CANopen implementation in the Zagreb tramcar

Sinisa Marijan, Mario Bilic, Kresimir Ivanus

KONCAR – Electrical Engineering Institute, Zagreb, Croatia

The paper deals with the several topics related to the development and production of the ultra low-floor tramcar, type TMK2200, for the city of Zagreb. During the development many electronic control units have been specified, designed and integrated into the vehicle. The communication between these control units is mostly based on CANopen. The reasons for selecting CANbus and CANopen application layer are discussed. Furthermore, several proprietary hardware and software solutions have been developed for this project. These solutions, among other, include redundant main vehicle control unit. The concept of this unit is presented, along with some details that increase vehicle reliability and availability. Finally, some experience facts and possible future improvements are also pointed out.

1 Introduction

The project objective of this work is related to the development of ultra-low floor tramcar, type TMK 2200, for the city of Zagreb, Fig. 1. KONČAR – Electrical Engineering Institute was responsible for the development of main vehicle control unit, traction units, static converters for auxiliary power supplies and driver-machine interface. Further, our responsibility was the choice of appropriate communication busses and accordingly, the system integration of all electronic control units. This is generally a rather demanding task. However, this paper deals only with the communication networks in the vehicle and the Vehicle Control Unit (VCU). Other parts are mentioned only where needed. Details on control units are given in [1] and [2]. Trends in light rail vehicles development concerning electronic control units have been significantly influenced by modern solutions already implemented in industry applications and especially in road vehicles, [10]. Solutions implemented in road vehicles are in many cases used in other high-tech segments such are: avionics, military and railway. Modern control and communication solutions demand a high level of availability, reliability and maintainability. Long life, with possible future improvements, simple integration and commissioning should also be supported. To achieve these goals, and taking into account the fact that sophisticated systems are usually integrated through equipment of different sub-suppliers, system integrators are facing challenges when interfacing the equipment.

Figure 1: TMK2200 ultra low-floor-tram

One demanding task during development of 100% low floor tramcar is a limited space under the floor. Therefore, all equipment are integrated into roof containers or into, also limited, space between the roof and passenger compartment. Here, the demand for fully air-conditioned tramcar puts additional requirements on the roof equipment.

Section 2 of this paper briefly describes main vehicle control unit. Sections 3 and 4 deal with the communication infrastructure and CANopen implementation. Section 5

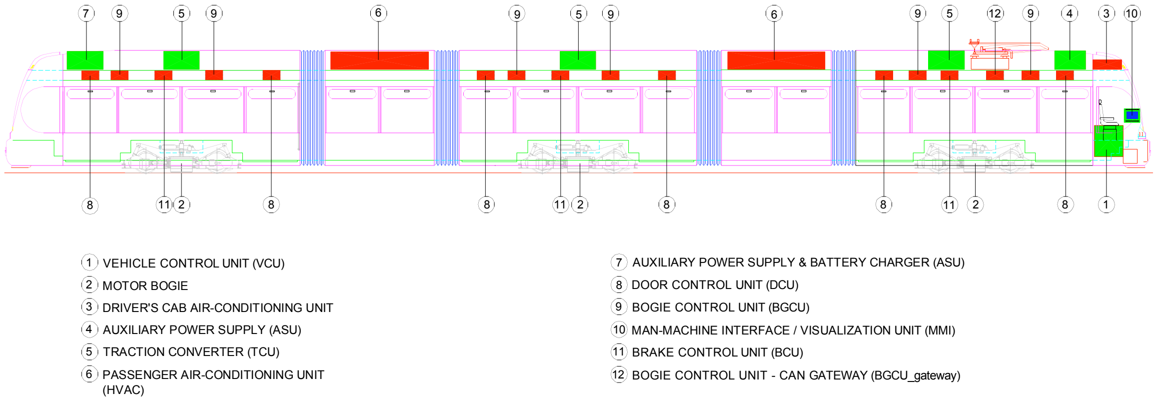

Figure 2: The position of electronic control units (ECUs) in the tramcar

points out some issues related to the CANbus utilization calculation, while section 6 is related to the commissioning and diagnostic tools. Section 7 presents some experience facts along with the possible future improvements.

2 Proprietary solutions

Taking into account electronic control units (ECUs) integrated into this tramcar, Fig. 2, proprietary developed units are: main vehicle control unit (VCU), 3 traction control units (TCU), 2 auxiliary power supplies (static converters) control units (ASU), 1 visualization unit/man-machine interface (MMI). Other suppliers delivered 3 brake control units (BCU), 2 heating/ventilation and air conditioning control units (HVAC), 6 door control units (DCU) and 7 bogie control units (BGCU).

Physical position of the above described units is given in Figure 2.

2.1 Vehicle control unit

The photo of the VCU is given in Figure 3. VCU frame consists of two 19” racks. The first 19” rack is used during normal operation as an active system, while the second one is used as redundant one. The core of the VCU are VMEbus based central processing module (CPM) and double channel CAN communication module, [1], [2].

Apart from redundant channel that can be used in a case of failure, the high error detection coverage has been required from operational channel to put the system in the fail-safe operation in case of malfunctioning. Fail-safe means that no undefined system state is allowed and therefore if such a case occurs,

Figure 3: Vehicle control unit (VCU)

appropriate actions will activate the process of stopping the vehicle, disconnecting the power line and sending appropriate message to the driver. To support this, a lot of additional hardware and software mechanisms have been implemented to detect different types of errors. Some of them that start fail-safe routines need to be mentioned here:

- CPM communicates with peripherals asynchronously by means of acknowledge signal. If it misses, CPM will try to perform recovery action and if it also fails an appropriate hardware error will be set

- software or hardware exception errors
- each power sub-system failure detection
- software watchdog that monitors system and application programs
- external watchdog that monitors processor and other vital components;
- controlled timing between power-off/on sequences to avoid risks related to vehicle battery malfunctioning
- malfunctioning of memory components
- failure of both CANbus channels

2.2 In-House software development

Software environment consists of 2 main parts: the system software and the integrated development environment (IDE), i.e. application program development tools that enable application program development. The basic principles, IDE and system software concept are the same, regardless of the hardware, i.e. of the processor type [1], [2]. Thus, all processors, controllers and even digital signal processors used in VCU, TCU and ASU have the same IDE for the application program development. It is obvious that the same or similar environments (component based development) can help in reducing development costs. Real–time scheduling policies used are mostly based on fixed priority rate monotonic scheduling algorithms, [7].

One of the unusual features, at least for modern development processes, is the fact that all the software for proprietary solutions is in-house developed, and this has been done in assembler. It means that both the system software and the block-diagram based IDE are assembler-based, in-house developed and thus completely under control of our own software developers.

2.3 Other sub-suppliers

The ECUs of other sub-suppliers are integrated by following the basic rule: critical functions are hard-wired and in the same time supported through communication networks. The only exceptions are bogie control units (BGCU) that communicate only through private CAN_3 communication network.

3 Communication infrastructure

During the project planning and initial project phase there was a lot of discussion about appropriate physical layer and communication protocol. The following solutions have been considered: RS485 physical layer with in-house protocol; RS485 physical layer with ModBus protocol; MVB (multifunction vehicle bus); FlexRay; CAN physical layer with custom protocol; CAN physical layer with CANopen protocol; CAN physical layer with TTCAN (time-triggered CAN).

RS485 with either in-house or ModBus protocol is a simple and cheap solution for system integrators. However, due to its limitations and opinions of other suppliers, it was considered only as the solution for connecting proprietary equipment. MVB as a part of Train Communication Network (TCN) international standard, [5], would be the most appropriate solution. TCN defines two hierarchical interfaces as connections to a data communication network. The first one called Wired Train Bus (WTB) is used for interconnecting vehicles in "Open Trains" such are international UIC trains. The second one, called Multifunction Vehicle Bus (MVB), is

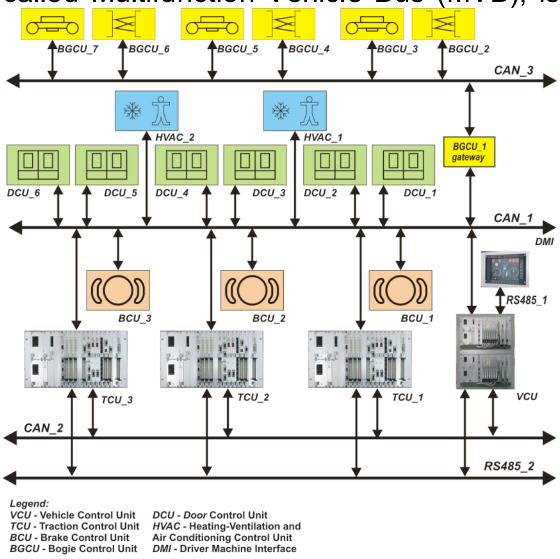

Figure 4: TMK2200 communication busses

used for connecting standard on-board equipment. MVB type of interface requires implementation of a proprietary ASIC (application specific integrated circuit) or FPGA (field programmable gate array). However, the lack of support on the component basis and development tools would demand high development costs. Furthermore, other suppliers preferred different solutions. TTCAN (or some of its derivatives), [12], and particularly FlexRay, [8], [9], are in many ways better and more technologically advanced then CAN itself. However, they are still under development (FlexRay) or have not reached the availability and support status of CAN, [3], [4], [10]. Therefore, CAN appeared to be a good base for such an application. Furthermore, as expected, all equipment sub-suppliers have encouraged the CAN use.

Finally, it was decided to build the system around three independent CAN busses with CANopen protocol and two RS-485 proprietary busses, Fig. 4. RS485 networks are used to connect proprietary equipment, while CAN_1 network connects all tramcar control units (except auxiliary power supplies). CAN_2 connects only VCU with TCUs, thus enabling redundancy on the system level. It is obvious that VCU is responsible for almost all data transactions. Therefore, high demands related to reliability and availability were put on VCU during development.

4 CANopen implementation

CANopen functions applied are reduced when compared to all given possibilities, i.e. specific application profiles are not used. Instead, the CiA ds301, [6], document that specifies what minimal functionality a CANopen device must provide, was a base for building a CAN communication network.

Because the majority of safety-relevant functions are also hard-wired, the vehicle is functional even in the case of CAN_1 or CAN_2 failure. Due to the large number of demanding nodes in the CAN_1 network, the amount of data that are to be transferred is relatively high for this type of application. CANopen uses object-oriented approach and defines communication objects. Process data objects are used for real-time data transfer. Process Data Object (PDO) distribution for CAN_1 is given in Table 1. Second column gives priorities of the messages, where the lowest number indicates the highest priority. PDO transmission type was chosen, according to CANopen, to be 254 (manufacturer specific). After some tests it was decided to trigger the message each time when appropriate transmit PDO (TxPDO) event

PDO TYPE & DESTINATION	PDO PRIORITY / COB-ID	PDO EVENT TIMER SETTINGS	PDO NUMBER & SOURCE
TX			
TCU_1-3	1 / 184	50 ms	PDO#1_VCU
TCU_1-3	5 / 188	50 ms	PDO#2_VCU
BCU_1	12 / 1B0	10 ms	PDO#3_VCU
BCU_2	13 / 1B1	10 ms	PDO#4_VCU
BCU_3	14 / 1B2	10 ms	PDO#5_VCU
BGCU	17 / 1C3	150 ms	PDO#6_VCU
DCU_1	18 / 201	100ms	PDO#7_VCU
DCU_2	19 / 202	100 ms	PDO#8_VCU
DCU_3	20 / 203	100 ms	PDO#9_VCU
DCU_4	21 / 204	100 ms	PDO#10_VCU
DCU_5	22 / 205	100 ms	PDO#11_VCU
DCU_6	23 / 206	100 ms	PDO#12_VCU
TCU_1-3	30 / 384	300 ms	PDO#13_VCU
TCU_1-3	34 / 388	300 ms	PDO#14_VCU
BCU_1	41 / 3B0	200 ms	PDO#15_VCU
BCU_2	42 / 3B1	200 ms	PDO#16_VCU
BCU_3	43 / 3B2	200 ms	PDO#17_VCU
HVAC12	45 / 3E0	250 ms	PDO#18_VCU
TIMESTAMP	0 / 100	10000 ms	PDO#54_VCU
RX			
	2 / 185	50 ms	PDO#19_TCU1
	6 / 189	50 ms	PDO#20_TCU1
	3 / 186	50 ms	PDO#21_TCU2
	7 / 18A	50 ms	PDO#22_TCU2
	4 / 187	50 ms	PDO#23_TCU3
	8 / 18B	50 ms	PDO#24_TCU3
	9 / 1A0	10 ms	PDO#25_BCU1
	10/ 1A1	10 ms	PDO#26_BCU2
	11/ 1A2	10 ms	PDO#27_BCU3
	15 / 1C0	150 ms	PDO#28_BGCU
	16 / 1C1	150 ms	PDO#29_BGCU
	24 / 211	100ms	PDO#30_DCU1
	25 / 212	100 ms	PDO#31_DCU2
	26 / 213	100 ms	PDO#32_DCU3
	27 / 214	100 ms	PDO#33_DCU4
	28 / 215	100 ms	PDO#34_DCU5
	29 / 216	100 ms	PDO#35_DCU6
	31 / 385	300 ms	PDO#36_TCU1
	35 / 389	300 ms	PDO#37_TCU1
	32 / 386	300 ms	PDO#38_TCU2
	36 / 38A	300 ms	PDO#39_TCU2
	33 / 387	300 ms	PDO#40_TCU3
	37 / 38B	300 ms	PDO#41_TCU3
	38 / 3A0	200 ms	PDO#42_BCU1
	39 / 3A1	200 ms	PDO#43_BCU2

	40 / 3A2	200 ms	PDO#44_BCU3
	44 / 3C0	1000 ms	PDO#45_BGCU
	46 / 3E1	1000 ms	PDO#46_HVAC
	47 / 3E2	1000 ms	PDO#47_HVAC
	48 / 3E9	1000 ms	PDO#48_HVAC
	49 / 3EA	1000 ms	PDO#49_HVAC
	50 / 3F1	1000 ms	PDO#50_HVAC
	51 / 3F2	1000 ms	PDO#51_HVAC
	52 / 3F9	250 ms	PDO#52_HVAC
	53 / 3FA	250 ms	PDO#53_HVAC

Table 1: Distribution of process data objects

timer elapses. The adjustments of event timers are given in column 3. Table 1 gives only the distribution of main PDOs. Apart from them, there are also other objects on the network (heartbeat, emergency objects, service data objects). The principle of CANopen implementation into main VCU is explained in Figure 5.

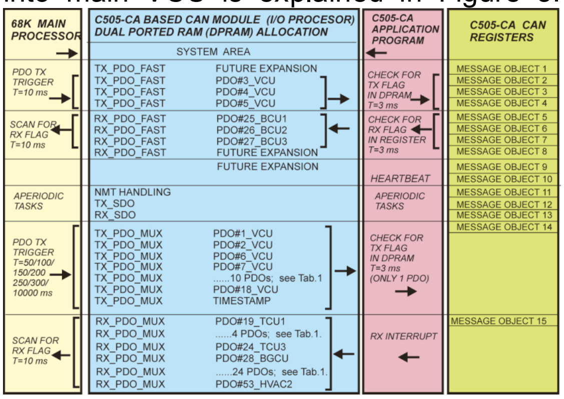

Figure 5: CANopen objects in VCU (CAN_1 bus)

Although it describes only CAN_1 channel, the same principle applies for the CAN_2.

Each CAN channel has it's own C505-CA processor and dual-ported RAM (DPRAM) for exchanging data with the main processor (CPM). Assigned address space on main processor side of DPRAM is divided into message buffers. Each message buffer can be assigned to one PDO described with its COB-ID, length and direction. C505-CA controller has got an integrated CAN controller on chip with 15 message buffers. The distribution of these buffers is also given in Fig. 5. They are the same, except message object 15 that is based on double buffer principle. PDO messages are, Fig. 5, divided into three groups: fast parallel PDOs, multiplexed transmitting PDOs, and multiplexed receiving PDOs. Fast parallel PDO messages are in fact copied from or to DPRAM according to the chosen direction. These messages are processed with 3-milliseconds period by pooling assigned flags. In case of reception, flag in message buffer on integrated CAN controller is checked, and in case of transmission, flag in DPRAM buffer is checked. Multiplexed transmitting PDOs are checked for their transmission flag in a circular manner with 3-milliseconds period. Only one of these messages is sent in this period. These messages are processed with the same priority as eight fast parallel messages. Multiplexed receiving PDOs use a special double-buffered receiving message object 15. It enables the processing of one message while another one is being received. This message is processed in interrupt procedure with the highest priority where it is saved in the temporary buffer to avoid DPRAM hand-shaking delays. Messages from the temporary buffer are copied to DPRAM in lower priority task. This concept introduces significant jitter in message transaction time and can even lead to message loss but is, according to the experience, acceptable and reliable.

5 CANbus utilization (load)

CAN is priority based protocol and when dealing with scheduling algorithms, the support of non-preemptive fixed priority scheduling can be considered. Due to the fact that the appropriate timer initiates PDOs transmission (Tx_PDO) and VCU transmits or receives all messages on the bus, the CANbus load can easily be calculated, [13], by means of the Table 1 and the following equation:

$$\text{CANbus\_load} = \sum_{1 \le i \le n} \frac{ML_i}{T_i}, \qquad (1)$$

where i=PDO number, n=number of TxPDOs on the bus, ML_i=length of i-th message, T_i=time base written in appropriate event timer. Each PDO, except DCU objects, is 8 bytes long. DCU objects are 2 bytes long. CAN_1 speed is 250 kBits/sec. Calculated load for CAN_1 that supports 54 PDOs is 45.2%. During the commissioning 48% was measured. The difference is caused by the fact that stuffing bits were ignored in the calculation, i.e. the message length of 111 bits (for 8 byte user data) was assumed.

6 Commissioning and diagnostic tools

In the early project phases dilemma was whether to use a commercial tool or to develop a proprietary solution for CANbus commissioning and monitoring. The final decision was to develop proprietary software solution based on commercially available CAN/USB hardware. As the development team had no experience with the CAN/CANopen applications it was a good chance to get familiar with this type of applications. Also, the flexibility is higher with proprietary software because of frequent customer demands for updates and changes. This gives the possibility to adjust the software according to new demands in a short time; usually not possible with commercial solutions.

Main features of the above mentioned tools are: easily upgradable but completely application related; ability to make record of complete CAN traffic regardless of baud rate and bus load; two CAN channels can be processed simultaneously; opportunity for detailed offline analysis of CAN traffic; no special technical skills required during commissioning; user interfaces according to the specific customer needs.

Three different tools for CAN network commissioning, development, monitoring and diagnostic were developed. First tool, called CBC (CanBusCommissioning) is intended for use during tramcar and CAN bus commissioning, Fig. 6. This tool is designed to be used by non-experienced personal, i.e. persons without CANopen knowledge. It gives a brief overview of CAN bus status and is capable of detecting some types of hardware errors (e.g. errors in wiring or control units physical layer) or errors in CAN nodes (e.g. missing PDO or incorrect PDO timing). Expected messages COBID and their time-out period are pre-defined in configuration file. Main advantage is that all errors that CAN/USB interface detects are presented and counted. The tool detects hardware and software overflows (situations when received messages aren't handled fast enough), error frames on the bus and errors in message content (stuffing error, form error and CRC error).

For each message there is one row in table, denoted “2” in Fig. 6, that presents message COBID, short description, expected period and current status. Statuses are: OK meaning that everything

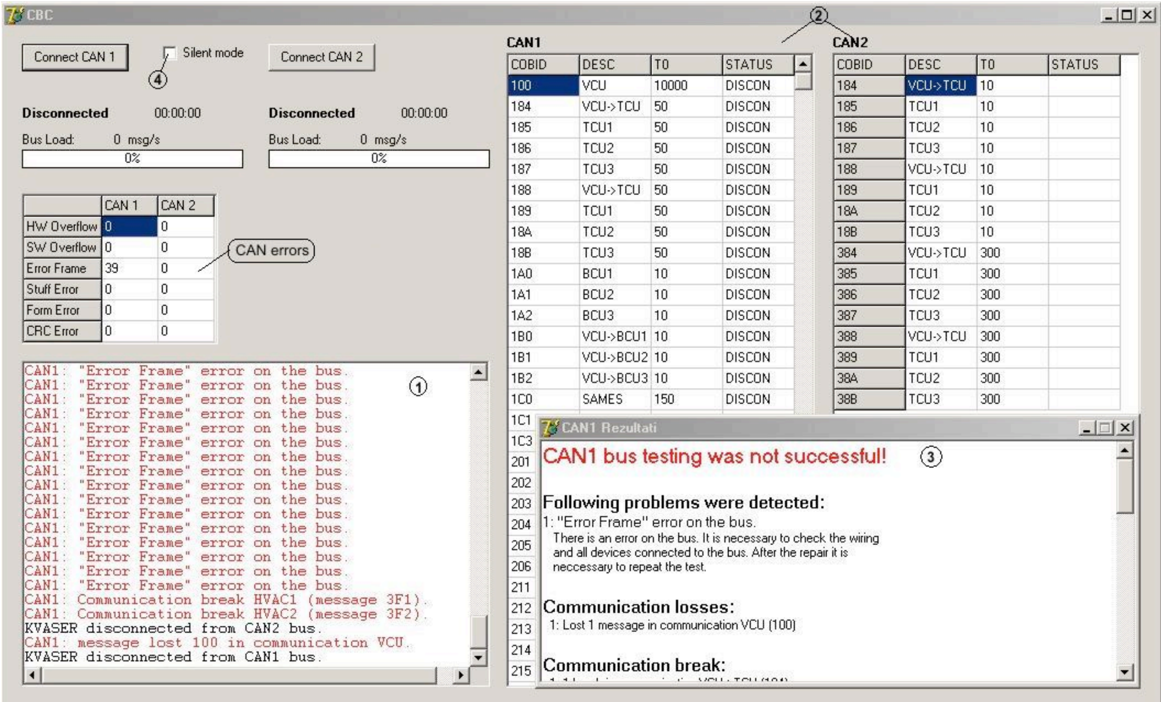

Figure 6: CANbus comissioning tool

is as expected; LOSS meaning that one or more messages is lost (message is considered lost if it's missing longer than 3 and less than 10 expected time-out periods); BREAK meaning that there is a communication break with node sending message with these COBID (break is declared when message is missing for more than 10 expected time-out periods); into log files on PC hard disk. Each message is logged with its COBID, time stamp, time between current and previous message with the same COBID, some flags and message content. Time stamp can be recorded with 10 μs resolution what enables off-line analysis of the bus traffic, statistical analysis of recorded data and advanced error detection, not only

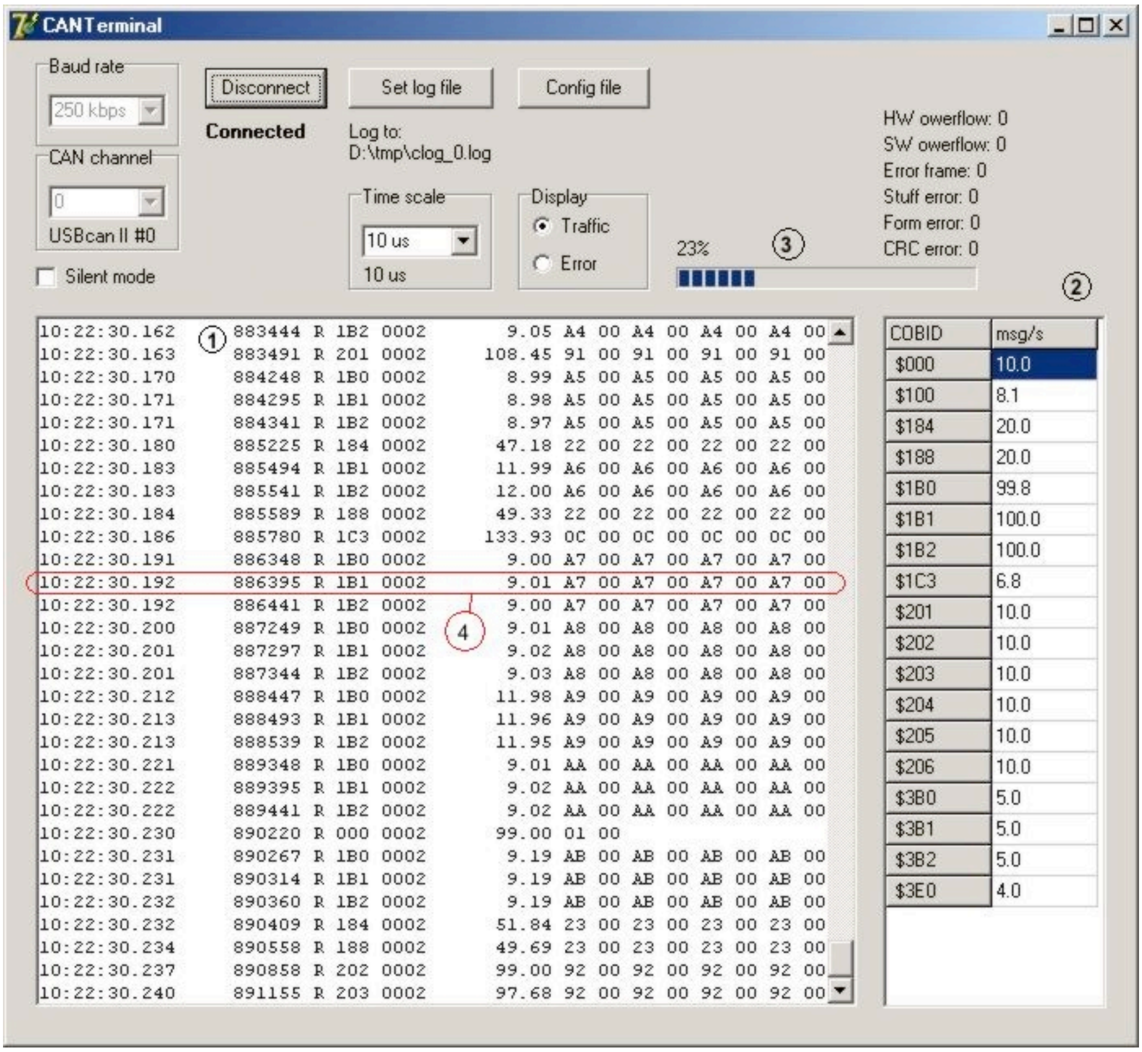

Figure 7: CAN terminal user interface

and DISCON meaning that recording has finished and CAN/USB test tool has disconnected itself. After the test is over, the window “3” gives clear report with all detected problems and short instructions for user how to handle some type of errors.

Second tool, CANTerm (CANbus Terminal), Fig. 7., is developed for use during CAN hardware and application program testing. This software tool is capable of logging complete CAN traffic related to CAN communication but also to the equipment connected to the CAN bus. User can select baud rate, one of two CAN channels, silent mode in which CAN/USB interface doesn't send anything to the network. There is also a window, “1”, showing bus traffic or recorded errors. Bus load is also shown, as well as list of all messages occurred on the bus, together with average load for each message (number of messages per second). Tool automatically splits recorded data into multiple files to reduce single file size and to simplify processing a large amount of data. Recorded data, for one channel, take about 250 MB for 1 hour at 50% bus load and speed of 250 kbps.

Data are recorded to the log file following the format marked with “4”. At the beginning of each file is a message time stamp in format hh:mm:ss:ms, then, there is time stamp represented as number of milliseconds or part of millisecond (depending on selected time scale) since connection. Next, there is a message direction, where "R" means that message is received, and "T" means that the message was sent by this tool. Next, COBID is given in hexadecimal data format. After COBID there is a four digit flag containing additional information about message, like message type and possible errors. Next, there is time since last message with the same COBID, in milliseconds. Last part of the displayed message is a message content. Each transferred byte is represented as a two digit hexadecimal number

The third tool, CANLogAn (CANLogger Analyzer), analyses data recorded by means of CANTerm. This tool calculates the distribution of time-outs between messages and detects problems in communication, lost messages etc. Such an information can also be a good criterion for grading overall communication quality.

7 Conclusion and future development

After system integration, commissioning and drive tests the vehicle was put into regular operation and series production is under way. The reliability and functionality of the control units and CAN busses appears to be as expected. For the time being, the CANbus load and processing power of proprietary solutions are sufficient, but the demand of higher calculation and communication possibilities is expected in the future projects. When VCU is an issue, processing power can be added by introducing additional module in multi-processing configuration, [1], [2]. Possible future problems with the limitations of 8-bit CAN controller can be solved with 16-bit controller, but in the same time there are faster 8-bit (8051-compatible) solutions available. Further tasks will also be the consideration of MVB and particularly FlexRay. Appropriate FlexRay hardware components are expected to be widely available in the very near future.

References

[1] S. Marijan: *Vehicle Control Unit for the Light Rail Applications.* 13th International Conference on Electrical Drives and Power Electronics, EDPE 2005, Dubrovnik, Croatia, 2005.

[2] S. Marijan: *Control Electronics of TMK2200 Type Tramcar for the City of Zagreb*. International Symposium on Industrial Electronics, ISIE 2005, Dubrovnik, 2005, volume IV, pp. 1617-1622.

[3] ISO 11898-1: *Road vehicles - Controller area network (CAN), Part 1: Data link layer and physical signaling.* International Organization for Standardization, 2003.

[4] ISO 11898-2: *Road vehicles - Controller area network (CAN), Part 2: High-speed medium access unit.* International Organization for Standardization, 2003.

[5] IEC 61375-1: *Electric railway equipment – Train bus, Part 1: Train Communication Network.* International Electrotechnical Commission, 1999.

[6] CiA ds301: *CANopen: Application Layer and Communication Profile.* CAN in Automation Draft Standard 301, version 4.02, 2002.

[7] L. Sha et al.: *Real Time Scheduling Theory: A Historical Perspective.* Real-Time Systems, Volume 28, Issue 2 - 3, Nov 2004, pp 101–155.

[8] FlexRay Consortium: *FlexRay Communications System, Electrical Physical Layer Specification, version 2.0,* 2004.

[9] FlexRay Consortium: *FlexRay Communications System, Protocol Specification, version 2.0*, 2004.

[10] M. Farsi and M. Barbosa: *CANopen implementation: applications to industrial networks.* Research Studies Press Ltd., 2000.

[11] H. Kopetz: *Real-Time Systems, Design Principles for Distributed Embedded Applications.* 3rd ed. Kluwer Academic Publishers, 1999.

[12] A. Albert, R. Strasser and A. Trächtler: *Migration from CAN to TTCAN for a Distributed Control System.* CAN in Automation, pp.05-9-05-16., 9th international CAN Conference, iCC, Munich, 2003.

[13] C.L.Liu, J.W.Layland: *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.* Journal ACM 20, 1 (Jan. 1973), pp.46-61.