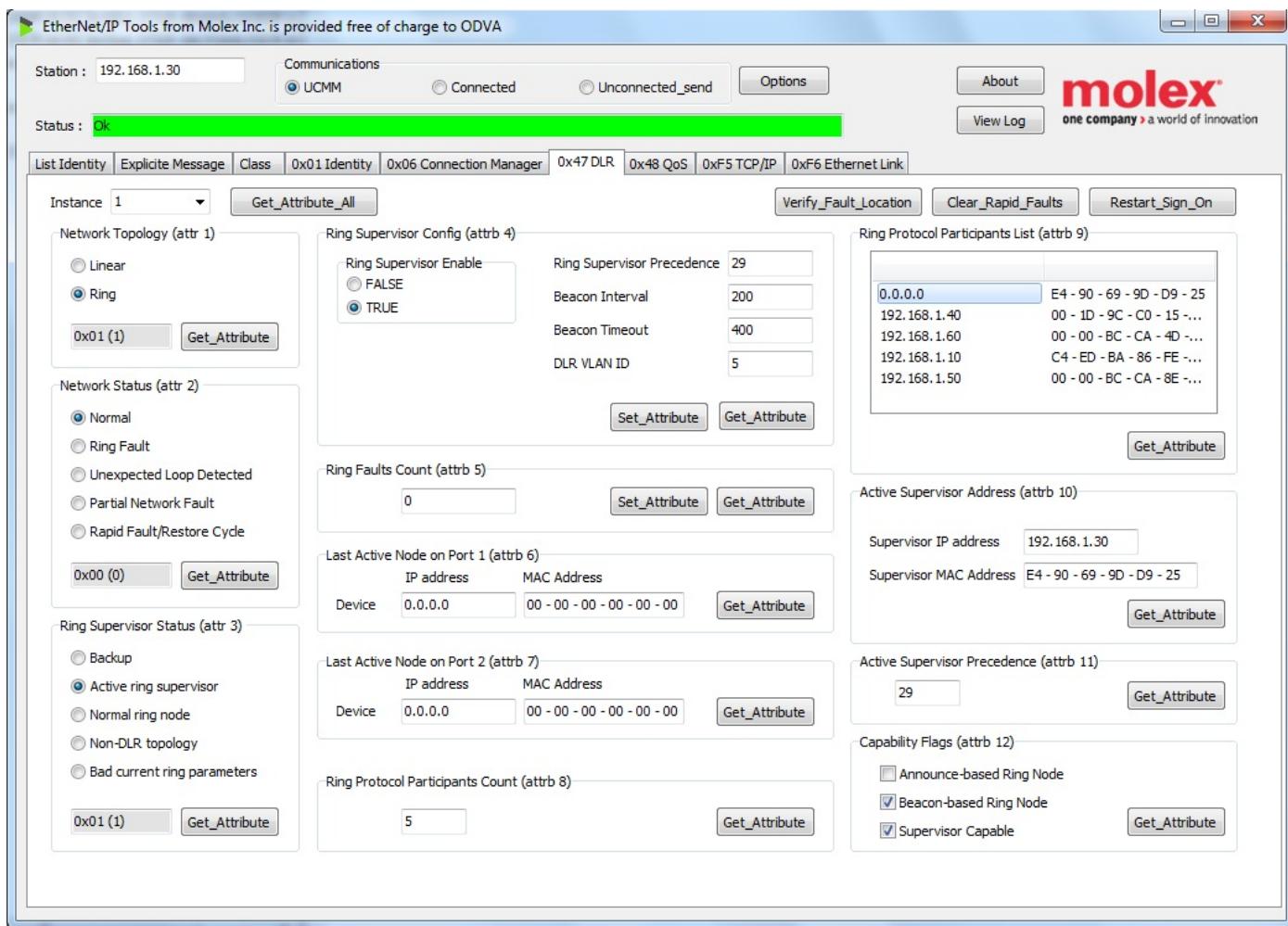
*0x01 Identity Object Window*

0x06 Connection Manager – All the supported Connection Manager related parameters will be read and displayed upon a Get\_Attribute\_All request from the tool.



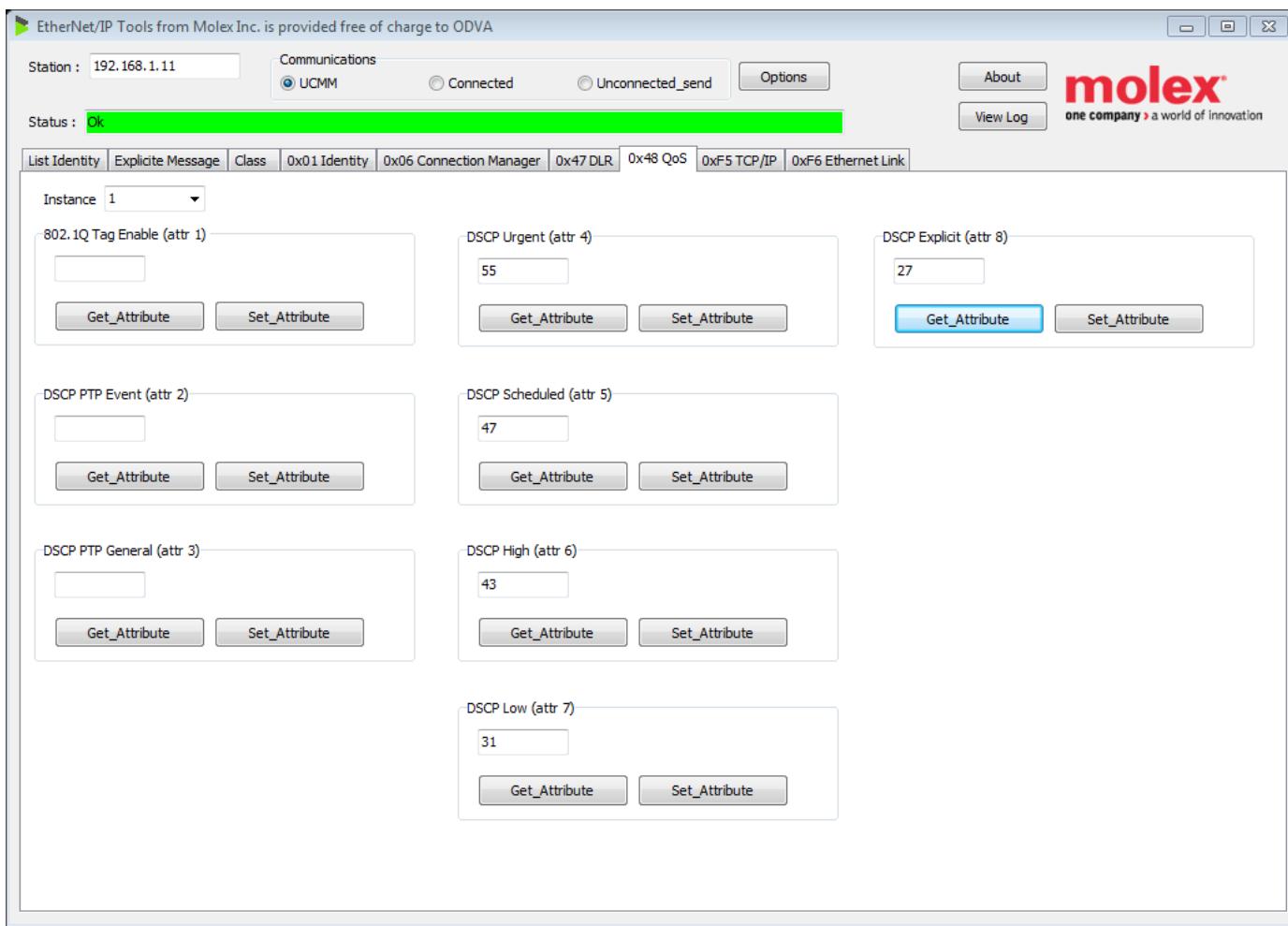
0x06 Connection Manager Object Window

0x47 DLR – Device Level Ring (DLR) object attributes are queried and displayed.



0x47 DLR Object Window

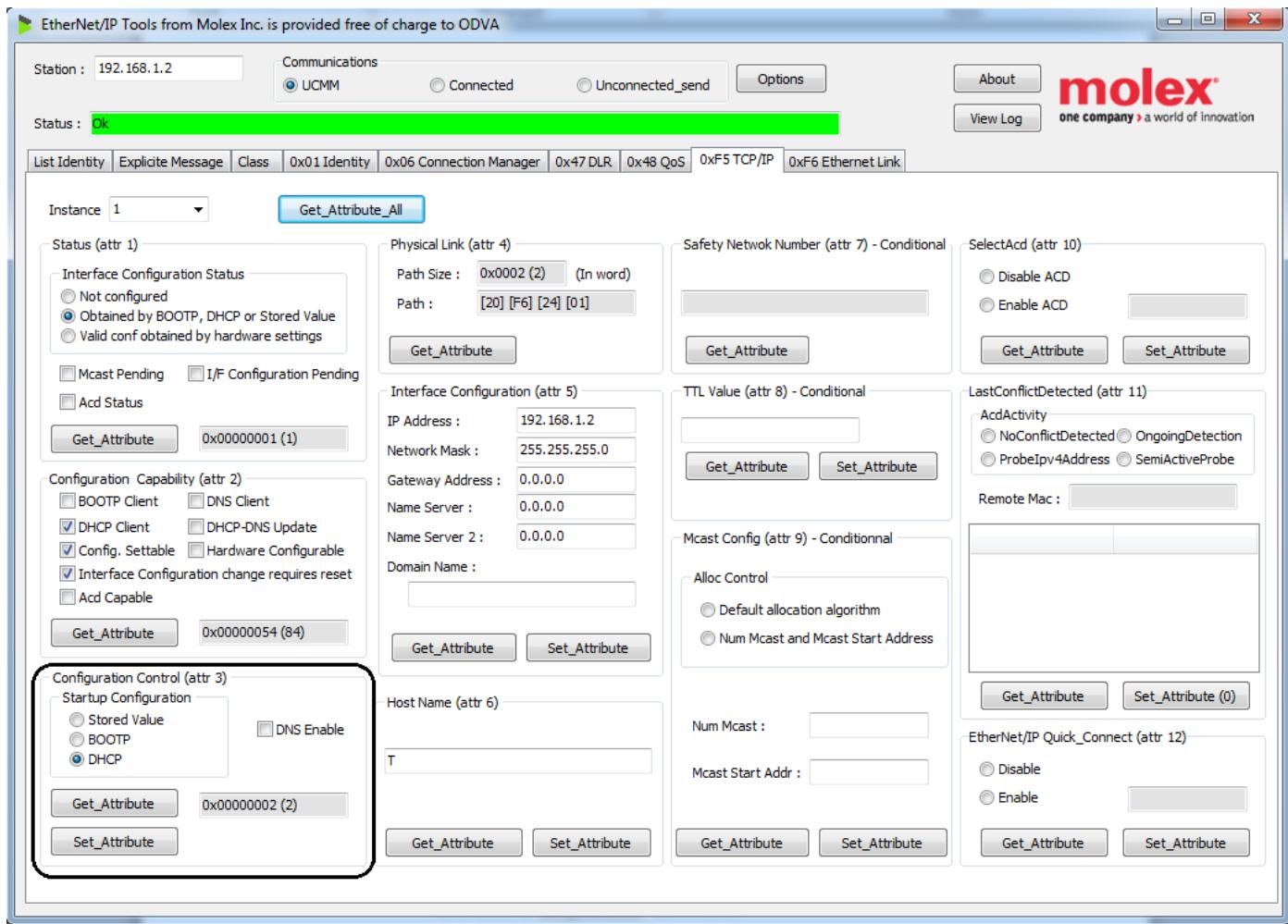
0x48 QoS – QoS object attributes values are displayed.



0x48 QOS Object Window

### TCP/IP Object

This object is used to read/write DUT interface settings and their configurations. The Configuration Control Attribute (Attr. 3) will show whether the DUT uses permanently stored IP or an IP address assigned via DHCP.



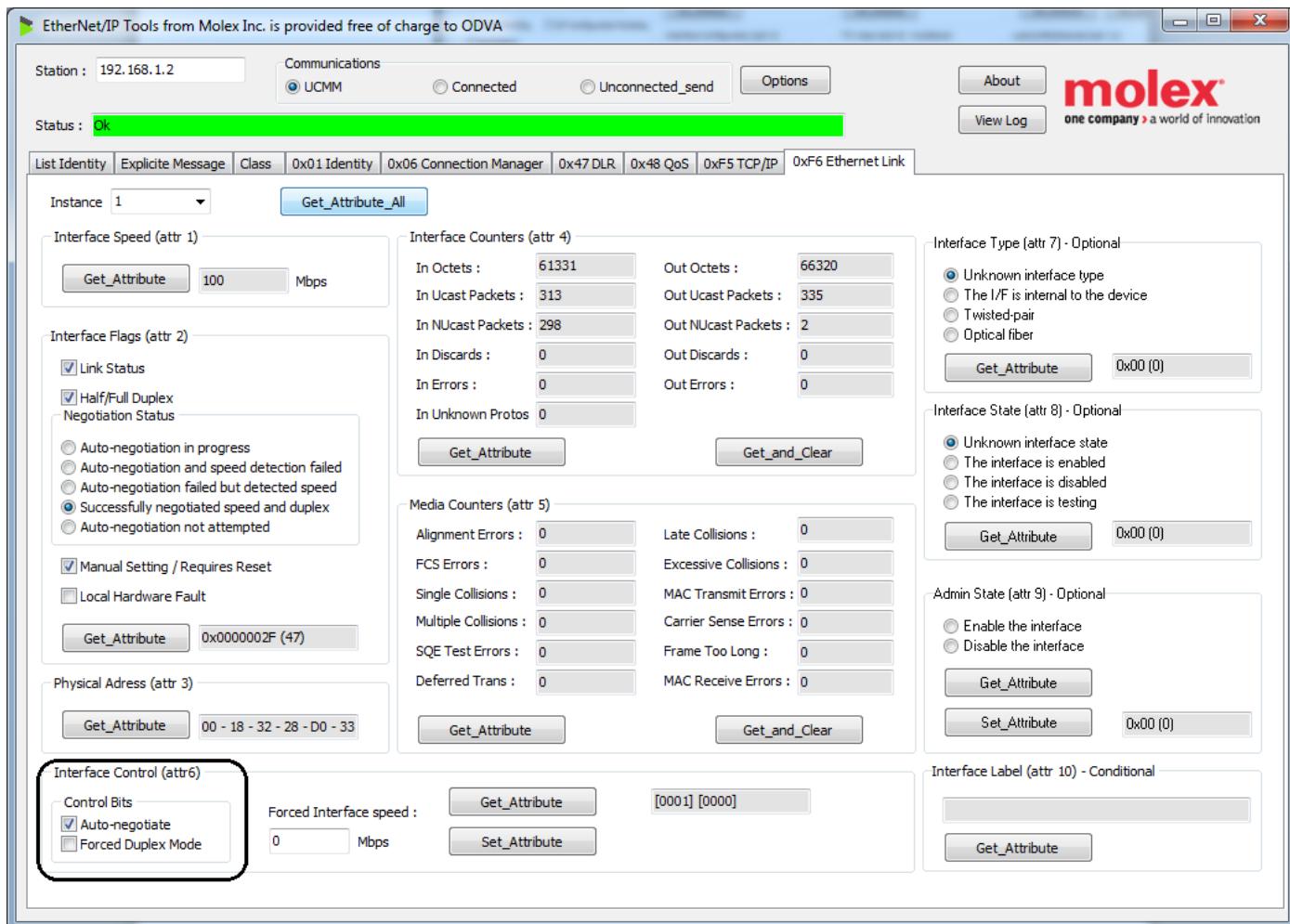
#### 0xF5 TCP/IP Object Window

**NOTE:** To make the IP Address permanently stored in the DUT memory, select 'Stored value' and issue a 'Set\_Attribute' Command. From the next reboot onwards, the DUT will use the permanently stored IP address (No need to assign IP from the DHCP server).

If we want to assign a different IP to the DUT, first change the Configuration Control to DHCP from Stored Value and issue a 'Set\_Attribute' command. Upon restart, the DUT will again wait for the IP address to be assigned from DHCP server.

#### Ethernet Link Object

Issue a Get\_Attribute\_All command and all the supported parameters are read and displayed. The Interface Control attribute (Attr. 6) will show the speed and duplexity of the device.



0xF6 Ethernet Link object Window

**NOTE:** Two instances of Link object can be tested by selecting option 1 and 2 in the instance drop down box.

**NOTE:** When we want to force the DUT speed to 10/100 Mbps or Auto-Negotiate mode, configure this attribute properly and issue a 'Set\_Attribute' command from the tool.

**Interface Control (attr6)**

Control Bits	<input type="checkbox"/> Auto-negotiate <input checked="" type="checkbox"/> Forced Duplex Mode	Forced Interface speed :	<input type="button" value="Get_Attribute"/>	[0001] [0000]
		0 Mbps	<input type="button" value="Set_Attribute"/>	

Forcing the DUT to 100 Mbps mode'

**Interface Control (attr6)**

Control Bits	<input checked="" type="checkbox"/> Auto-negotiate <input type="checkbox"/> Forced Duplex Mode	Forced Interface speed :	<input type="button" value="Get_Attribute"/>	
		0 Mbps	<input type="button" value="Set_Attribute"/>	

Forcing the DUT to Auto negotiate mode

**Interface Control (attr6)**

Control Bits	<input type="checkbox"/> Auto-negotiate <input checked="" type="checkbox"/> Forced Duplex Mode	Forced Interface speed :	<input type="button" value="Get_Attribute"/>	
		10 Mbps	<input type="button" value="Set_Attribute"/>	

Forcing the DUT to 10 Mbps mode

## Device Performance Testing

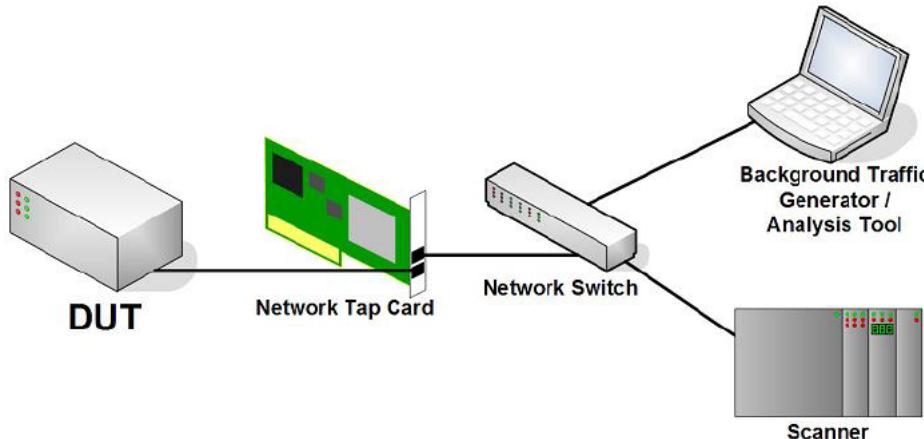
This test is intended to verify that the DUT behaves is capable of handling network traffic under possible conditions that may occur on a plant-floor EtherNet/IP network. The performance tests conducted during the PlugFests are representative of some situations that may occur on a plant floor, but are not intended to be exhaustive.

#### 1. Tools Used:

- Software scanner tool like Pyramid Solutions' EIP Scan or Hardware scanner like Rockwell PLC
- Molex Plugfest Performance Packet Generator to generate the background traffic.
- Hilscher netAnalyzer PCI-card (NANL-C500-RE) to capture the communication traffic
- NIST Industrial Ethernet Network Performance (IENetP) to analyze the traffic captured by netAnalyzer.

#### 2. Test Setup:

When setting up the performance test system, either hardware or software scanner can be used. If a software scanner is chosen, it is important that the scanner software not be run on the same computer as the background traffic generator. The Molex traffic generator and the EIPScan software have been can load down a computer to the point of causing connection issues with the DUT. The performance analysis tool can run on either the computer housing the network tap card or the computer used for the background traffic generator. If using a software scanner, the performance analysis tool should not be run on that computer due to the process intensive analysis that will run extremely slow while the scanner is running.



""Device performance Testing- Test Setup""

#### 3. Performance Test Categories:

There are 5 different types of performance measurements based on the presence or intensity of the background traffic applied. They are as follows-

##### 3.1 Baseline Performance Test

- No Background Traffic
- The mean of the device's measured packet interval (MPI) should be within 10% of the device's API.
- The standard deviation jitter of the device's MPI should be within 10% of the mean MPI.
- The maximum jitter of the device's MPI should be within 50% of the mean MPI.

##### 3.2 Steady-State Managed Background Traffic Test

- Generate Steady-State Managed Background Traffic using Molex Plugfest Performance Packet Generator
- The mean of the device's measured packet interval (MPI) should be within 10% of the device's API.
- The standard deviation jitter of the device's MPI should be within 25% of the mean MPI.
- The maximum jitter of the device's MPI should be within 100% of the mean MPI.

##### 3.3 Steady-State Unmanaged Background Traffic Test

- Generate Steady-State Unmanaged Background Traffic using Molex Plugfest Performance Packet Generator
- The mean of the device's measured packet interval (MPI) should be within 10% of the device's API.
- The standard deviation jitter of the device's MPI should be within 25% of the mean MPI.
- The maximum jitter of the device's MPI should be within 100% of the mean MPI.

##### 3.4 Burst Managed Background Traffic Test

- Generate Burst Managed Background Traffic using Molex Plugfest Performance Packet Generator
- The mean of the device's measured packet interval (MPI) should be within 10% of the device's API.
- The maximum jitter of the device's MPI should be within 400% of the mean MPI.
- Generate Burst Unmanaged Background Traffic using Molex Plugfest Performance Packet Generator
- The mean of the device's measured packet interval (MPI) should be within 10% of the device's API.
- The maximum jitter of the device's MPI should be within 400% of the mean MPI.

#### 4. Common Test Procedure:

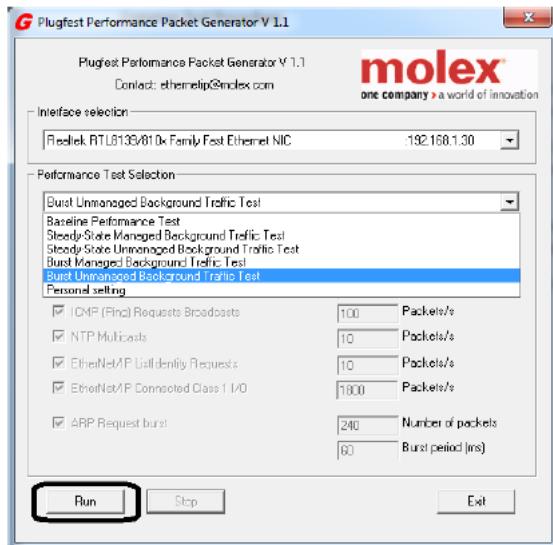
##### 4.1 Start capturing traffic using the network tap card.

##### 4.2 Establish a Connected Class 1 I/O connection from the scanner to the DUT at the minimum RPI (fastest RPI speed) that the DUT supports and at the desired connection size.

**NOTE:** The connection multiplier should be set as high as possible (512x multiplier recommended) to allow the analysis software to measure potential 4x multiplier connection timeouts without the scanner actually shutting down and reestablishing the connection.

4.3 Maintain the connection with the DUT for a minimum time (For baseline test, minimum time is 60 seconds. For all other categories, minimum 30 seconds will be enough)

4.4 Based on the test category, Generate the corresponding background traffic using 'Performance Packet Generator' tool.

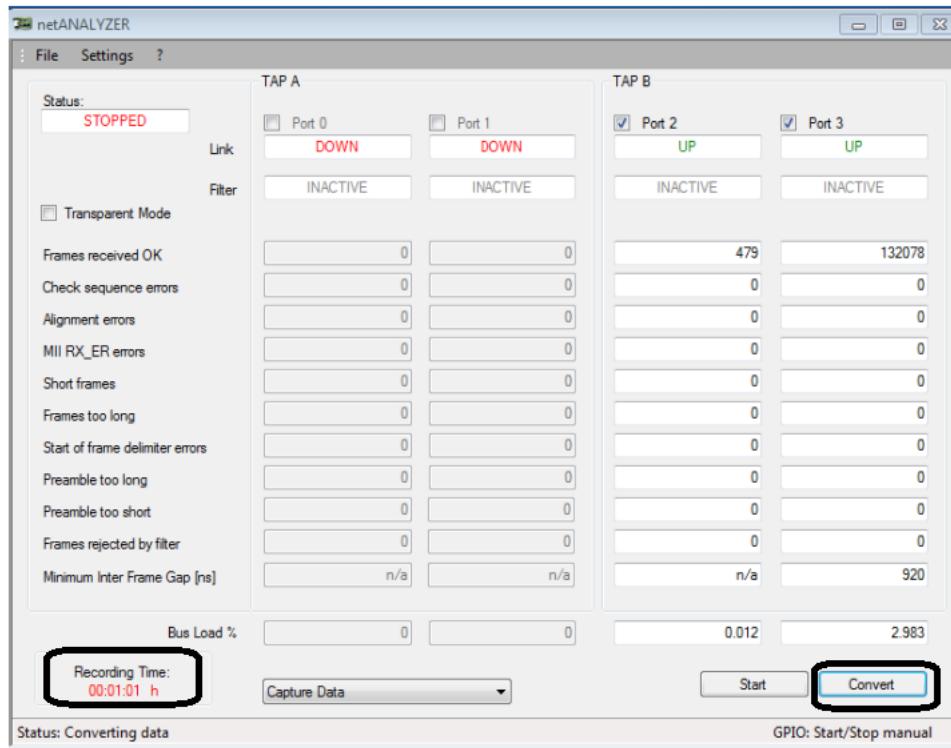


""Generating background traffic for performance test""

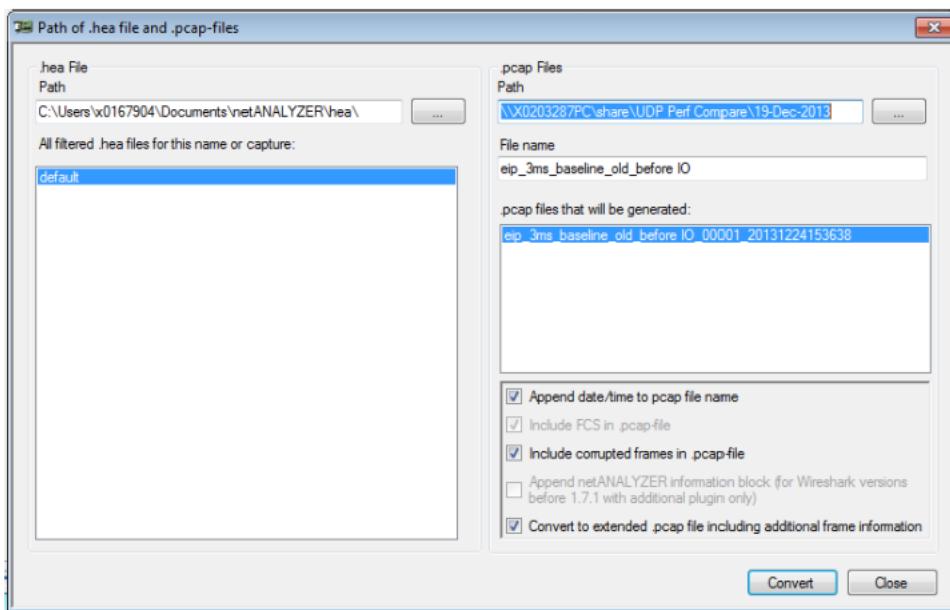
**NOTE:** Capturing the traffic should be started first and then start the I/O connection and background traffic at the same time.

##### 5. Capturing the traffic:

Once the traffic is captured for the required time period, convert the capture into a .peap file so that the file can be used for further analysis.



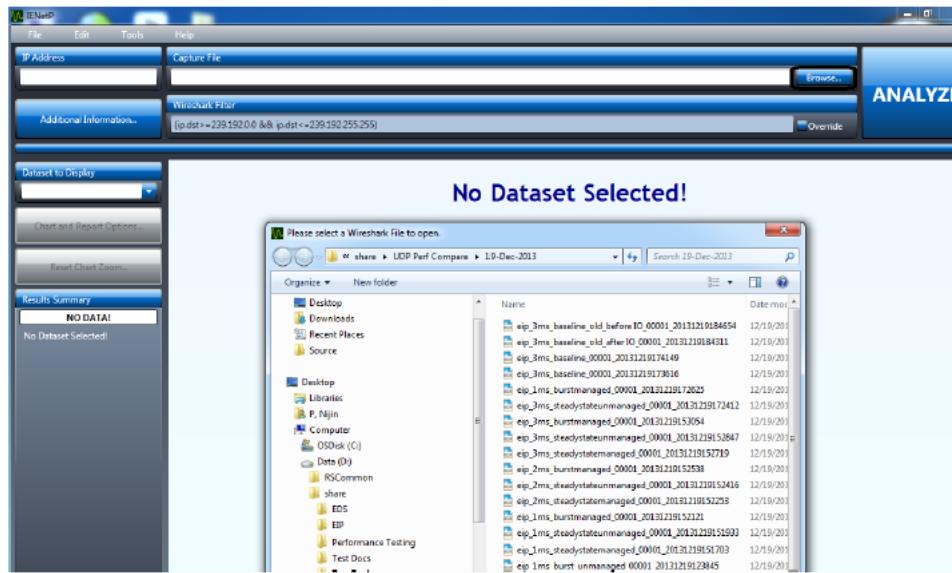
""Capturing the traffic""



""Saving the captured traffic""

## 6. Analyzing the captured traffic:

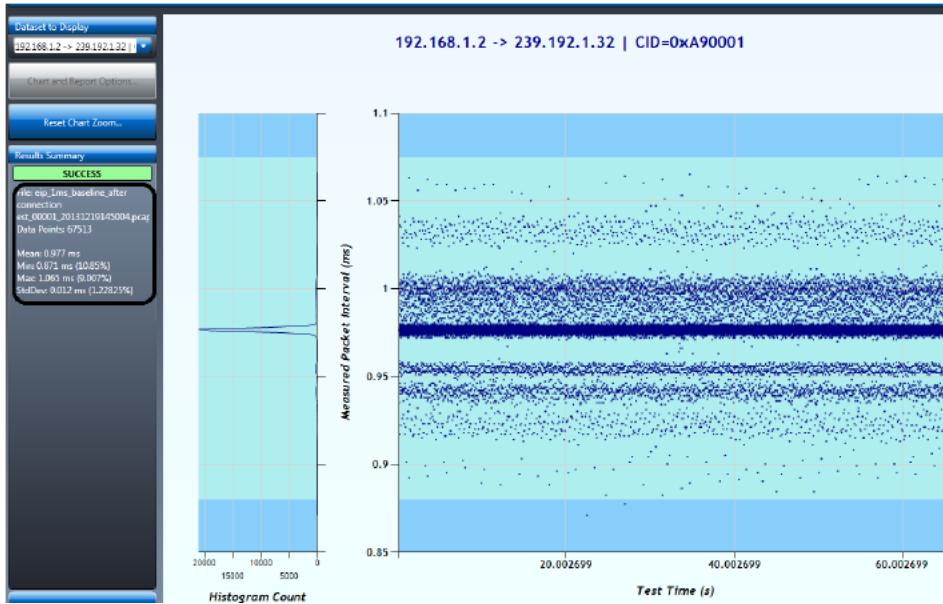
6.1 Launch the performance analysis tool. Load the capture file that needs to be analyzed using Browse option (or File->Open)



"" Loading the capture file to the performance analysis tool""

6.2 Load the proper file and click on 'ANALYZE' button.

The tool will analyze the capture and the performance data will be displayed in the graphical window. We can see the percentage of maximum jitter and the standard deviation jitter. Compare these data against the allowed percentage limit for a particular test.



""Performance analysis tool – Output window""

**NOTE:** If any of the performance data is not met with the specified/allowed limit in particular the test category, increase the RPI used for establishing the I/O connection and re-run the tests.

#### Communication between EtherNet/IP Adapter and Scanner devices

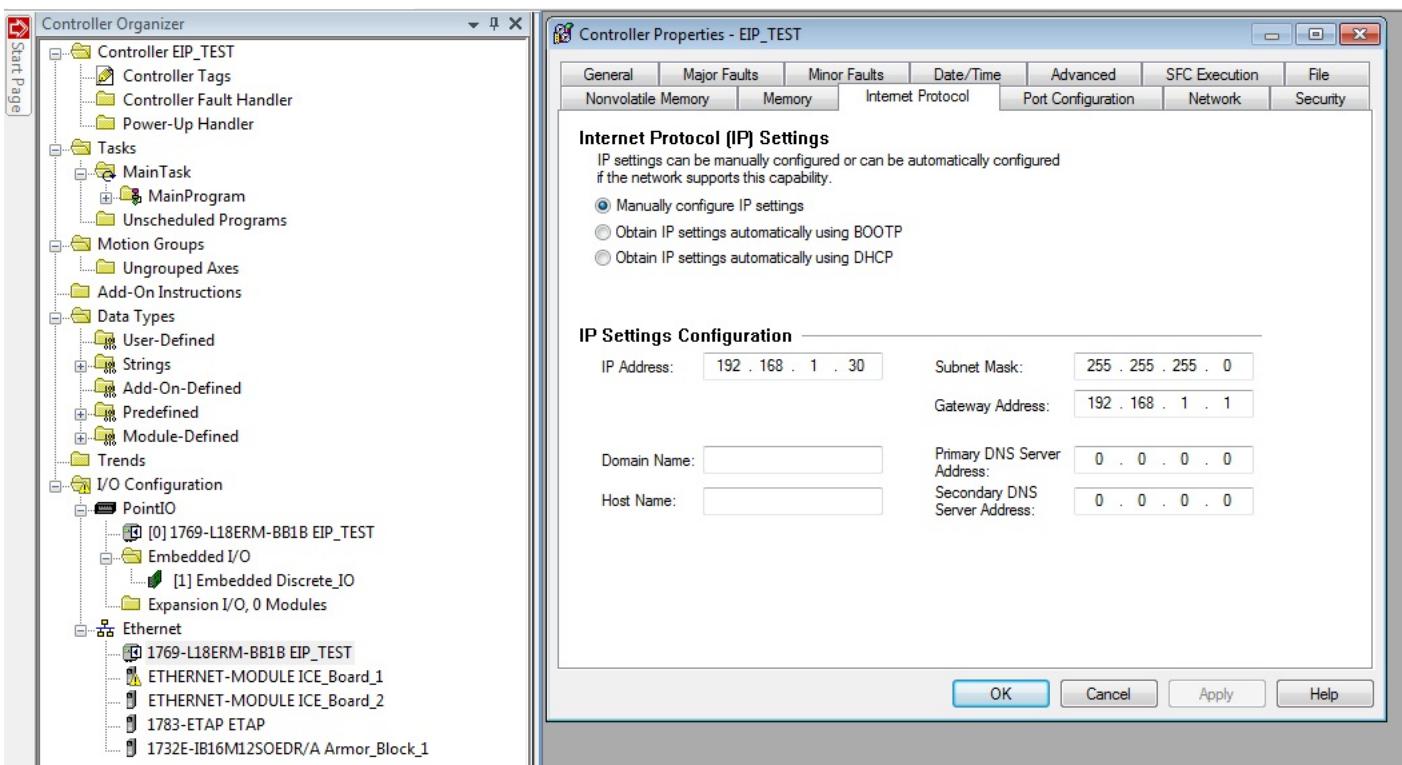
The EtherNet/IP adapter can be connected and communicated to any standard scanner device by establishing an Exclusive Owner (E/O) Connection. The connection parameters for E/O connection are shown below-

Description	Assembly Instance	Data Size (Bytes)
T->O	101	1
O->T	102	1
Configuration	103	0

*E/O Connection parameters*

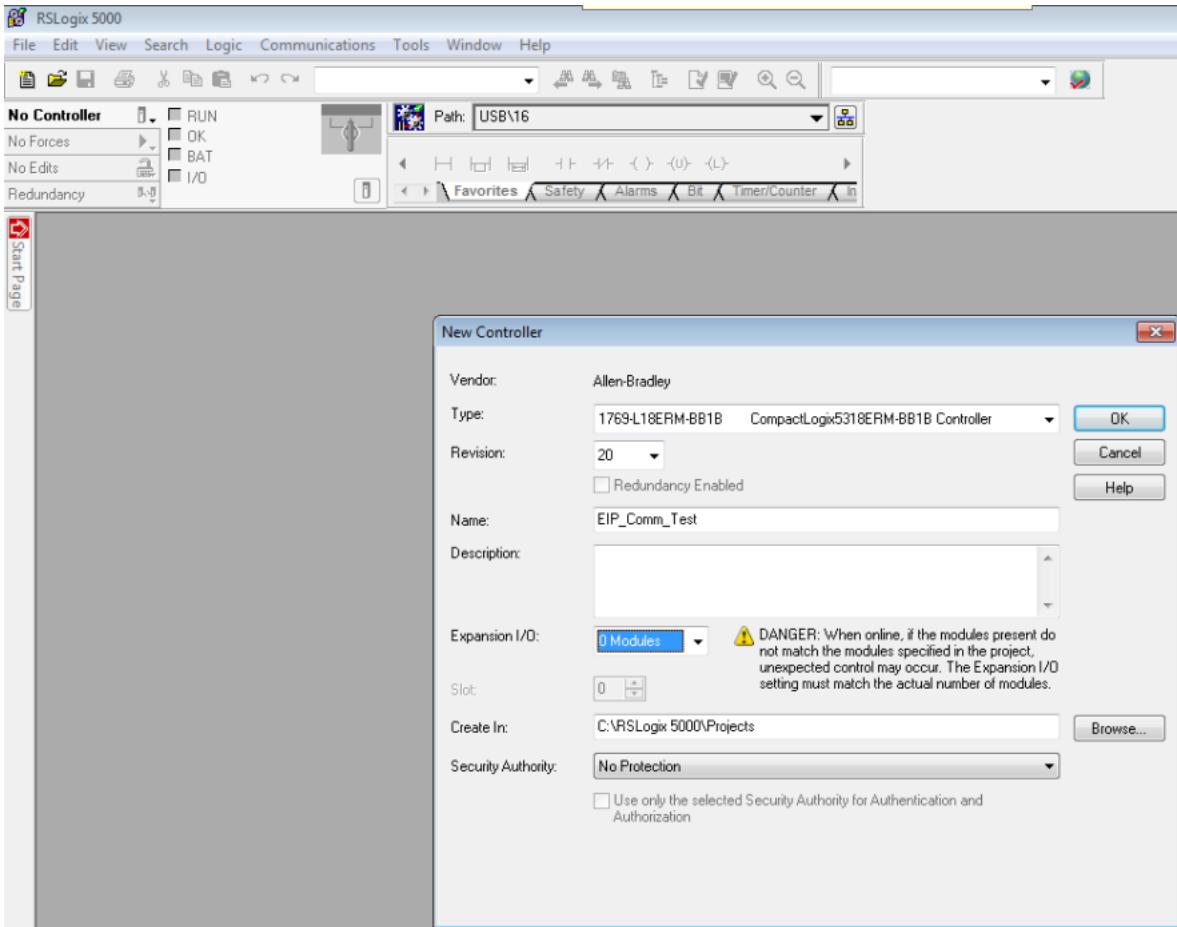
**I/O Messages Exchange with Rockwell PLC** The Rockwell EtherNet/IP PLC should be setup properly and the PLC can be configured via RSLogix 5000 software (version V20.01.00 and above).

**PLC IP Configuration:** The PLC IP can be configured to the desired subnet from RSlogix5000. Right click on the 1769 PLC -> Properties-> Internet Protocol Tab. Here we can select the appropriate IP settings.



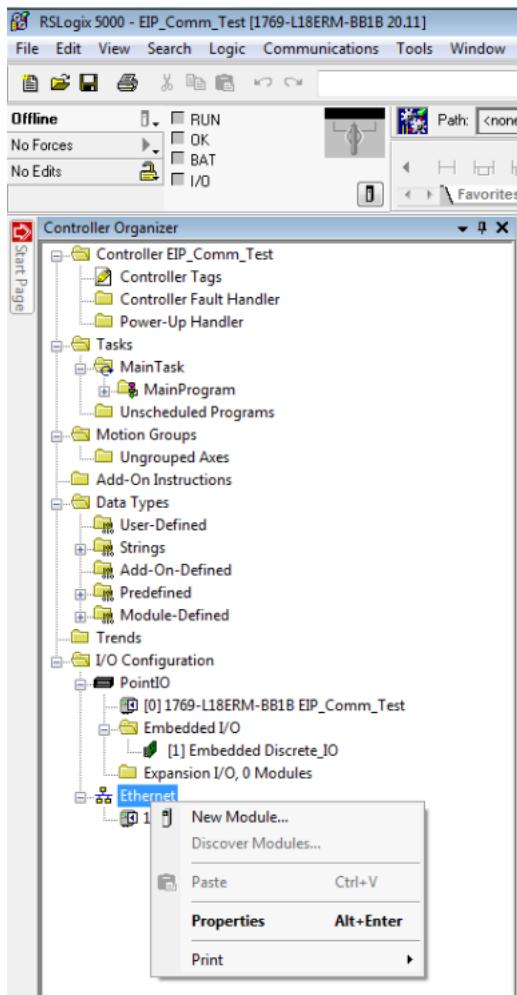
#### Configuring the IP address for PLC

The steps for establishing I/O connection between the PLC and DUT are as follows - (i) Create a new project (File -> New) and give a project name. Select the Expansion I/O as o Modules



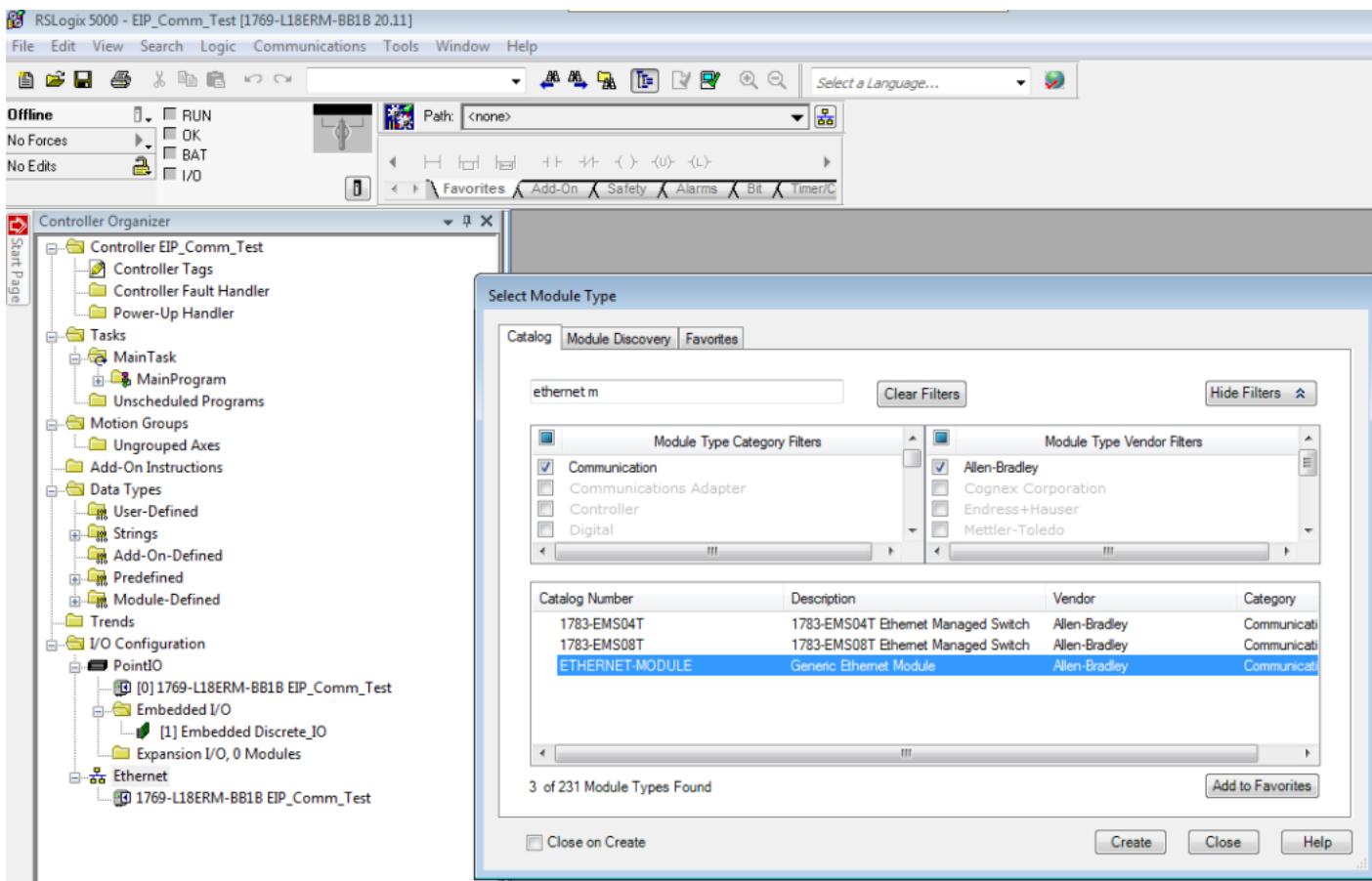
#### Starting new project in RSLogix 5000

(ii) Once the project is created, click on Ethernet and select 'New Module'



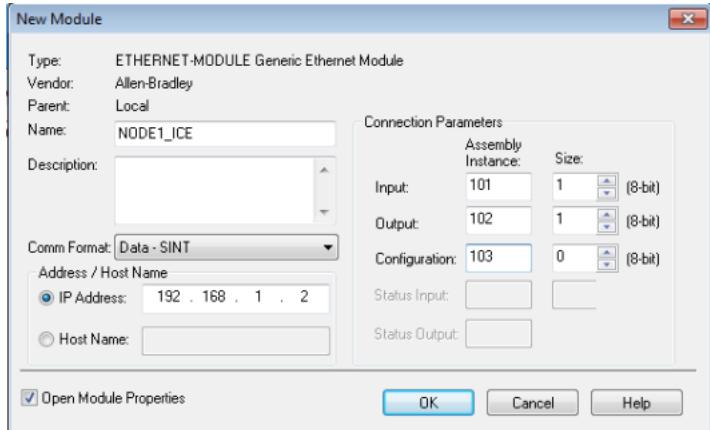
Adding a new module to the project

(iii) In the 'Select Module Type' Window, select Generic Ethernet Module by searching from the list and click on Create.



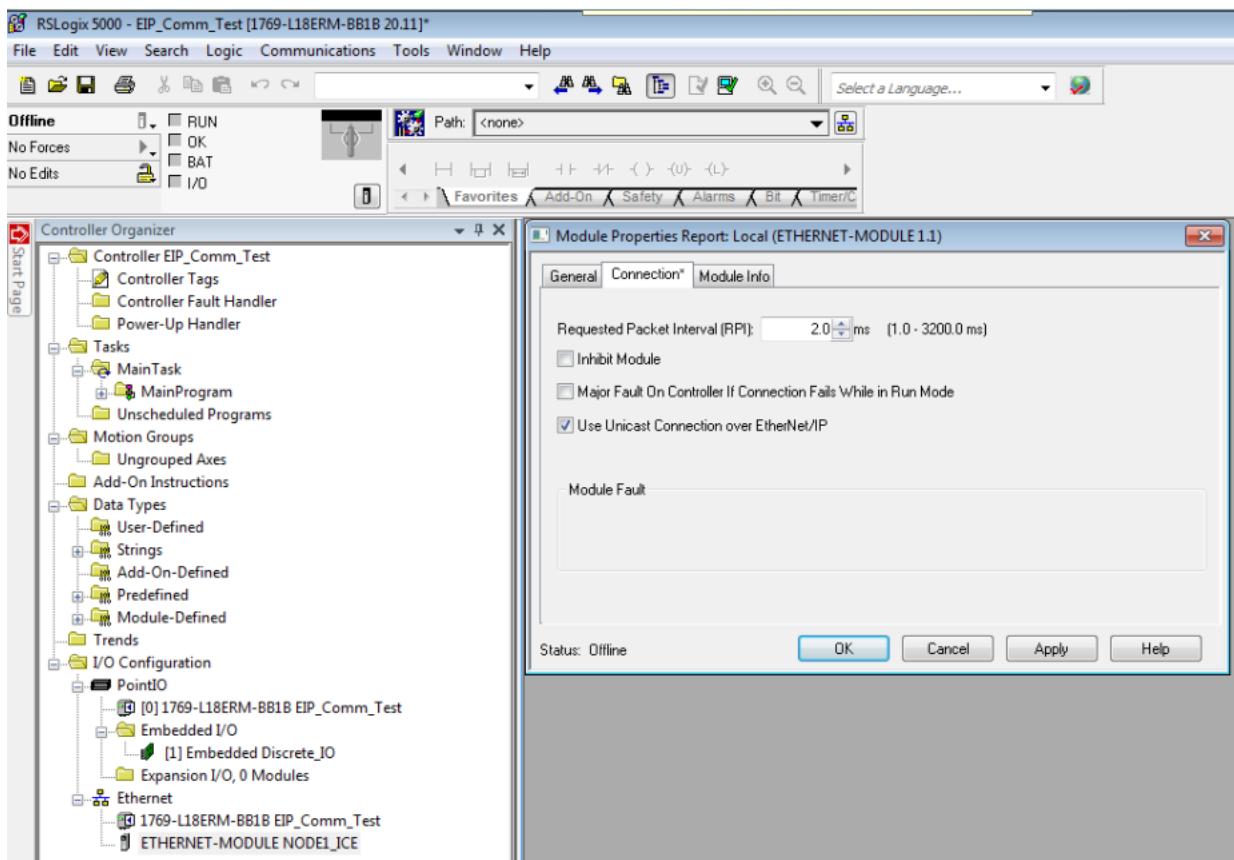
#### Selecting the module type

(iv) In the 'New Module' window select Comm Format as 'Data-SINT', IP address as DUT IP and Connection parameters as shown below-



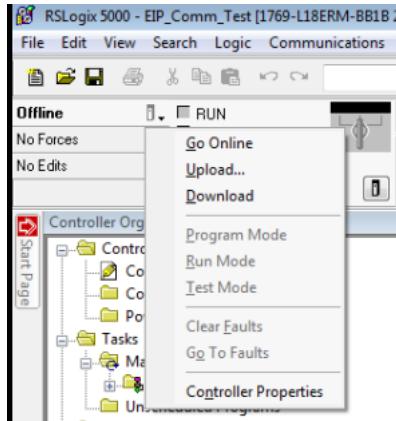
#### New module configuration

(v) In the 'Connection Tab' configure the required Requested Packet Interval (RPI) which completes the addition of a new module into the project



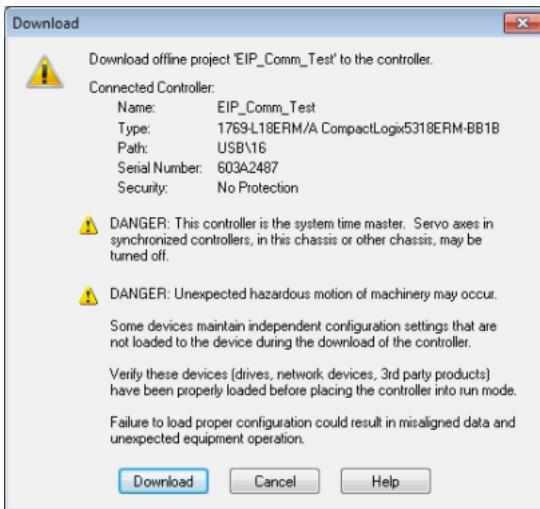
#### Configuring the connection parameters

(vi) Click on the drop-down box near the Offline mark and select 'Download' to download the configuration



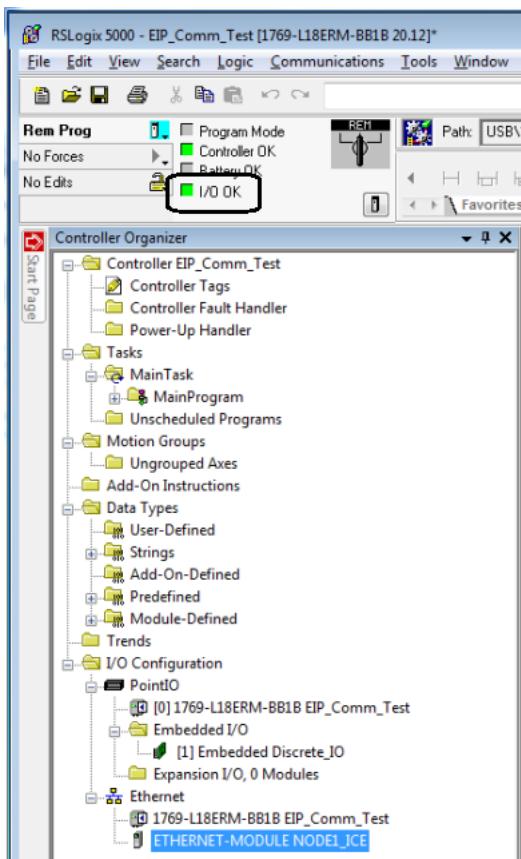
#### Downloading the configuration

(vii) A confirmation for downloading the configuration will be there. Proceed by clicking OK



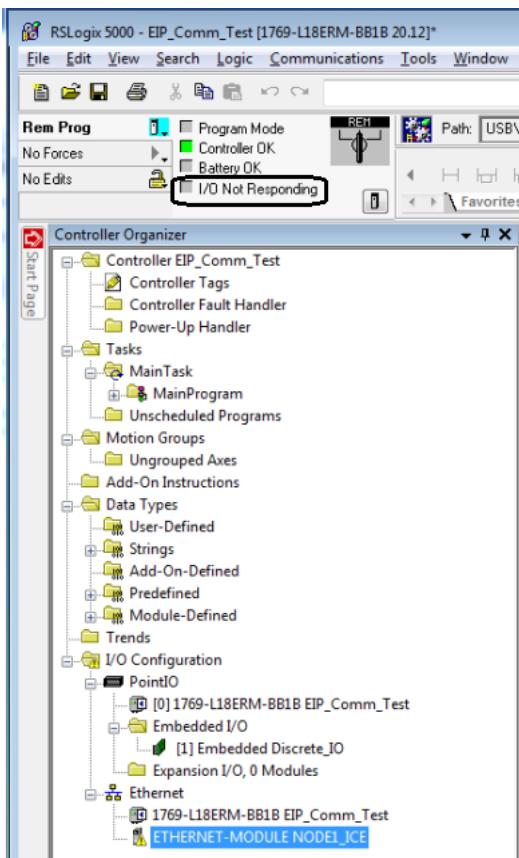
- Confirmation while downloading the configuration

(vii) Now the I/O connection will be established at the configured RPI. Successful I/O connection will be indicated by 'I/O OK' as shown below-



Successful I/O message exchange shown as 'I/O OK'

(ix) When there is problem with the I/O connection (Communication timed out, Unsupported RPI etc...), PLC will show 'I/O Not Responding'



Communication problem due to unsupported RPI shown as 'I/O Not Responding'

	Keystone=	C2000=	MSP430=	OMAP35x=	OMAPL1=	MAVRK=	
{		For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum	DaVinci=For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.	MSP430=For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.	OMAP35x=For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.	OMAPL1=For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.	MAVRK=For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.
1. switchcategory:MultiCore=		For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum	DaVinci=For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.	MSP430=For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.	OMAP35x=For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.	OMAPL1=For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.	MAVRK=For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.
■ For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum		■ For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum	■ For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.	■ For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.	■ For technical support on OMAP35x please post your questions on The OMAP Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.	■ For technical support on OMAPL1 please post your questions on The OMAP Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.	■ For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article ICSS EIP Adapter Developer Guide here.

## Links



[Amplifiers & Linear](#)

[Audio](#)

[Broadband RF/IF & Digital Radio](#)

[Clocks & Timers](#)

[Data Converters](#)

[DLP & MEMS](#)

[High-Reliability](#)

[Interface](#)

[Logic](#)

[Power Management](#)

[Processors](#)

- [ARM Processors](#)
- [Digital Signal Processors \(DSP\)](#)
- [Microcontrollers \(MCU\)](#)
- [OMAP Applications Processors](#)

[Switches & Multiplexers](#)

[Temperature Sensors & Control ICs](#)

[Wireless Connectivity](#)

Retrieved from "[https://processors.wiki.ti.com/index.php?title=ICSS\\_EIP\\_Adapter\\_Developer\\_Guide&oldid=236360](https://processors.wiki.ti.com/index.php?title=ICSS_EIP_Adapter_Developer_Guide&oldid=236360)"

This page was last edited on 8 March 2019, at 12:06.

Content is available under [Creative Commons Attribution-ShareAlike](#) unless otherwise noted.