Vol 58, No 3—July 2024

ANALOG DEVICES | ADI Analog Dialogue™

A Deep Dive into the CANopen Protocol for Low Power, Industrial Motor Control

Atul Kumar, Applications Engineer

Abstract

A robust communication protocol and interface play a substantial role in industrial motor control applications. Where multiple processor elements are required to communicate continuously to accomplish complex tasks, CANopen® has emerged as the popular technique among engineers in industrial drive applications due to features such as easy integration. It is highly configurable, providing efficient, and reliable real-time data exchange. This article provides an in-depth understanding of CANopen from the perspective of low power motor control applications.

Background of Controller Area Network

Developed in 1983 by Robert Bosch Gmbh, Controller Area Network (CAN) is a highly robust communications protocol and interface. It was created to address the limitations of conventional serial communication networks, like RS232, which were unable to facilitate real-time communication between multiple controllers. The automotive industry was the first to adopt CAN, as it required continuous and simultaneous data transmission for multiple sensors. CAN allows multiple nodes to communicate with each other using small messages, making it ideal for automotive applications.

Over time, CAN gained popularity in various industries due to its proven robustness and benefits. However, integrating multiple devices from different vendors into a single system using the CAN protocol proved challenging and sometimes impossible due to proprietary coding rules. To overcome this limitation, international users of CAN in Automation (CiA) and manufacturers associations developed a high layer protocol called CANopen.

In the following section, we will explore the CANopen protocol architecture and its application in controlling a multi-axis motor driver. This article will delve into the intricacies of this high layer communication protocol and its impact on the motor and motion control domain. By analyzing a real-time communication log of ADI Trinamic™ TMCM-6212 multi-axis motor controller/driver module with the QSH4218-35-10-027 stepper motor we aim to provide readers with an understanding of the CANopen protocol. Specifically, we will focus on the network management (NMT) state and the client-server-based CANopen protocol. Additionally, case studies will be presented to demonstrate how to decipher the communication log and determine the status of the drive.

CANopen Architecture

This section of the article explains various application principles of the CANopen protocol, including NMT and SDO (service data object).

Network Management: NMT is a crucial communication principle in CANopen that every CANopen-compatible device must adhere to. It operates as a state machine and plays a vital role in coordinating applications within the CANopen framework.

Network Management State Machine Architecture: The NMT state machine is illustrated in Figure 1, and is composed of three distinct states as detailed in the following:

- Initialization state
- Preoperational state
- Operational state

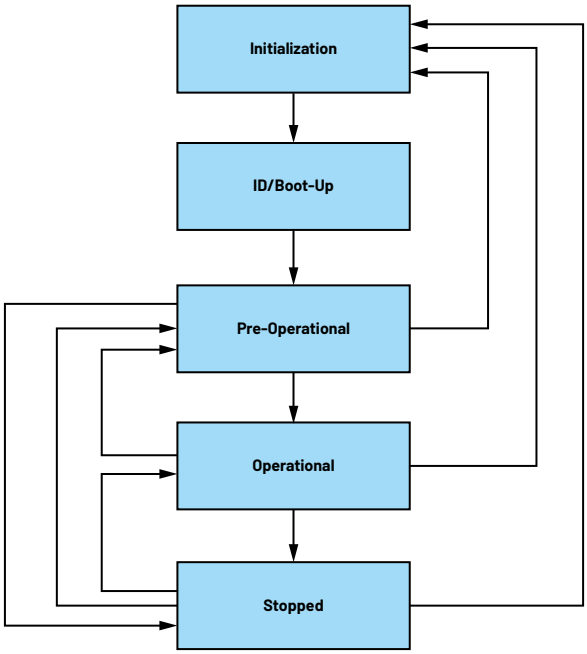

Figure 1. An NMT state machine.

The client node undertakes the pivotal role of overseeing the communication status of associated server nodes across different operational states. This is achieved through the implementation of the NMT mechanism. Two distinct methodologies, namely heartbeat and node guarding, enable the client node to assess the communication integrity of the server nodes. In the case of the TMCM-6212 the heartbeat technique is employed to validate proper communication. Each node emits a heartbeat signal at a user-configurable cyclic time interval, measured in milliseconds, utilizing the object 1017_h. This ensures that all nodes are active and alive for communication.

Table 1. State Configuration in NMT Communication

	Initializing	Pre-Operational	Operational	Stopped
Boot-Up	•			
SDO		•	•	
Emergency		•	•	
Sync/Time		•	•	
Heartbeat/Node-Guard		•	•	•
PDO (Process Data Object)			•	

Table 1 shows the combination of all the communication objects used in different communication states. When the device enters an initialization state after power-on or reset, it generates a boot-up message. The device then transitions to a preoperational state, where it is ready for the desired operation. In the preoperational state, all the nodes in the network can transfer all the objects related to SDO, heartbeat/node guarding, emergency, and time/sync. In the operational state, the PDO objects can be mapped in addition to all the objects available in the preoperational state. Lastly, in the stopped state the device disables the communication of all the SDO and PDO objects, allowing only NMT commands.

Service Data Object: The SDO communication protocol is mainly used in the preoperational state of the NMT state machine. It operates in a client-server configuration, in which the client can access all the objects available in the object dictionary of all connected servers (nodes). In this protocol, the client always initiates a read/write transaction with the server and the server acknowledges the completion of the task. This process ensures that every transaction in SDO is acknowledged.

Figure 2 depicts a client-server-based configuration for the SDO protocol in a multinode network. Each node is assigned a channel through which they can communicate with the client. In this case, the Trinamic TMCM-6212 sextuple stepper motor driver/controller acts as a server, and the connected PC serves as the client, initiating the read/write transaction with the specific node, that is, NODE-1 in this case. While all nodes receive the SDO client message, only the intended node will respond, while the other servers ignore the client request.

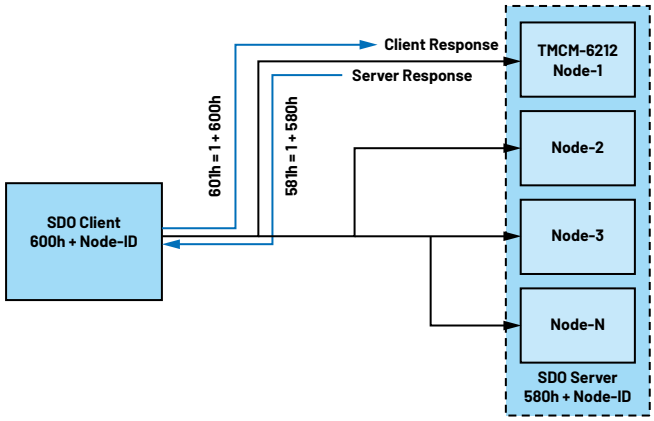

Figure 2. A multinode SDO configuration.

Service Data Object Datagram

Figure 2 illustrates the comprehensive structure of the SDO datagram. The SDO header consists of the COB-ID (connection object ID), which is a unique number assigned for specific tasks such as read and write functionalities. Therefore, two COB-IDs are required in SDO communication. The first COB-ID represents the NODE-ID+ function code for the client's upload/download request, which is 600_h + NODE-ID. The second COB-ID, 580_h+ NODE-ID, is used for the server's response.

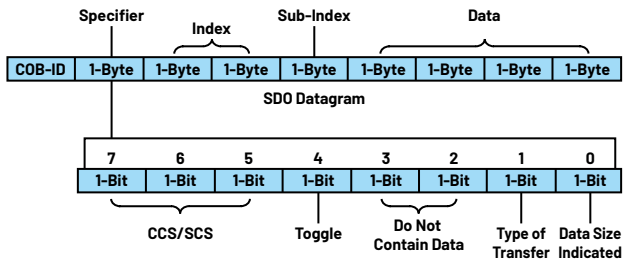

Figure 3. An SDO datagram structure.

The first byte in an SDO message, known as the specifier, plays a crucial role in determining the nature of the message. It indicates whether the client intends to write (download) or read (upload) the data and also signifies any errors in the transaction through abort messages. The specifier byte is divided into eight bits, which are shown in Figure 3. The three bits (7-5) known as the client command specifier (CCS) provide key information about the nature of the message. The client command specifier has different configurations depending on the client's operation, such as read, write, segmented/expedited transfers, or errors in transactions. In the server's response, the three bits of the specifier (SCS, sever command specifier) determine the success of the transaction. Table 2 shows the

various combinations of CCS and SCS bits for different operations. Bit 4 in the specifier datagram is a toggle bit used in data transfers exceeding four bytes. Bits 3-2 do not contain any data and are valid only if bits 0-1 are set. Bit 1 determines the type of message transferred through the SDO channel, indicating whether it is a segmented or expedited transfer. In the SDO datagram, as shown in Figure 3, the last four bytes are dedicated to the data that needs to be transferred. I If the data exceeds four bytes, it will be sent in a segmented manner. Alternatively, if the SDO datagram contains the complete data, it is considered an expedited transfer. Therefore, if bit 1 is high it indicates an expedited transfer, while a low bit indicates a segmented transfer. In the segmented transfer, the data is transferred in packets. The server responds to the initial read/write request from the client by providing the data size in the data field, and then the fourth bit (toggle bit) will start to toggle with the transfer of each data packet to the client. Lastly, if bit 0 in the specifier datagram is set, it indicates the data size in bits 3-2, as mentioned earlier.

Table 2. CCS and SCS Configuration

Operation	Client Request (CCS)	Server Response (SCS)
SDO Download	1	3
SDO Upload	2	2
SDO Download Segmented	0	1
SDO Upload Segmented	3	0

Bytes 2-3 and 4 in the SDO datagram correspond to the index and subindex bytes, respectively, as shown in Figure 3. These bytes are used to access all the objects available in the device's object dictionary. The object dictionary contains all the device parameters, allowing users to configure the device's functionality based on real-time application requirements. This concept of device profiling brings standardized behavior to devices, whether they are control devices like drives or a simple I/O components. The last four bytes in the SDO datagram are dedicated to the data that needs to be transferred through the SDO layer, as explained earlier.

In the event of an error, the SDO transmission will be aborted and the reason for the transmission stoppage can be identified by referring to the error code explanation provided in the manual of the target device. In this case, the CCS bits value is 4, the index and subindex specify the affected parameters in the device during the transmission, and the last four bytes indicate the error code.

Real-Time Communication Analysis

This section explains the SDO datagram using a real-time communication log window while the machine is in a pre-operational state. The ADI Trinamic TMCM-6212 sextuple stepper motor driver/controller[4] is used in conjunction with the QSH4218-35-10-027 [5] stepper motor. For this setup, maximum current of the motor (Object 2003_h) is set to 200. The upload and download transactions between client and server are further explained using the messages highlighted in the log window of the software interface of the targeted setup, as shown in Figure 4.

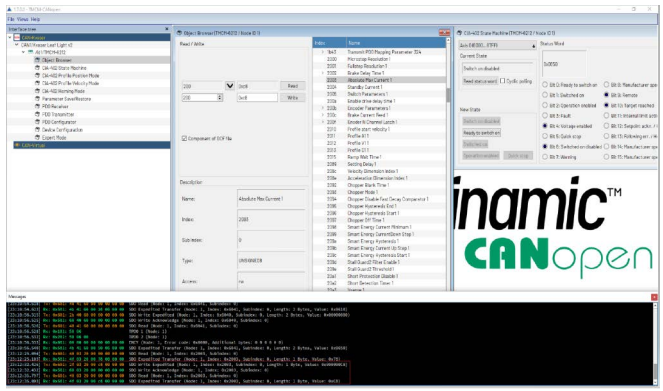

Figure 4. A CANopen IDE.

Case 1: Download Operation Between Client and Server

Initiated by the client: 0x601: 2f 03 20 c8 00 00 00 (Figure 5).

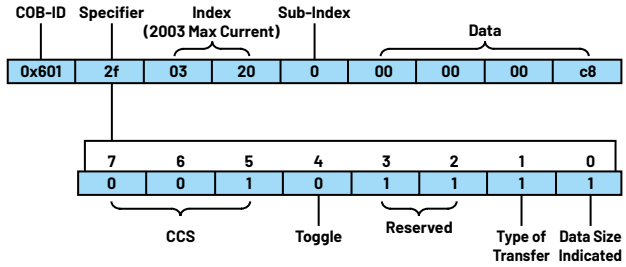

Figure 5. An initiate download request by the client.

Response by the server: 0x581: 60 03 20 00 00 00 00 (Figure 6).

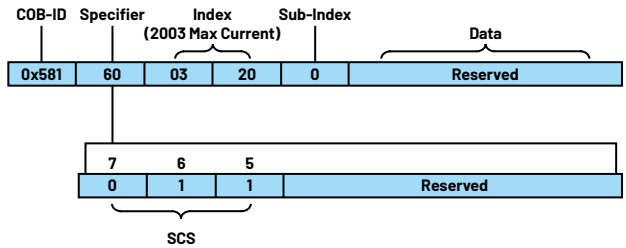

Figure 6. A download response by the server.

In the operation shown in Figure 6, the combination of CCS and SCS bits shows the successful write operation from the client and the server's response, also seen in Table 2.

Case 2: Upload Operation Between Client and Server

Initiated by the client: 0x601: 40 03 20 00 00 00 00 (Figure 7).

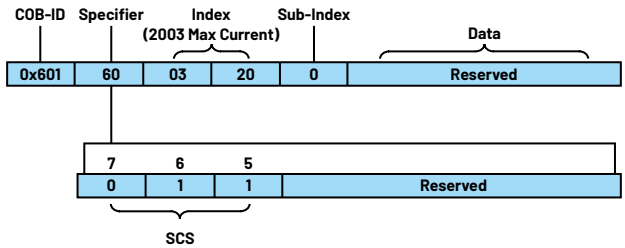

Figure 7. An initiate upload request by the client.

Response by the server: 0x581: 4f 03 20 00 c8 00 00 00 (Figure 8)

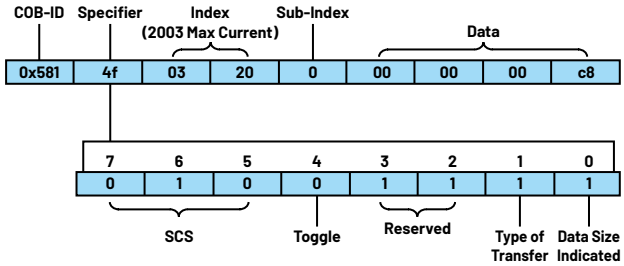

Figure 8. An upload response by the server.

Conclusion

The combination of CCS and SCS bits indicates the successful upload operation between the client and server. The examples mentioned in this article can be applied to other objects in the device's object dictionary, providing insights into the state of the machine. The main objective of this demonstration is to empower users to decipher the communication log and monitor the drive's status. Users can troubleshoot errors in real time and explore the advanced features of ADI Trinamic CANopen more efficiently. The integration of CANopen protocol in ADI products offers customers the flexibility to integrate their own PLC's with ADI Trinamic modules, enabling the development of multivendor systems. This interface is particularly valuable for customers working on complex applications such as, lab automation, robotics, liquid handling, semiconductor handling, and more. The next article in this CANopen series will cover the in-depth analysis of the process data object (PDO) CANopen protocol while exploring the TMCM-6212's more advanced features for motor control applications.

References

Olaf Pfeiffer, Andrew Ayre, and Christian Keydel. "Embedded Networking with CAN and CANopen." Copperhill Technologies Corporation, 2008.

"TMCM-6212 CANopen Firmware Manual." Trinamic Motion Control, 2018.

About the Author

Atul Kumar is an applications engineer in Central Applications Dublin. His primary expertise is in motor control, closed-loop control architecture for low power stepper motors, and BLDC/PMSM motors. He has done his postgraduate studies in Dublin City University and joined Maxim Integrated (now a part of Analog Devices), as an associate application engineer in February 2022.