DSPowerlink

Interfacing Network Adapter with Ethernet Powerlink (EPL)

Rev.2.2
August 20, 2009

DSP Control Group, Inc.

WHY ETHERNET POWERLINK (EPL)

ETHERNET Powerlink (**EPL**) is a deterministic real-time protocol for standard Ethernet. EPL expands Ethernet with a mixed polling and time-slicing mechanism. That brings:

guaranteed transfer of time-critical data in very short isochronous cycles with configurable response time

time-synchronization of all nodes in the network with a very high precision of sub-microseconds

transmission of less time-critical data in a reserved asynchronous channel

DSPCG implementations reach cycle-times of 200µs and a time-precision (jitter) of less than 1µs.

This communication profile meets timing demands typical for high-performance automation and motion applications without changing basic principles of the Fast Ethernet Standard IEEE 802.3; it also extends it towards Real Time Ethernet (RTE).

EPL is a cyclic communications network. The diagram below represents one EPL cycle.

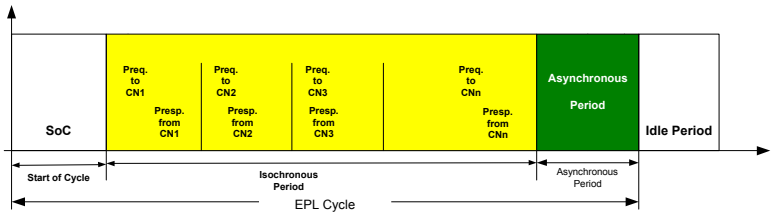


Figure 1: An EPL cycle consists of three segments

To avoid collisions and to make maximum use of the bandwidth, data exchange between the devices is time slot-based. One device on the EPL network takes on the function of the Managing Node, which controls the communication, defines the clock pulse for synchronization of all nodes, and assigns the right of transmission to the individual devices. The Controlled Nodes only transmit when requested to by the manager. An EPL cycle is divided into three periods (Fig. 1):

Start of Cycle (SOC): Here the manager transmits a "Start of Cycle" frame (SoC) as a broadcast message to all controllers. All devices in the EPL network synchronize on the SoC.

Isochronous Period: Cyclic data exchange takes place in this time period. According to a preset (configurable) schedule, the manager transmits a Poll Request frame (PReq) sequentially to each controller. The addressed controller responds with a Poll Response frame (PRes). All nodes involved with these data can receive them, whereby a real producer (or consumer) communication between the nodes is achieved similar to CAN.

Asynchronous Period: This time interval is available for asynchronous, non-time-critical data exchange. The Master Node grants access to one of the Controlled Nodes based on a priority which the request contains.

The application interface of EPL is based on the mechanisms defined in the CANopen communication profile DS301 of CAN in Automation. This opens up

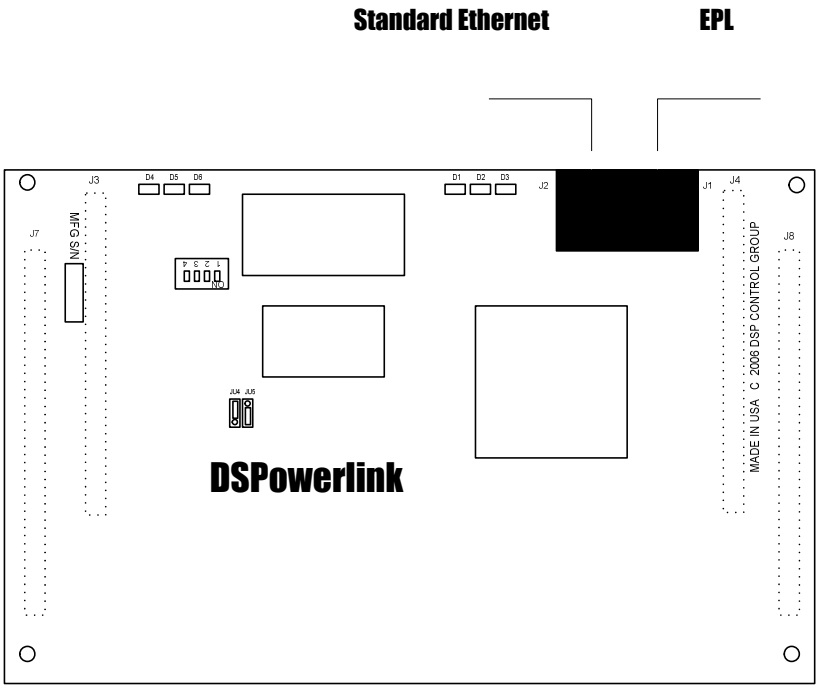
a wide range of readily available and usable device and application profiles for EPL.

The reference model for EPL is similar to the communication mechanisms and elements of CANopen. Supported elements include PDO (Process Data Objects, which transmit cyclic real-time data such as I/O and encoder feedback), SDO (Service Data Objects, which are transmitted during the Asynchronous period of EPL), object dictionaries, and network management. Also, the SDO protocol can be implemented via UDP/IP and therefore using standard IP messages. This enables direct access to the object dictionaries of EPL devices by devices and applications outside the EPL system via special EPL routers.

As a result of its impressive features, EPL is suitable for implementing applications with hard real-time requirements in the range of microseconds.

Ethernet Powertlink

Hardware Description



Four DIP switches determine the MAC address the card will use on the EPL network. The MAC address is determined as follows:

node	sw1	sw2	sw3	sw4	MAC address	DSPNET Node
MN	0(off)	0	0	0	00:CA:FE:F0:0D:00	0
CN	0	0	0	1	00:CA:FE:F0:0D:01	1
CN	0	0	1	0	00:CA:FE:F0:0D:02	2
CN	0	0	1	1	00:CA:FE:F0:0D:03	3
MN	0	1	0	0	00:CA:FE:F0:0D:04	4
CN	0	1	0	1	00:CA:FE:F0:0D:05	5
CN	0	1	1	0	00:CA:FE:F0:0D:06	6
CN	0	1	1	1	00:CA:FE:F0:0D:07	7

Note that no two devices on an EPL network can share a MAC address. Also note that the EPL standard requires one node to be set to the first entry in the table above, to configure it as the Managing Node. In a single motion control card application, a DSPowerlink card will serve as Managing Node. Above, switch setting illustrates a system with two Managing Nodes each with a unique DSPNet address setting.

A Simple, Single-DSPowerlink Network

Figure 3 illustrates the topology of a simple system inclusive of a single DSPowerlink card and associated Mx4 controller, and one bank of remote I/O. As noted earlier, the Managing Node (DSPowerlink) must be programmed at address 0 (all switches are set to off). The remote I/O modules (e.g. B&R X20 BC0083) can be set to any non-zero address and, finally, the Ethernet connector of the DSPowerlink card may be connected to the Ethernet terminal of an ordinary PC.

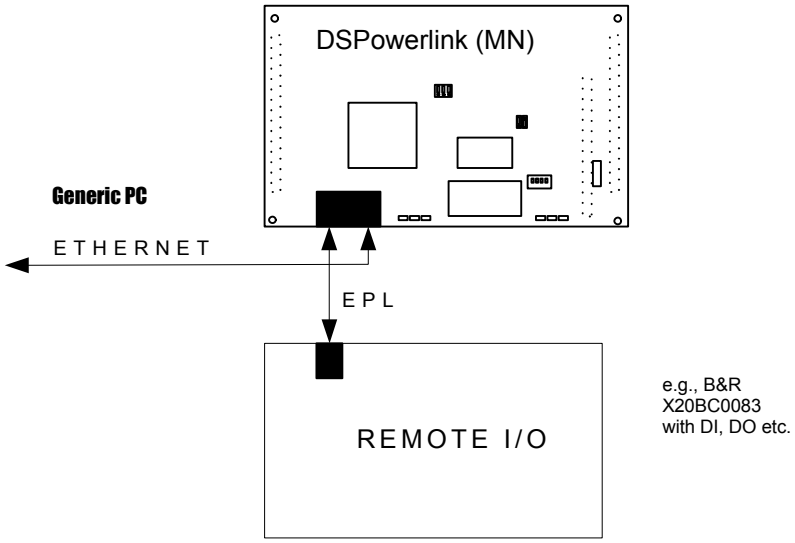


Figure 3: A system inclusive of one DSPowerlink and a remote I/O

Using Multiple DSPowerlink Cards in an EPL Network

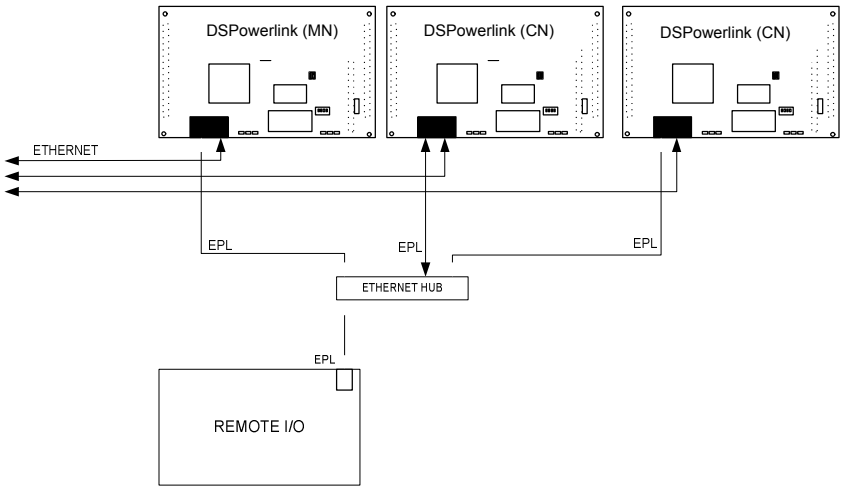
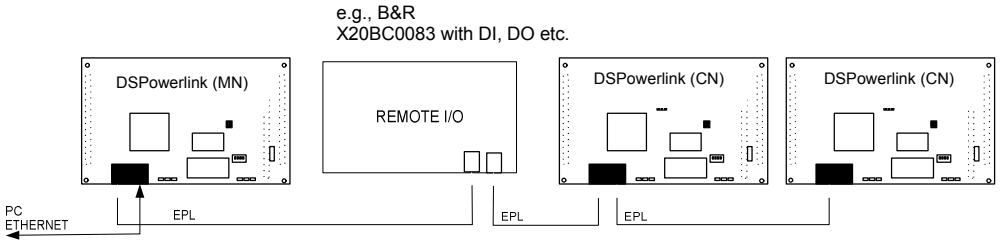


Figure 4: Using DSPowerlink cards as both Controlled and Managing Nodes

Figure 4 illustrates the use of several DSPowerlink cards in an EPL network, along with a remote I/O unit. Clearly, only one DSPowerlink card can be the Managing Node, while all others must be the Controlled Nodes. It is noteworthy that this by no means will impose a constraint on the functionality of any Mx4 controller. Thus, from the application point of view all Mx4 boards are alike and they all share the same remote I/O unit.



Users of DSPowerlink with an Mx4 controller can use one of two different software interfaces to control the Mx4 from a host PC. The first is via DSP Control Group's Mx4Pro software environment, which permits the user to write, compile, and download DSPL programs for an Mx4 controller; and view DSPL program operation on the Mx4 controller by monitoring DSPL variables in real time.

The second method is to use C routines and Windows 2000/XP drivers provided by DSP Control Group to create your own Windows applications to communicate with and control each Mx4 Controller.

Installing Mx4Pro (applicable only to star connected firmware version)

To install Mx4Pro, simply run *Mx4Pro5302\setup.exe* from the installation CD. The installation process will prompt you for the directory to install to on your hard drive.

You will need to install the latest version of Mx4Pro's DSPL compiler to accommodate all of DSPowerlink's features. Copy the file *DSPLCO32.EXE* from the root directory of the installation CD to the directory into which you have installed Mx4Pro. When asked whether you want to overwrite the existing file in the Mx4Pro directory, click *OK*.

To install the DSPowerlink driver for Windows,

1. Select *Network and Dial-up Connections* from Windows Control Panel
2. Select the *Networking* tab
3. Click the *Install...* button
4. From the component list, select *Protocol*
5. Click the *Add...* button
6. Click the *Have Disk...* button (these are instructions for Win XP. For Win 2K these instructions are slightly different see: Readme.txt)
7. Browse to the *Drivers\Win2000_Drivers* directory on the installation CD if you are installing under Windows 2000, or *Drivers\WinXP_Drivers* if you are installing under Windows XP
8. Select the *DSPNet* protocol
9. Windows may prompt for a .sys file; this can be found in the *Drivers\Win2000_Drivers* or *Drivers\WinXP_Drivers* directory selected in step 7. above.
10. NOTE: The DSPNet protocol will NOT appear in the list of network protocols. This is normal.

After installing the protocol, all you need to do is connect an Ethernet cable from the host PC to any DSPowerlink card. A connection will be established automatically when you run Mx4Pro.

Installing a Simple C-Language Project

Multiple C-language projects are included on the installation CD to demonstrate interfacing to a DSPowerlink card from a user's application. The projects are designed to run from a DOS command-line window under Windows 2000 or Windows XP, and are simple command-line applications which permit the user to access dual-port RAM (DPR) on the Mx4 controller on which the DSPowerlink card is installed, and to download various types of data tables. It is assumed that the user is familiar with DOS command-line operations.

Interfacing to a DSPowerlink card from C requires first that WinPcap, an open-source freeware Ethernet interface, be installed on your computer. A version of WinPcap in included on the installation CD in the root directory. Run the executable *WinPcap_4_0.exe* to install WinPcap to your computer. The same version of WinPcap works for both Windows 2000 and Windows XP.

Two sample projects generated using Microsoft Visual C++ 6.0, are included on the CD, and each project comes in two versions. One version has its interface routines compiled into a DLL; the other has the routines included as source files which you can view and modify.

The projects are included as compressed zip files on the installation CD. Simply decompress whichever version you want into a directory on your hard drive, and modify or run the project as desired. The project *DPR* provides a simple interface to DPR on an Mx4 controller. Run the executable *DPR.EXE*, located in the *Debug* subdirectory of the project, without any command-line parameters to see an information screen describing program operation.

For more information about understanding DPR, see the user manual for your Mx4 controller.

The project *DWNLD* permits the user to download DSPL code and various cam and cubic spline tables to the Mx4 controller. Run the executable DWNLD.*EXE*, located in the *Debug* subdirectory of the project, without any command-line parameters to see an information screen describing program operation.

For more information about DSPL and the various tables supported by your Mx4, see the user manual for your Mx4 controller.

Ethernet Powertlink

GLOSSARY

Process Data

The data exchanged during the isochronous phase are known as Process Data Objects (PDO) and are device and application dependent. You choose what data you wish to exchange at ‘design time’. Examples would be control and status information, speed, position, torque demand for a drive, digital and analogue I/O information, or perhaps absolute position from an EPL encoder.

Multiplexing to Optimize Bandwidth

Each of the time slots can actually be multiplexed amongst a number of Controlled Nodes, which then effectively operate at a lower cycle rate. This allows some devices to operate at an optimum fast cyclic rate, while lower priority devices operate at lower bandwidth. This provides a great deal of design flexibility for system configuration and optimization.

Asynchronous Period

The remaining cycle time is allocated for asynchronous, or non-real-time, communications. The Managing Node will grant access to this period to one of the Controlled Nodes based on a request and prioritization scheme. Messages are standard IP messages, so communication with the outside world for maintenance or production monitoring and control can take place.

Service Data

During the asynchronous period, devices can request or respond with Service Data Object (SDO) messages, which can contain any information which is generally not time critical. This can be for control purposes such as gain changes, or configuration and maintenance purposes from outside the EPL network.

Internet and Network Connection

EPL uses standard Ethernet infrastructure and physics, Hubs, CAT5e STP cabling, PHYs and MACs. EPL gateways (routers) provide a means of networking an EPL system to higher-level IT network infrastructure with integrated firewall and NAT services your IT specialists will be familiar with.

EPL Security

EPL is designed to operate as a protected network segment. Connection to other networks is done via an EPL router/gateway. An EPL router/gateway is similar to traditional routers, but with the behavior of an EPL device on one side and the behavior of a standard Ethernet router on the other side. These routers act as a mediator and also a security barrier between the open and non-deterministic Ethernet, and the deterministic EPL network. EPL routers integrate firewall technologies, which prevent unauthorized access, thereby maintaining network security and keeping hackers out.

EPL_INP1_REG, EPL_INP2_REG
IDENTIFIER

IDENTIFIER Real-time status of EPL inputs 0 to 31

USAGE DSPL (PLC, Motion)

DESCRIPTION

These input bit registers may be assigned to DSPL variables, or they may be used in conjunction with logical bitwise operators as part of the following conditional expressions in the DSPL language: IF, WHILE and WAIT_UNTIL.

`EPL_INP1_REG`

A 16-bit value coding the status of EPL inputs (0 to 15). A bit set to 1 indicates the corresponding EPL input is in an active state.

`EPL_INP2_REG`

A 16-bit value coding the status of EPL inputs (16 to 31). A bit set to 1 indicates the corresponding EPL input is in an active state.

EPL_INP1_REG, EPL_INP2_REG cont.

EXAMPLE

The following DSPL segment holds the program until the EPL input 31 is on:

```
wait_until(EPL_INP2_REG & 0x8000)
```

The following DSPL program segment increments var2 whenever EPL input 1 is on:

```
var1                          =                          1
while ( var1 == 1 )
  if (EPL_INP1_REG & 0x0001)
     var2=var2+1
  endif
wend
```

The following DSPL program segment assigns the contents of EPL_INP1_REG to var42:

```
var42 = EPL_INP1_REG
```

EPL_OUTP_ON

FUNCTION Sets EPL Outputs to 'On' State

EXECUTION 200 microseconds

SYNTAX

[Mx4 Octavia and Mx4 and Mx42 family]

`EPL_OUTP_ON (outp1,outp2)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

`outp1` A 16-bit value coding the EPL outputs (0 to 15) to be set to ON. A bit set to 1 sets the correspondingEPL digital output to 1. A bit set to zero leaves the corresponding EPL digital output unchanged.

`outp2` A 16-bit value coding the EPL outputs (16 to 31) to be set to ON. A bit set to 1 sets the corresponding EPL digital output to 1. A bit set to zero leaves the corresponding EPL digital output unchanged.

A complete bank of 32 digital outputs is sent cyclically to all nodes via the EPL network.

DESCRIPTION

This command sets the corresponding EPL digital output to an “ON” state.

SEE ALSO `EPL_OUTP_OFF, OUTP_ON, OUTP_OFF`

APPLICATION

This command can be used for a general purpose logical output operation.

EPL_OUTP_ON cont.

EXAMPLE

The following DSPL command turns on EPL outputs 2,3 and 31:

```
EPL_OUTP_ON (0x000C, 0x8000)
```

Also, the following DSPL program segment toggles all 32 EPL outputs on and off every 0.5 seconds.

```
var1 = 1
while ( var1 == 1 )
     epl_outp_on(0xFFFF, 0xFFFF)
     delay (2500)
     epl_outp_off(0xFFFF, 0xFFFF)
     delay(2500)
wend
```

EPL_OUTP_OFF

FUNCTION Sets the EPL Outputs to 'Off' State

EXECUTION 200 microseconds

SYNTAX

[Mx4 Octavia and Mx4 and Mx42 family]

`EPL_OUTP_OFF (outp1,outp2)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

`outp1` A 16-bit value coding the EPL outputs (0 to 15) to be set to OFF. A bit set to 1 sets the corresponding EPL digital output to 0. A bit set to zero leaves the corresponding EPL digital output unchanged.

`outp2` A 16-bit value coding the EPL outputs (16 to 31) to be set to OFF. A bit set to 1 sets the corresponding EPL digital output to 0. A bit set to zero leaves the corresponding EPL digital output unchanged.

A complete bank of 32 digital outputs is sent cyclically to all nodes via the EPL network.

DESCRIPTION

This command sets the corresponding EPL digital output to an “OFF” state.

SEE ALSO `EPL_OUTP_ON, OUTP_ON, OUTP_OFF`

APPLICATION

This command can be used for a general purpose logical output operation.

EPL_OUTP_OFF cont.

EXAMPLE

The following DSPL command turns off EPL outputs 1,2 and 30:

```
EPL_OUTP_OFF (0x0006, 0x4000)
```

Also, the following DSPL program segment toggles all 32 EPL outputs on and off every 0.5 seconds.

```
var1 = 1
while ( var1 == 1 )
     epl_outp_on(0xFFFF, 0xFFFF)
     delay (2500)
     epl_outp_off(0xFFFF, 0xFFFF)
delay(2500)
wend
```

EPL_NODE1_WRITE, EPL_NODE2_WRITE, EPL_NODE3_WRITE

FUNCTION Write data at a given offset to a controlled node

EXECUTION 200 microseconds

SYNTAX

[Mx4 Octavia and Mx4 and Mx42 family]

```
EPL_NODE1_WRITE(offset, data)
EPL_NODE2_WRITE(offset, data)
EPL_NODE3_WRITE(offset, data)
```

USAGE DSPL (PLC, Motion)

ARGUMENTS

`offset`	A DSPL variable or constant in the range 0-3, referring to which of four words of the controlled EPL device that is to be written to.
`data`	16 bits of device-specific data.

EPL_NODE1_WRITE, EPL_NODE2_WRITE, EPL_NODE3_WRITE … continued

DESCRIPTION

This command will write two of the available eight bytes for a given controlled node.

SEE ALSO `EPL_NODEx_IN0, EPL_NODEx_IN1`

APPLICATION

This command can be used for device-specific logical output operations, as the significance of 'data' will depend upon the controlled device. Also, the numbers "1", "2", and "3" that refer to the specific controlled node are relative to the master. Therefore, if the master is node 4 then `EPL_NODE1_WRITE` will be writing to node 5. If the master is node 8, then `EPL_NODE1_WRITE` will be writing to node 9, etc.

EXAMPLE

The following DSPL program can be used for turning a Baldor EPL motor. In this example the Baldor motor is node 2.

```
plc_program:
   run_m_program (my_first)
end


my_first:
   var1 = 1;

   epl_node2_write( 0x0, 0x0000 );
   epl_node2_write( 0x1, 0x0000 );
   epl_node2_write( 0x2, 0x0000 );
   epl_node2_write( 0x3, 0x0000 );

   epl_node1_write( 0x0, 0x0000 );
   epl_node1_write( 0x1, 0x0000 );
   epl_node1_write( 0x2, 0x0000 );
   epl_node1_write( 0x3, 0x0000 );

; Condense into 16-bit writes.
```

EPL_NODE1_WRITE, EPL_NODE2_WRITE, EPL_NODE3_WRITE … continued

```
epl_node2_write( 0x0, 0x0000 );
epl_node2_write( 0x1, 0x0000 );
epl_node2_write( 0x2, 0x0000 );
epl_node2_write( 0x3, 0x0000 );
delay( 100 );

epl_node2_write( 0x0, 0x0080 );
delay( 500 );
epl_node2_write( 0x0, 0x0000 );
delay( 500 );
epl_node2_write( 0x1, 0x0001 );
delay( 500 );

epl_node2_write( 0x0, 0x0046 );
delay( 500 );
epl_node2_write( 0x0, 0x0047 );
delay( 500 );
epl_node2_write( 0x0, 0x004F );
delay( 500 );


while( var1 == 1 )
   epl_node2_write( 0x2, 0x09c4 );
   epl_node2_write( 0x3, 0x0000 );

   epl_node2_write( 0x0, 0x005F );
   delay( 500 );
   epl_node2_write( 0x0, 0x004F );
   delay( 2000 );

   epl_node2_write( 0x2, 0xf63b );
   epl_node2_write( 0x3, 0xffff );

   epl_node2_write( 0x0, 0x005F );
   delay( 500 );
   epl_node2_write( 0x0, 0x004F );
   delay( 2000 );

   epl_node2_write( 0x2, 0x1388 );
   epl_node2_write( 0x3, 0x0000 );

   epl_node2_write( 0x0, 0x005F );
   delay( 500 );
   epl_node2_write( 0x0, 0x004F );
   delay( 2000 );

   epl_node2_write( 0x2, 0xec77 );
   epl_node2_write( 0x3, 0xffff );

   epl_node2_write( 0x0, 0x005F );
   delay( 500 );
```

EPL_NODE1_WRITE, EPL_NODE2_WRITE, EPL_NODE3_WRITE … continued

```
            epl_node2_write( 0x0, 0x004F );
            delay( 2000 );

            epl_node2_write( 0x2, 0x2710 );
            epl_node2_write( 0x3, 0x0000 );

            epl_node2_write( 0x0, 0x005F );
            delay( 500 );
            epl_node2_write( 0x0, 0x004F );
            delay( 2000 );

            epl_node2_write( 0x2, 0xD8EF );
            epl_node2_write( 0x3, 0xffff );

            epl_node2_write( 0x0, 0x005F );
            delay( 500 );
            epl_node2_write( 0x0, 0x004F );
            delay( 2000 );

            epl_node2_write( 0x2, 0x4e20 );
            epl_node2_write( 0x3, 0x0000 );

            epl_node2_write( 0x0, 0x005F );
            delay( 500 );
            epl_node2_write( 0x0, 0x004F );
            delay( 2000 );

            epl_node2_write( 0x2, 0xb1DF );
            epl_node2_write( 0x3, 0xffff );

            epl_node2_write( 0x0, 0x005F );
            delay( 500 );
            epl_node2_write( 0x0, 0x004F );
            delay( 2000 );
        wend

    end
```

EPL_NODE1_IN0, EPL_NODE1_IN1, EPL_NODE2_IN0, EPL_NODE2_IN1, EPL_NODE3_IN0, EPL_NODE3_IN1

FUNCTION Real-time status of a controlled node's 16 bit input words

EXECUTION 200 microseconds

USAGE DSPL (PLC, Motion)

DESCRIPTION Each of these variables will reflect one of the 16 bit words associated with a given controlled node. The meaning of the returned data is specific to the type of controlled node that the data is coming from.

These values may be assigned to DSPL variables, or they may be used in conjunction with logical bitwise operators as part of the following conditional expressions in the DSPL language: IF, WHILE, and WAIT_UNTIL.

`EPL_NODEx_IN0`	A 16-bit value coding the status of a controlled node's inputs 0-15.
`EPL_NODEx_IN1`	A 16-bit value coding the status of a controlled node's inputs 16-31.

Also, the numbers "1", "2", and "3" that refer to the specific controlled node are relative to the master. Therefore, if the master is node 4 then EPL_NODE1_WRITE will be writing to node 5. If the master is node 8, then EPL_NODE1_WRITE will be writing to node 9, etc.

SEE ALSO `EPL_NODEx_WRITE`

EPL_NODE1_IN0, EPL_NODE1_IN1, EPL_NODE2_IN0, EPL_NODE2_IN1, EPL_NODE3_IN0, EPL_NODE3_IN1 … continued

EXAMPLE

The following DSPL segment holds the program while waiting for controlled node 3's low 16 input bits to be a certain value:

```
wait_until(EPL_NODE3_IN0 & 0x0001)
```

The following DSPL segment assigns the contents of controlled node 2's input:

```
var2low = EPL_NODE2_IN0
var2high = EPL_NODE2_IN1
```

The following DSPL program can blink the lights on a PLC in an EPL network when the PLC is node 1:

```
plc_program:
        run_m_program (my_first)
end

my_first:
var1 = 1;

epl_node1_write( 0x00, 0x0001 );

while( var1 == 1 )
        var2 = epl_node1_in0;
        var3 = epl_node1_in1;
        var4 = var2 & 128;

        if( var2 & 32 )
                var10 = 1;
        else
                var10 = 0;
        endif;

        if( var2 & 64 )
                var11 = 2;
        else
                var11 = 0;
        endif;

        if( var2 & 16 )
```

EPL_NODE1_IN0, EPL_NODE1_IN1, EPL_NODE2_IN0, EPL_NODE2_IN1, EPL_NODE3_IN0, EPL_NODE3_IN1 … continued

```
                var12 = 4;
        else
                var12 = 0;
        endif;

        if( var2 & 128 )
                var13 = 8;
        else
                var13 = 0;
        endif;

        var14 = var10 + var11;
        var14 = var14 + var12;
        var14 = var14 + var13;

        epl_node1_write( 0, var14 );
wend;
end
```