Ethernet Peripheral Manual

https://www.nodeudesign.com
sales@nodeudesign.com

1. Hardware

1.1. Pinouts

- Add graphic showing pinout

1.1. Electrical Characteristics

Absolute Maximum Ratings	
	Max
Voltage at any Output terminal	45 V
V(ESD) Electrostatic discharge	16000V
Temperature (Storage & Operating)	-40℃ - 80℃
Operating Temperature	-20℃ - 80℃

Recommended Maximum Ratings			
	Min	Typical	Max
Operating Voltage	7.5V	-	36V
Output Voltage	0	-	36V

2. Overview

The Ethernet peripheral provides a wide array of methods for transferring data from the control system to the outside world and to receive input from auxiliary systems.

2.1. Hardware

The Ethernet Peripheral offloads communication tasks from the Brain, allowing the Brain to focus on control tasks. The peripheral supports auto MDI-X to simplify cabling procedures and automatic negotiation of 10/100 speeds.

A microSD card is supported for local file storage.

The Ethernet Peripheral provides one Aux I/O line. TODO: What can we do with this line?

2.2. Software

The software stack on the Ethernet Peripheral supports a variety of communication protocols. Support is provided for protocols including UDP, TCP, DHCP Client, Web Server and MQTT Client.

The Ethernet Peripheral currently supports IPv4 addressing only, both via DHCP and static assignment.

Protocol configuration and use is done through the Brain interface.

3. Hardware Installation

3.1. Peripheral to Brain Connection

Locate the active port marked with an outline silhouette. Rotate to match with an open Brain port on the stack, then seat the connections. Please make a note of the port number. This will be required to finish wiring and will be needed to program the peripheral later.

3.2. Ethernet Wiring Connection

The ethernet cable pairs number 2 & 3 (Orange/OrangeWhite and Green/GreenWhite) must be connected to your terminal header for the selected port. Connect a solid color wire to connector 1 and the matching white wire to connector 2. Connect the other solid wire to connector 3 and the matching white wire to connector 4. The order of the pair connection does not matter due to the auto MDI-X feature of the peripheral.

Connect the other end of your ethernet cable to your switch.

4. Software Configuration

4.1. Imports

The following are the recommended imports to use the Ethernet Peripheral. All following examples will assume that these imports are used.

```
import NUD.System as sys

import NUD.Peripherals.Ethernet.Ethernet as Eth
import NUD.Peripherals.Ethernet.Values as EthV
import NUD.Peripherals.Ethernet.Types as EthT
```

4.2. Connecting to the Ethernet Peripheral

The following is an example of how to create an Ethernet Peripheral in the system. Please see the NUD Brain Manual for information on specifying the correct side and slot for the peripheral.

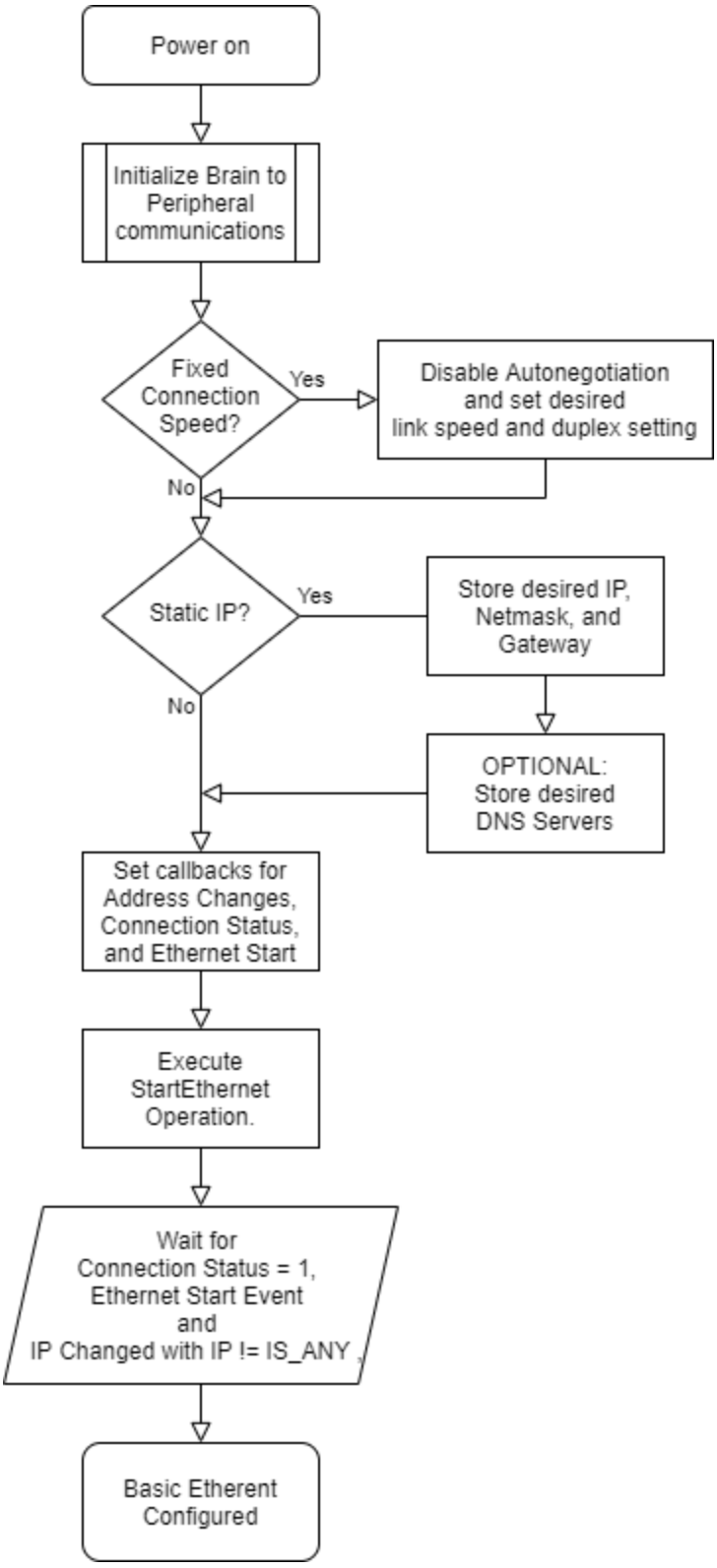
The Ethernet peripheral is complicated and has many user configurable parameters and cached state. While not strictly necessary, we recommend that your initialization process include requesting a reset of the Ethernet peripheral on Brain startup to avoid any synchronization issues.

```
nudSystem = sys.System( )
eth = Eth.Ethernet( )
nudSystem.attachPeripheral( eth, sys.SideSlot.SIDE_1_SLOT_B )
nudSystem.startAutoUpdate()
nudSystem.flushPeripheral( 1 )
```

All following examples will assume you have created an ethernet peripheral object named ‘eth’, properly attached it, started auto update, and flushed the peripheral.

4.3. Configuration Overview

There are many parameters that can be configured on the Ethernet interface, many of which should be set prior to initializing the ethernet connection. The following flow chart will help you initialize the peripheral to your desired settings. Setting individual parameters will be covered in their own sections.


Power on
Initialize Brain to Peripheral communications
Fixed Connection Speed?
Yes
Disable Autonegotiation and set desired link speed and duplex setting
No
Static IP?
Yes
Store desired IP, Netmask, and Gateway
No
OPTIONAL: Store desired DNS Servers
Set callbacks for Address Changes, Connection Status, and Ethernet Start
Execute StartEthernet Operation.
Wait for Connection Status = 1, Ethernet Start Event and IP Changed with IP != IS_ANY
Basic Etherent Configured

4.4. Autonegotiation, Link Speed, and Duplex Configuration

The Ethernet Peripheral supports link speeds of 10 or 100 mbps in either full or half duplex modes. By default the Ethernet Peripheral will perform autonegotiation on startup. Manual configuration is possible prior to the StartEthernet operation.

TODO

4.5. Initialization Event Callbacks

The Ethernet Peripheral provides several events to indicate the status of the device. The callbacks are Ethernet Cable Link Status, IP Changed event, and Ethernet Start status. Event callbacks need to be set for these events.

It is recommended that you ensure all states are good prior to performing any other operations on the Ethernet Peripheral. This can be accomplished by waiting on flags set by your event callbacks.

Example:

```
while not (isEthernetStarted and isCableConnected and isIpValid):
      time.sleep_ms(100)
```

Ethernet Start

Ethernet Start status indicates that the basic ethernet initialization process is finished. No configuration changes or protocol commands should be sent to the peripheral between the StartEthernet Operation and receiving the StartEthernet event.

Example:

```
isEthernetStarted = False

def ethernetStartCallback(peripheral, status):
      print("Etherenet Start Callback");
      global isEthernetStarted
      isEthernetStarted = True

eth.events.StartEthernet.setCallback( ethernetStartCallback )
```

Cable Link

Cable Link status reports if the cable is plugged into the ethernet port. This status can be queried through the ConnectionStatus property or obtained on change through the ConnectionStatus event.

Example:

```
isCableConnected = False

def connectionCallback( peripheral, status):
      print("Cable Connection Status Callback: " + str(status))
      global isCableConnected
      isCableConnected = bool(status)

eth.events.ConnectionStatus.setCallback( connectionCallback )
```

IP Addresses

The current IP Address configuration (IP Address, Netmask, and Gateway) can be queried through properties or monitored for changes through the PeripheralAddresses event. When a valid IP is sent, the value of the ip will not be IS_ANY. This can be tested by calling the isAny() function on the IP address.

Example:

```
isIpValid = False

def ipChangedCallback( peripheral, ipAddresses):
      print("IP Change Callback:" + str(ipAddresses))
      global isIpValid
      isIpValid = not ipAddresses.ip.isAny()
      print("IP Changed Valid IP?:" + str(isIpValid))

eth0.events.PeripheralAddresses.setCallback( ipChangedCallback )
```

4.6. Ensuring cleanup

Several of the ethernet functions require cleanup on exit, such as closing TCP connections and finishing UDP listens. To ensure that these happen, you should wrap your code with a try block and provide a finally block that cleans up any resources.

Example:

```
try:
      eth.operations.UdpStartListen.execute(port)
      while(RUN):
            # periodic code here

finally:
      eth.operations.UdpEndListen.execute(port)
      print("Done")
```

5. IP Configuration

5.1. Overview

Ethernet devices require configuration of an Internet Protocol address (IP address) to allow addressing on the network. The Ethernet Peripheral supports IPv4 addresses, which are specified as 4 numbers separated by a period, such as 192.168.1.1.

The Ethernet Peripheral supports two methods of configuration: Manual and DHCP. DHCP allows the device to query your network to determine the correct values to use and ensures there are no addressing conflicts.

By default the Ethernet Peripheral uses DHCP to configure the device.

5.2. Manual Configuration

Manual Configuration requires the user to specify the following:

- IP Address
- Network Mask
- Gateway
- DNS Server (optional)

If you do not know the values to assign, please consult with your network administrator to avoid conflicts and ensure the correct values.

There are two ways to set the parameters: they can be set individually or combined in one command.

Example:

```
ipValue = EthT.IPAddr((192,168,1,15))
eth.properties.PeripheralIP.store(ipValue)
ipValue = EthT.IPAddr((255,255,0,0))
eth.properties.PeripheralNetMask.store(ipValue)
ipValue = EthT.IPAddr((192,168,1,1))
eth.properties.PeripheralGateway.store(ipValue)
```

Example:

```
ip = EthT.IPAddr((192,168,1,15))
nm = EthT.IPAddr((255,255,0,0))
gv = EthT.IPAddr((192,168,1,1))
addresses = EthT.IPInfo( ip, nm , gw)
eth.properties.PeripheralAddresses.store(addresses)
```

5.3. DHCP Configuration

DHCP configuration requires that a valid DHCP server is present on the same physical network as the Network Peripheral. If you are unsure, please consult your network administrator.

To use DHCP, set the IP address to “IP_ANY” or (0,0,0,0).

Example:

```
ipValue = EthT.IPAddr(EthT.IPAddr.IP_ANY)
eth.properties.PeripheralIP.store(ipValue)
```

6. DNS

6.1. DNS Overview

The Domain Name Service (DNS) provides lookup to translate host names like google.com to their corresponding network IP addresses. It is often compared to a phone book, translating names that humans can easily recognize into numerical addresses that the system is able to handle.

The Ethernet Peripheral supports basic DNS lookup capabilities. Prior to using DNS a service provider server must be set. Usually a primary server and a secondary server are specified.

6.2. DNS Configuration

If the system is set to use DHCP, the DNS servers should be automatically set by the DHCP server. If you are using manual IP Configuration you must set them yourself. You can check the current configuration by querying the DNS servers. If they are IP_ANY you should manually configure them before performing any queries.

Example:

```
eth.properties.DnsServer.fetch()
time.sleep_ms(100)
dns_servers = eth.properties.DnsServer.value()
print( "DNS Servers: ", str(dns_servers[0]), str(dns_servers[1]))
if(dns_servers[0].isAny()):
      print("DNS Servers not set: Setting to Google’s public servers")
      dns_servers = [EthT.IPAddr((8,8,8,8)),EthT.IPAddr((8,8,4,4))]
      eth.properties.DnsServer.store(dns_servers)
      time.sleep_ms(100)
      eth.properties.DnsServer.fetch()
      time.sleep_ms(100)
      dns_servers = eth.properties.DnsServer.value()
      print( "DNS Servers: ", str(dns_servers[0]), str(dns_servers[1]))
```

6.3. DNS Usage

The DNS queries work in an Operation -> Event pattern. The user sends a query operation. A variable amount of time later the Ethernet peripheral returns a result by an event. If the address is IP_ANY then the system was unable to resolve the hostname.

Example:

```
def dnsCallback(peripheral, dnsResponse):
      if dnsResponse.ip.isAny()
            print(“DNS query failed for “, dnsResponse.hostname)
```

```
      else
            print("DNS Response for ", dnsResponse.hostname, " : ", str(dnsResponse.ip))

eth.events.DnsQuery.setCallback(dnsCallback)
eth0.operations.DnsQuery.execute("google.com")
```

The DNS Query string should be the target Fully Qualified Domain Name, like “google.com” or “drive.google.com”

- Do not include a protocol header (“http://google.com”)
- Do not include trailing slashes or paths (“www.youtube.com/feed/explore”)

7. UDP

7.1. Introduction

User Datagram Protocol (UDP) is a basic network protocol for sending and receiving messages with no requirement for prior communication and with no confirmation of reception. It's primarily used for establishing low-latency and loss-tolerating connections between applications.

The Ethernet Peripheral supports both sending and receiving UDP messages.

7.2. UDP Send

In order to send UDP you need to know both the IP address and the port number of the receiving system.

The basic steps to send a UDP message are:

1. Initialize the Ethernet peripheral
2. Specify the target IP address
3. Create a UDPPacket using the IP address, destination port, local port (optional) and the data to send.
4. Execute the SendUDP operation using the specified packet.

Example:

```
remoteIp = EthT.IPAddr((192,168,1,2))
remotePort = 20001
localPort = 0 # When zero the system chooses an available port.
data = bytearray([1,2,3,4,5,6,7,8,9]`

udpPacket = EthT.UDPPacket( remoteIp, remotePort, localPort, data ))
eth.operations.SendUDP.execute(udpPacket)
```

Note: There is no acknowledgement that the packet was sent.

7.3. UDP Receive

UDP Receive is an asynchronous process using a callback. To receive UDP you only need to provide a local port to listen on.

The basic steps to receive are:

1. Specify the callback to be executed when data is received.
2. Execute the UDPStartListen operation.
3. When finished, execute the UdpEndListen operation.

Example:

```
def udpCallback( peripheral, packet ):
      print("UDP Received:", packet)

eth.events.UdpPacketReceived.setCallback( udpCallback )
eth.operations.UdpStartListen.execute(port)
```

The UDPPacket type contains the following member variables:

addr: Source IP Address
srcPort: Source Port
destPort: Destination Port
data: bytearray containing the data from the UDP packet.

8. TCP

8.1. Introduction

Transmission Control Protocol (TCP) is a basic network protocol that provides reliable, ordered, and error checked delivery of a stream of bytes. It is the protocol that underlies most major networking applications. TCP is a connection-oriented protocol and requires that a socket connection be established before any data can be sent. A handshake process ensures that data is reliably received but lengthens latency.

The Ethernet Peripheral supports both listening as a server and connecting as a client.

The Ethernet Peripheral supports multiple TCP connections. It uses a user provided connection ID number between 1 and 65,535 to identify connections. This number will be needed to properly route data and state information.

8.2. Socket States

The Ethernet Peripheral handles many of the socket states and transitions for you, simplifying the process. Socket states are reported to the Brain via the TcpSocketState event.

The states that the Ethernet Peripheral reports are:

```
Failed      = const(0)
Closed      = const(1)
New         = const(2)
Connecting  = const(3)
Open        = const(4)
Listening   = const(5)
```

The user MUST ensure the socket is in the Closed state when finished with the socket. Sockets can enter the closed state due to a connection failure, the other party closing their socket, or when requested by the User.

Socket state changes are communicated using the TcpSocketState event.

8.3. Callbacks

TCP requires two callbacks to be set, the TcpSocketState and the TcpReceived.

TcpSocketState callback handles the socket state transitions. This callback should signal your application that it is safe to start transmitting data requests when the socket enters the “Open” state. It should signal your app to stop sending data when the socket enters the “Closed” state and provide error handling for unexpected socket closures. Error handling can also be provided for Failed socket connections.

Example:

```
def tcpStateCallback( peripheral, tcpconnection):
      print("On Connection." + str(tcpconnection))
      if tcpconnection.status == EthT.ConnectionState.Open:
            startDataSend(tcpconnection)
      elif tcpconnection.status == EthT.ConnectionState.Failed:
            print("Connection Failed: " + str( tcpconnection))
      elif tcpconnection.status == EthT.ConnectionState.Closed:
            print("Connection Closed: " + str( tcpconnection))
      elif tcpconnection.status == EthT.ConnectionState.Listening:
            print("Listening for connections: " + str( tcpconnection))
      elif tcpconnection.status == EthT.ConnectionState.Connecting:
            print("Connecting on: " + str( tcpconnection))
      else:
            print("ConnectCallback: " + str(tcpconnection))



eth.events.TcpSocketState.setCallback( tcpStateCallback )
```

The TcpReceived callback handles the arrival of TCP data from the socket. Data is guaranteed to be in order. How data is handled depends on your application.

Example:

```
myConnectionNumber = 1
remoteIp = EthT.IPAddr((192,168,1,2))
remotePort = 20001
localIp = None
localPort = 0 # When zero the system chooses an available port.

tcpConnection = EthT.TCPConnection(myConnectionNumber, localIp, localPort, remoteIp,
remotePort)

print("Starting TcpConnect")
eth.events.TcpReceived.setCallback( tcpCallback )
eth.events.TcpSocketState.setCallback( tcpStateCallback )
eth.operations.TcpConnect.execute(tcpConnection)
```

8.4. Connecting to a Server

The process of connecting to a server involves establishing a connection, using a callback function to monitor the socket state, and using a callback function to receive data from the server. Generally the server waits for requests from the client and then transmits the requested data back to the client. The user must wait for the socket Open state before attempting to send data.

When establishing a connection the following data must be provided:

- User provided connection ID number.
- The Local IP address (set to “None” when connecting to a server.)
- The Remote IP address for the connection.
- The Local port to use for the connection (set to 0 or “None” to allow the Ethernet Peripheral to choose)
- The Remote port for the connection.

After the TcpConnect operation is executed, the socket will transition to the “Connecting” or “Failed” state. Reasons for failure at this point:

- Out of Memory
- The socket is already in use.

Example:

```
myConnectionNumber = 1
remoteIp = EthT.IPAddr((192,168,1,2))
remotePort = 20001
localIp = None
localPort = 0 # When zero the system chooses an available port.


tcpConnection = EthT.TCPConnection(myConnectionNumber, localIp, localPort, remoteIp,
remotePort)

print("Starting TcpConnect")
eth.events.TcpReceived.setCallback( tcpCallback )
eth.events.TcpSocketState.setCallback( tcpStateCallback )
eth.operations.TcpConnect.execute(tcpConnection)
```

9.1. AllMaxValues(Fetch/Reset)

Gets the maximum value of all channels in a single request

Description:

Data Type(s):

Fetch:

Parameter(s):

Response:

Example:

Reset:

Parameter(s):

Response:

Example:

10.1. AllMinValues(Fetch/Reset)

Description:

Gets the minimum value of all channels in a single request.

Data Type(s):

Type depends on channel mode.

Fetch:

Parameter(s):

Response:

Example:

Reset:

Parameter(s):

Response:

ACK on success
NACK otherwise

Example:

11. Examples

11.1. Importing Output Peripheral

```
import NUD.System as s

import NUD.Peripherals.CAN.CAN as can
import NUD.Peripherals.CAN.CANValues as cv
import NUD.Peripherals.CAN.CANTypes as ct

CANperiph = can.CAN()

system = s.System()
# Attach the CAN peripheral to Side 1 Slot A
system.attachPeripheral(CANperiph, s.SideSlot.SIDE_1_SLOT_A)
```

11.2. Filter Configuration

11.3. Scaling and Offset

12. Terms and Abbreviations