AN076: How to map objects to a PDO on a CANopen slave

Document Revision V1.00 • 2022-JAN-25

This appnote describes how CANopen objects are mapped to PDOs on a CANopen slave. It also shows how to change PDO definitions by changing the appropriate communication objects.

Contents

1 Introduction

With CANopen®, there are two ways of communicating with the objects of a slave: either via SDO or via PDO. SDO communication can be used with all objects at any time, but has a great amount of overhead. PDOs are a method of communication with mimimized overhead, but they have to be configured before they can be used. Mostly the factory default PDO definitons of a slave are sufficient, but sometimes it is desirable to change PDO definitions.

2 How PDO Mappings are defined

For defining PDO mappings, the PDO communication parameter objects and the PDO mapping parameter objects are used. These objects are located in the communication segment of the object dictionary. For each PDO there is a PDO communication parameter object and a PDO mapping object. The PDO communication parameter object defines overall parameters of the PDO, like its COB-ID or if the PDO is synchronous or asynchronous. The PDO mapping parameter object defines which objects are mapped to this PDO.

2.1 RPDO Communication Parameters

The RPDO communication parameter objects start at index 1400_h. Object 1400_h contains the communication parameters of RPDO #1. They contain the following sub-indices:

- 0: *Number of entries*: always 2.
- 1: *COB-ID:* Bits 10…0 contain the COB-ID to be used by this PDO. For RPDO #1 this normally is a value of 200_h+Node ID. Bit 30 is always set, as we do not support RTRs. Bit 31 enables or disables this PDO. The PDO is disabled if this bit is 1, and the PDO is enabled if this bit is 0.
- 2: *Transmission Type:* Setting this value to 254 or 255 makes this PDO be an asynchronous PDO. Setting it to a value between 1 and 240 makes it a synchronous PDO which means that the PDO will be processed after this number of SYNC messages have been received. Other values are not supported by Trinamic Motion Control modules.

2.2 TPDO Communication Parameters

The TPDO communication parameter objects start at index 1800_h. Object 1800_h contains the communication parameters of TPDO #1. They contain the following sub-indices:

- 0: *Number of entries*: always 5.
- 1: *COB-ID:* Bits 10…0 contain the COB-ID to be used by this PDO. For TPDO #1 this normally is a value of 180_h+Node ID. Bit 30 is always set, as we do not support RTRs. Bit 31 enables or disables this PDO. The PDO is disabled if this bit is set, and the PDO is enabled if this bit is not set.
- 2: *Transmission Type:* Setting this value to 254 or 255 makes the PDO be an asynchronous PDO. The PDO will then be sent each time the value of any object mapped to this PDO has changed. Setting it to a value between 1 and 240 makes it a synchronous PDO. This means that the PDO will be sent after this number of SYNCs have been received. Other values are not supported by Trinamic Motion Control modules.
- 3: *Inhibit Time:* After the PDO has been sent, it will not be sent again after this time has elapsed. It is given in 1/10ms. This parameter can only be changed while the PDO is disabled.
- 5: *Event Timer:* This value is given in ms. When set to a value greater than zero, the PDO will be sent each time this timer has elapsed (but also when the value of an object mapped to this object has changed).

2.3 RPDO and TPDO Mapping Parameters

The RPDO mapping parameter objects start at index 1600_h. TPDO mapping parameter objects start at index $1A00_h$. They contain the following sub-indices:

- 0: *Number of entries*: Contains the number of objects mapped to this PDO. With most Trinamic modules the maximum is 3, some also support a maximum of 4 mappings per PDO.
- 1…3: *Mapping Entry:* Each mapping entry defines a mapping. Bits 31…16 contain the index of the mapped object. Bits 15…8 contain the sub-index of the mapped object. Bits 7…0 contain the number of mapped bits. This must be dividable by 8 (so it can be 8, 16, 24 or 32).

3 Changing PDO Mappings

PDO mappings can be changed. This is done the same way for RPDOs and for TPDOs:

- Before a PDO can be changed, it has to be disabled. Do this by setting bit 31 of the COB-ID (Sub-index 1 of the PDO Communication Parameter object).
- Set the number of mappings of this PDO to 0. Do this by setting sub-index 0 of the PDO Mapping Parameter object to 0.
- Now the mappings of the PDO (sub-indices 1…3) can be changed. Write the desired values into these sub-indices. Use sub-index 1 for the first object that is to be mapped to this PDO. It will be mapped to the first bytes of the PDO.
- Set the number of mappings to the number of mapped objects (1…3).
- Now, the PDO can be enabled again. Do this by clearing bit 31 of the COB-ID (Sub-index 1 of the PDO Communication Parameter object).

4 Examples

Here we provide some examples that show how to set PDO mappings. For all examples we assume a Trinamic module with factory default settings and node ID 1.

4.1 Example 1: Change the Mappings of RPDO #3

In this example, we will change the mappings of RPDO #3. We will map the target velocity (60FF$_h$) and the state of the digital outputs (2703$_h$ sub-index 1) to this RPDO. Both objects are 32 bit integers, so together they will need eight bytes, which is the maximum length of a PDO.

1. We first have to disable RPDO #3. To do this, read object 1402$_h$ sub-index 1. Then, set bit 31 of this value and write it back to the same object. With default settings and node ID 1, the value read will be 40000401$_h$. With bit 31 set, this is C0000401$_h$ which has to be written to object 1402$_h$ sub-index 1.
2. Set the number of mappings to zero. Do this by writing 0 to object 1602$_h$ sub-index 0.
3. The first mapping shall be object 607A$_h$. The sub-index is 0 (as this object does not have sub-indices), and its length is 32 bits. So we will have to write a value of 607A0020$_h$ to sub-index 1 of object 1602$_h$. The value means object 607A$_h$, sub-index 0, 32 bits to be mapped.
4. The second mapping shall be object 2703$_h$ sub-index 1. This means to write 27030120$_h$ to sub-index 2 of object 1602$_h$. The value means object 2703$_h$, sub-index 1, 32 bits to be mapped.
5. Now set the number of mappings. As we are using two mappings, write 2 to sub-index 0 of object 1602$_h$.
6. The last step is to activate the RPDO. This is done by clearing bit 31 of object 1402$_h$ sub-index 1. Taking the value from step 1, we can write a value of 40000401$_h$ to this sub-index.

Now we can use the RPDO. With the value written in the last step, its COB-ID is 401$_h$. So to use this RPDO, we will have to send a CAN message with COB-ID 401$_h$ and a length of eight bytes. The first four bytes will be written to object 607A$_h$, and the last four bytes go to object 2703$_h$ sub-index 1. Remember that with CANopen the least significant byte (LSB) is always sent first. As an example we will write a value of 500000 (0007A120$_h$) to the target position and 00FF0000$_h$ to the state of the digital outputs. We then have to send (all in hexadecimal):

```
COB-ID: 401 Data:  20 A1 07 00 00 00 FF 00
```

4.2 Example 2: Change the Mappings of TPDO #1

In this example, we will put the status word (object 6041_h), the modes of operation disaply (object 6061_h) and the actual position (object 6064_h) onto TPDO #1. So our TPDO will consist of a 16 bit object, an 8 bit object and a 32 bit object. Together this makes 56 bits or seven bytes which fits into one PDO without exceeding its maximum length.

1. First we have to disable the PDO. This is done by setting bit 31 of object 1800_h sub-index 1. Read this value, then set bit 31 and write it back. Default settings assumed, we will read a value of 40000181_h and write back a value of $C0000181_h$.
2. Before we can change the mappings we will have to set their number to zero. Do this by writing 0 to object $1A00_h$.
3. The first mapping shall be object 6041_h. It is a 16 bit value and has no sub-indices, so we have to write a value of 60410010_h to object $1A00_h$ sub-index 1. The value means object 6041_h, sub-index 0, length 16 bits.
4. The second mapping shall be object 6061_h. It is an 8 bit value and has no sub-indices, so we have to write a value of 60610008_h to object $1A00_h$ sub-index 2. The value means object 6061_h, sub-index 0, length 8 bits.
5. The third mapping shall be object 6064_h. It is a 32 bit value and has no sub-indices, so we have to write a value of 60640020_h to object $1A00_h$ sub-index 3. The value means object 6064_h, sub-index 0, length 32 bits.
6. Now set the number of mappings. As we are using three mappings, write 3 to sub-index 0 of object $1A00_h$.
7. The last step is to activate the TPDO. This is done by clearing bit 31 of object 1800_h sub-index 1. Taking the value from step 1, we can write a value of 40000181_h to this sub-index.

Now this TPDO will be sent each time when at least one of the mapped objects has changed its value. The PDO will be sent using COB-ID 181_h and a length of seven bytes. The first two bytes contain the value of object 6041_h (LSB first), the third byte contains the value of object 6061_h and the last four bytes contain the value of object 6064_h (LSB first).

5 Other PDO Communication Parameters

The PDO communication parameter objects also contain other parameters that control the behaviour of PDOs. For RPDOs there is only the transmission type. TPDOs have the transmission type, the event timer and the inhibit time.
The transmission type controls if the PDO is an asynchronous PDO or a synchronous PDO. The event timer makes a TPDO being sent at least each time the time set there has elapsed. The inhibit time prevents a TPDO from being sent too frequently.

5.1 Synchronous vs. Asynchronous PDOs

Setting the transmission type to 254 or 255 (both values have the same meaning) makes the PDO be an asynchronous PDO. For RPDOs this means that the PDO data will be processed as soon as the PDO has been received by the slave. The received data will then immediately be copied to the mapped objects. For TPDOs this means that the TPDO will be sent each time the content of at least one of the objects mapped to it has changed.

Setting the transmission type to a value between 1 and 240 makes the PDO be a synchronous PDO. For RPDOs this means that the received PDO data will not immediately be copied to the mapped objects. Instead, this will be done after the number of SYNC messages given by this value has been received. This way, the point of time at which the data is being processed can be synchronized. For TPDOs this means that the TPDO will only be sent when the number of SYNC messages given by this value has been received. The TPDO will then be sent with the actual contents of the mapped objects.

5.2 Event Timer

Setting the event timer of a TPDO to a value >0 will make the TPDO being sent always when this time (in ms) has elapsed. But the TPDO will also be sent in between when at least one of the mapped values has changed. So setting this value to 100 for example will make the TPDO being sent at least every 100ms. This only works when the transmission type is set to asynchronous. The event timer value can be changed at any time.

5.3 Inhibit Time

The inhibit time prevents an asynchronous TPDO from being sent too frequently. The value is given in units of 1/10ms. Setting this value to 1000 for example means that after the TPDO has been sent it will not be sent again before 100ms have elapsed. This is useful when objects that change very frequently are mapped to an asynchronous TPDO. Analog inputs are a good example: Because of noise, such values change continually. Without using the inhibit time in such a case, the CAN bus will be flooded with this PDO, and it will hardly be possible to sent out any other CAN messages (it might not even be possible any more to send an NMT message for switching off PDO communication). Always set the inhibit time to a reasonable value when mapping such objects to asynchronous TPDOs. The inhibit time can only be changed while the PDO is disabled.

6 Other Hints

6.1 PDO Definition Tools

There are also tools for changing PDO defintions so that the steps described in sections 3 and 4 of this application note do not necessarily have to be done manually. The Trinamic TMCM-CANopen software for example also contains a tool called PDO Configurator. This tool does all necessary steps automatically. Please see the TMCM-CANopen user manual for instructions on how to use this tool. TMCM-CANopen can be downloaded free of charge from the Trinamic website.

6.2 Save PDO Definitions

PDO definitions can also be saved to the EEPROM of the module. This can be done using object 1010_h sub-index 2 (save communication parameters) or sub-index 1 (save all parameters). To do this, write the save signature (65766173_h) to the appropriate sub-index. After an NMT reset command or a power cycle all PDO defintions will be those that have been stored before.

Please note that older CANopen firmware releases for Trinamic modules do not support saving PDO definitions. In such cases you will have to udapte the module firmware in order to get this feature supported.

6.3 Which Objects can be mapped?

Not all objects can be mapped to PDOs. The CANopen firmware manual for the specific module and firmware version contains information about this for each object. Most objects that are only used during

the intialization phase (or used only rarely during the operation phase) are cannot be mapped to PDOs. Most objects that are used very often during the operation phase can be mapped to PDOs. Objects can either be mapped to RPDOs or TPDOs, not to both. This depends on whether the object is a read-only object or a read-write object.
The EDS file of the module also contains this information. The TMCM-CANopen software for example uses this information and presents it to the user in the PDO Configurator.

7 Revision History

Version	Date	Author	Description
V1.00	2022-JAN-25	OK	First release version.

Table 1: Document Revision