PDO – Process Data Objects – CanOpen

November 7, 2015

Pages

Process Data Objects is a protocol for rapidly moving data from one node to another, data can be moved when it changes, at a preset interval or via the SYNC mechanism so multiple nodes can synchronize data gathering.

Process Data objects can be configured to move multiple object dictionary entries from one node to another. The mapping of which dictionary get end and where they end up is totally programmable. PDOs are a maximum of 8 bytes of data but in those 8 data bytes you can pick and choose which object dictionary values to send.

A simple example is you may have a device with multiple analogue inputs and if each analogue input is 16 bit you could potentially send 4 different channels at once via the PDO. Or you could send one channel and some other data that the node may have that is useful to your overall system.

The PDO (TX) protocol is very simple and is just :-

Can header	rtr	len	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x180 + node	0	1-8	Len bytes of data							

PDOs may have 1 byte of data or 8, What is contained in the data bytes and where it ends up is the devil of the detail. PDOs are configured by Object Dictionary entries and it is the understanding of these that is key to understanding and using PDOs. PDOs also are configured in pairs with a matching TX and RX PDO.

The TX PDO is controlled by two Object Dictionary entries 0x1Axx and 0x18xx. The RX PDO is controlled by two Object Dictionary entries 0x16xx and 0x14xx. The format of the 0x1Axx and 0x16xx entries are identical and also they need to be a matched pair between the transmitting and receiving nodes

The PDO configuration registers 0x1Axx and 0x18xx simply specify which object dictionary entries will be transmitted and received with the PDO packet. It is possible to specify any OD entries that you are interested in. On the receiving side you specify where the received data will be put and again you can specify any OD entries that are of the correct size to receive the data

Up two 8 configuration bytes are available, but the total amount of data they specify must be 8 bytes or less, so you cannot for example specify 8x32bit values to transfer but you can specify 8×8 bit values

Each configuration entry has the following format and is 4 bytes long

Index	Sub	Length
2 bytes	1byte	1byte

This just specifies which OD entry to send in the packet. the packet will be populated in the order specified in the 0x1Axx entry.

The complete format of the 0x1Axx and 0x18xx entries looks like

Number of elements (0-8)	Entry 0	Entry 1	Entry 2	Entry 3	Entry 4	Entry 5	Entry 6	Entry 7

So that 0x1Axx and 0x18xx is just an array of 32 bit values.

A quick example suppose you have some sensors reading analog input, The Canopen standard will find analog input 16 bit sensors in OD entry 0x6401 and individual channels will be in sub indexes 1-8. So lets say you are interested in sending Analog channels 1 and 2 back to the host node. And lets also assume the same node also has digital inputs that you are interested in and you wish to read the 1st digital input (8 bit) this will be found at 0x6000 Sub 1. Lets also say this is the first TX PDO we wish to configure. So your TX PDO entry for 0x1Axx would look like

Index	Number of sub indexes	Entry 1	Entry2	Entry3				
0x1A00	3	64010110	64010210	60000108	0	0	0	0

Note all values in hex following the previously defined format, of index,sub,length. So in this case we have defined a PDO with 5 bytes so 3 are spare if we need them later.

You also need to configure the RX PDO mapping on the receiving node in the same way (index 0x18xx) so lets once again say its the first RX PDO we are using so we can put the data in 0x1800 we could put exactly the same as the TX PDO mapping eg :-

Index	Number of sub indexes	Entry 1	Entry2	Entry3				
0x1600	3	64010110	64010210	60000108	0	0	0	0

This would have the effect of every time the PDO was transmitted the receiving node would get an exact copy of the 6401 Sub 1 and SUb 2 and 6000 sub 1 entries in its own object dictionary. Of cause you do not need to put the data in the same object dictionaries as the TX node had them, they can go anywhere, maybe you want to keep them in some custom entries at 0x3000

Index	Number of sub indexes	Entry 1	Entry2	Entry3				
0x1600	3	30000110	30000210	30010108	0	0	0	0

Here the two analog readings are placed in 0x3000 sub 1 and 0x3000 sub 2 and the digital reading in 0x3001 sub 1. Its all down to what you need to do and wish to achieve.

Two more registers are important with the PDO mapping and these are for the tx side 0x18xx and rx side 0x14xx. These registers control the COB-ID the TX and RX nodes act on as well as a number of parameters on the TX side that effect when the PDO is transmitted.

Sub Index	Description	Size
0	Number of sub indexes	Byte
1	COB ID	4 Bytes
2	Type	Byte
3	Inhibit time (ms)	2 Bytes
5	Event timer (ms)	2 Bytes

For the RX PDOs only sub index 0 and 1 are used. Its important to note that the COB ID of the configured RX and TX PDO must match. So either the RX or TX PDO must be configured with a different COB ID to the default or they will never talk to each other. I usually leave the TX in its default COB ID of 0x180+node id and change the RX PDO COB ID in 0x14xx sub 1 to match.

The Type field specifies the details of when the PDO is transmitted

Type	Description
0	Acyclic synchronous
1 – 240	Cyclic synchronous

Type	Description
241 – 251	Reserved
252	Synchronous RTR only
253	Asynchronous RTR only
254-255	Asynchronous

Types 254 and 255 are the simplest, these are transmitted when ever the value that the mapping parameter refers to changes, so in our example above if the analog input changed value a PDO would be transmitted with the new value (and also the other analog channel and the digital input info as well). The Inhibit time sub index can prevent this being updated too often and it limits the maximum update rate. The Event timer can also be used to make Asynchronous PDOs transmit automatically at a given rate when the timer expires. Giving a minimum update rate.

Cyclic Synchronous PDOs are transmitted with the SYNC message. In fact the SYNC message causes all nodes with Synchronous PDOs configured to sample their inputs and store the data, the actual data is transmitted on the next SYNC message. The SYNC message ensures that the sample time minimizes jitter between nodes as they all sample at the same time (with in limits) but there is lag to actually getting the data, so factor this in to any control loops. The Cyclic synchronous options 1-240 can be used to divide down when the PDO is transmitted so that a value of 2 is transmitted every other SYNC message etc. the device still samples on the SYNC message as before so sample lag is only ever SYNC interval behind, but the update rate can be divided down.

With Acyclic synchronous these only transmit when a Remote request from another device ‘pre-triggers’ the PDO or A device (profile) specific event ‘pre-triggers’ the PDO.

The modes involving RTR work the same way except Synchronous RTR requires a SYNC message and an RTR packet where as Asynchronous RTR only requires the RTR packet.