DSPEtherCAT User's Guide v1.1

EtherCAT Motion Controller & Yaskawa Sigma 7

DSPEtherCAT – Master EtherCAT Motion Controller DSP Control Group, Inc.

Contents

DSPEtherCAT

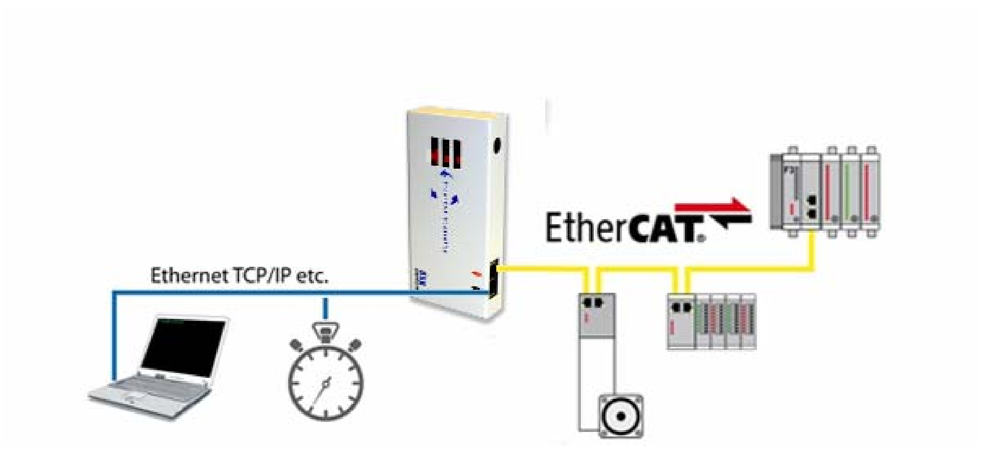
DSPEtherCAT Motion Controller & Yaskawa Sigma 7 v1.1

DSP Control Group, Inc.
4445 W 77th Street
Minneapolis, MN 55435
Phone: (952) 831-9556
FAX: (952) 831-4697

1 Introducing DSPEtherCAT

DSPEtherCAT is a high-speed, high-precision motion controller that uses real-time industrial EtherCAT protocol and supports both the line and star topologies. DSPEtherCAT is offered in three platforms of

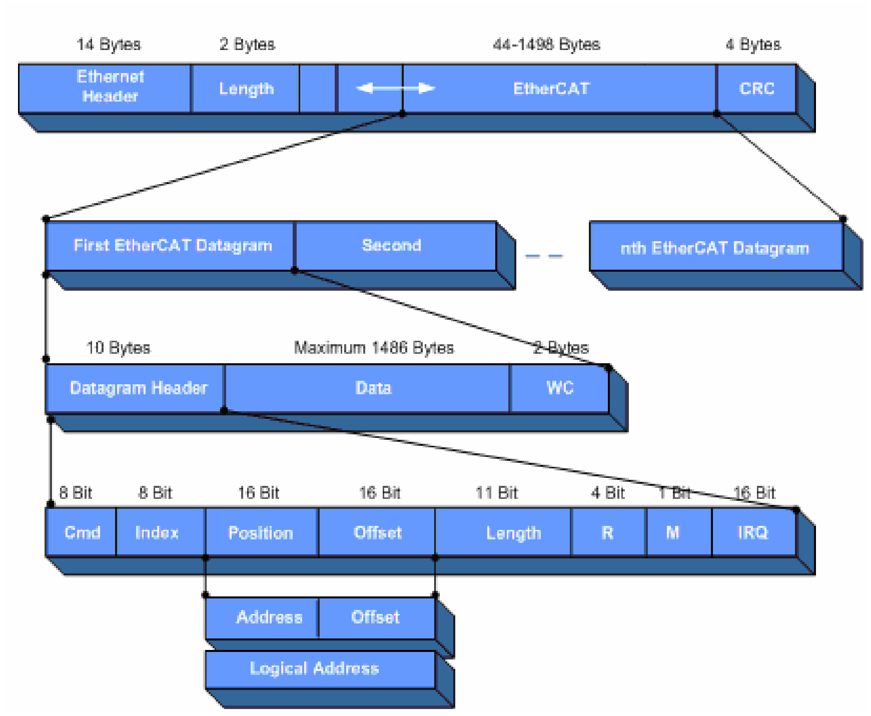
- PCI-based DSP motion control hardware,
- Stand-alone DSP motion controller and,
- Soft Motion controller (meaning,using a real-time operating system, commands are generated inside the PC by the user program.)



DSPEtherCAT Master Motion Controller and Network Manager

DSPEtherCAT Timing Cycle

With EtherCAT there is only one packet per cycle; each slave's data is in a specified part of that packet. Since Ethernet allows for a maximum packet size of 1518 bytes, subtracting the 18 bytes for Ethernet overhead leaves 1500 bytes for EtherCAT to work with (also there will be 14 bytes of EtherCAT header, leaving 1486 bytes for data). This lends itself to optimal bandwidth utilization. When you incorporate the 32 bytes of overhead per packet, one large packet is certainly more optimal than many small packets.



Ethernet frame structure/ EtherCAT Datagram used by DSPEtherCAT

EtherCAT protocol has a unique feature that distinguishes it from other industrial networking protocols. With EtherCAT there is only one packet per cycle; each slave's data is in a specified part of that packet.

Other protocols require one packet per slave, with each slave also sending one packet back to the master. Hence when there are many slaves, EtherCAT's approach does not put much of a load on the master's CPU. This can allow for faster cycle times.

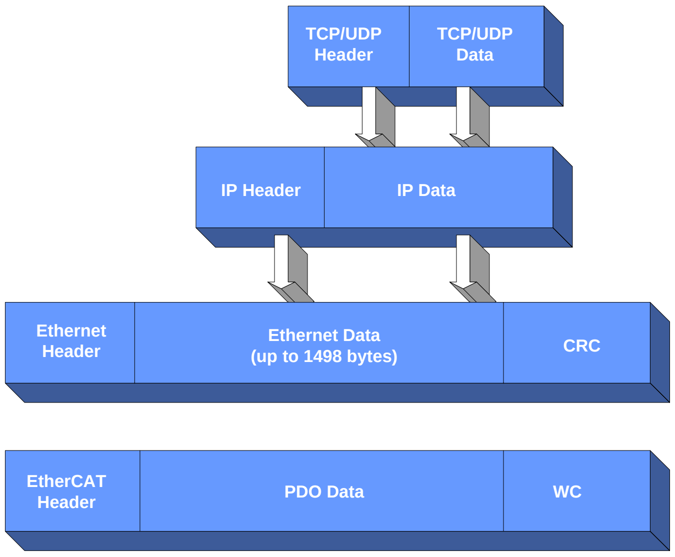
Network Delay

The EtherCAT packet leaves the master and then traverses the network. The delay introduced by processing (forwarding) at each slave is bounded by approximately 500 nanoseconds. (The reason that the delay from each slave is so low is because of special hardware that each EtherCAT slave must have. This hardware allows for data to be read or written while the packet passes through the slave.) Taking into account the PHY delay and cable delay, the addition of a slave can add 1 microsecond of delay into the system.

The EtherCAT protocol is a real-time industrial Ethernet protocol. As a matter of fact, it was one of the first, and is supported well and widely used. Most people are now familiar with the term "Ethernet". Because this technology has been so developed and tested, it now presents itself as a solution for industrial networking as well. The components, such as Ethernet cabling and PHY interfaces, are well-tested and widely available. These components are used in an EtherCAT solution.

2 EtherCAT Principle of Operation

The EtherCAT protocol transfers data directly within a standard Ethernet frame without changing its structure. When the master controller and slaved drives are on the same network, the EtherCAT protocol merely replaces the Internet Protocol (IP) in the Ethernet frame.



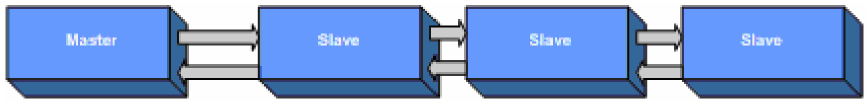
Ethernet Frame Structure With EtherCAT

Data is communicated between a master and drives in the form of Process Data Objects (PDOs). Each PDO has an address to one particular drive/IO device or multiple slaves, and this “data and address” combination (plus the Working Counter for validation) makes up an EtherCAT telegram. One Ethernet frame can contain multiple telegrams.

Data Exchange

With most real-time protocols, the master controller sends a data packet and must wait for the Process Data to be interpreted and copied at every slave node. However, this method of determinism may be difficult to sustain because the master controller must add and manage a relatively long processing time and jitter per slave.

EtherCAT overcomes this limitation by processing each frame on the fly. For example, suppose the Ethernet frame is a moving train, and the EtherCAT telegrams are train cars. The bits of PDO data are people in the cars who can get off or get in at the appropriate slaves. The whole “train” passes through all the slave drives/IOs without stopping, and the end slaves end sit back through all the slaves again.



In the same way, when device 1 encounters the Ethernet packet sent by the master, it automatically begins streaming the packet to device 2, while reading and writing to the packet with only a few nanoseconds of delay. Because the packet continues passing from slave to slave, it could be present in multiple drive slaves at the same time.

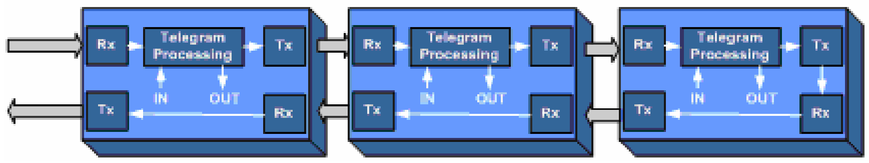
What does this mean practically? Consider having 5 slave devices, and different data is sent to each slave. For non-EtherCAT implementations, this may mean sending 5 different packets. For EtherCAT, one long packet that touches all slaves is sent, and thepacket contains 5 devices’ worth of data. However, if all the slaves need to receive the same data, one short packet is sent, and the slaves all look at the same part of the packet as it is streaming through, which optimizes the data transfer speed and bandwidth.

Timing and Synchronization

Another factor in achieving deterministic networks is the master controller's responsibility to synchronize all slave devices with the same time using distributed clocks. One of the slave devices must contain the master clock that synchronizes the other slave devices' clocks. Accurate synchronization is particularly important when widely distributed processes requires simultaneous actions such as coordinated motion between motion axes.

Transfer Mechanism in EtherCAT

Seen from Ethernet view, a EtherCAT bus is a single Ethernet participant. This participant receives and sends Ethernet telegrams. Inside this Ethernet participant there are many EtherCAT® slave devices, which process data on the fly.
The telegrams are only delayed for a micro seconds by each slave. Each telegram is processed by all the slaves until it arrives at the last slave in the line. The processed telegram is sent back from the last slave to the first slave, which relays it back to
the master controller.



Telegram Processing

As was explained earlier, the telegrams are processed on the fly. During forwarding the telegram to the next device, the slaves interpret the EtherCAT commands and read its input data. Output data for the master or for another slave is also inserted in that same telegram.

The processing of telegrams is performed by the slave hardware. Thus, the processing speed of an eventually connected processor does not influence the processing.

Several EtherCAT Process Data Units (PDU) can be embedded in one EtherCAT® telegram. Each can address one or more slaves.

An EtherCAT PDU consists of

- Header
- Data Area
- Working Counter

The Working Counter is incremented by each slave which is addressed by this PDU and which has successfully carried out the command embedded in the header of the PDU. The Master-Communication-Module compares the received Working Counter with the expected value (No. of slaves).
There are two types of EtherCAT® telegrams: with and without UDP/IP. Telegrams without UDP/IP can only be used in an Ethernet subnet. Telegrams with UDP/IP allow IP routing and perform communication over router.

Distributed Clock

Highly accurate synchronization between the slaves is required if distributed slaves are used to carry out actions simultaneously. One of those applications for example is the control of several servo drive axes, which must perform coordinated motion. EtherCAT uses synchronization according to IEEE 1588. This method performs synchronization that is robust to communication disturbances. This synchronization mechanism eliminates:

- different start-up times
- delay times between the slave with the master clock and all other slaves
- drift in the local clocks.

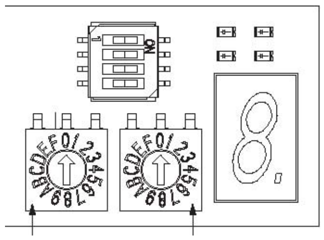
CANopen over EtherCAT (CoE)

The CANopen protocol is preferably used for transferring configuration parameters to the slaves. It can also be used for the transmission of Process Data. Instead of a standard CANopen protocol, which can only transfer 9 bytes of user data, CoE transfers a maximum of 1478 bytes. Therefore, the CoE protocol can be used to transfer process data with variable length.

3 DSPEtherCAT Master Communication

DSPEtherCAT (CoE) Communications Settings

In a DSPEtherCAT, servo drive secondary addresses (set on the amplifier’s front panel) can be used in order to identify slave devices on the network.



Setting the address on servo drive : Upper four bits Lower four bits of EtherCAT secondary address

Normal Device Recognition Process at Startup

When communication is started, the DSPEtherCAT master uses auto-increment addressing to detect the slaves. The Identity objects read from the slaves are compared with the master configuration information - set in advance with an EtherCAT Servo Drive configuration tool. Therefore, the slaves must be connected in the network in the same manner as they appear in the DSPEtherCAT master configuration.

Device Configuration on Servo Drives

The master uses auto-increment addressing to read the station addreses set by their address switches. It then compares the detected station addresses with the master configuration to get the topology that was set as the network topology.

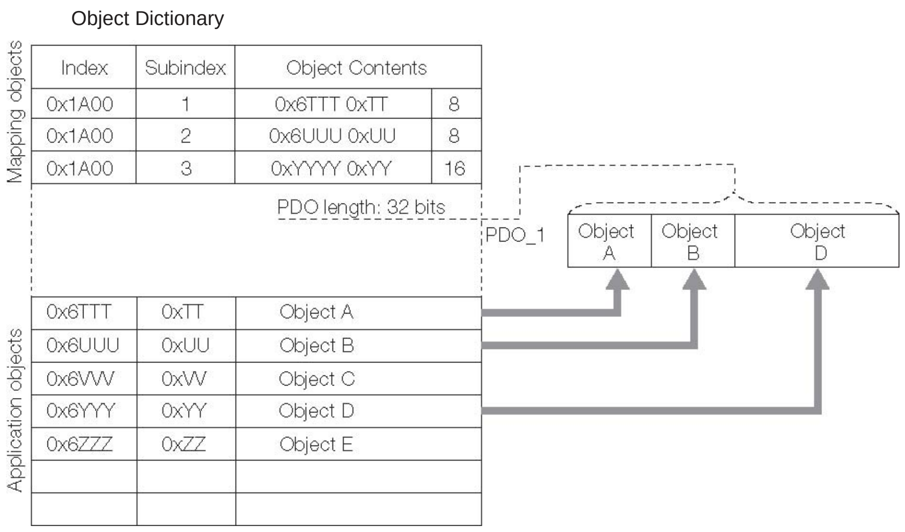
Example: Yaskawa SERVOPACK Station Alias Register (0x0012)

The value of the register can be read as follows:
Configured station alias = (S1 set value) × 16 + (S2 set value)

PDO Mappings

This section is informative only to more advanced developers who use Yaskawa SGDV EtherCAT as their slave drives. The process data that is used in process data communications is defined in the PDO mappings. PDO mappings are definitions of the applications objects that are sent with PDOs. The PDO mapping tables are in indexes 1600h to 1603h for the RxPDOs and indexes 1A00h to 1A03h for the TxPDOs in the object dictionary.

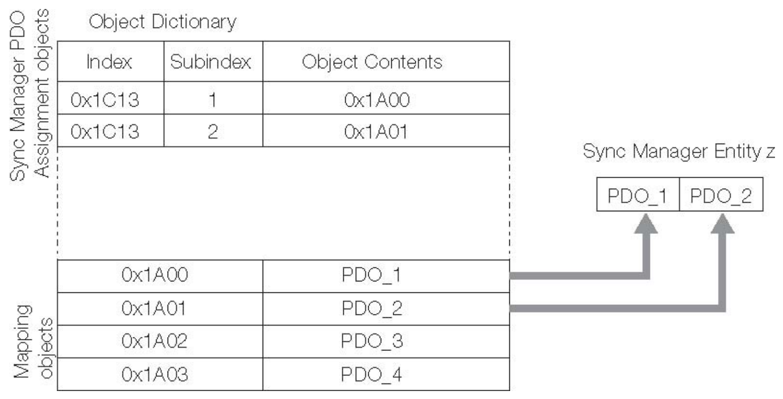
The following figure shows an example of PDO mappings.



In addition to the above PDO mappings, PDOs have to be assigned to the Sync Managers to exchange EtherCAT process data.The Sync Manager PDO assignment objects (1C12h and 1C13h) establish the relationship between these PDOs and the Sync Managers.

The following figure shows an example of a Sync Manager and the PDO mappings.

Object Dictionary



Setting Procedure for PDO Mappings

4 DSPEtherCAT and Drives' Inner-Workings

Before reading this sections there are a few abbreviations you need to know.

- **CoE** CANopen over EtherCAT - Profile for motion control
- **DC** Distributed Clocks - Synchronization mechanism
- **ESC** EtherCAT Slave Controller - Interface between EtherCAT bus and slave application

Synchronous & Asynchronous Motions

There are two timing modes in any EtherCAT motion control.

- Cyclic Synchronous Motion, CoE, in which points on a coordinated trajectory are transmitted each communication cycle
- Acyclic motion (asynchronous motion), in which the trajectory is transmitted as one command in one communication cycle.

Cyclic Synchronous motion

Synchronization is required to support cyclic motion.
Cyclic motion support is necessary if realtime interaction with slaves is necessary.

Examples:

- Coordinated motion – synchronizing position of

multiple axes
- Example: x-y-z milling machine cutting a cyllinder
- Time-critical events to occur based on another event – for instance an input signal or set an output when motor passes a specific position.

Cyclic motion support allows operation in every most servo drive motion modes:

- Profile modes (Position, Velocity, Torque)
- Cyclic modes (Position, Velocity, Torque)
- Interpolated Position mode – Mode 1 & Mode 2
- Homing mode

Distributed Clock (DC) is the synchronization mechanism for an EtherCAT network in tightly coordinated or interpolated motions.

Acyclic motion

Synchronization is not required to perform acyclic motion. Acyclic motion support is limited to operation in the following modes:

- Profile modes (Position, Velocity, Torque)
- Interpolated Position mode – Mode 2 (Yaskawa)
- Homing mode

Distributed Clocks do not need to be implemented on the DSPEtherCAT master for acyclic motion support.
The Yaskawa Drive is set to "Free-Run", by setting ESC register 0x980 = 0x0000.
In this mode, the cycle during which the drive samples physical inputs and sets physical outputs is not synchronized with the network communication cycle.

Cyclic Update Rate

The network's cyclic update rate is defined as the frequency of the DSPEtherCAT master transmitting process data to the slaves. The cycle time or communication cycle can be determined from the cyclic update rate.

Example: A network's cyclic update rate of 1 kHz has a cycle time of 1 ms, and a communication cycle of 1 ms.
The following factors affect the selection of a cyclic update rate that meets application requirements:

- Application requirements
- Example: Closing the position loop in the master.
- Cycle time of other devices on the network and features the master supports.

Example: Yaskawa drive operates at cycle times that are a multiple of 125 us (and up to 4 ms maximum), and another slave that operates at cycle times that are a multiple of 100 us, the DSPEtherCAT master may need to set a lowest common multiple of the two devices (in this case, 500 us).

Process Requirements

The process requirements dictate the required DSPEtherCAT network cycle time. Consider the following requirements:

- The rate at which the DSPEtherCAT master equires input process data from the slave (e.g. for data logging).
- Whether or not the master is required to react based on the process data from the drive (e.g. if the master must close a position or velocity control loop through the network).

- When possible, make a system that avoids closing control loops through the network and instead fully utilizes the processing power available on the drive slave (e.g., by using interpolated position or CSP) you can minimize the need for network bandwidth:

- EtherCAT network bandwidth is used more efficiently, allowing a reduced network update to be used
- EtherCat network and processing demands on the DSPEtherCAT master are reduced, allowing lower performance hardware to be used without sacrificing system performance

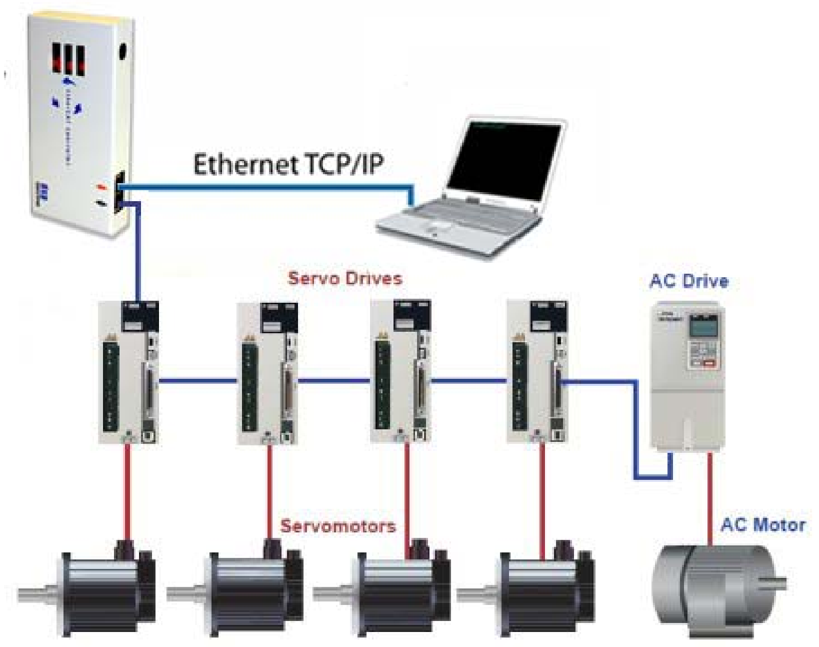
Example: In CoE Cyclic Synchronous Position mode, the DSPEtherCAT master transmits a new position value to the drive each communication cycle. The position values are calculated by the DSPEtherCAT from a pre-determined path (such as in CNC contouring motions or a coordinated packaging application). The next position values are not dependent on the feedback position values from the drive. Velocity is not transmitted by the master DSPEtherCAT in this mode, because the velocity is calculated as the position delta over time (the communication cycle being the time). The tuning algorithms in the drive amplifier may be used so that the position of the motor matches closely to the position values transmitted by the DSPEtherCAT. In this application, the master is not required to react based on process data from the slave. The slave drive is closing the position loop. An acceptable cycle time may be 2 ms in this example.

5 DSPEtherCAT Instruction Set

In a master motion control card application, either DSPEtherCAT PCI or its stand-alone version will serve as the Master EtherCAT Controller.

DSPEtherCAT Network Network Topology

The figure below illustrates the topology of a simple system inclusive of a single DSPEtherCAT unit and associated drive slave units. The remote I/O modules can be set to any non-zero address and, finally, the Ethernet connector of the DSPEtherCAT card may be connected to the Ethernet terminal of an ordinary PC.



Users of DSPEtherCAT can use an Mx4pro development software on their PC. The Mx4Pro software environment permits the user to write, compile, and download DSPL programs for an DSPEtherCAT controller; and view DSPL program operation on the controller by monitoring DSPL variables on PC in real time.

The second method is to use C routines and Windows XP/Win 7 drivers provided by DSP Control Group to create your own Windows applications to communicate with and control a DSPEtherCAT Controller.

Installing Mx4Pro Development Software

To install Mx4Pro, simply run *Mx4Pro5304\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.

7 Browse to the *Drivers* on the installation CD if you are installing under Windows 7, 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\Win7* 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 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 7 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 7 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 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.

DSPEtherCAT Configuratin File

The DSPEtherCAT .cfg file must be downloaded to the DSPEtherCAT flash memory, before the drives are powered up for the first time. This file includes the following values:

Drive Number Value = x	(where x = 1, 2, 3, … 8)
Drive's Front Panel Address Value =	(sw1) + 16 x (sw2)
Motion Control Mode Value = x	(where x = 1, 2, 3, … 6)

Each mode is to be supplied with its respective parameters:

Mode 1: Profile Position Mode

a. Target position,
b. software position limit,
c. profile velocity,
d. max profile velocity,
e. profile acceleration,
f. profile deceleration,
g. quick stop deceleration.

Mode 2: Interpolated Position Mode

h. Interpolated time period,
i. software position limit,
j. quick stop deceleration,
k. profile deceleration

Mode 3: Cyclic Synchronous Position Mode

i) Torque offset,
ii) velocity offset,
iii) target position,

iii) software position limit,
iv) quick stop deceleration,
v) profile deceleration,
vi) interpolation time period

Mode 4: Homing Mode

vii) Homing Speed,
viii) Homing Method,
ix) Homing Acceleration,
x) Homing Offset

Mode 5: Profile Velocity Mode

xi) Target velocity,
xii) max profile velocity,
xiii) profile acceleration,
xiv) profile deceleration,
xv) quick stop deceleration

Mode 6: Cyclic Synchronous Velocity Mode

xvi) Torque offset,
xvii) velocity offset,
xviii) target velocity,
xix) quick stop deceleration,
xx) profile deceleration

Synchronization Mode

For an EtherCAT (CoE) operation, the synchronization mode can be changed by setting the Synchronous Control Register (ESC register 0x980 & 0x981). The two possible modes to consider are:

Free Running Mode
In this mode local cycle is independent from the communication cycle and master cycle. (DSPEtherCAT sets ESC register 0x980 to 0x0000)

Distributed Clock Mode
In this mode, the Yaskawa servo pack is synchronized with DSPEtherCAT on the Sync0 event. (DSPEtherCAT sets ESC register 0x980 to 0x0300)

ETHERCAT_INP1_REG, ETHERCAT_INP2_REG

IDENTIFIER Real-time status of ETHERCAT 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.

`ETHERCAT_INP1_REG`

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

`ETHERCAT_INP2_REG`

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

ETHERCAT_INP1_REG, ETHERCAT_INP2_REG cont.

EXAMPLE

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

```
wait_until(ETHERCAT_INP2_REG & 0x8000)
```

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

```
var1 =
while ( var1 == 1
)
if
(ETHERCAT_INP1_REG
& 0x0001)
var2=var2+1
 endif
wend
```

The following DSPL program segment assigns the contents of ETHERCAT_INP1_REG to var42:

```
var42                =                ETHERCAT_INP1_REG
```

ETHERCAT_OUTP_ON

FUNCTION

Sets ETHERCAT Outputs to 'On' State

SYNTAX

```
ETHERCAT_OUTP_ON (outp1,outp2)
```

USAGE

DSPL (PLC, Motion)

ARGUMENTS

`outp1`	`A 16-bit value coding the ETHERCAT outputs (0 to 15) to be set to ON. A bit set to 1 sets the correspondingETHERCAT digital output to 1. A bit set to zero leaves the corresponding ETHERCAT digital output unchanged.`
`outp2`	`A 16-bit value coding the ETHERCAT outputs (16 to 31) to be set to ON. A bit set to 1 sets the corresponding ETHERCAT digital output to 1. A bit set to zero leaves the corresponding ETHERCAT digital output unchanged.`

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

DESCRIPTION

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

SEE ALSO

`ETHERCAT_OUTP_OFF, OUTP_ON, OUTP_OFF`

APPLICATION

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

ETHERCAT_OUTP_ON cont.

EXAMPLE

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

```
ETHERCAT_OUTP_ON (0x000C, 0x8000)
```

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

```
var1 = 1
while ( var1 == 1 )
    ETHERCAT_outp_on(0xFFFF, 0xFFFF)
    delay (2500)
    ETHERCAT_outp_off(0xFFFF, 0xFFFF)
    delay(2500)

wend
```

ETHERCAT_OUTP_OFF

FUNCTION Sets the ETHERCAT Outputs to 'Off' State

SYNTAX

[Mx4 Octavia and Mx4 and Mx42 family]

```
ETHERCAT_OUTP_OFF (outp1,outp2)
```

USAGE DSPL (PLC, Motion)

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

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

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

DESCRIPTION

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

SEE ALSO `ETHERCAT_OUTP_ON, OUTP_ON, OUTP_OFF`

APPLICATION

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

ETHERCAT_OUTP_OFF cont.

EXAMPLE

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

```
ETHERCAT_OUTP_OFF (0x0006, 0x4000)
```

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

```
var1 = 1
while ( var1 == 1 )
     ETHERCAT_outp_on(0xFFFF, 0xFFFF)
     delay (2500)
     ETHERCAT_outp_off(0xFFFF, 0xFFFF)

delay(2500)
wend
```