TECHNISCHE UNIVERSITÄT
CHEMNITZ

Designing and Development of a Data Logging and Monitoring Tool

Master Thesis

Submitted in Fulfillment of the

Requirements for the Academic Degree

M.Sc. Automotive Software Engineering

Dept. of Computer Science

Chair of Computer Engineering

Submitted by: Habib Ur Rehman

Student ID: 331342

Date: 27.07.2016

Supervisors: Prof. Dr. Wolfram Hardt

Dipl.-Inf. Stephan Dörr

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Declaration of Authorship

I, Habib Ur Rehman, assure that the thesis “Designing and Development of a Data Logging and Monitoring Tool” is my own work under the guidance of my technical supervisors at Torqeedo GmbH. The data collected during the literature review and referred in this document is given due acknowledgment. All the references and helping materials are enlisted in the Bibliography with all sincerity.

Signature: _____________________

Date: _________________________

Abstract

Since the mid 90's computer communication has become more and more common in cars and other auto mobiles. CAN based networks with sensors transmitting small data packets are utilized in the automotive industry to operate and supervise vehicles' functionality. To ease communication several higher layer protocols for CAN based networks have been developed. In some applications it is necessary to exchange information between networks using different protocols, and by connecting the two networks to a gateway, the information is translated and forwarded and intercommunication is enabled. This master thesis is conducted at Torqeedo GmbH, Munich. Theme of the thesis was “Designing and Development of a Data Logging and Monitoring Tool”.

Term “data logging” refers to the gathering or collection of specific data over a period of time. Monitoring means evaluate the data we are logging. Tools for data logging and monitoring are used in variant application these days. In medical, in-vehicle data logging and environment monitoring. This data could be voltage, current temperature, Time stump, heartbeat of the patient, vehicle fuel level etc. To capture and log data various communication channels used. Such channel varies from simple data cable to satellite link. There are variant protocols used for different communication channels. For our DBHS logging and monitoring tool we are using CANopen protocol. Main goal of this thesis is to develop a tool which can make debugging easy and log connection box data so we can use logged data later on for offline data analysis and simulation purposes.

Keywords: Protocol Analyzer, CANopen protocol, CAN communication

Acknowledgements

In the name of Allah, the Most Gracious, the Most Merciful.

I cannot thank enough to my company supervisor at Torqeedo Gmbh, Mr. Stephan Dörr who was abundantly helpful and offered invaluable guidance throughout my project. Indeed something as complex as Master thesis could not have been completed without his valuable suggestion and dedicated time, his response always in form of "hmm yes" to my question "whether he has time?" kept me motivated throughout my project.

I would also like to take this opportunity to express my gratitude to Michael Lorenz and Mario Wohlwender for introducing me to this topic as well as their support and guidance all the way.

Moving to the academic side, my Professor, Dr. Wolfram Hardt contributed to the development of my skills even before my thesis had started. These skills certainly helped me better understand the engineering problems and taught me the art of solving them.

Mr Felix Hänchen had been coordinating with me throughout the thesis as internal supervisor. His hints on my documentations and concept presentations were beyond helpful. Thanks for your time and effort.

There were two aspects of support during the course of my thesis; practical and moral. Having mentioned the practical support, and beginning with the moral support, I would like to thank my parents and siblings. Though they were miles away from me, their prayers reflected in the success. It was their moral support that allowed me to keep this commitment, yet meet all the deadlines.

Table of Contents

List of Abbreviations

Abbreviations	**Full form**
AF	Acceptance Filtering
AMP	Arbitration on Message Priority
ACK	Acknowledgment
CAN	Controller Area Network
CiA	CAN in Automation
COB-ID	COB Identifier
COB	Communication Object
CRC	Cyclic Redundancy Check
CSMA	Carrier-Sense Multiple Access
CSMA/CD	Carrier-Sense Multiple Access / Collision Detection
DLC	Data Length Count
DBH	Deep Blue Hybrid
DBHS	Deep Blue Hybrid System
ER	Error Register
ECU	Electronic control unit
EEC	Emergency Error Code
EMCY	Emergency
HB	Heartbeat
IDE	Identifier Extension
ISO	International Standardization Organization
LLC	Logic Link Control
MAC	Medium Access Control
MSB	Most Significant Bit/Byte
NMT	Network Management

OD	Object Dictionary
OSI	Open Systems Interconnection
PDO	Process Data Object
RPDO	Receive Process Data Object
RTR	Remote Transfer Request
SDO	Service Data Object
SSDO	Server SDO
SYNC	Synchronization Object
TPDO	Transmit Process Data Object
UART	Universal Asynchronous Receiver/Transmitter

List of Figures

List of Tables

Code Listings

Chapter 1

1. Introduction

Communication is an exchange of information between two or more Objects. There are several methods and means for exchanging this information. Humans can exchange information by variant different means. Now in our world machines are continuously replacing human due to several reasons. And their role in today's industry is increasing day by day. There are many jobs in today's fast growing industry that only machine can do precisely, accurately with speed. We know that while working humans need to communicate and exchange information with other coworker's while working at workplace. The question comes in our mind that when we replace humans with machines than how this exchange of information between machines would be possible? In communication among two people, one participant is transmitter and second is receiver. For exchange of information among machines we must keep in mind that both participants of communication can understand each other. So there had to be common standards. One option to make exchange of information among machines is communication protocol.

Communications protocol is a set of rules for data exchange, which allows two or more entities of a communication network to exchange information over a communication channel.[1] These are standard that defines the syntax, semantics and synchronization of communication and possible error recovery methods. Protocols may be implemented by hardware, software, or a combination of both. If you want to design your own protocol according to your specific requirements you can do this, just following some basic procedural principles. Your communication protocol needs to be easy to interact if some IT person wants to implement it or use it. Further it should be reliable for detecting errors and resilient which means less network failure.

1.1. Motivation

In past three decades automotive industry has been in the transition from simple point to point wiring to highly complex network topologies and communication bus systems. This is because the customer is demanding more and more sophisticated features in the vehicle to make it more convenient for traveling. And on the other hand there is a high competition in market among all manufacturers. In these three decades all the previous standards upgraded and many new standards introduced. In these new standards there are few standards used to reduce the complexity of inter component communication in any system. In 1980 Robert Bosch Gmbh introduced the CAN (Control Area Network) also known as CAN bus. CAN-bus gradually make place in the automotive industry and nowadays CAN-bus is going to be used in variant industries e.g. in Aerospace, in Medical field, in Industrial control and similar. Today each vehicle has around 100 ECUs and this trend is gradually growing with time because of the addition of new features. [20] According to OSI communication system model CAN covers the Data link layer and Physical layer out of 7 OSI communication model layers. CANopen is a higher level communication protocol which covers the top 5 layers. It is developer by CiA (CAN in Automation). CANopen standardized communication between devices and Application of different manufacturers [4]. To analyze the performance of the protocol we need a protocol analyzer to check the performance of the CANopen protocol. Protocol analyzer increases efficiency and improves the performance of the network. There are several open source CANopen protocol analyzers e.g. CANFestival.

1.2 Overview

This thesis is structured as follows; the first chapter gives a brief introduction, motivation, objectives. The second chapter provides the basic knowledge to understand the work, which I will explain in coming chapters. It contains basic information of CAN bus and what CAN and CANopen protocol is and why we need these?. In last part of chapter 2 there is a brief over view of deep blue Hybrid system. The third chapter is all about “State of the Art”. First, we will take the overview of CAN bus analyzer. Later on we will introduce problem and our approach to solve this problem. Later on I will introduce some commonly known open source solutions. The 4th Chapter is Concept which introduces readers with the problem for which we are looking for solution. To understand the problem readers must know basics about the system. I will introduce the. The Fifth Chapter starts the Implementation of our concept phase outcome. Actually our implementation is in Three Phases. I will overview all of them one by one. Phase_1 is Online Monitoring tool. Second phase is Offline Analysis and Phase 3 is embedded platform. Second last chapter tells the reads what is the outcome of our implementation and there would be a cross check of our results and expected results. Last chapter is the guidance for future work how we can make our system more result oriented and detect protocol violations in minimum time frame and reasons behind these violations which leads to failures more quickly.

For the sake of readers comfort and make it easy to read following modifications are adopted. Content of the chapters is divided into 3 levels at most. The names of tools, files and folders are in italic. Full forms of the abbreviations are in separate table in the beginning of the document. And first usage of all the abbreviations is written with its full form in brackets ( ) e.g. PDO (*Process Data Object*). All the quotations inside the document would be in "quotation marks".

1.3 Objective

Main objective of this master thesis is to design and develop a logging and monitoring tool for our deep blue hybrid system CANopen network. This tool will help us to play back our logged communication data to debug network. We are using customized CANopen protocol as per our project requirements. The very basic requirements are to analyze network data communication and keep an eye on CAN bus utilization. Our goal is to be able to calculate the payload of all the devices of the network. Payload analysis of each node helps us for finding errors/problems in network and monitoring the behavior of each node while communication with other nodes of the network.

Lastly it is important for to evaluate logging and monitoring tool on Linux based embedded platform as well. Evaluation of tool on embedded platform is valuable for future work to expand the tool efficiency.

Chapter 2

2. Basics

This chapter describes the basics which help the readers to understand the concept and implementation of the project. This chapter will give readers the good insight and basic understanding of CAN bus and CANopen Protocol. In the first part of this chapter our focus would be CAN-bus. In the last part of introductory chapter, High-level protocol CANopen working and key objects of protocol will be discussed.

2.1 CAN Introduction

CAN is an ISO (International Standardization Organization) standardized serial communication bus firstly introduced in early 1980's by a German company Robert Bosch GmbH specifically for automotive applications. But nowadays it is being used in various industries. It is being used by industrial control, medical equipment and electric boats. CAN is a most common communication channel. As embedded components communication channel, its cost efficient, highly reliable and also has high data transmission rate [6]. The primary goal of introducing CAN bus was to replace complex wiring system inside the vehicle with single CAN bus. Bus was standardized by ISO in 1993 as ISO 11898-1.

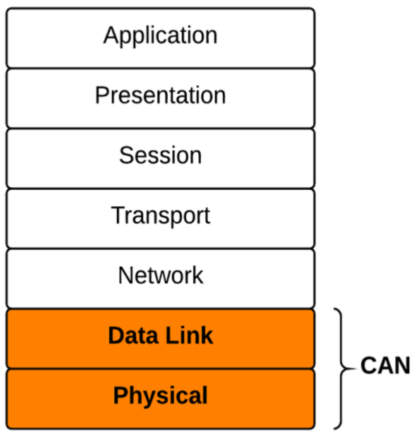


Figure 1. OSI 7 layer communication system model [5]

CAN bus cover the last two layers of the OSI (Open Systems Interconnection) 7 layer Model. Last two layers are physical layer and Data Link layer. Physical layer defines how bits are encoded into signals to transmit over network from one node to other [23]. Physical layer consists of three sub layers:

- Physical Signaling
- Physical Medium attachment
- Medium dependent Interface

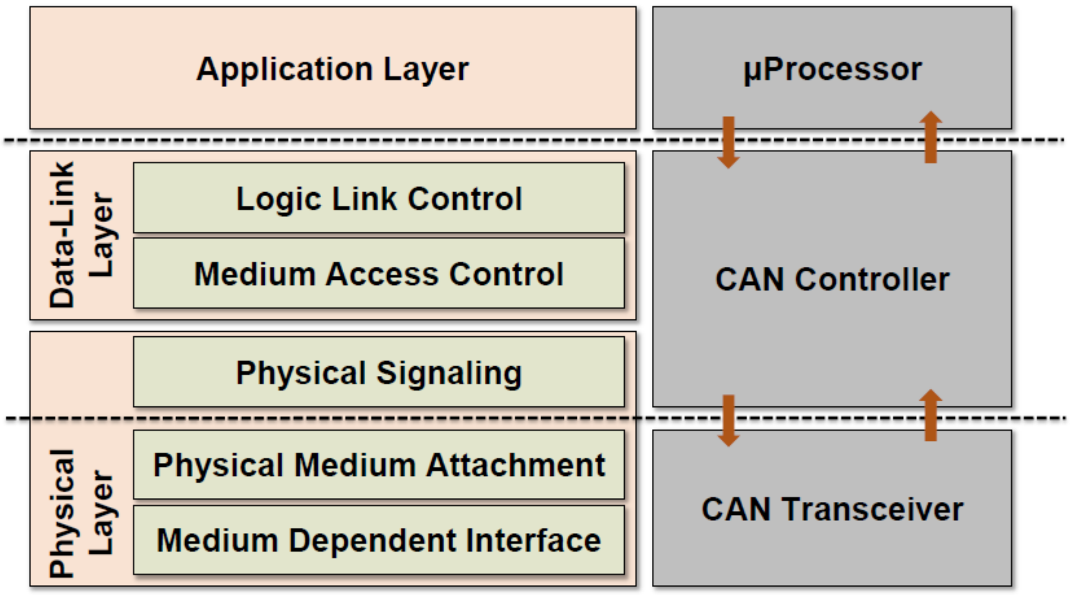


Figure 2 Layered ISO 11898 Standard architecture [8]

Data-Link layer consist of two sub layers:

- Logic Link control
- Medium access control

Logic Link Control (LLC) do message filtering and recovery management. On the other hand, Medium Access Control (MAC) is responsible for bit stuffing which means the insertion of one or more bits into a data packet. After five consecutive identical bits, one extra bit is inserted into the message and this bit has a compliment of the previous five consecutive bits. The receiver of the message also knows how to detect extra inserted bit from the message. MAC also guarantee that message with higher priority (low identifier) will be transmitted first all the low priority messages need to wait until the bus is free after finishing the transmission.

CAN bus describe how data is transferred between all the nodes connected to the network [7]. Its communication is broadcast based all nodes can transfer and receive messages. When the master node transmits the message, all the nodes read the message and decide either it needs to accept it or not. This method of filtering is called Acceptance Filtering (AF) [6].

2.1.1 CAN Frame Format

CAN communication is Carrier-Sense Multiple Access / Collision Detection (CSMA/CA). CSMA/CD is a modified form of Carrier-Sense Multiple Access (CSMA). Each node on the bus waits for some time before transmitting frame to avoid collisions. Collision can be resolved through bit-wise arbitration called Arbitration on Message Priority (AMP). Message with higher priority wins the access of the channel [7]. CAN bus have two standard formats:

- Standard CAN with 11 bit identifier
- Extended CAN with 29 bit identifier

The Standard CAN has 11 bit identifier which would be unique within the network and determine the priority of the message. Node on the Bus with lower identifier has higher priority and node with higher identifier has lower priority to access the CAN bus.

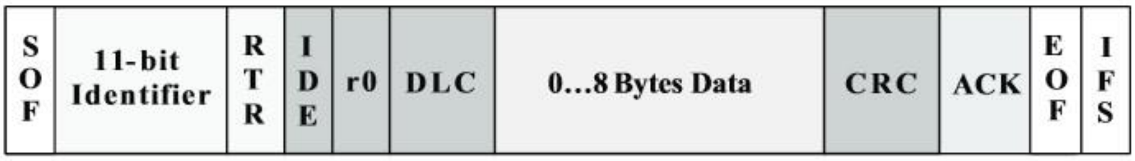


Figure 3. Standard CAN 11 bit Identifier [7]

Structure of the CAN standard frame (CAN 2.0 A) is given in the figure 3 above. Message frame always starts with 1 bit SOF (Start Of Frame). SOF bit would always be dominant and indicates the start of the frame. It always synchronizes all the nodes on the bus. There is 11 bit identifier after SOF bit. Followed by identifier bits there is one bit RTR bit (Remote Transmission Request). RTR bit has to be dominant in data frame otherwise recessive. IDE (ID Extension) identifies a standard frame or message. R0 is reserved bit. 4 bit DLC (Data Length Code) contains the information about the data section of the frame. Value of DLC would be between 0 and 8, because data section of CAN frame contain maximum 8 bytes of data. The next one after DLC is actual data field which could vary from 0 to 8 Byte. One CAN message frame contain maximum 8 bytes of data. After data section next is 15 bit (Cyclic Redundancy Check) CRC. CRC is used for the checking of the transmitted data. The receiving bit overwrite the ACK (Acknowledgments) bit with dominant to make sure to the sender message received successfully otherwise transmitter retransmit the message frame. There is a 3 bits space between two CAN message frames called IFS (Inter frame space).

The message identifier of Extended CAN (CAN 2.0 B) is 29 bit long. This is divided further into two parts. 11 standard identifier and 18 bits extended frame identifier [6]. IDE bit is used to determine the priority of the message when multiple notes try to get access of the bus and transmit their messages frames. Priority of 11 bit identifier is higher than the 29 Bit Identifier. 1 Bit IDE would be 1 in case of extended frame and 0 when frame is with 11 bit identifier.

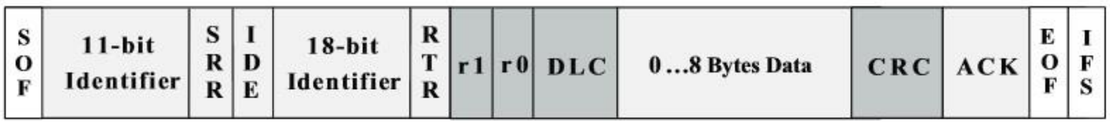


Figure 4. Extend CAN with 29 Bit Identifier [7]

2.1.2 Standard CAN Vs Extended CAN Frame Format

Standard CAN allow 2048 different messages on the other hand extended frame allowed message are 536870912 messages. Standard CAN 2.0 and Extended CAN with 29 bit identifier both exist in the same CAN. During bus arbitration Standard CAN message has higher priority over 29 bit extended frame. Extended frame CAN always has more latency than standard frame. It also need 20% extra bandwidth comparatively to the bandwidth required for standard CAN. Overall reliability of message decreases because in CAN 2.0B (Extended frame) CRC need to check extra 18 bits in each message.

2.2. CANopen at Glance

CANopen is high-level communication protocol which is based on CAN protocol. CANopen is “open” in three different ways. The first and foremost important is it does not require any payment or any license fees to use. So it is open for everyone. The second feature of CANopen is that it can easily be customized as per your application requirements because it has a small set of functionality which is mandatory and a huge set of functionality is optional. So implementing own application specific functionality after configuring the mandatory functionality is easy. [9] CANopen data transmission follows the “Little Endian” rule this means lowest value byte always transmitted first and high value data byte transmitted in the end.

If talking about OSI communication model, CAN protocol works on first two layers of the mode, Physical layer and Data link layer. [4] CANopen covers all the top five layers. Their names are as following:

- Network layer
- Transport layer
- Session layer
- Presentation layer
- Application layer

2.2.1 Advantages of CANopen Protocol

There are many technical advantages which offered by CANopen protocol to make CAN network more resilient and flexible. Some of these technical advantages given below [3]:

- Configuration flexibility
- Prioritization of messages
- System wide data consistency
- Multi cast reception
- Error detection & error signaling

CANopen has some very confusing or even sometimes conflicting technical terms. These interrelated terms can confuse someone easily especially if readers have their first interaction with CANopen. First term is "identifier" All the IDs referred to as identifiers there are different kind of IDs in CANopen: [9]

- Node ID
- Object Dictionary index
- COB ID

Node ID is used to identify each node in the network. Object Dictionary index is used to identify specific object in the device object dictionary. COB ID is used to identify each CAN message on the network.

Second confusing term of CANopen is "Object" which also has variant meanings within CANopen. Object could be Process Data Object, Connection Object, Service Data Object or object of object dictionary. Detail explanation of these topics will be coming chapters.

2.3 CANopen Communication Model

There are 3 different possible communication models in CANopen. Each communication objects of CANopen use one of these 3 communication relationships: [11].

- Client/Server Relationship
- Master/Slave Relationship
- Producer/Consumer Relationship

2.3.1 Client/Server Relationship

Client/server relationship is always need two participants. One of them is a client, who requests some data and second participant is server to perform the requested operation. One device in the network can be client and server simultaneously. SDO follow this relationship. SDO client request for required information and SDO server respond to SDO client with the requested piece of data.

2.3.2 Master/Slave Relationship

In CANopen network there is one master node and all the others are slaves. This master slave relationship could be one to one but mostly one to many (one master many slaves). Master node controls the slave nodes.

2.2.3 Producer/Consumer Relationship

This relationship is also one to one or one to many. Producer node pushes the data over the network and there could be one or many receivers of this data. The entire CANopen communication objects except SDO and NMT follow producer consumer relationship model.

2.4 CANopen Device Model

CANopen is divided into three main parts:

- Communication interface
- Object Dictionary
- Application process

The communication interface provide the services of transfer and receive communication objects over the CAN bus. Object dictionary contain configuration and process data in tabular form. Object dictionary is also the interface to the application process. Application process comprises device functionality according to the requirements of interaction with external process environment. [12]

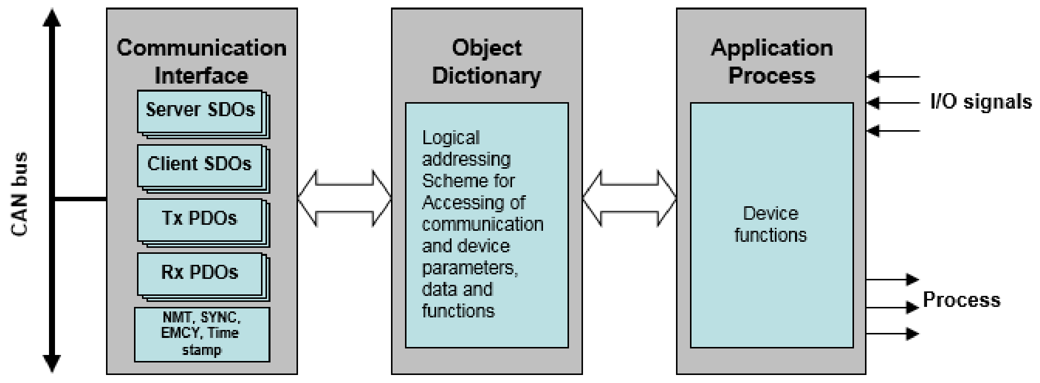


Figure 5. CANopen device model [13]

2.4.1 Object Dictionary

The Object Dictionary (OD) is a table which contains all the data and their meanings, this data could be process data or configuration data. This is the mandatory functionality of CANopen. All the nodes must implement their own Object Dictionary. OD holds the functionality of the CANopen node and any other node of the network node can write and read the functionality of the node into its object dictionary. Object dictionary contain data in table structure. All information of node is in the OD entry and all the entries have their identification number called index. The index is 16 bit is size. Maximum numbers of entries in OD are 65,536 (2^{16}). Each index of the object dictionary has 8 bit sub-index.

Index Range	Index Description
0000h	Reserved
0001h - 0FFFh	Data Types
1000h – 1FFFh	Communication Entries
2000h – 5FFFh	Manufacturer Specific
6000h – 9FFFh	Device profile
A000h - FFFFh	Reserved

Table 1. Object Dictionary domain and their ranges [9]

There are maximum 256(2^8) Sub entries of each OD entry. All the entries of OD must have minimum one sub index because the entries contain only one value have one sub index at 00h. All the indexes in the object dictionary are divided into 6 sections. All the sections have their index ranges as mentioned in Table 1.

Node can read the object dictionary entries of the other node in the network. With this method one node of the network can get the data of one node by accessing its object dictionary entries. Object Dictionary can store pre-defined and manufacturer specific data types.

Index	Object Type	Entry Name	Data Type	Action
1000h	Variable	Device Type	UNSIGNED32	RO
1001h	Variable	Error register	UNSIGNED8	RO
1017h	Variable	Producer HB time	UNSIGNED16	RW
1018h	Structure	Identity Object	IDENTITY	RO

Table 2. Mandatory object dictionary entries [11]

2.5 CANopen Communication Objects

CANopen have several communication objects. These communication objects play the key role while implementing the desired behavior of network. [14]. Key communication objects of CANopen are as following:

- Process data object
- Service data Object
- Network management
- Timestamp
- Emergency messages
- Synchronization Object

2.5.1 Process Data Object

Most of CANopen application SDO is not enough to exchange the real data; It need some method with less overhead to access data. PDO gives the fast way to share the process critical data without any additional protocol overhead. Both producer and consumer know how to interpret the data of PDO. Because of this COB-ID is enough to recognize the PDO [10]. Basic principle of data transmission is “Broadcasting” means one node produce and all the other nodes can consume. This data could be like input from Weather sensor. Which send the weather information after a specific period of time. PDO consist of one CAN frame with maximum 8 Bytes of payload data. PDO have two type of parameters first one is PDO communication parameters and the second one is PDO mapping parameters. PDO mapping tells us which individual variable (process variable) are going to transmit and how data is arranged which data type and length they have. PDO data is stored into the object dictionary. There are two types of PDOs:

- Transfer Process Data Object
- Receive Process Data Object

Transfer Process Data Object (TPDO) is the data which is coming from the data producer. Receive Process Data Object (RPDO) is the data coming to the data receiver in other words called data consumer. Parameters of PDO are also of two types. Either they would be configuration parameters or mapping parameters. Mapping parameter tells which object will be transmitted with which PDO and configuration parameters specify the type of PDO transmission. There are several methods to initiate the PDO transfer. These methods are given below:

- Even driven PDO transmission
- Time driven PDO transmission
- Remote request transmission
- SYNC transmission

PDO transmission after a specific period of time is called synchronous data transmission. In event driven, PDO transmission is linked with the event. When the particular event occurred or specific condition satisfied, for example when object value altered, than PDO transmission occurred. In Remote request transmission method PDO transmission can be initiated from consumer node with RTR bit of message frame. The Fourth method of PDO triggering is linked with the SYNC object. PDOs are only transmitted when a synchronous object is received. In this method, PDO transmission is synchronous in the whole network.

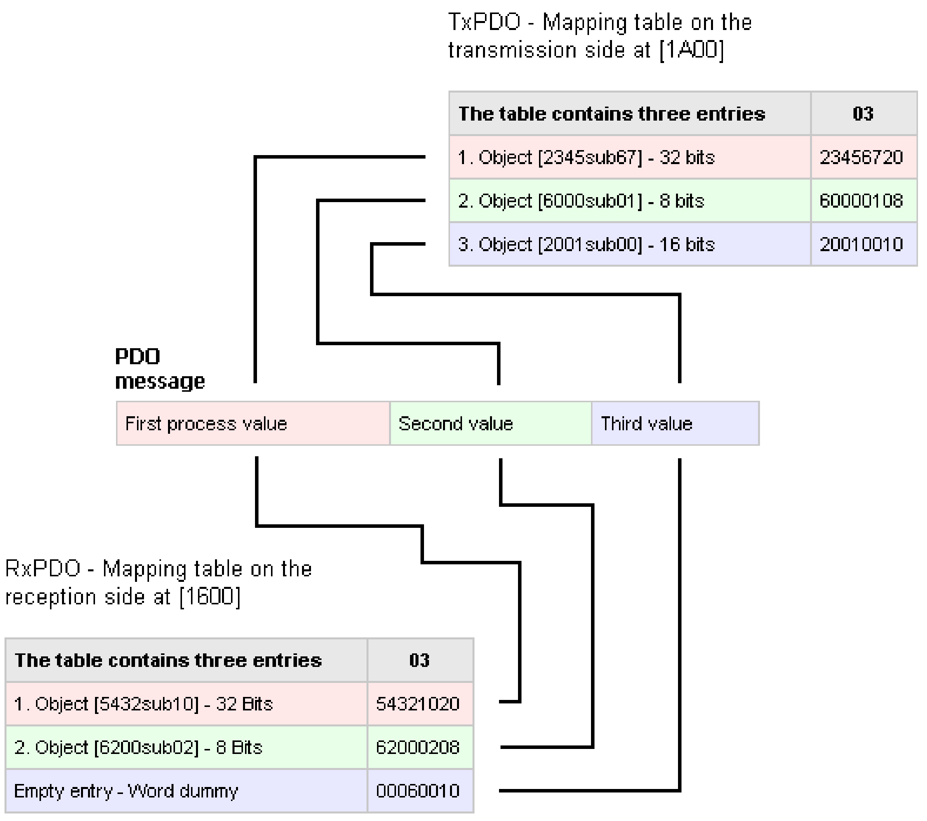

TxPDO - Mapping table on the transmission side at [1A00]
The table contains three entries
03
1. Object [2345sub67] - 32 bits
23456720
2. Object [6000sub01] - 8 bits
60000108
3. Object [2001sub00] - 16 bits
20010010
PDO message
First process value
Second value
Third value
RxPDO - Mapping table on the reception side at [1600]
The table contains three entries
03
1. Object [5432sub10] - 32 Bits
54321020
2. Object [6200sub02] - 8 Bits
62000208
Empty entry - Word dummy
00060010

Figure 6. Event driven PDO Transfer [4]

The basic principle of PDO mapping is illustrated in the figure 6 above. The entire process variables are in the object dictionary as OD entries and the mapping of each process variable of a PDO. In example above there are two objects. First object is from PDO producer index 2345 sub-index 67 to the PDO consumer of index 5432 sub-index 10. Second object is from object index 6000 sub-index 01 to the consumer index 6200 sub-index 02.

Sub-Index	Contents	Data type
0	Largest sub index	BYTE
1	COB-ID	DWORD
2	Type	BYTE
3	Inhibit time	WORD
5	Event timer	WORD

Table 3. sub index data types

2.5.2 Service Data Object

Device configuration Objects commonly known as Service Data Object (SDO). They are mostly used for the direct access of the devices on the network. SDO can write and read the object dictionary entries. CANopen always make sure that each device of the network must have a server. That can perform the read and write operations on the object dictionary of the device. SDO use the client server communication. Device whose object dictionary is accessed is called SDO server and the device which access the object dictionary of the SDO server is called SDO client. Communication is always initiated by the SDO client. There is always a peer to peer communication between network devices. Where one node is acting as a SDO server and second one is SDO client.

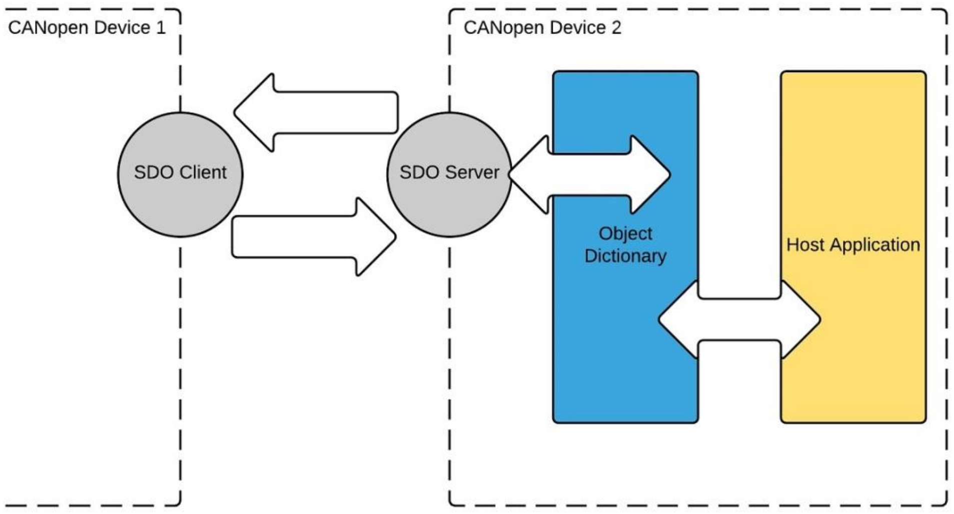


Figure 7. SDO client and SDO server communication

Data transfer is carried out with acknowledgment service which always takes two CAN messages per transfer. First CAN message would be SDO request transmitted from SDO client to the network. Than network targeted node (SDO Server) respond with requested object with CAN message.[10] Data of any length can be transmitted if data is more than 8 bytes it would be divided into segments and one segment per CAN message frames that's why theoretically SDO has ability to transfer unlimited data. "SDO download" service starts when SDO client request the SDO server to download data from object dictionary. "Initiate SDO download request" message is used to start the SDO download. Message contains full address of object address in form of index and sub-index to download from object dictionary. Server responds with "initiate SDO download response" to SDO client as confirmation.

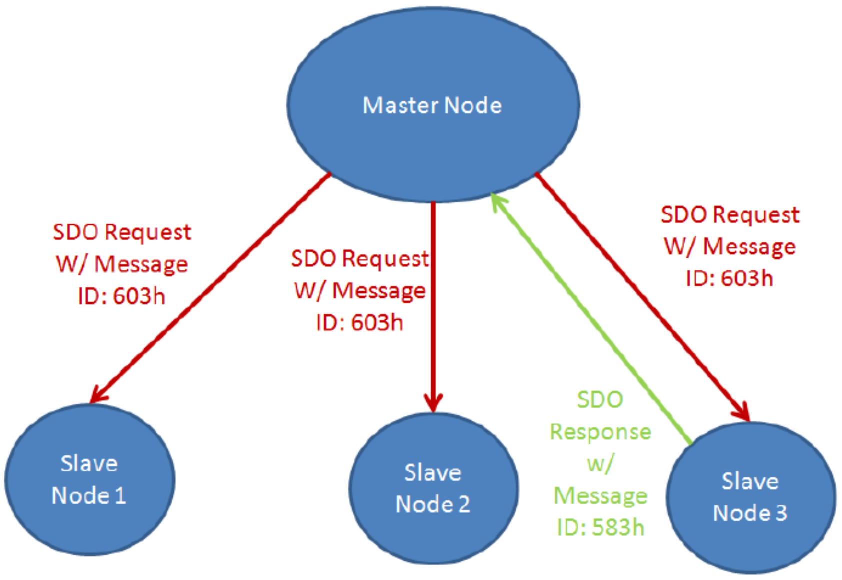

Master Node
SDO Request W/ Message ID: 603h
SDO Request W/ Message ID: 603h
SDO Request W/ Message ID: 603h
SDO Response w/ Message ID: 583h
Slave Node 1
Slave Node 2
Slave Node 3

Figure 8.SDO communication [4]

Communication is always started by CANopen master node which would be SDO client. Master node starts the communication and will send the request to the network which is called SDO request. As it is illustrated in figure 8 above CANopen master transmit SDO request with specific ID (e.g: ID. 603h). CANopen reserve range of indexes for each type of communication. For SDO request index range is 0x600h+Node ID and similarly SDO response index range is 0x580+Node ID. When CANopen master transmits SDO request all the nodes on the network get this message but only the node which is specified in ID will respond with CAN-ID 0x580+Node ID the data requested form CANopen master. SDO server understand this is SDO request (0x600) the data field of the message contain the index and sub index of the object tells which information SDO client is looking for. SDO server copies the requested data into the data field and response with CAN-ID (0x583). SDO client (CANopen master) will identify the response with the Node ID attached in the CAN-ID (SDO Response + Node ID).[4]

2.5.3 Network Management

Network management commonly known as NMT state machine has the ability to change the state of all slave devices of the network. CANopen can change the state of the slave devices remotely. In simple words NMT state machine defines the communication behavior of CANopen network nodes. Each slave node of the network would always be in one of the following four states:

- Initialization state
- Pre Operational state
- Operational state
- Stopped state

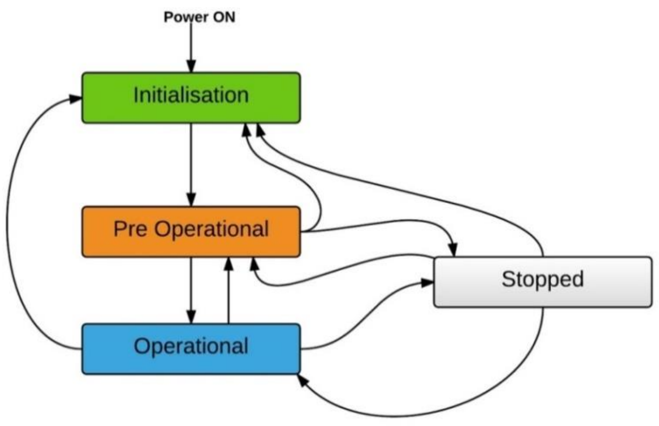


Figure 9. Network management states

The device is set into the "initialization" state after the power ON or reset of the hardware. This state is further divided into three sub states. Initializing, Reset Application, and reset communication. In Initializing the device starts and initializes its basic parameters. After initialization node transmit boot-up message. [10] Second sub class "Reset Application" all the object dictionary entries from index 2000h to 9FFFh set to default values. During third subclass communication parameters are set to default values. After finishing the initialization state (successfully sending boot up message) the device automatically entered into the pre-operational state. "Preoperational" state is mainly used for network device configuration. Each device can perform certain actions in specific NMT states. For instance PDOs can only be transmitted in the operational state not in pre-operational state because PDO contain process data. SDO communication is possible in pre-operational state because SDO contain configuration data. Only heartbeat messages are transmitted in "Stopped" state. There is no SDO and PDO transmission is last state.[14]

NMT startup

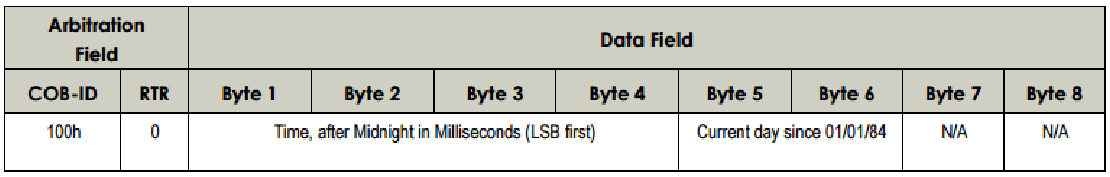
NMT startup contains 32 bits data which control the MNT node response. This NMT node could be NMT slave or NMT master. Value and description of each bit of UNSIGNED32 is given in table below.

Bit	Description
0	If 0 it means node is not an NMT Master If 1 it means node is NMT Master
1	If 0 means start only assigned nodes If 1 means start all nodes
2	If 0 means automatically enter into operational state If 1 it means node will not go for operational state
3	If 0 NMT master node will start nodes automatically If 1 NMT will not start nodes automatically
4	If 0 then reset only the node which fails to respond in HB request If 1 means reset all nodes if one node fails to respond HB request
5	-
6	If 0 then reset only one node which did not respond on HB call If 1 then reset all node if any one did not respond on HB call
7-31	Always zero

Table 4. Control bits for NMT startup [9]

2.5.4 Timestamp

Timestamp object provides time reference to the network. And push model is used for the transmission. Timestamp is mapped to one CAN frame with 6 bytes data length. These 6 bytes are divided into two parts, 4 byte contain number of milliseconds since midnight. 2 byte contains present day since January 1, 1984. [19]



Arbitration Field		Data Field							
COB-ID	RTR	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
100h	0	Time, after Midnight in Milliseconds (LSB first)				Current day since 01/01/84		N/A	N/A

Figure 10. Time stamp data frame.[19]

2.5.5 Emergency Messages

Emergency message can be generated and transmitted in any state of the state machine. Emergency message is generated when any internal error with in the device is detected or in could be a logical error for example wrong address within the object dictionary. There are many reasons which can trigger EMCY message. There is always only one error message per error frame. EMCY message is divided into three parts. Error code, Error register and Manufacturer specific error

2.5.6 Synchronization Object

SYNC message is periodically transmitted from SYNC producer and received by device. Time difference between two consecutive SYNC messages is known as communication cycle period. This communication cycle period is defined in the object dictionary (index 1006h). This communication cycle period is divided into synchronous transmission and asynchronous transmission. All synchronous communication occurred in synchronous part of communication cycle. This is the section used to transmit process data updates. Asynchronous communication goes in asynchronous section. Half of the SYNC messages communication cycle period is used for synchronous communication and second half for asynchronous communication. Data length of SYNC message is zero it does not contain any data. In some standards it contains 1 byte SYNC counter value. Identifier of SYNC message is 0x80h.

2.6 Protocol Analyzer

The Protocol analyzer is a program to analyze the data over the communication channel. It shows us what actually is happening over the communication channel.

Responsibilities of Protocol analyzer are as following: [22]

- It provide detail information for all the recent communication over the network
- Determine the load over the communication channel
- Easily identify the packets source and packet destination
- Can look for the specific data over the communication channel.
- Display all the data information in the form of statistics one can easily understand

2.7 Payload Analysis

Word "Payload" refers to the carrying capacity of any subject, this could be a vehicle, carrying weight or data cable carrying data packet. In IT, payload is the data carried from sender to end user by any transmission unit. Without payloads data transmission units (either data packets or messages) are just like sending empty envelops, without mail inside.

2.8 Deep Blue Hybrid system

In deep blue hybrid system, connection box contains Box Control Unit (BCU), System Control Unit(SCU) upto 5 Device control units (DCU) and Throttle control Unit (TCU). Each BCU can go to the autonomous driving mode without communication. It is necessary to allow an operation within 5 Sec before SCU would be available. After system booting SCU take the control of the entire system. There are multiple TCU in different positions of the boat. Each TCU includes two controllers that operate separately and independently both have UART(Universal Asynchronous Receiver Transmitter) connection.

Display is connected using Ethernet to SCU only. There could be one or more display boxes as per requirements. TCU are connected with the CAN bus of the BCU. This CAN bus is different from the CAN bus used between BCU and DCU. CAN bus is used to connect all DCUs ,BCU and SCU. This CAN is named Box-CAN. In this CAN bus BCU is the master. All the DCUs are also sharing Box-CAN to respond to

BCU with the data requested from BCU or SCU. SCU in this network is connected as backup master. All DCU get the data from the devices they are connected and transfer updated process data to BCU using bus they are sharing with BCU and SCU. TCU is also connected with SCU using other dedecated bus named Throttle-CAN.

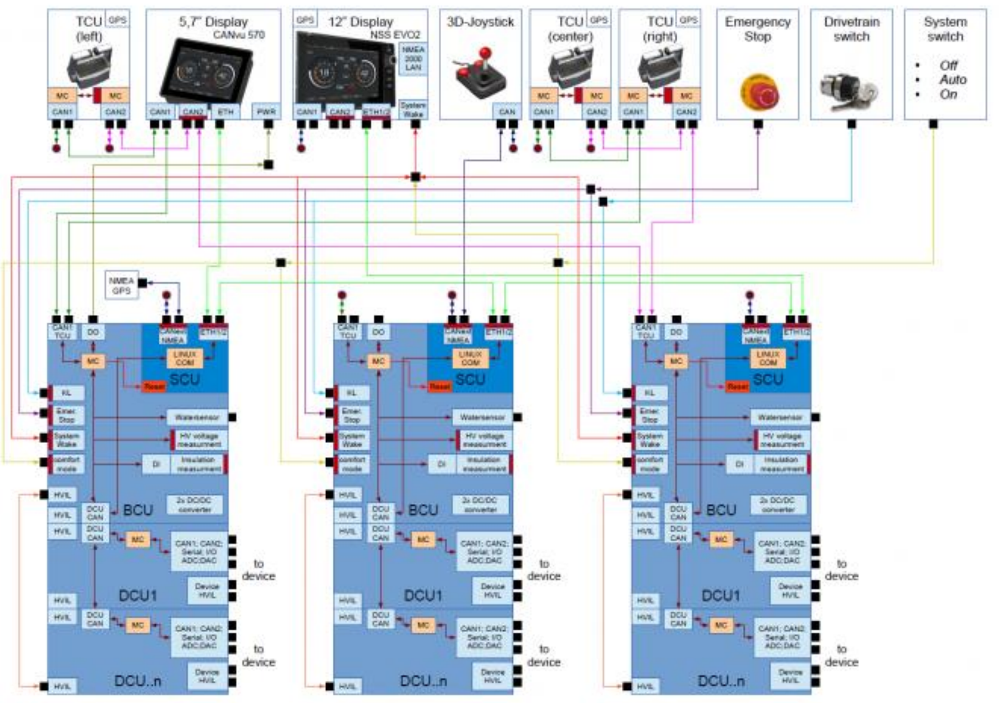


Figure 11.Deep blue hybrid system structure [16]

In some midsize boats it is possible that they don’t have only one control stations. There could be multiple control station. One TCU in each control station. In system there are 2 drive trains 2 boxes where drives and batteries are connected. In emergency drive there is not need of SCU. TCU is directly connected with BCU of the connection box with CAN bus.

3D Joystick module is not implemented yet. According to our DBHS structure design it need to be connected directly with SCU using NMEA CAN bus. This CAN is shared among 3D joystick and fuel tank sensor. Main intention was to separate the system if there is problem in one, system still can drive the boat. If power supply of TCU is from Box-1 only and due to some reason Box-1 fails than driver cant drive the boat. That was the reason to have multiple throttle controls installed with all connection boxes independently.

2.9 System Connection Box

In Deep blue hybrid system each connection box consist of 4 different types of control units as mentioned above. Each DCU has minimum 1 and maximum 16 devices units. ID allocation of each device is deterministic. First ID of each DCU index is reserved for DCU itself. Which is used as DCU base-ID. First node connected with DCU would get ID, DCU Base-ID + 1. All the DCUs of connection box and BCU are sharing the CAN bus named “BoxCAN” as communication channel. SCU is also connected with BoxCAN. Here SCU is a silent listener of the communication for initial 5 to 7 seconds meanwhile all the nodes get ready. After this SCU take the charge. BCU is connected with Throttel Control Unit (TCU) through separate CAN called “ThrottleCAN”. Number of throttle could be vary between 1 to 4. There could be more than one connection boxes in one DBH System. All the connection boxes are connected through SCU. SCU of each connection box is connected through Ethernet with other connection boxes of the system.

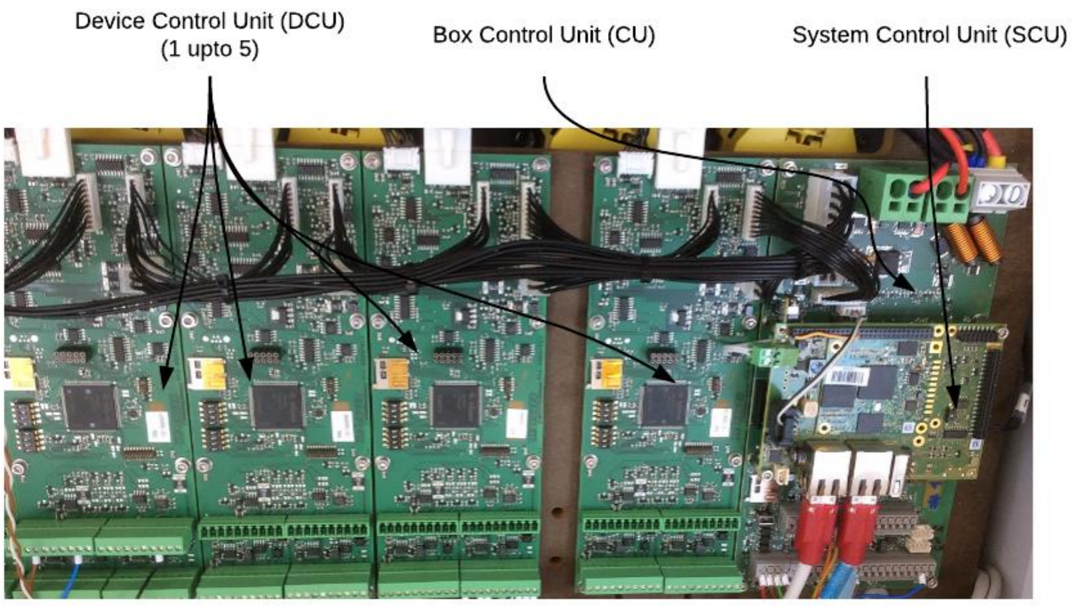


Figure 12. Connection Box of DBHS [16]

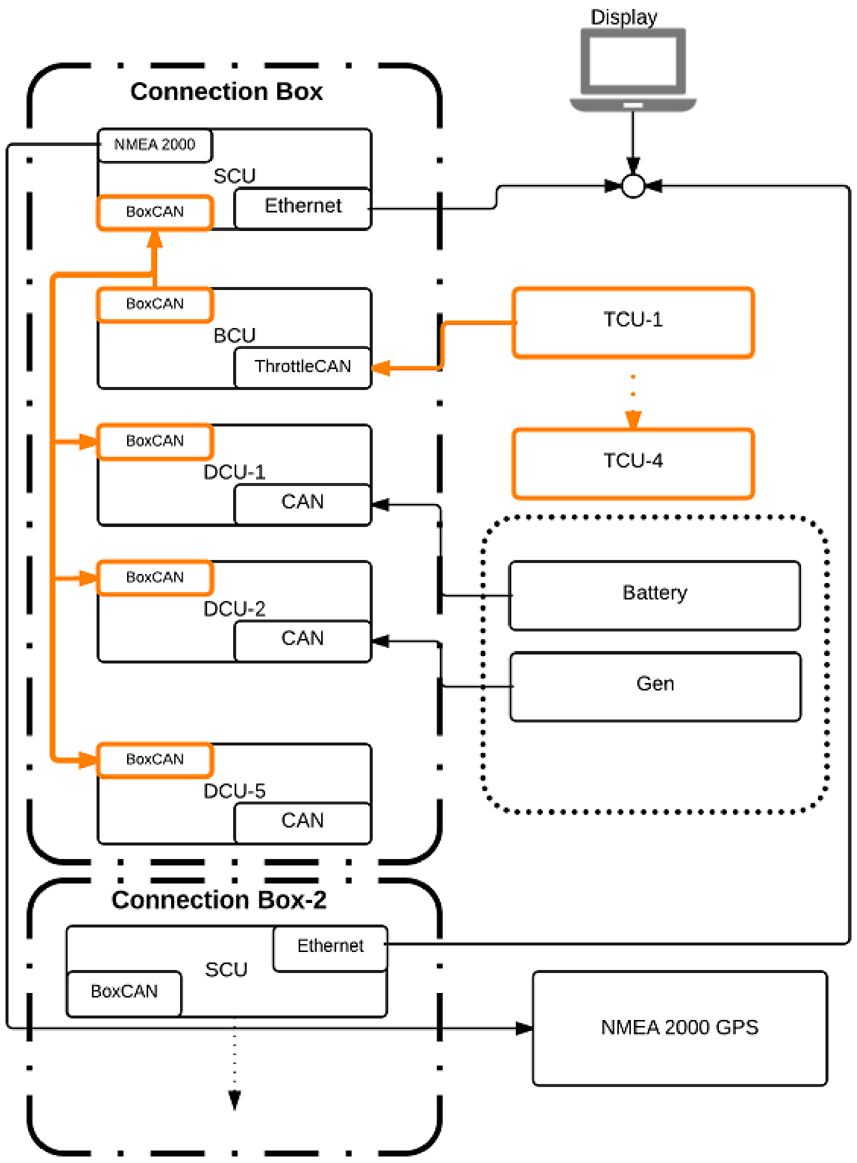


Figure 13.Deep blue hybrid system Connection box [16]

2.10 Summary

To sum up the basics of “*logging and monitoring tool*” it is clear that the role of Protocol analyzer is must to make network more efficient and reliable. Protocol analyzer is used to capture and analyze data over a communication channel. This communication channel could be simple networking cable or wireless link. CANopen is a standardized protocol for distributed automated systems. All CANopen objects use index and sub-index as an address pointer to a data location in the Object dictionary. All the data (communication data or Process data) stored in these object dictionary locations. Communication objects of protocol use different communication model to transfer and receive data within the network. SDO and PDO are two methods of reading and writing data over objects. SDO objects carry mostly configuration data because of this SDO transfer starts in pre operational state of the nodes whereas PDO can be transferred in operational state of the node only.

Chapter 3

3. State of the art

Torqeedo is developing a complex hybrid propulsion system for boats with high flexibility in the system structure. This flexibility needs a complex and flexible communication between the numerous components. For this communication a CAN network with CANopen protocol (with proprietary additions) is designed. The reliability of the system is dependent on correct communication with strict timing requirements. To be able to support distributed software over the network the content/meaning of each CAN-message is auto generated by a tool with database access. Manually debugging/monitoring of the communication bus with off the shelf tools is hard and time consuming process.

3.1 CAN Bus Analyzers

CAN bus analyzer is a tool which is used for communication analysis and logging data over a communication channel. Bus analyzer helps engineers in debugging process. Filtering data as per requirements is easy with CAN bus analyzer. Bus analyzer is normally designed for one or few dedicated protocols. Sometimes Bus analyzer is interchanged with another term “Logic analyzer”. I think they are totally different both with their own features and specifications. Logic analyzer is more generic analyzer and has less speed than Bus analyzer. Bus analyzer has limited capacity of logging data.

3.2 Data Transmission

CAN bus data transmission is without any central master node which control all the communication. Individual nodes of network get access of the CAN Bus for read or write. Node broadcast its message over Bus which all the other nodes can receive. Getting the access of the CAN bus for any node in the network is procedural. When any node of the network is ready to transmit its message over bus it will check the availability of the Bus. If the bus is idle node can transmit message otherwise needs to wait for the Bus to be idle. After getting bus access Node transmit message over Bus. Data frame transmit on Bus don’t have any information about transmitter or receiver but has arbitration ID which is unique for each data frame. If more than one node tries to transmit data frame over the bus, Node with highest priority (lowest arbitration rate) will get the access of the bus. Nodes with lowest Arbitration ID must wait until bus will be available after transmitting high priority data frames. [18]

3.2.1 CANopen Data Transmission

The primary reason of implementing CANopen protocol is transmission of data. Mainly this data is process data, for this purpose, object dictionary entries has been used. Goal of transmitting data using CANopen is to avoid protocol overhead of data transmission. Process data is transmitted according to producer consumer principle. This will be explained in following section of this chapter below.

3.2.2 Flexible Transmission

All the real process data is transmitted only when node is in “Operational” state. On the other hand transmission of configuration is possible in pre-operational state and initialization state of the node. CANopen standards use only SYNC start counter to identify the SYNC (Synchronization) slot of any PDO (Process Data Object) or SDO (Service Data Object). SYNC message period (20 milliseconds in current implementation) is basically divided into two sections (Asynchronous data transmission period, Synchronous data transmission period). Process data is transmitted in Synchronous period and configuration data is transmitted in Asynchronous period. E.g. to transmit TPDO-1 of Device A in the 9th sync slot of SYNC window .Reason behind this this single transmission is that CANopen is using sync start only to transmit PDO. In CANopen standard TPDO-1 can be transmitted only once in SYNC overflow period. In Torqeedo CANopen implementation, two parameters (SYNC start and Sync Gap) have been used to transmit TPDO or RPDO. Now network node can transmit each TPDO/RPDO multiple times in one SYNC overflow period. Minimum once per SYNC overflow period and maximum 200 (current SYNC overflow) times in one sync window. Data transmitted with PDO is one which transmitted regularly during communication. To get data after short time period just decrease the Sync gap of that TPDO. In the Figure 11 below it’s depicted that if TPDO/RPDO has SYNC gap of 50 and sync overflow is 200 SYNC. This PDO will be transmitted four times in one SYNC overflow period.

$$\text{Over flow periood} = \quad (SYNC\ Period) * (\ Number\ of\ messages\ per\ sync\ window)$$

$$Over\ flow\ period\ = (\ 4\ Seconds)$$

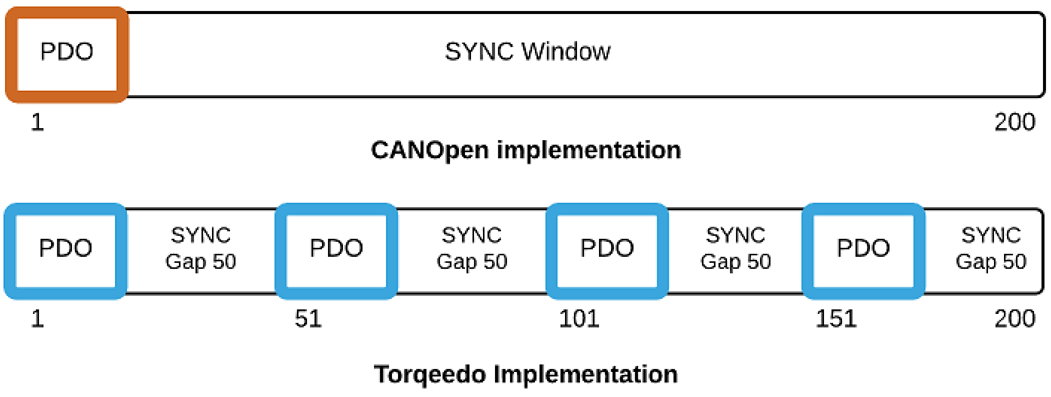


Figure 14.CANopen standards vs Torqeedo implementation[16]

3.3 Bus Balancing Algorithm

Balancing the CANopen objects equally on CAN bus is one of complicated task, which is achieved through especially developed algorithm. This algorithm distributes all the PDOs equally among SYNC slots (SYNC 01 - SYNC 200). Otherwise all the PDOs rush to transmit in the initial few SYNCs of the SYNC window. When turn on DBHS connection box, all the devices connected with network get boot up starts some configuration messages (SDO) to network master, meanwhile BCU sort the PDOs of all the devices connected with the network. BCU sort the PDOs with respect to their SYNC gap. All PDOs with SYNC gap 1 would have higher priority than rests. Steps of Bus balancing algorithm is as following:

Step-1: Sort all ready to transmit PDOs with respect to their transmit intervals (SYNC gaps). BCU will do this sorting.

Step 2: Initialize array containing all SYNC slots. Length of array would be equal to the Sync overflow period.

Step 3: Check that number of elements in array should not be more than SYNC overflows value.

Step 4: Size of array should not be greater than maximum number of frames transmitted in one SYNC slot.

Step 5: Get the emptiest Sync slot and transmit PDO from top of the sorted stack. SYNC slot number should be less than SYNC gap of the PDO.

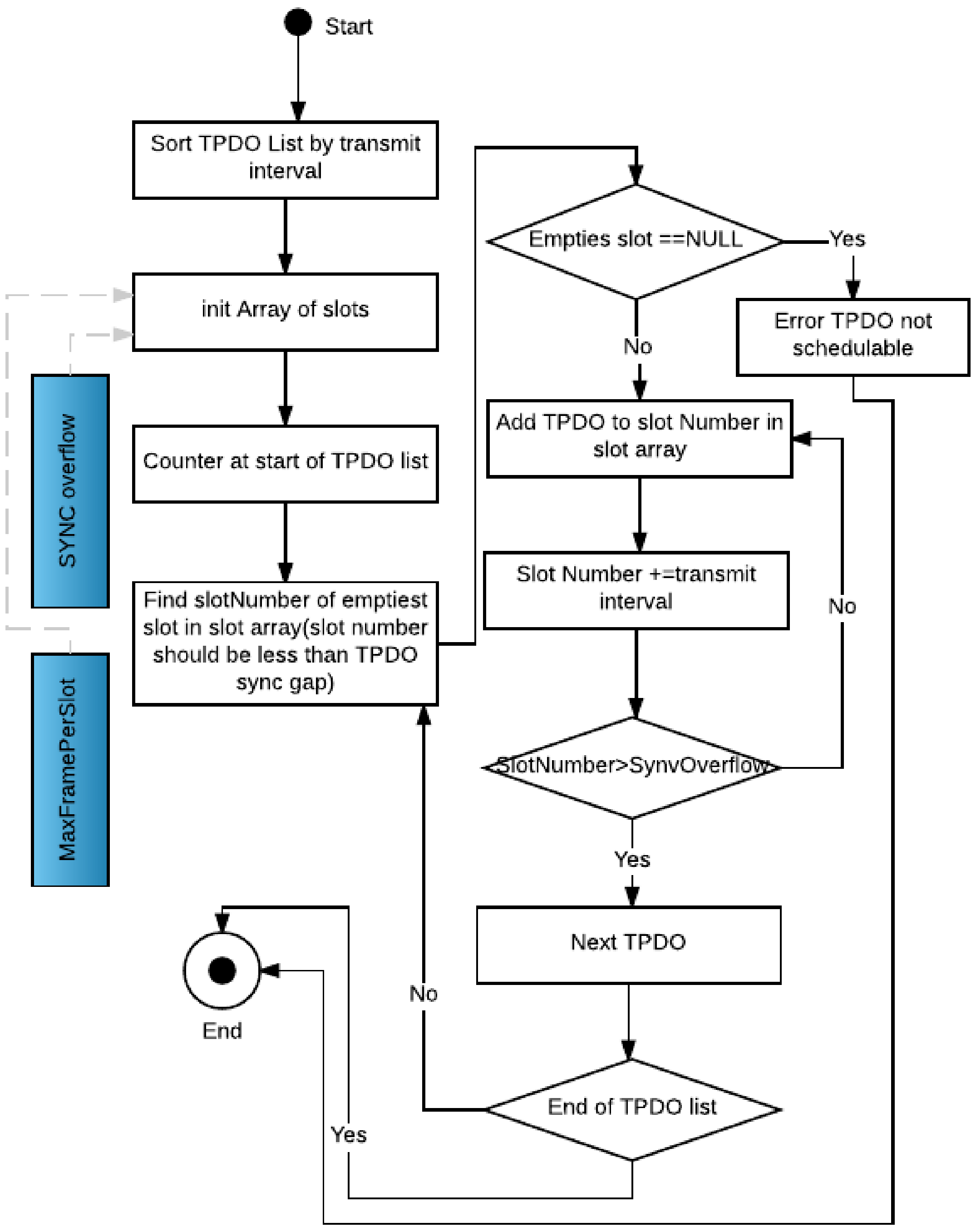

Start
Sort TPDO List by transmit interval
init Array of slots
Counter at start of TPDO list
Find slotNumber of emptiest slot in slot array(slot number should be less than TPDO sync gap)
SYNC overflow
MaxFramePerSlot
Empties slot ==NULL
Yes
Error TPDO not schedulable
No
Add TPDO to slot Number in slot array
Slot Number +=transmit interval
SlotNumber>SynvOverflow
No
Yes
Next TPDO
End of TPDO list
No
Yes
End

Figure 15.Flow chart of Bus balancing algorithm[16]

Step 6: In case there is no more place in any SYNC slot to transmit PDO of sorted stack.

Step 7: Next slot number to transmit this PDO again would be current slot Number + Sync gap. If sum will be greater than Sync overflow counter go to Next PDO of sorted list otherwise add PDO into the SYNC slot.

Step 8: If current TPDO was the last TPDO of sorted list than end the bus balancing algorithm.

Step 9: If there is no more possibility to transfer TPDO.It will also go to end the algorithm,

In the figure given below, there is a list of PDOs to be transmitted on CAN, All PDOs are in the stack. All the devices connected with the network put their ready to transmit data in form of PDOs, in this stack. Step 2 is responsibility of BCU, to sort all the PDOs (in stack) with respect to their SYNC gaps. Minimum SYNC gap among 2 same PDOs is 1 (200 times in one SYNC window) and maximum is 200 (only once in one SYNC window). End of the step 2 sorted stacks of PDOs has been prepared. Now next step is to transmit these PDO on the CAN bus. PDOs with the SYNC gap of 1 would be on the top of the sorted stack and transmitted first. Followed by PDOs with the gap of 2 and have modulus divided by sync gap is 0. While sync counter is 1 (P5 and P11) transmitted in first sync. Sequence of PDOs in initial three SYNCs is shown under step 3 of figure 16 given below.

While placing the PDO from sorted stack to transmission bus, check for the emptiest SYNC slot where SYNC number is less than Sync gap of the PDO to be transmitted. For example PDO-4 with the sync gap of 5 will be scheduled in one of the emptiest SYNC of initial 5 SYNC slots. Same for all the remaining PDOs remained in stack. In the end of this balancing, resultant CAN bus would be much balanced where all PDOs distributed relatively more balanced on the bus than without using any scheduling mechanism.

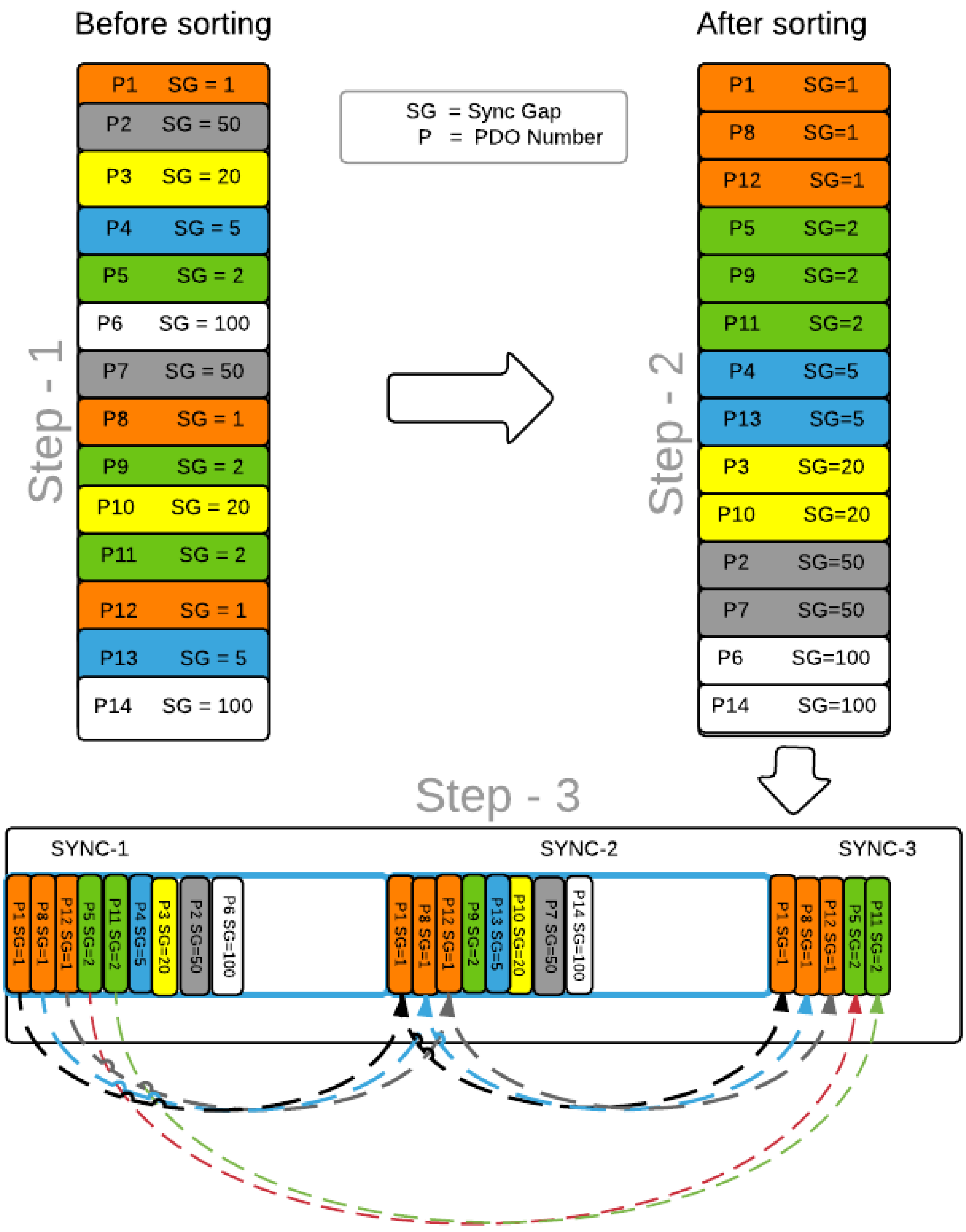


Figure 16.Example of Bus balancing algorithm[16]

3.4 Off the Shelf Solutions

Now a days there are many off the shelf and open source CANopen solutions in the market. They are ready to use and sometimes they are even at quite affordable prices as well. But on the contrary, they also have some disadvantages for example, for off the shelf software there is no source code access. Sometimes they have functionality which is not required for your project in this case you have extra code in your project. For open source project, they are mostly generic not developed for the specific project because of this sometimes 70% code is not required but user can't remove it. This code can create problems for other parts of the source code in future. Following are the some of the commercial and open source off the shelf solutions.

3.4.1 PCAN

PCAN is the commercial project of PEAK Systems. There are many versions of PCAN which include Software and hardware. From simple monitoring of CAN bus tool "PCAN View" to PCAN-Explorer 4. PCAN-Explorer is a tool for monitoring data traffic on a CAN network. [15] For details about PEAK system products visit company website

3.4.2 CANalyzer

Developed by Vector for development and analyzing of data traffic over CANopen network. [16]. CANalyzer offers devices graphical representation and project specific representation in the trace window. It also monitor the individual message sequence. The configuration of CANopen devices is also easy. Feature which make CANalyzer unique is its ability to adjust user project specific requirements by inserting graphical blocks. [6]

3.4.3 CANFestival

CANfestival is open source framework made in 2001 which provides an ANSI-C platform independent CANopen stack, that can be built as master of slave nodes. [17] Its run-time code is licensed under General Public License(GPL) and Licensed General Public License (LGPL) license. CANFestival GUI is developed in Qt. It also provide some very useful features:

- Simulation mode
- Separate thread for GUI and handling CAN messages
- Value on the CANopen network could be in Hexadecimal, decimal or binary format.
- Its easy to load and unload device drivers.

3.5 Summary

To sum up the State of the art, it is evident; that our implementation of data transmission on CAN bus using CANopen protocol is bit changed from CANopen standard implementation. Using SYNC start and SYNC gap to identify the SYNC slot of the data add more flexibility in communication hence it allows to transmit important data with minimum transfer interval of 20 milliseconds (Sync slot time period) and maximum 4 Sec. Without some special bus balancing mechanism it is very challenging to distribute all PDOs equally on the bus. Whenever CANopen master start working. After configuration of all the nodes try to transmit their PDOs in first available SYNC slot. Result will be too much rush in the initial some SYNC slots. This makes bus unbalance. Bus balancing algorithm does balance distribution of data frames over the CAN Bus.

Chapter 4

4.Concept

Theme of this chapter is to get the overview of Deep Blue Hybrid System (DBHS) and proposed monitoring tool. Current implementation of communication channel is complex and error prone. New monitoring tool will make this complex communication network more efficient and will reduce debugging time. Than later on there would be an overview of Torqeedo DBHS connection box. Readers will get the inside view of DBHS connection box. System components their roles, and communication scheme would be highlighted. In the end readers will get the overview of all off the shelf solutions many of them are free to use and open source as well.

4.1 Problem Definition

Torqeedo is looking for a data logging and monitoring tool for deep blue hybrid system to track inter connection box communication errors more efficiently and reduce debugging time. CAN Bus load has to be calculated to check how much resource has been used. Analyze the payload of each device connected with communication network is also in requirement list. Data logging and monitoring could be Online monitoring or offline monitoring. Online means doing realtime analyses while logging. Off line analysis is to use already logged file and do analyses process. Goal of this research work is to develop a monitoring tool to log errors and protocol violations in error file. There are many off the shelf products. All these solutions are as per CANopen standards and in DBH system CANopen standards has been not followed exactly. They has been altered as per requirments. Many OD entries are modified.

4.2 Methods of data Analyses

There are 4 possible methods to provide input to the analyzing module. First method is to read from control box using PCAN USB. In this method, data stream from SCU will go through all monitoring steps, in the end logged into data logger file. Second method is to use previously logged data as input. There are two different formats of CANdump files.

- SCU file format
- BusMaster file format

Difference is in time format and data sequence. Third method is stream from SCU directly. Fourth method of input is to parse SCU format file to data analysis function.

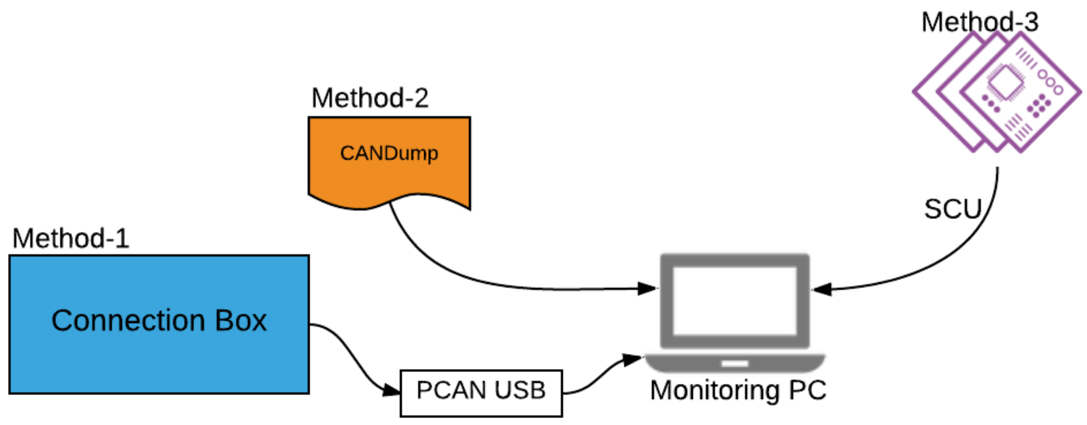


Figure 17: Methods of data input

4.3 Proposed Solution

Developing a logging and monitoring tool as per implementations of CANopen protocol in DBHS is the way to go. Proposed solution is divided into three steps. First requirement is to develop one basic online logging and monitoring application to log data from communication channel and check for timing violations while logging. Bus load and payload calculations of each node is also part of our application, last step would be to execute developed application on embedded platform. For this purpose Qt frame work is used for application development because it is platform independent.

4.4 Connection Box Units

There are three types of modules inside the connection box of DBH system. BCU, SCU and DCU. TCU is not installed inside the connection Box, but connected with BCU using Throttle CAN. All the modules installed inside the connection box have their certain responsibilities. BCU and DCU are sharing CAN bus named “Box_CAN” to transmit their data frames. Key units of all connection box units is given below in detail.

4.4.1 Box Control Unit (BCU)

BCU is the master of Box-CAN which manage all device control units of the connection box. Each connection box has one BCU and 1 upto 5 device control units. BCU, SCU and all DCUs of connection box are sharing the common CAN called “Box-CAN”. It sends process data requests to all DCU to get process and configuration data. All the safety functions are also managed by box controller. e.g. Turn OFF the generator if SOC is greater then certain limit. It also sets the throttle and handles station requests (box with lowest ID is master).

Data transmission from BCU to TCU

- Box state information and active station information
- TCU power enable signal

4.4.2 System Control Unit (SCU)

SCU calculates system wide power and state information to manage system energy distribution strategy. It is possible to be more than one connection boxes connected in serial Ethernet. It also handles master SCU selection . System control handles system configuration and to set system wide parameters. Data transmitted from SCU to BCU is mostly basic system information. e.g. Active station , Throttle current state. SCU can also request BCU for any specific device state.

4.4.3 Device Control Unit (DCU)

Each connection box has 5 device control units. Upto 15 devices can be connected with each DCU in the connection box. These device control units abstracts information from devices. Handling device parameters is also done by DCU. Periodice data transmitted from DCU to box control unit is key feature of DCU.

Data transmission from DCU to BCU:

- Module state information
- Data requested from BCU

4.5 Summary

This chapter explains the basic concept of proposed solution for making network debugging easier and efficient. It is evident to use protocol analyzer to capture problems in the complex network. There are multiple reasons of developing CANopen analyzer tool for capturing protocol violations. Most important is CANopen implementation in the project DBH system. There are several modifications made in protocol configuration and object dictionary entries as per project requirements. For data analysis tool there are 4 possible ways to analyze CAN bus communication. In offline analysis there would be two possible file formats, monitoring tool need to debug.

Chapter 5

5. Implementation

In this chapter implementation of proposed data logging and monitoring tool will be explained. Implementation of data logging and monitoring tool is divided into three phases:

1. Basic PC based online Monitoring tool
2. Offline data analysis
3. Embedded platform

5.1 Tools and Technologies

Following are the main tools and technologies that have been used for the development process of data logger and monitoring tool:

- QtCreator as IDE
- PCAN USB adapter (system interface)
- C++
- MS Access database

Qt is cross-platform framework used widely in embedded development world. It makes project OS independent with some libraries like STL and many more. On the other hand Qt is also open source developed under QT group, Nokia. PEAK PCAN USB adapter is used as data logging hardware tool.

5.2 PC Based Online Monitoring Tool

The first phase of this data logging and monitoring tool is to develop a PC based tool to log data and analyze data. To log data from CAN-based network to PC, An adapter is required which can connect PC with the CANopen network to log data traffic on PC. There are several adapters for this purpose with different features. Thesis priority is the one, which has most efficient time stamp feature. Requirements of project are strict with the timing behavior of network nodes. There were two options initially, either to use “PCAN_USB adapter” or “Kvaser leaf light V2”. Key features of “PCAN_USB adapter” and “Kvaser leaf light V2” are in the table 5 below. [16][17]

PCAN_USB adapter	Kvaser leaf light V2
Compatible with all USB 1.1, USB 2.0 and USB 3.0.	High speed USB interface.
Bit rates range is from 5 Kbits/s to 1 Mbit/s.	Bit rates range is in between 40 Kbit/s to 1 Mbit\s.
Time resolution is 42 microseconds	Time resolution is 42 microseconds
Compliant with CAN specifications 2.0A and 2.0B	Compliant with CAN specifications 2.0A and 2.0B
Connection with D-Sub is also possible	D-Sub connector support
Power supply with USB	Power supply with USB is also possible.
24000 messages per second	Send /Receive 8000 message per second
Operates in temperature range -40 °C to 85 °C	Temperature range is -20 °C to 75 °C.

Table 5. PCAN Basic Vs kvaser V2

PEAK CAN adapter was closer to thesis requirements because of its minimum time stamp period. Phase-I of implementation is to develop PC based logging and monitoring tool. PCAN_USB adapter has been used for logging data from DBHS Connection Box to PC. The first phase has following tasks:

1. PCAN USB adapter configuration
2. Timing of CANopen SYNC-Message
3. Monitoring of SYNC-Window
4. Bus Load Measurement

5. Period/Sync Slot monitoring of PDO Messages
6. Period/Timeout monitoring of Heartbeat Messages
7. EMCY Message logging
8. Error triggering conditions

5.2.1 Timing of CANopen SYNC-Message

SYNC-Message is periodically transmission of synchronous object from SYNC producer to SYNC consumer. SYNC object provides the mechanism for network synchronization. Period of transferring SYNC-Message is defined in object dictionary index 1006h. As per CANopen standards it is conditional to set communication cycle period. In DBHS (Deep Blue Hybrid System), communication cycle is 20 milliseconds. There is option to add jitter, which is adjustable as per requirements. Default Jitter allowed for SYNC message is 1 millisecond (ms). So any SYNC message with time period of 20 ± 1 ms is considers within the period. Jitter of SYNC slot varies between maximum 2 ms and minimum 0.5 ms from our logging tool. Any SYNC message out of time period ± jitter window would violate allowed time of CANopen protocol configurations. Monitoring tool needs to log these errors as SYNC timing violations. Each SYNC message is divided into two sub-windows:

- Synchronous communication window
- Asynchronous communication window

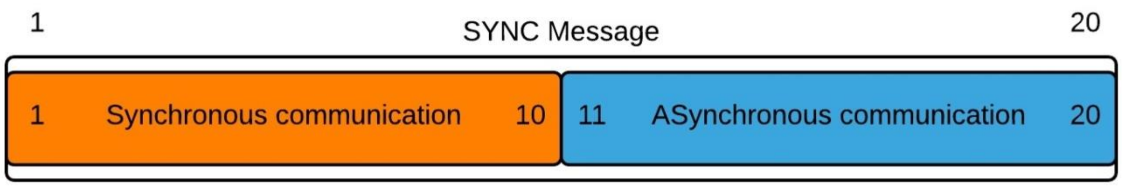


Figure 18. SYNC Message

All the synchronous communication occurs in first half window SYNC slot (e.g PDO) and asynchronous communication occurs in last half of our SYNC slot.

5.2.2 Slot Monitoring of PDO Messages

PDO gives the fastest way to share the process critical data without any additional protocol overhead. CANopen has four transmit PDO and four receive PDOs. All the PDOs need one COB-ID. Following table 6 contain all TPDOs and RPDOs.

PDO number	Transmission Type	COB-ID(s) hEX
TPDO-1	Transmit PDO	0x180 + Node ID
RPDO-1	Receive PDO	0x200 + Node ID
TPDO-2	Transmit PDO	0x280 + Node ID
RPDO-2	Receive PDO	0x300 + Node ID
TPDO-3	Transmit PDO	0x380 + Node ID
RPDO-3	Receive PDO	0x400 + Node ID
TPDO-4	Transmit PDO	0x480 + Node ID
RPDO-4	Receive PDO	0x500 + Node ID

Table 6.TPDO and RPDO of network node

Maximum there are 127 nodes in the network. Therefore Node ID of any node of network would be between 1 and 127. Two type of data to be transmitted; either it could be synchronous or asynchronous data. All the PDOs would be transmitted in the synchronous communication period of SYNC message and SDOs are in asynchronous communication period. If there is a node in the network which produces the update data in every 20 ms. Sync gap of this node is 1. Data would be transmitting 200 times in one sync overflow period of 200 syncs. Similarly as sync gap would increase transmission frequency of data will be decreased. As given in following table.

Message Interval(Sync Gap)	Transmission(Frequency per Sync window)
1	200 times
2	100 times
20	10 times
50	4 times
100	2 times

Table 7. Sync gap and transmission recurrence.

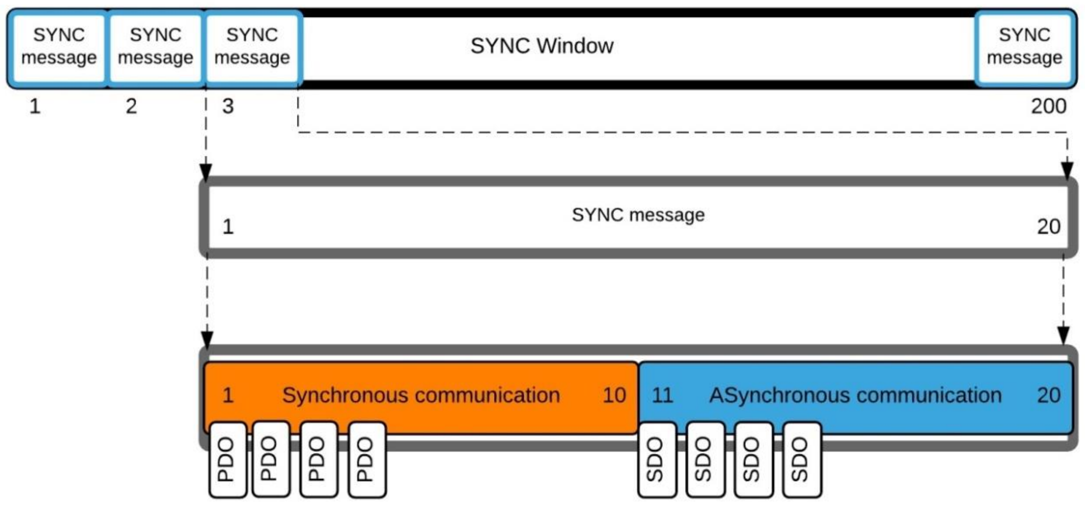


Figure 19. SYNC window and SYNC message

While monitoring the PDO messages, make sure that all the PDOs are whether they are TPDOs or RPDOs, need to be in Synchronous communication portion of SYNC slot (initial 10 milliseconds). Sync gaps of all process data objects (TPDO and RPDO) are in *SyncGap.conf* file. This file is generated from database using template file. Every node on the network can transfer max 4 TPDOs and receive max 4 RPDOs. All information of each node for example, total TPDOs, RPDOs with respected gaps are in the *Syncgap.conf* file. ID numbers of all devices are defined in *systemconfig* file. Example of *syncGap.conf* file is given below.

```
Battery_TPDO__GAP1_5__GAP2_200__GAP3_50__GAP4_0
Battery_RPDO__GAP1_5__GAP2_0__GAP3_0__GAP4_0
```

Listing 1: Node Battery Sync Gaps

Configuration given is listing-1 depicts that the device "*Battery*" has 3 TPDOs. TPDO-1 with gap of 5 Sync messages, TPDO-2 with gap 200 and TPDO-3 with gap 50. Zero gap means this PDO is not going to be used by the device. On the other hand, device "Battery" has only one RPDO with sync gap of 5. Data section of SYNC message contain only 1 byte SYNC counter, which always starts from 1, when connection box gets started , sync counter starts from 1. Supposedly the first occurrence of any TPDO and RPDO of any node/device is correct and after this, calculating the sync gap with the occurrence of the previous TPDO or RPDO. To verify the gap of TPDO/RPDO, current sync gap has to be compared with the expected gap in *syncGap.conf*. If current sync gap is greater or less than what is defined in *syncgap* file, Monitoring tool log it in error log file.

There are two possibilities of Transmit Process Data Object (TPDO)/ Transmit Process Data Object (RPDO) missing their sync message slot. Either message frame arrived one sync message late or either it is missing. e.g. Sync gap of network node "Battery" TPDO-3 is 50. If producer transmits TPDO-3 after 51 sync messages, its means TPDO-3 is one SYNC late. If it is transferred after 100 sync messages which mean one TPDO is missing in between. Reasons behind this missing SYNC message could be that the device state maybe changed from operational to pre-operational or back to Initialization state (Node restarted). Network node can't transmit or receive PDO in pre operational state. In this situation nodes will wait to come back to the operational state and to resume TPDO/RPDO service.

5.2.3 Synchronization (SYNC) Slot Delay

BCU is real time unit which reads from the CAN bus and stack the PDO for SCU. SCU assumed to be real time too. There is a probability that SCU can make some delay while handling real time data. When SCU made some delay for starting SYNC all the data to be transmitted within this SYNC slot would keep waiting in stack. In current implementation we can adjust maximum jitter of 2 milliseconds. Greater than 2 milliseconds would be considered error SYNC late start. This error can leads to data objects (PDOs and SDOs) to miss their deadlines. In figure-19 given below, Sync-3 starts on timestamp 49 rather than 41(Expected starting time stamp). There are only 2 milliseconds to transmit Synchronous data (PDOs) of Sync-3 slot. When SYNC starts on time, there are 10 milliseconds to transmit process data accordingly 10 ms for configuration data (SDO).

This jitter zone between SYNC-2 and SYNC-3 (41 to 48) belongs to none of the SYNCs. Any transmission of data (possibly SDO of SYNC-2) in this time period would be out of Asynchronous section. PDOs of SYNC-3 can't use this period because they are dependent on the start of SYNC-3.

In implementation, there was no way to make sure either the first SYNC slot is on time or not. For every new SYNC slot afterword, first SYNC is benchmark. Expected start time of SYNC calculated as following.

$$New\ SYNC\ start\ time = current\ time - start\ time\ of\ prev\ SYNC$$

Equation 1: SYNC time period

If this gap is greater than 20+Jitter or less than 20-jitter, it means new arrived SYNC slot is late. Normally jitter should not be more than 1 millisecond. In offline analysis 2 milliseconds is also acceptable.

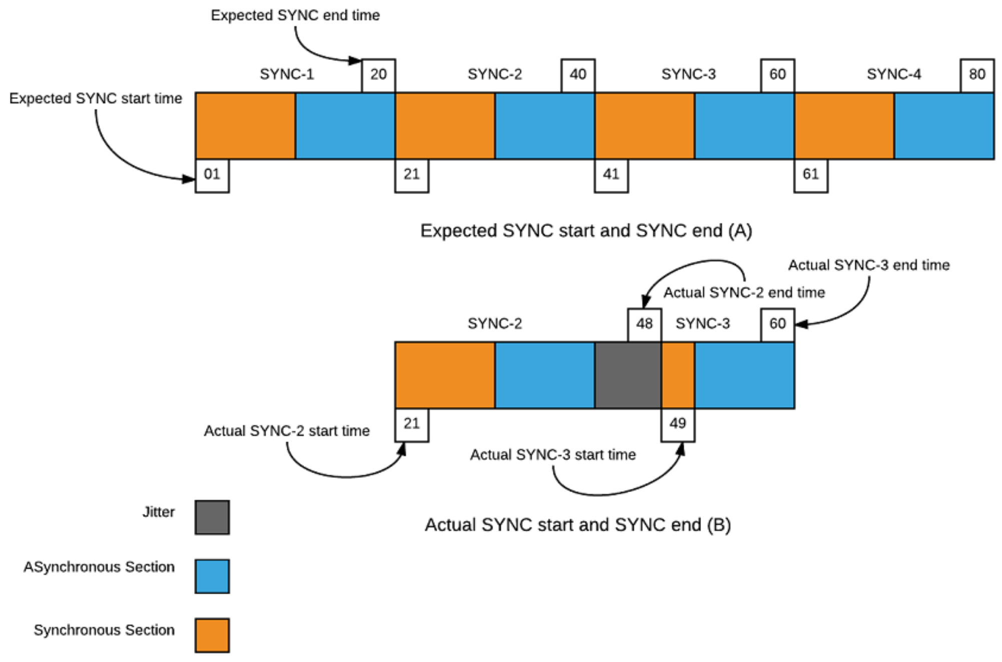


Figure 20. SYNC slot stating delay

5.2.4 Bus Load Measurement

Bus load measurement is the calculation of the amount of time bus is being used over a time interval.[16] The first principle of CAN bus load measurement is, all messages meet their deadlines. To make sure this utilization of the CAN bus would not be more than 100%. If utilization would be more than 100% it means definitely some messages would miss their deadline.

Bus being used means transmission phase. Logically bus utilization can be 100% but to be on the safe side it is recommended and also as per industry standard to keep utilization of CAN bus under 45%. Above 60% utilization can cause problems (e.g frames missing the deadline).

CAN frames	CAN 2:0 B	CAN 2.0A
SOF	1	1
ARB	32	12
CTRL	6	6
DATA	64	64
CRC	16	16
ACK	2	2
EOF	7	7
IFS	3	3
TOTAL	131	111
OVERHEAD BITS	67	47
DATA	64	64

Table 8.CAN frame bits

To calculate the total load on CAN bus, Total number of CAN messages over a specific period of time need to be calculated. In Deep Blue Hybrid System (DBHS) there is a sync period of 20 ms. Data transfer rate is 500 Kbits/sec. First calculate the total number of messages transferred in sync period.

There are two types of CAN messages on the bus either CAN standard frame with 11-bit identifier (from DCU and BCU) and CAN extended frame with 29-bit identifier (from SCU). Data carried by CAN standard frame and CAN extended frames are same (Maximum 8 Bytes) number of overhead bits are different. Detail of frame bits is given in the table 8.

5.2.5 Heartbeat Monitoring

There are multiple methods to make sure that all the nodes connected with network are functional. The Heartbeat (HB) message is one of them. This is a cyclic transmitted message that informs the HB consumers about the availability of the node. The cycle time of HB message is defined in object dictionary object 1017h. HB message is not mandatory to transmit. Value in object dictionary index 1017h can be 0. The value 0 means this method is disabled. [10] Current cycle time of DBHS, HB producer is 500 milliseconds. All the nodes connected with network transmit its communication state after 500 milliseconds to network master. Allowed HB jitter is ±20 milliseconds. HB message arrival time should not be greater than 520 and less than 480 milliseconds. If any HB message is transmitted later than 520 or earlier than 480 milliseconds, monitoring tool logs this as HB error.

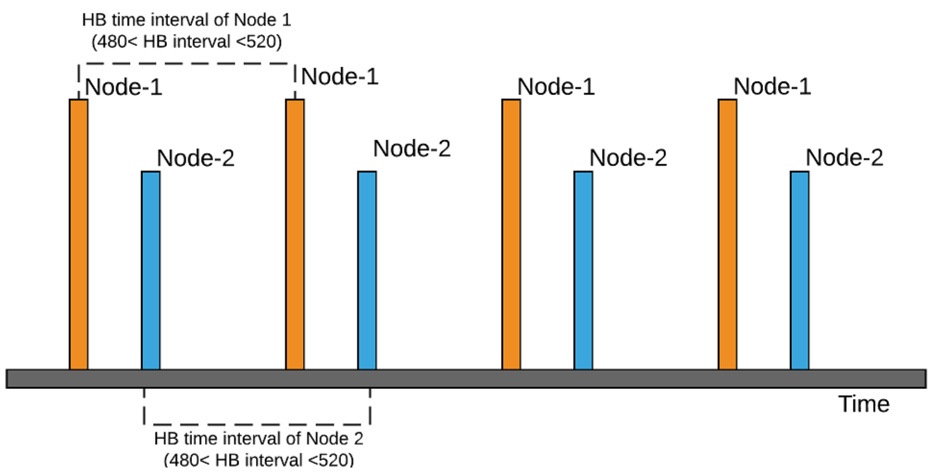


Figure 21.Heartbeat monitoring.[9]

During implementation, there was no source to verify about the first HB time from the producer either this messages is on time or not. Assume that the first HB of producer node is always on time. Use this as criterion to check the heartbeat timing of upcoming heartbeats. Equation to calculate the HB time of node is given in Equation-2.

$$Node\ HB\ time = current\ time\ stamp + previous\ HB\ time\ stamp\ of\ node$$

Equation 2: HB time period

HB message also carries the current NMT state of the node in 1 byte CAN frame. In figure 18 two slave nodes transmitting Heartbeat messages over a specific period of time (In current implementation it is 500 milliseconds defined in OD index 1017h, sub index 00h [UNSIGNED 16]). Possible NMT states of the node in the network are in the following Table 9.

State	State values
Boot loader	0x00 07
Boot up	0x00
Pre-Operational	0x7F
Operational	0x05
Stopped	0x04

Table 9. NMT states

5.2.6 CANopen Emergency Message Handling

Emergency message is triggered when any internal error occurs in CANopen device. This message is sent to other nodes of the network. Node that produces this emergency message is called emergency message producer and nodes that receive this message are called emergency message receiver. There could be multiple receivers of the same emergency message but there should be only one emergency producer for each emergency object. Emergency message data frame has three parts as depicted in figure 19 below.

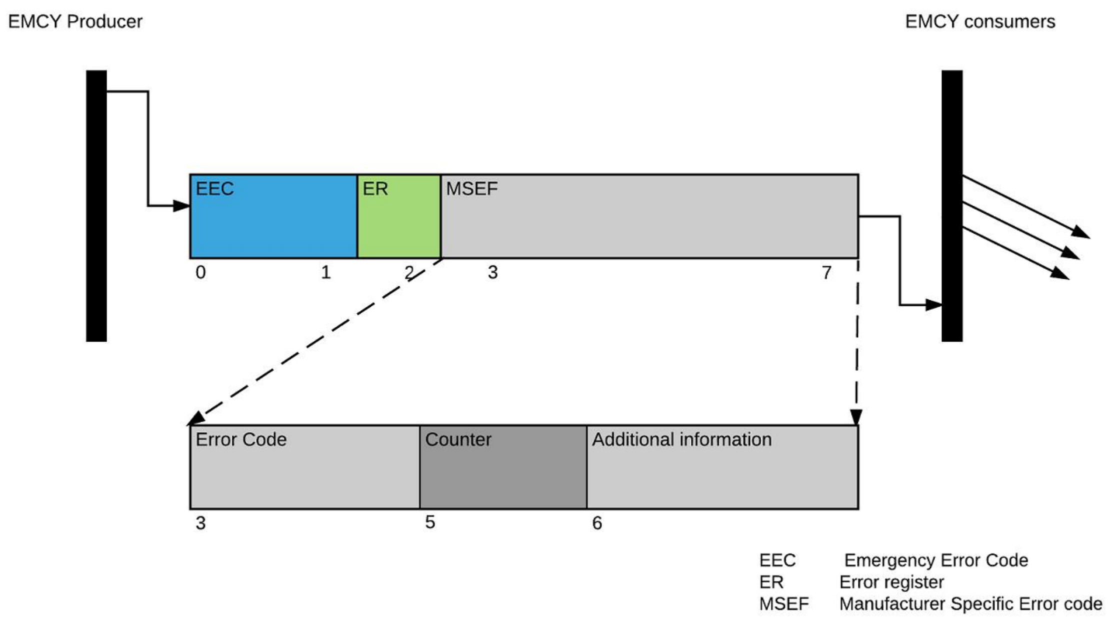


Figure 22. Structure of EMCY message

Nodes of the CANopen network supporting Emergency messages will always be in “Error occurred state” or “Error free state”. Emergency error flow chart is given below in figure 20.[12]

When system has been started, after initialization system enters into the Error Free State because no error is detected yet so no emergency message is transmitted. CANopen device enters into error state when device detects an internal error (e.g: weather sensor is not transmitting current temperature) and EEC is set to 0x00FF.

Emergency Error Code (EEC)	Description
00FFh	Generic Error
0000h	No Error\Reset error

Table 10. Emergency error code values

Error register (ER)	Description
00h	No Error
21h	Generic Error

Table 11.Error code values

Last 5 bytes of EMCY message are Manufacturer Specific Error Function (MSEF). These 5 bytes are subdivided as given in following table. [12]

EMCY data frame Bytes	Description
Byte-3	Lower byte of error code
Byte-4	High byte of error code
Byte-5	Error counter
Byte-6	Not used in current implementations
Byte-7	Not used in current implementations

Table 12. EMCY Message data bytes

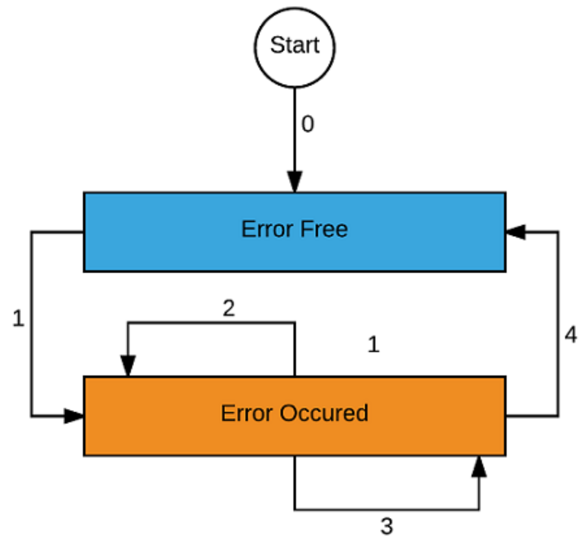


Figure 23. Emergency object state transitions

If error code (value of byte 3 and 4) are set twice in a row it will be logged as “Double set” error. Similarly, if same error code resets twice will be “Double reset”. Error counter is used to count the number of times same error occurred. The value of error counter is increased with each occurrence of corresponding error. Maximum value of counter is $(FF)_{16}$ if counter reached its maximum value it will remain FF after this.

5.3 Offline Data Analysis

In first phase of thesis task was to develop a PC based tool to log CAN bus communication and log the errors and timing violations in a separate file. SYNC gap of all devices are defined in a file *syncGap.conf*. To make sure all the nodes transmit their PDO with the sync gap mentioned in the sync file.

As per first phase there was no source to cross check the number of SYNC gaps between the occurrence of two TPDO or RPDO. So I was assuming that the first SYNC gap between two PDOs (TPDO or RPDO) is correct. There is a possibility that the first occurrence of Process Data Object (PDO) is wrong (PDO missed the expected SYNC slot). If first occurrence of PDO is wrong, it will lead to bundle of errors (missing SYNC slot) but in reality they was only first PDO missing the SYNC slot. E.g, Node GenSet TPDO-1 (Node ID 65) SYNC gap is 5 which means if first occurrence would be in SYNC slot 1 next would be Sync 6, 11, 16 and so on. If first occurrence of TPDO-1 missed the SYNC slot due to some technical problem then it would be difficult to catch this error. Sample log data is given below.

```
1) 13:40:22:805:000, Rx, 1, 0x080,           01,SYNC 01
2) 13:40:22:806:000, Rx, 8, 0x1C1,           00 00 00 00
   7D 00 7D 00,    TPDO1 Node 65
3) 13:40:22:825:000, Rx, 1, 0x080,           02,SYNC 01
4) 13:40:22:845:000, Rx, 1, 0x080,           03,SYNC 03
5) 13:40:22:865:000, Rx, 1, 0x080,           04,SYNC 04
6) 13:40:22:885:000, Rx, 1, 0x080,           05,SYNC 05
7) 13:40:22:905:000, Rx, 1, 0x080,           06,SYNC 06
8) 13:40:22:906:000, Rx, 8, 0x1C1,           00 00 00 00
   7D 00 7D 00,    TPDO1 Node 65
9) 13:40:22:925:000, Rx, 1, 0x080,           07,SYNC 07
10) 13:40:22:945:000, Rx, 1, 0x080,          08,SYNC 08
11) 13:40:22:965:000, Rx, 1, 0x080,          09,SYNC 09
```

Listing 2: Logger sample data

SYNC gap of node 0x1C1 (GenSet TPDO 1) is 5 sync. There is no option to cross check whether this gap is correct or not. To make sure this gap is correct, use the configuration file in Offline analysis phase. Which contain all the nodes their IDs and respective SYNC gaps of TPDOs and RPDOs.

In first phase there was no configuration file. Sync slot difference between first 2 occurrences between two PDOs (TPDO or RPDO) is the sync gap. To overcome this problem, "*SyncGap.conf*" and "*SystemConfig.conf*" file used in second phase.

5.3.1 Generation of SYNCgap and SystemConfig File

System configuration file (*SystemConfig.conf*) contains the category of the component, component name, component Identification number, IP and reference. File format is defined in the DB. Super admin can easily change the format of the file. And can generate new configuration file. From "*SystemConfig.conf*" it is easy to get the name of the component. This Component name, served as foreign key in second file "*SyncGap.conf*" to get the number of TPDO, RPDOs and their sync gaps.

```
CAT::Battery.COMP::JCI.ID::97.IP::3.REF::3_5
```

Listing 3: System configuration of Node Battery

In example data of file "*SystemConfig.conf*" given above, Component name is JCI, component lies in "Battery" category, component ID is 97(Component ID is primary key of file "*SystemConfig.conf*") , IP is 3 and reference is 3_5.

"*SyncGap.conf*" file contain the information, how much TPDO and RPDO each device has and their respective synchronization gap.

```
Battery_TPDO__GAP1_5__GAP2_200__GAP3_50__GAP4_0
Battery_RPDO__GAP1_5__GAP2_0__GAP3_0__GAP4_0
```

Listing 4: Sync Gap of Node Battery

Lines given above are from the file *SyncGap.conf* , depicts that Node "Battery" has 3 TPDOs and 1 RPDO. First TPDO with the SYNC gap of 5 *(TPDO 1* will be transmitted after 5 Sync. In DBHS configuration, sync window contains 200 sync slots it means TPDO 1 will be transmitted 40 times in each sync window). Second TPDO of node battery will be transmitted with sync gap of 200(once per sync window). TPDO 3 of node battery will be transmitted with the sync gap of 50(4 times in each sync Window). Example of Node battery TPDO1 is given below.

```
1) 10:04:56:152:000, Rx, 1, 0x080,          01,      SYNC 01
2) 10:04:56:155:000, Rx, 8, 0x1E1,          02 99 00 7D 54
   0E 52 0E,    TPDO1 Node 97
3) 10:04:56:172:000, Rx, 1, 0x080,          02,      SYNC 02
4) 10:04:56:192:000, Rx, 1, 0x080,          03,      SYNC 03
5) 10:04:56:212:000, Rx, 1, 0x080,          04,      SYNC 04
6) 10:04:56:232:000, Rx, 1, 0x080,          05,      SYNC 05
7) 10:04:56:252:000, Rx, 1, 0x080,          06,      SYNC 06
8) 10:04:56:254:000, Rx, 8, 0x1E1,          02 99 00 7D 54
   0E 52 0E,    TPDO1 Node 97
```

Listing 5:Sample logger data

First occurrence of TPDO1 of node battery occurred in sync slot 01. Sync Gap of TPDO1 is 5. Next occurrence of node “Battery” TPDO1 will be in Sync slot 06, if second occurrence of node “Battery” transmitted late monitoring application can now catch this delays with the use of configuration files from SCU. On the other hand node “Battery” has only one RPDO. RPDO 1 with sync gap of 5 sync.

5.3.2 SYNCgap and SystemConfig File

In CANopen communication network SCU is working as secondary master after system boots up. The duration between start or restart of the system and system boot up (initial 5 to 7 seconds) SCU is master because BCU is not configured yet. SCU configures all the devices configure with the network and cross verify these devices with “*SystemConfig*” file.

If devices configured in “*SystemConfig*” file are different than in the actual network. SCU will generate some errors “Unknown devices connected with network”. System admin compulsorily generates new “*SystemConfig*” file after configuring new nodes in the network. To generate new configuration file super admin need to reset the SCU on “factory reset” state. SCU will generate new configuration files with updated information of all the devices connected to the network. After updating the configuration files now SCU start logging data in dump file.

5.3.3 Watchdog problem

There are two different mechanisms of watchdog. In our implementation Heartbeats monitoring is used to detect Watchdog:

- Node Guarding
- Heartbeat monitoring

Node Guarding

Network is polling each device to check the health after the configured period of time. NMT master sends the request to all NMT slaves to send their current communication state to NMT master within defined life time. If slave nodes will not respond to master node with current communication state in data section, NMT slave would be considered dead node of the network. On the other hand NMT slaves also check whether they are receiving request from NMT master within "node life time" or not. If there are no sign of life from NMT master node, NMT slaves considered master node is not alive anymore.

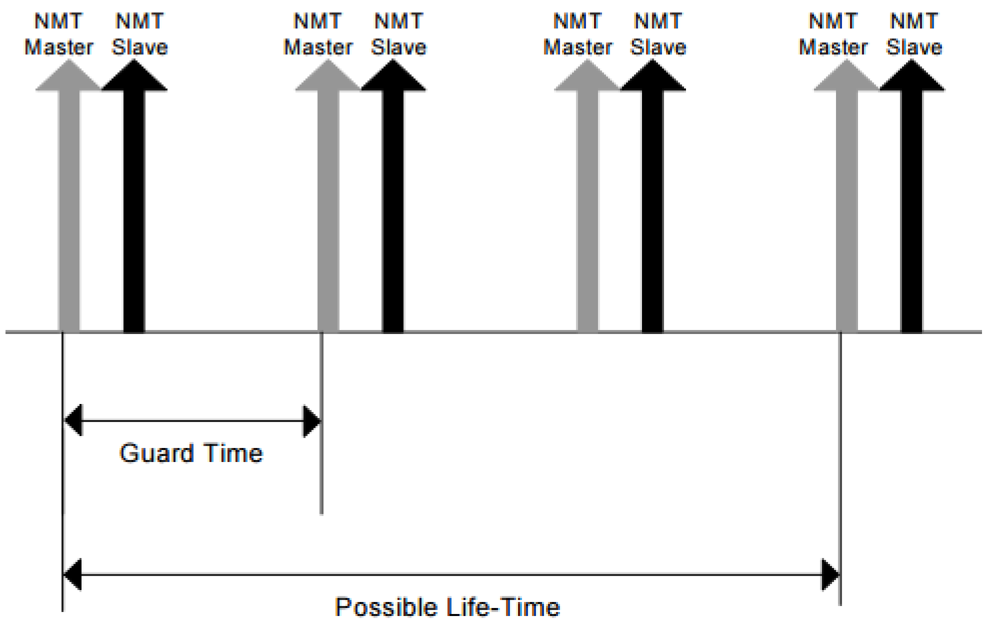


Figure 24.Node guard time Vs Life time.[9]

Heartbeat Monitoring

Each device gives an operational sign to the network manager or to the other devices of the network with their current communication state (pre operational, operational or stopped). As per CANopen standards Heartbeat data section contain only 1 byte data which is the NMT state of the node.

```
if ((hex($1) & 0x01) == 0x01){$interpret .= " PORST";}
if ((hex($1) & 0x02) == 0x02){$interpret .= " SWD";}
if ((hex($1) & 0x04) == 0x04){$interpret .= " PV";}
if ((hex($1) & 0x08) == 0x08){$interpret .= " CPU_SYS";}
if ((hex($1) & 0x10) == 0x10){$interpret .= "
CPU_LOCKUP";}
if ((hex($1) & 0x20) == 0x20){$interpret .= " WDT";}
if ((hex($1) & 0x40) == 0x40){$interpret .= " RESERVED";}
if ((hex($1) & 0x80) == 0x80){$interpret .= "
PARITY_ERR";}
```

Listing 6: Watchdog Values of 2nd Byte of Heatbeat data

In DBH system implementation, HB DLC is extended to 2 bytes. 2nd byte contains some information of BCU failure. 2nd byte contains one of the values given in the list below. There could be more than one reasons of rebooting the BCU. If data section of Heartbeat message (0x700+Node ID) contains 2 bytes data it means BCU is interrupted and getting restarting so timing check of all objects after BCU rebooting has to be done in separate data logging file.

5.4 Embedded Platform

Third phase is to go for embedded platform and enable to execute monitoring tool on Linux based SCU. In Embedded online monitoring CAN data has been passed as standard input to analysis thread. In analysis thread all the objects of CANopen will be monitored. Any protocol violation has to be logged as error in error log file. Default allowed jitter for nodes heartbeat is 20 ms.

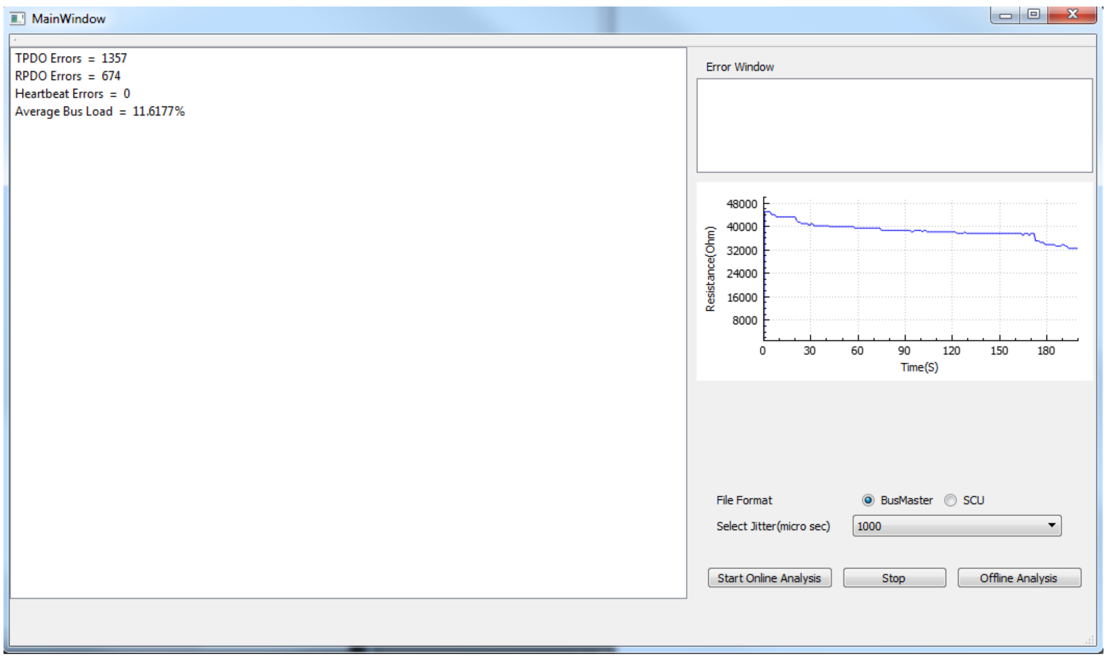
5.5 IsoMon Payload Analysis

Node “Isolation monitor” of CANopen network carries 5 bytes of data in message data section while sending data to master (TPDO) or responses from master (RPDO). In TPDO 5 bytes of IsoMon CAN message carry three pieces of information. These are as follows:

- Resistance(2 bytes)
- Voltage(2 bytes)
- Status(1 byte)

When IsoMonitor receive any CAN message. Data section of CAN message frame must contain 5 bytes of information:

- Command
- Warning Level
- Error Level



TPDO Errors = 1357
RPDO Errors = 674
Heartbeat Errors = 0
Average Bus Load = 11.6177%

File Format [✓ BusMaster] SCU
Select Jitter(micro sec) 1000

Figure 25. Screenshot: IsoMon PDO list

Screenshot above is the resultant screen of Offline analysis of logged data. Graph in right depicts the payload of node “IsoMonitor”. Top left window of screenshot display summary of file analysis.

In Isolation monitor "Resistance" should be between 1 and 65535000 Ohm. Minimum voltage of "IsoMonitor" is 0 and maximum should not be more than 1000. Graph of node "IsoMonitor" Resistance vs time is shown in figure 22 above. Possible state values of device are in table below.

IsoMon state	State value
0x0	Reset
0x10	Self-check
0x20	Running with invalid value
0x30	Running with valid value
0x40	Device error

Table 13. Iso Monitor possible values

5.6 Summary

Overall implementation of my research works has been divided into three phases (online PC based monitoring tool development, Offline data analysis, embedded platform). First task was to select system logging hardware to log communication channel data into windows based tool. After hardware selection of PEAK CAN adapter next step was configuration of PCAN adopter. After successfully configuration and logging data next step was to start data analysis. All the objects like SYNC, Heartbeats, SDOs and PDO. Logging all the errors and timing violations from any component over the network into logError.txt file. Second phase of thesis was to develop offline data analysis. In 2nd phase task was to be able to analyze already logged data independently. While logging data using dedicated hardware like PCAN logger is accurate. When read from already logged dump files, it's easy to analyze because here I can check the same data with different configurations. E.g. analyze one dump files with SYNC jitter 1 millisecond and 2 milliseconds or even more. There are different formats of input files accordingly different timestamp accuracy. PCAN give accuracy in microseconds however while logging with SCU timestamp is in milliseconds.

System configuration files were integrated to verify objects timing behavior. In last phase, tasks were to calculate the payload analysis of one sample node of the network and execute logging and monitoring tool on SCU platform. This phase enable our connection box SCU to do real time data logging and analysis.

Chapter 6

6. Results

In this chapter of thesis I am going to explain the end results of implementation phases which we explained in chapter 5(Implementation). As I already mentioned that our implementation is in three different phases (PC based monitoring tool development, Offline analysis and embedded platform).

6.1 Results Criteria

The results criteria of each phase are different. End results criterion of success is based on expected results. We will check how much of our expected results are satisfied with our results achieved after implementation. This thesis is the starting point to develop a logging and monitoring tool for DBH system. There is much work to do further to increase efficiency. These future tasks and suggestion will be explained in coming chapter “Future work”.

The result success criterion is based on the desired features of the thesis mentioned in thesis tasks. These desired characteristics of the thesis are given below:

1. Reduce debugging time
2. Log inter network communication
3. Log protocol violations
4. Flexible jitter testing
5. Payload of nodes
6. Bus load measurement
7. Scale down system complexity

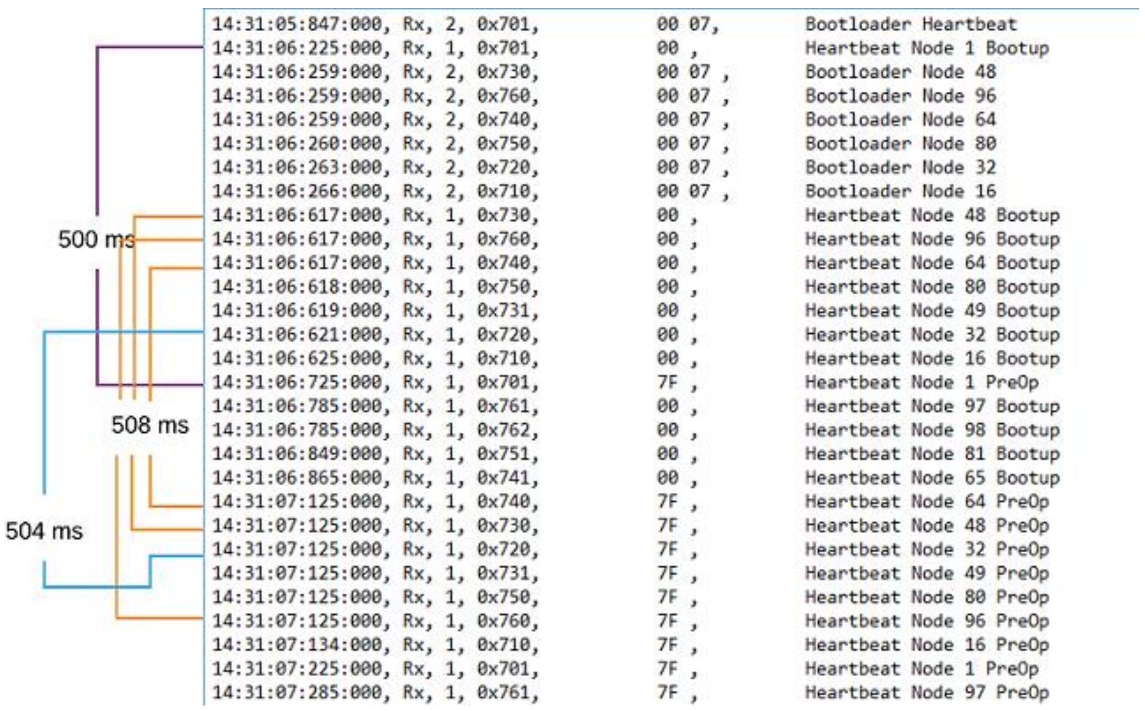
6.2 Logging and Monitoring Results

First phase of desired, logging and monitoring tool was “ PC based monitoring tool development”. The expected end result of this phase was to have a logging tool that can log CAN bus data using external hardware, PCAN USB adapter and monitor following CANopen objects:

- Monitoring SYNC slot Jitter
- Monitoring of Heartbeat messages and Jitter
- Analysis of PDOs and SDOs
- Emergency messages

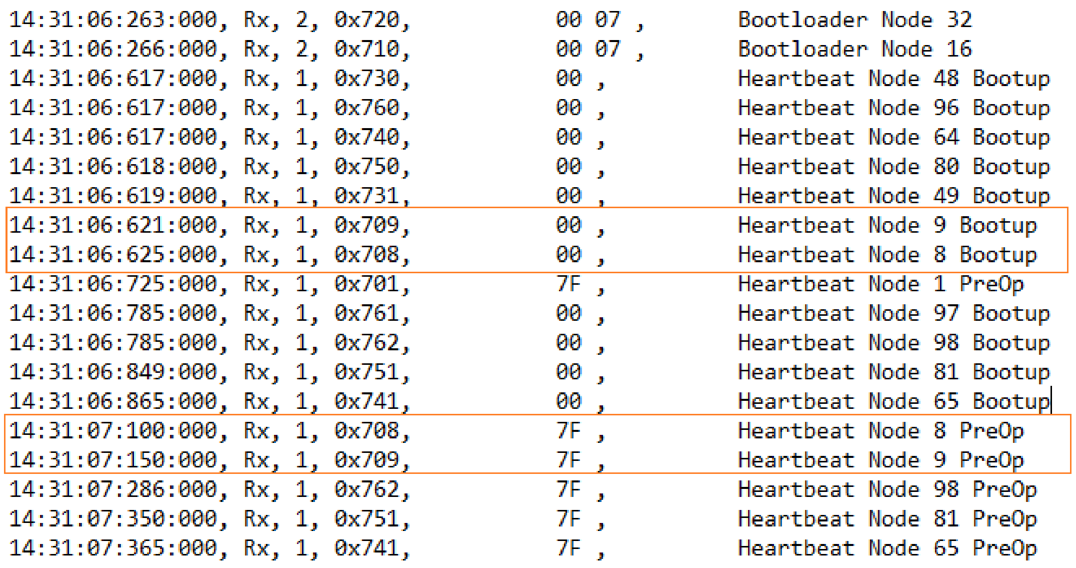
6.2.1 Monitoring of HB Messages and Jitter

All the nodes in the network must inform the master node about their current status. As per first phase implementation time interval for all nodes to report to the master node of the network is 500 milliseconds. Allowed Jitter is ± 20 milliseconds. Node of the network sending "Node status" to the master node before 480 milliseconds or after 520 milliseconds will be considered network node is not alive anymore. The analyses tool must log this as an error that in the error log file. Example HB log file and error log file is given below.



```
14:31:05:847:000, Rx, 2, 0x701,          00 07,        Bootloader Heartbeat
14:31:06:225:000, Rx, 1, 0x701,          00 ,          Heartbeat Node 1 Bootup
14:31:06:259:000, Rx, 2, 0x730,          00 07 ,       Bootloader Node 48
14:31:06:259:000, Rx, 2, 0x760,          00 07 ,       Bootloader Node 96
14:31:06:259:000, Rx, 2, 0x740,          00 07 ,       Bootloader Node 64
14:31:06:260:000, Rx, 2, 0x750,          00 07 ,       Bootloader Node 80
14:31:06:263:000, Rx, 2, 0x720,          00 07 ,       Bootloader Node 32
14:31:06:266:000, Rx, 2, 0x710,          00 07 ,       Bootloader Node 16
14:31:06:617:000, Rx, 1, 0x730,          00 ,          Heartbeat Node 48 Bootup
14:31:06:617:000, Rx, 1, 0x760,          00 ,          Heartbeat Node 96 Bootup
14:31:06:617:000, Rx, 1, 0x740,          00 ,          Heartbeat Node 64 Bootup
14:31:06:618:000, Rx, 1, 0x750,          00 ,          Heartbeat Node 80 Bootup
14:31:06:619:000, Rx, 1, 0x731,          00 ,          Heartbeat Node 49 Bootup
14:31:06:621:000, Rx, 1, 0x720,          00 ,          Heartbeat Node 32 Bootup
14:31:06:625:000, Rx, 1, 0x710,          00 ,          Heartbeat Node 16 Bootup
14:31:06:725:000, Rx, 1, 0x701,          7F ,          Heartbeat Node 1 PreOp
14:31:06:785:000, Rx, 1, 0x761,          00 ,          Heartbeat Node 97 Bootup
14:31:06:785:000, Rx, 1, 0x762,          00 ,          Heartbeat Node 98 Bootup
14:31:06:849:000, Rx, 1, 0x751,          00 ,          Heartbeat Node 81 Bootup
14:31:06:865:000, Rx, 1, 0x741,          00 ,          Heartbeat Node 65 Bootup
14:31:07:125:000, Rx, 1, 0x740,          7F ,          Heartbeat Node 64 PreOp
14:31:07:125:000, Rx, 1, 0x730,          7F ,          Heartbeat Node 48 PreOp
14:31:07:125:000, Rx, 1, 0x720,          7F ,          Heartbeat Node 32 PreOp
14:31:07:125:000, Rx, 1, 0x731,          7F ,          Heartbeat Node 49 PreOp
14:31:07:125:000, Rx, 1, 0x750,          7F ,          Heartbeat Node 80 PreOp
14:31:07:125:000, Rx, 1, 0x760,          7F ,          Heartbeat Node 96 PreOp
14:31:07:134:000, Rx, 1, 0x710,          7F ,          Heartbeat Node 16 PreOp
14:31:07:225:000, Rx, 1, 0x701,          7F ,          Heartbeat Node 1 PreOp
14:31:07:285:000, Rx, 1, 0x761,          7F ,          Heartbeat Node 97 PreOp
```

Figure 26. Nodes Heard beats and time intervals respectively



```
14:31:06:263:000, Rx, 2, 0x720,         00 07 ,      Bootloader Node 32
14:31:06:266:000, Rx, 2, 0x710,         00 07 ,      Bootloader Node 16
14:31:06:617:000, Rx, 1, 0x730,         00 ,         Heartbeat Node 48 Bootup
14:31:06:617:000, Rx, 1, 0x760,         00 ,         Heartbeat Node 96 Bootup
14:31:06:617:000, Rx, 1, 0x740,         00 ,         Heartbeat Node 64 Bootup
14:31:06:618:000, Rx, 1, 0x750,         00 ,         Heartbeat Node 80 Bootup
14:31:06:619:000, Rx, 1, 0x731,         00 ,         Heartbeat Node 49 Bootup
14:31:06:621:000, Rx, 1, 0x709,         00 ,         Heartbeat Node 9 Bootup
14:31:06:625:000, Rx, 1, 0x708,         00 ,         Heartbeat Node 8 Bootup
14:31:06:725:000, Rx, 1, 0x701,         7F ,         Heartbeat Node 1 PreOp
14:31:06:785:000, Rx, 1, 0x761,         00 ,         Heartbeat Node 97 Bootup
14:31:06:785:000, Rx, 1, 0x762,         00 ,         Heartbeat Node 98 Bootup
14:31:06:849:000, Rx, 1, 0x751,         00 ,         Heartbeat Node 81 Bootup
14:31:06:865:000, Rx, 1, 0x741,         00 ,         Heartbeat Node 65 Bootup
14:31:07:100:000, Rx, 1, 0x708,         7F ,         Heartbeat Node 8 PreOp
14:31:07:150:000, Rx, 1, 0x709,         7F ,         Heartbeat Node 9 PreOp
14:31:07:286:000, Rx, 1, 0x762,         7F ,         Heartbeat Node 98 PreOp
14:31:07:350:000, Rx, 1, 0x751,         7F ,         Heartbeat Node 81 PreOp
14:31:07:365:000, Rx, 1, 0x741,         7F ,         Heartbeat Node 65 PreOp
```

Figure 27. (Screenshot) Logged data with late\early heartbeat messages

Our monitoring tool detects this error and logs it into the *ErrorLog.txt* file. Content of *loggerError.txt* file is given below in listing 7.

```
0x709   5:50:47:193:000    [Node   ID]=9   TcuNode_HeartBeat
late/earlier [Time gap: 529] [Allowed Jitter: +/- 20]

0x708   5:50:47:193:000    [Node   ID]=8   ScuNode_HeartBeat
late/earlier [Time gap: 475] [Allowed Jitter: +/- 20]
```

Listing 7: Errorlog data

Node-ID 9 has HB time gap 529 which is more than 520. It has been logged into the error file. Similarly Node-ID 8 also logged in the error file because node has HB time gap is 475 less than 480.

6.2.2 Monitoring PDO Messages

There are two kind of Process data objects (PDO) in CANopen either they could be TPDO or RPDO. All the nodes in the network can transmit maximum 4 TPDO and they can receive 4 RPDO. Each PDO (RPDO or TPDO) must be transmitted /received after a specific SYNC gaps defined in *syncGap.txt*. In first phase of thesis idea was to assume that first gap between two PDOs would be considered correct and all the PDOs in afterword would be compared with the difference between first two PDOs of any network node.

```
10:04:56:012:000, Rx, 1, 0x080,         79,
10:04:56:014:000, Rx, 8, 0x1A1,         04 21 4E E8 03 00
7D 3C,
10:04:56:032:000, Rx, 1, 0x080,         7A,
7D 07,
10:04:56:034:000, Rx, 8, 0x1A1,         04 21 4E E8 03 00
7D 3C,
10:04:56:052:000, Rx, 1, 0x080,         7B,
7D 07,
10:04:56:074:000, Rx, 8, 0x1D1,         00 F8 2F 02 03 00
7D 00,,
10:04:56:072:000, Rx, 1, 0x080,         7C,
7D 07,
10:04:56:034:000, Rx, 8, 0x1A1,         04 21 4E E8 03 00
7D 3C,
```

Listing 8: Sample Logger data to check PDO sync slot

In the log data given above node 0x1A1 (TPDO-1 of Node-33) transmitted in SYNC 79 and second transmission of 0x1A1 is in SYNC 7A. This log data depicts that the sync gap of TPDO-1 of Node-33 is 1. In future, if log file has sync gap of TPDO-1 Node-33 greater or less than 1. The tool must log this as timing violation of node. Example log file and error file is given below.

```
10:04:56:012:000, Rx, 1, 0x080,           79,
10:04:56:014:000, Rx, 8, 0x1D1,           00 F8 2F 02 03
00 7D 00, TPDO1 Node 81
10:04:56:014:000, Rx, 8, 0x1C1,           00 00 00 00 7D
00 7D 00, TPDO1 Node 65
10:04:56:032:000, Rx, 1, 0x080,           7A,
10:04:56:034:000, Rx, 8, 0x1D1,           00 F8 2F 02 03
00 7D 00, TPDO1 Node 81
10:04:56:052:000, Rx, 1, 0x080,           7B,
10:04:56:054:000, Rx, 8, 0x1D1,          00 F8 2F 02 03
00 7D 00, TPDO1 Node     81
10:04:56:072:000, Rx, 1, 0x080,          7C,
10:04:56:074:000, Rx, 8, 0x1D1,          00 F8 2F 02 03
00 7D 00, TPDO1 Node 81
10:04:56:092:000, Rx, 1, 0x080,          7D,
10:04:56:094:000, Rx, 8, 0x1A1,          04 21 4E E8 03
00 7D 3C, TPDO1 Node 33
10:04:56:092:000, Rx, 1, 0x080,          7E,
10:04:56:094:000, Rx, 8, 0x1D1,          00 F8 2F 02 03
00 7D 00, TPDO1 Node 81
10:04:56:092:000, Rx, 1, 0x080,          7F,
10:04:56:094:000, Rx, 8, 0x1D1,          00 F8 2F 02 03
00 7D 00, TPDO1 Node 81
10:04:56:094:000, Rx, 8, 0x1C1,          00 00 00 00 7D
00 7D 00, TPDO1 Node 65
```

Listing 9: Logger data to check Sync

```
0x1D1 5:50:44:759:000  [Node ID]=81 GenSet_TPDO_1
late/earlier [Current SYNC Gap: 2] [Expected SYNC gap:1]

0x1C1 5:50:44:760:000  [Node ID]=65 AcCharger_TPDO_1
late/earlier [Current SYNC Gap: 6] [Expected SYNC gap:5]
```

Listing 10: ErrorLog file with missing SYNC error message

It's clear from the error log file window monitoring tool detects TPDO-1 of the Node-ID 81 is missing 1 SYNC slot. Expected SYNC gap was 1 but in SYNC slot "7E" TPDO 1 of node "GenSet" is missing. Similarly TPDO 1 of node 65 ("AcCharger") is also missing the expected SYNC slot. Expected gap was 5 but current gap is 6. This is the expected behavior of monitoring tool to detect all the nodes missing their respective sync slots.

6.2.3 Using Configuration Files

System admin can generate configuration file from database which contain all the required information about the nodes e.g. ID, Sync gap, number of PDO and SDO. Monitoring tool can verify the SYNC gaps of each node from configuration files. If current gap of node (TPDO or RPDO) is more than what is defined in the *syncGap.conf* file, our logging and the monitoring tool log it as an error. Sync gap of all the nodes are given in below:

```
DcDc_TPDO__GAP1_5__GAP2_0__GAP3_0__GAP4_0
DcDc_RPDO__GAP1_5__GAP2_0__GAP3_0__GAP4_0
GenSet_TPDO__GAP1_1__GAP2_5__GAP3_0__GAP4_0
GenSet_RPDO__GAP1_5__GAP2_0__GAP3_0__GAP4_0
Drive_TPDO__GAP1_1__GAP2_50__GAP3_0__GAP4_0
Drive_RPDO__GAP1_1__GAP2_5__GAP3_0__GAP4_0
Battery_TPDO__GAP1_5__GAP2_200__GAP3_50__GAP4_0
Battery_RPDO__GAP1_5__GAP2_0__GAP3_0__GAP4_0
BoxConnection_TPDO__GAP1_0__GAP2_0__GAP3_0__GAP4_0
BoxConnection_RPDO__GAP1_0__GAP2_0__GAP3_0__GAP4_0
Dcu_TPDO__GAP1_5__GAP2_20__GAP3_200__GAP4_0
Dcu_RPDO__GAP1_10__GAP2_0__GAP3_0__GAP4_0
DriveRotation_TPDO__GAP1_0__GAP2_0__GAP3_0__GAP4_0
DriveRotation_RPDO__GAP1_0__GAP2_0__GAP3_0__GAP4_0
AcCharger_TPDO__GAP1_5__GAP2_5__GAP3_50__GAP4_0
AcCharger_RPDO__GAP1_5__GAP2_0__GAP3_0__GAP4_0
DcAc_TPDO__GAP1_0__GAP2_0__GAP3_0__GAP4_0
DcAc_RPDO__GAP1_0__GAP2_0__GAP3_0__GAP4_0
IsoMon_TPDO__GAP1_50__GAP2_0__GAP3_0__GAP4_0
IsoMon_RPDO__GAP1_20__GAP2_0__GAP3_0__GAP4_0
Bcu_TPDO__GAP1_5__GAP2_5__GAP3_0__GAP4_0
Bcu_RPDO__GAP1_5__GAP2_0__GAP3_0__GAP4_0
```

Listing 11: SyncGap.conf

This is the required behavior of tool. "*SyncGap.conf*" file make it easy to check, which node in the network is missing assigned Sync slot. Sometimes one node in the network create problem for rest of the nodes in the network which leads to the system failure.

6.3 SYNC and Emergency Messages Monitoring

SYNC period of each slot is 20 milliseconds (10 milliseconds for the synchronous communication and 10 milliseconds for the Asynchronous communication) with the allowed jitter could vary in-between 0.5 to 2 milliseconds. If any sync slot take time more or less than allowed (SYNC period ± Jitter). It will be logged as SYNC error in "*LoggerError.txt*" file. Example of SYNC log file and error file is given below.

```
10:04:56:052:000, Rx, 1, 0x080,          7B,      SYNC 7B
10:04:56:053:000, Rx, 2, 0x281,          2D 00,
10:04:56:054:000, Rx, 5, 0x2D1,          64 5A 43 6E 52,
10:04:56:065:000, Rx, 2, 0x709,          05 B5,
10:04:56:072:000, Rx, 1, 0x080,          7C,      SYNC 7C
10:04:56:073:000, Rx, 5, 0x241,          00 00 7D 00 7
10:04:56:073:000, Rx, 5, 0x251,          02 02 03 00 7D,
10:04:56:074:000, Rx, 5, 0x2D1,          64 5A 43 6E 52,
10:04:56:092:000, Rx, 1, 0x080,          7D,      SYNC 7D
10:04:56:093:000, Rx, 3, 0x220,          00 00 00,
10:04:56:093:000, Rx, 7, 0x221,    02 20 4E 65 04 A5 A5,
10:04:56:094:000, Rx, 5, 0x2D1,          64 5A 43 6E 52,
10:04:56:112:000, Rx, 1, 0x080,          7E,      SYNC 7E
10:04:56:113:000, Rx, 5, 0x251,          02 02 03 00 7D,
10:04:56:113:000, Rx, 6, 0x181,        00 04 02 03 03 0D
10:04:56:115:000, Rx, 5, 0x2D1,          64 5A 43 6E
10:04:56:132:000, Rx, 1, 0x080,          7F,      SYNC 7F
10:04:56:133:000, Rx, 5, 0x251,          02 02 03 00
10:04:56:135:000, Rx, 5, 0x2D1,          64 5A 43 6E 52,
10:04:56:144:000, Rx, 1, 0x701,          05,
10:04:56:154:000, Rx, 1, 0x080,          80,      SYNC 80
```

Listing 12: logger data to check Sync timing (BusMaster file format)

The time difference between *Sync 7F* and *Sync 80* is 22. CANopen Protocol was expecting sync after 21 milliseconds (SYNC period + Jitter). The monitoring tool detects and logs these errors into error file. Before this tool it was time consuming process to make sure none of the sync is exceeding from the allowed period (Sync period ± Jitter).

6.4 Summary

Over all this logging and the monitoring tool increase our system efficiency to detect errors and hidden protocol timing violations. Debugging time also reduced from hours to seconds. Before this analysis tool it was very difficult and time consuming process to detect the errors and timing misbehavior. Now with newly developed analyzer we can easily log data and errors with flexibility. There is an option to play back previously logged data anytime with different configurations. This analyzer tool also tells about the status of the node while error occurred and what is the possible reason of error either it was timing violations or some data frames lost. Node payload analysis graph tells the parameters inside the specific node. Check the NMT state of all the other devices connected with network during occurrence of error.

Chapter 7

7. Future Work

Logging and monitoring tools help the development team to analyze node communications and their timing behavior which are connected with the CAN network. If there are some irregularities and protocol violations or node is not alive anymore, analysis tool can easily log these violations in error log file. There are many 3rd party tools to monitor and analyze data over a CAN bus but using off the shelf analyzer has some disadvantages like security and relatively extra source to carry. To overcome all these disadvantages the solution on the table was to go for developing our own logging and monitoring tool. In the future we can go for XMC4500 microcontroller so our monitoring tool can log inter nodes communication directly on micro SD card. This MC card has an option to configure 64 GB SD card.

In Online data analysis that detects the first occurrences of any node PDO nodes communication is correct or not. Currently our sorting algorithm is serving our purpose. To increase system efficiency this must be solved. One proposed solution is to move the data frame into adjacent SYNC frames. Checking the data frame with adjacent frame reduces the possibility of first SYNC missing error.

In our current implementation we are calculating the payload of only one node of the network “IsoMonitor”. Next steps would be to calculate the payload of all nodes of the network separately. We will be able to playback the log file in offline analysis with payload analysis of each node of the network. This is significant for debugging our network.

During data logging, either using SCU or PCAN USB we don’t know anything about the expected arrival time of first sync slot. There is no method currently to check whether the occurrence of the first SYNC is correct or not. One of the possible solutions is to use stacking sync slot time period to take the mode of first 10 sync slot time period and select the one which is repeated maximum number of times. Addition of fast playback simulation would also cut off debugging time. In normal simulation, analyses of 2 hours logged data of connection box will take two hours to play it back.

In offline analysis, monitoring tool input is a previously logged data file. CPU is capable of reading much faster than reading from CAN bus. CAN communication sync slots are not fully filled with process data. In most of the sync slots, synchronous section is free 50% to 60% or even more. Mostly process data of sync slot get transmitted in initial 3 milliseconds out of 10 milliseconds.

Bibliography

[1] Marco Di Natale.Haibo Zeng, Paolo Giusto. Arkadeb Ghosal . (2012).*Understanding and Using the Control Area Network Communication Protocol*. Springer-Verlag. NY. ISBN 978-1-4614-0314-2

[2] Wilfried Voss. (2005). *A Comprehensive Guide to Control Area Network*.Copperhill Technologies Corporation. Amherst, MA. ISBN 0-976511-6-0-6.

[3] Mohammad Farsi, Manuel Bernardo, Martins Barbosa. (1999). *CANopen Implimentation: Application to industrial network*. Research Studies Press. London. ISBN 978-0863802478.

[4] National Instruments. (2016).The Basics of CANopen National instrument, [Online] Available at: http://www.ni.com/white-paper/14162/en/ [Accessed: 01.05.2016]

[5] Wolfhard Lawrenz. (2013). *CAN system engineering. from theory to practical application*, second edition. Springer-Verlag. London. ISBN 978-1-4471-5613-0.

[6] Milo˘s Gajdo˘, (2008), *CAN bus Communication Protocol Support and Monitoring*.

[7] Steve Corrigan. (2016). Application Report. *Introduction to Controller Area Network (CAN).* Available at: http://www.ti.com/lit/an/sloa101a/sloa101a.pdf [Accessed: 19.06.2016]

[8] Prof. Alejandro Masrur. (2013). Software Platform for Automotive Systems. Lecture notes distributed in class at The University of technology Chemnitz.

[9] Olaf Pfeiffer , Andrew Ayre , Christian Keydel.(2008) *Embedded Networking with CAN and CANopen*. Copperhill Technologies Corporation, Greenfield, MA. ISBN 978-0-9765116-2-5.
[10] canopensolutions.com, (2016) Object Dictionary and Electronic Data Sheet, [Online], Available at: http://www.canopensolutions.com/english/about_canopen/communication.shtml [Accessed: 20.06.2016]

[11] Matej Kubicka. (2011). *CANopen implementation*.

[12] CAN in Automation,(2011) *301 CANopen application layer and communication profile*. Version 4.2.0, Available at: http://www.can-cia.org/standardization/specifications/

[13] CAN EDS (2011). Introduction to CANopen ESA.

[14] can-cia.org, CANopen knowledge, [Online] Available at: http://www.can-cia.org/can-knowledge/canopen/canopen/ [Accessed: 12.05.2016]

[15] Peak-system.com, [Online], CAN interface for USB, User manual. Available at: http://www.peak-system.com/PCAN-USB.199.0.html?&L=1 [Accessed: 23.04.2016]

[16] Torqeedo Gmbh, Internal documents

[17] Kvaser, Kvaser leaf Light V2 user's guide. [Online]. Available at: https://www.kvaser.com/ products/kvaser-leaf-light-hs-v2/ [Accessed: 12.03.2016]

[18] National Instruments, [Online] Available at :http://www.ni.com/white-paper/2732/en/#toc7 [Accessed: 12.03.2016]

[19] AMC, CANopen Communication Manual [Online]. Available at: http://www.a-m-c.com/download/sw/dw5-4-2/AMCCANopenManual5-4.pdf [Accessed: 12.6.2016]

[20] Prof. Alejandro Masrur. *Software Platform automotive systems*, lecture notes distributed in class at The University of technology Chemnitz.

[21] Searchnetworking.com, *Network analyzer vs packet analyzer*, [Online] Available at: http://searchnetworking.techtarget.com/definition/network-analyzer [Accessed: 01.05.2016]

[22] R. Bosch. (1991). CAN specification, [Online]. Version 2.0. Stuttgart.

[23] Marco Di Natale. (2008*). Understanding and using the Controller Area Network*. Springer-Verlag. NY. ISBN 978-1-4614-0314-2.

Appendix A

Emergency Error Code Classes

Error code	**Description**
00xx$_h$	Error reset or no error
10xx$_h$	Generic error
20xx$_h$	Current
21xx$_h$	CANopen device input side
22xx$_h$	Current inside the CANopen device
23xx$_h$	CANopen device output side
30xx$_h$	Voltage
31xx$_h$	Mains voltage
32xx$_h$	Voltage inside the CANopen device
33xx$_h$	Output voltage
40xx$_h$	Temperature
41xx$_h$	Ambient temperature
42xx$_h$	CANopen device temperature
50xx$_h$	CANopen device hardware
60xx$_h$	CANopen device software
61xx$_h$	Internal software
62xx$_h$	User software
63xx$_h$	Data set
70xx$_h$	Additional modules

80xxh	Monitoring
81xxh	Communication
82xxh	Protocol error
90xxh	External error
F0xxh	Additional functions
FFxxh	CANopen device specific

Object Dictionary Entries

Index	Name	Type	Access
1000h	Device Type	UNSIGNED32	RO
1001h	Error register	UNSIGNED8	RO
1002h	Manufacturer Status Register	UNSIGNED32	RO
1003h	Pre-defined Error Field	UNSIGNED32	RO
1004h	Reserved	-	-
1005h	SYNC COB ID	UNSIGNED32	RW
1006h	Communication Cycle Period	UNSIGNED32	RW
1007h	Synchronous Window Length	UNSIGNED32	RW
1008h	Manufacturer Device Name	VISIBLE_STRING	RO
1009h	Manufacturer Hardware Version	VISIBLE_STRING	RO

100Ah	Manufacturer Software	VISIBLE_STRING	RO
100Bh	Reserved	-	-
100Ch	Guard Time	UNSIGNED16	RW
100Dh	Life time factor	UNSIGNED8	RW
100Eh	Reserved	-	RW
100Fh	Reserved	-	RW
1010h	Store parameters	UNSIGNED32	RW
1011h	Restore default parameters	UNSIGNED32	RW
1012h	Time COB ID	UNSIGNED32	RW
1013h	High Resolution Time Stamp	UNSIGNED32	RW
1014h	Emergency COB ID	UNSIGNED32	RW
1015h	Emergency Inhibit Time	UNSIGNED16	RW
1016h	Consumer Heartbeat Time	UNSIGNED32	RW
1017h	Producer Heartbeat Time	UNSIGNED16	RW
1018h	Identity	IDENTITY	RW
1019h	Reserved	-	-
1020h	Verify Configuration	UNSIGNED32	RW
1021h	Store EDS	DOMAIN	RW
1022h	Storage format	UNSIGNED32	RW
1023h	OS command	DEBUGGER_PAR (0025h)	RW
1024h	OS command Mode	UNSIGNED8	RW

1025h	OS debugger interface	DEBUGGER_PAR (0024h)	RW
1026h	OS prompt	UNSIGNED8	RW
1027h	Module list	UNSIGNED16	RW
1028h	Emergency consumer	UNSIGNED32	RW
1029h	Error behavior	UNSIGNED8	RW
102Ah – 11FFh	Reserved	-	-
1200h	1st SDO Server Parameters	SDO_PARAMETER (0022h)	RO
1201h – 127Fh	-	SDO_PARAMETER (0022h)	RW
1280h	-	SDO_PARAMETER (0022h)	RW
1281h – 12FFh	Additional SDO client parameters	SDO_PARAMETER (0022h)	RW
1300h – 13FFh	Reserved	-	-
1400h	1st Receive PDO Parameter	PDO_COMMUNICATION_ PARAMETER (0020h)	RW
1401h	2nd Receive PDO Parameter	PDO_COMMUNICATION_ PARAMETER (0020h)	RW
1402h	3rd Receive PDO Parameter	PDO_COMMUNICATION_ PARAMETER (0020h)	RW
1403h	4th Receive PDO Parameter	PDO_COMMUNICATION_ PARAMETER (0020h)	RW

Appendix B

Data log file PCAN USB

```
14:12:58:847:000, Rx, 2, 0x701,     00 07 ,                     Bootloader    Heartbeat
Node 1 Bootup PORST SWD PV

14:12:59:214:000, Rx, 1, 0x080,     02 ,                 SYNC 0x02

14:12:59:225:000, Rx, 1, 0x701,     00 ,                 Heartbeat Node 1 Bootup

14:12:59:234:000, Rx, 1, 0x080,     03 ,                 SYNC 0x03

14:12:59:245:000, Rx, 8, 0x081,     00 FF 21 00 01 01 00 00 ,   EMCY Node 1

14:12:59:254:000, Rx, 1, 0x080,     04 ,                 SYNC 0x04

14:12:59:259:000, Rx, 2, 0x730,     00 07 ,                     Bootloader    Heartbeat
Node 48 Bootup PORST SWD PV

14:12:59:259:000, Rx, 2, 0x760,     00 07 ,                     Bootloader    Heartbeat
Node 96 Bootup PORST SWD PV

14:12:59:259:000, Rx, 2, 0x740,     00 07 ,                     Bootloader    Heartbeat
Node 64 Bootup PORST SWD PV

14:12:59:260:000, Rx, 2, 0x750,     00 07 ,                     Bootloader    Heartbeat
Node 80 Bootup PORST SWD PV

14:12:59:263:000, Rx, 2, 0x720,     00 07 ,                     Bootloader    Heartbeat
Node 32 Bootup PORST SWD PV

14:12:59:266:000, Rx, 2, 0x710,     00 07 ,                     Bootloader    Heartbeat
Node 16 Bootup PORST SWD PV

14:12:59:274:000, Rx, 1, 0x080,     05 ,                 SYNC 0x05

14:12:59:294:000, Rx, 1, 0x080,     06 ,                 SYNC 0x06

14:12:59:314:000, Rx, 1, 0x080,     07 ,                 SYNC 0x07
```

```
14:12:59:334:000, Rx, 1, 0x080,        08 ,                   SYNC 0x08
14:12:59:354:000, Rx, 1, 0x080,        09 ,                   SYNC 0x09
14:12:59:374:000, Rx, 1, 0x080,        0A ,                   SYNC 0x0A
14:12:59:394:000, Rx, 1, 0x080,        0B ,                   SYNC 0x0B
14:12:59:414:000, Rx, 1, 0x080,        0C ,                   SYNC 0x0C
14:12:59:434:000, Rx, 1, 0x080,        0D ,                   SYNC 0x0D
14:12:59:454:000, Rx, 1, 0x080,        0E ,                   SYNC 0x0E
14:12:59:474:000, Rx, 1, 0x080,        0F ,                   SYNC 0x0F
14:12:59:494:000, Rx, 1, 0x080,        10 ,                   SYNC 0x10
14:12:59:514:000, Rx, 1, 0x080,        11 ,                   SYNC 0x11
14:12:59:534:000, Rx, 1, 0x080,        12 ,                   SYNC 0x12
14:12:59:554:000, Rx, 1, 0x080,        13 ,                   SYNC 0x13
14:12:59:574:000, Rx, 1, 0x080,        14 ,                   SYNC 0x14
14:12:59:594:000, Rx, 1, 0x080,        15 ,                   SYNC 0x15
14:12:59:614:000, Rx, 1, 0x080,        16 ,                   SYNC 0x16
14:12:59:617:000, Rx, 1, 0x730,        00 ,                   Heartbeat Node 48 Bootup
14:12:59:617:000, Rx, 1, 0x760,        00 ,                   Heartbeat Node 96 Bootup
14:12:59:617:000, Rx, 1, 0x740,        00 ,                   Heartbeat Node 64 Bootup
14:12:59:617:000, Rx, 8, 0x0B0,        00 FF 21 00 20 01 00 00 ,    EMCY Node 48
14:12:59:617:000, Rx, 8, 0x0B0,        00 FF 21 01 70 01 00 00 ,    EMCY Node 48
14:12:59:618:000, Rx, 8, 0x0C0,        00 FF 21 00 20 01 00 00 ,    EMCY Node 64
14:12:59:618:000, Rx, 8, 0x0E0,        00 FF 21 00 20 01 00 00 ,    EMCY Node 96
14:12:59:618:000, Rx, 8, 0x0E0,        00 FF 21 00 10 01 00 00 ,    EMCY Node 96
14:12:59:618:000, Rx, 1, 0x750,        00 ,                   Heartbeat Node 80 Bootup
```

```
14:12:59:619:000, Rx, 8, 0x0D0,    00 FF 21 00 20 01 00 00 ,  EMCY Node 80
14:12:59:619:000, Rx, 1, 0x731,    00 ,                       Heartbeat Node 49 Bootup
14:12:59:621:000, Rx, 1, 0x720,    00 ,                       Heartbeat Node 32 Bootup
14:12:59:621:000, Rx, 8, 0x0A0,    00 FF 21 00 20 01 00 00 ,  EMCY Node 32
14:12:59:625:000, Rx, 1, 0x710,    00 ,                       Heartbeat Node 16 Bootup
14:12:59:634:000, Rx, 1, 0x080,    17 ,                       SYNC 0x17
14:12:59:654:000, Rx, 1, 0x080,    18 ,                       SYNC 0x18
14:12:59:674:000, Rx, 1, 0x080,    19 ,                       SYNC 0x19
14:12:59:694:000, Rx, 1, 0x080,    1A ,                       SYNC 0x1A
14:12:59:714:000, Rx, 1, 0x080,    1B ,                       SYNC 0x1B
14:12:59:725:000, Rx, 1, 0x701,    7F ,                       Heartbeat Node 1 PreOp
14:12:59:726:000, Rx, 8, 0x601,    40 00 10 01 00 00 00 00 ,  SDO Request to
Node 1
14:12:59:726:000, Rx, 8, 0x601,    40 00 10 02 00 00 00 00 ,  SDO Request to
Node 1
14:12:59:727:000, Rx, 8, 0x601,    40 00 10 03 00 00 00 00 ,  SDO Request to
Node 1
14:12:59:728:000, Rx, 8, 0x581,    4B 00 10 01 0A 00 00 00 ,  SDO Response from
Node 1
14:12:59:728:000, Rx, 8, 0x581,    43 00 10 02 28 00 00 00 ,  SDO Response from
Node 1
14:12:59:728:000, Rx, 8, 0x581,    43 00 10 03 00 00 04 00 ,  SDO Response from
Node 1
14:12:59:734:000, Rx, 1, 0x080,    1C ,                       SYNC 0x1C
14:12:59:754:000, Rx, 1, 0x080,    1D ,                       SYNC 0x1D
```

Appendix C

Data Logged by SCU

```
8 0372: <0x080> [1] ac

8 0372: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0372: <0x261> [1] 01

8 0373: <0x1d1> [8] 04 21 4e e8 03 00 7d 31

8 0384: <0x740> [1] 7f

8 0384: <0x730> [1] 7f

8 0384: <0x731> [1] 7f

8 0384: <0x720> [1] 7f

8 0384: <0x750> [1] 7f

8 0384: <0x760> [1] 7f

8 0392: <0x080> [1] ad

8 0392: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0393: <0x1d1> [8] 04 20 4e e8 03 00 7d 31

8 0393: <0x1e3> [8] 02 8c 05 7d 27 0e 24 0e

8 0404: <0x710> [1] 05

8 0412: <0x080> [1] ae

8 0412: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0412: <0x351> [2] a5 a5

8 0413: <0x263> [1] 01

8 0413: <0x1d1> [8] 04 21 4e e8 03 00 7d 31

8 0432: <0x080> [1] af
```

```
8 0432: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0433: <0x1d1> [8] 04 22 4e e8 03 00 7d 31

8 0452: <0x080> [1] b0

8 0452: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0453: <0x1d1> [8] 04 21 4e e8 03 00 7d 31

8 0453: <0x1e1> [8] 02 8d e2 7c 29 0e 23 0e

8 0472: <0x080> [1] b1

8 0472: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0472: <0x261> [1] 01

8 0473: <0x1d1> [8] 04 20 4e e8 03 00 7d 31

8 0483: <0x701> [1] 7f

8 0492: <0x080> [1] b2

8 0492: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0493: <0x1d1> [8] 04 22 4e e8 03 00 7d 31

8 0493: <0x1e3> [8] 02 8c 05 7d 27 0e 24 0e

8 0512: <0x080> [1] b3

8 0512: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0513: <0x351> [2] a5 a5

8 0513: <0x263> [1] 01

8 0513: <0x1d1> [8] 04 21 4e e8 03 00 7d 31

8 0532: <0x080> [1] b4

8 0532: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0533: <0x1d1> [8] 04 22 4e e8 03 00 7d 31

8 0544: <0x761> [1] 05
```

```
8 0544: <0x762> [1] 7f

8 0552: <0x080> [1] b5

8 0552: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0553: <0x1d1> [8] 04 20 4e e8 03 00 7d 31

8 0553: <0x1e1> [8] 02 8d e2 7c 29 0e 23 0e

8 0572: <0x080> [1] b6

8 0572: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0572: <0x261> [1] 01

8 0573: <0x1d1> [8] 04 21 4e e8 03 00 7d 31

8 0592: <0x080> [1] b7

8 0592: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0593: <0x1d1> [8] 04 20 4e e8 03 00 7d 31

8 0593: <0x1e3> [8] 02 8c 05 7d 27 0e 24 0e

8 0603: <0x741> [1] 7f

8 0604: <0x751> [1] 05

8 0612: <0x080> [1] b8

8 0612: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0612: <0x351> [2] a5 a5

8 0612: <0x263> [1] 01

8 0613: <0x1d1> [8] 04 20 4e e8 03 00 7d 31

8 0632: <0x080> [1] b9

8 0632: <0x251> [7] 02 20 4e 65 04 a5 a5

8 0633: <0x1d1> [8] 04 20 4e e8 03 00 7d 31
```